

# Machine Learning Methods in Geoscience

Gerard T. Schuster with labs by Yuqing Chen, Yongxiang Shi, Shihang Feng,  
Zhaolun Liu, Zongcai Feng, Yunsong Huang, Yan Yang and Tushar Gautam

March 29, 2022





# Contents

<b>Preface</b>	<b>xv</b>
<b>Notation Convention</b>	<b>xxiii</b>
<b>Abbreviations</b>	<b>xxvii</b>
<b>1 Introduction to ML in Geosciences</b>	<b>1</b>
1.1 Introduction . . . . .	1
1.2 Three Classes of Machine Learning . . . . .	5
1.2.1 Supervised Learning . . . . .	5
1.2.2 Unsupervised Learning . . . . .	7
1.2.3 Reinforcement Learning . . . . .	7
1.3 Summary . . . . .	9
<b>I Mathematical Background</b>	<b>13</b>
<b>2 Data Prediction by Least Squares Inversion</b>	<b>15</b>
2.1 Least Squares Inversion . . . . .	15
2.1.1 Overdetermined, Inconsistent, and Ill-conditioned Equations . . . . .	16
2.1.2 Least Squares Solution . . . . .	16
2.1.3 Regularized Least Squares Solution . . . . .	21
2.1.4 Geometrical Interpretation of Gradient $\nabla\epsilon$ . . . . .	26
2.1.5 Preconditioning . . . . .	28
2.1.6 Overfitting Data . . . . .	29
2.1.7 Inclusion of Bias Factor . . . . .	34
2.2 Steepest Descent Optimization . . . . .	34
2.3 Non-Linear Optimization . . . . .	36
2.3.1 MATLAB Examples of Newton's Method . . . . .	37
2.4 Summary . . . . .	41
2.5 Exercises . . . . .	42
2.6 Appendix: Exact Step Length . . . . .	46

<b>3</b>	<b>Specialized Gradient Descent Methods</b>	<b>49</b>
3.1	Ad Hoc Gradient Descent Methods . . . . .	49
3.1.1	Momentum Gradient Descent . . . . .	49
3.1.2	Adagrad Gradient Descent . . . . .	56
3.1.3	Adadelta Gradient Descent . . . . .	58
3.1.4	Nesterov Accelerated Gradient (NAG) . . . . .	59
3.1.5	Adam Gradient Descent . . . . .	60
3.1.6	AdaMax Gradient Descent . . . . .	63
3.1.7	Hybrid Step-length Strategy . . . . .	63
3.2	Numerical Line Search . . . . .	64
3.3	Conjugate Gradient . . . . .	64
3.3.1	MATLAB Conjugate Gradient Code . . . . .	67
3.4	Summary . . . . .	70
3.5	Exercises . . . . .	71
3.6	Appendix: Rosenbrock Plotting Code . . . . .	73
3.7	Appendix: Step Length Plotting Codes . . . . .	73
<b>II</b>	<b>Supervised Learning</b>	<b>75</b>
<b>4</b>	<b>Introduction to Neural Networks</b>	<b>77</b>
4.1	Introduction . . . . .	78
4.1.1	Biological Inspiration of a Neural Network . . . . .	80
4.1.2	General Neural Network . . . . .	83
4.1.3	Geometrical Interpretation of a Neural Network . . . . .	86
4.1.4	Generality of a Neural Network . . . . .	88
4.1.5	Neural Network Workflow . . . . .	91
4.1.6	Recursive Feed-forward Formula . . . . .	91
4.1.7	Recursive Back-propagation Formula . . . . .	93
4.2	Activation Functions . . . . .	95
4.3	Examples of Neural Networks . . . . .	98
4.3.1	Single-Layer Neural Network . . . . .	99
4.3.2	Two-layer Neural Network . . . . .	100
4.3.3	Neural Network MATLAB Code . . . . .	103
4.3.4	NN with a Cross-Entropy Objective Function . . . . .	104
4.4	Types of Classification . . . . .	111
4.4.1	Multilabel Classification . . . . .	111
4.4.2	Multiclass Classification . . . . .	111
4.5	Best Practices . . . . .	114
4.6	Summary . . . . .	121
4.7	Exercises . . . . .	124
4.8	Computational Labs . . . . .	132
4.9	Appendix: Neural Network Workflow . . . . .	132
4.10	Appendix: MATLAB Code for Validation . . . . .	135

<b>5</b>	<b>Processing Data with FCNN</b>	<b>137</b>
5.1	FCNN Filtering of Migration Images in $\mathbf{z} = (x, z)$	137
5.2	Filtering Angle-Domain Images	141
5.2.1	Angle Domain Transform	141
5.2.2	Filtering Marine Data	143
5.3	NN Filtering of Surface Waves	144
5.4	MATLAB Codes	147
5.5	Summary	149
5.6	Exercises	150
5.7	Computational Labs	150
5.8	Appendix: skeletonized Characteristics in an Image	150
5.9	Appendix: MATLAB Code for Logistic Regression	151
<b>6</b>	<b>Multilayer FCNN</b>	<b>155</b>
6.1	Introduction	155
6.2	Feed-forward Operation	155
6.3	Back-propagation Operation	156
6.3.1	Formula for $\partial\epsilon/\partial w_{jk}^{[N]}$	157
6.3.2	Formula for $\partial\epsilon/\partial w_{ij}^{[N-1]}$	157
6.3.3	Formula for $\partial\epsilon/\partial w_{ij}^{[N-2]}$	158
6.4	MATLAB Code	160
6.5	Summary	162
6.6	Exercises	163
6.7	Computational Labs	163
6.8	Appendix: Vectorized Steepest Descent Formula	163
<b>7</b>	<b>Support Vector Machines</b>	<b>165</b>
7.1	Introduction	165
7.2	SVM Theory	166
7.2.1	SVM Applications	167
7.3	Linear SVM	169
7.4	Nonlinear SVM	172
7.5	Primal and Dual Solutions	174
7.6	Kernel Methods	177
7.7	Numerical Examples	181
7.7.1	MATLAB Codes	182
7.8	Multiclass SVM	183
7.9	Hinge-Loss and Soft-Margin SVM	183
7.10	SVM Lab with CoLab	187
7.11	SVM Regression	192
7.12	Practical Issues for Implementing SVM	192
7.12.1	Scaling	193
7.12.2	Data Augmentation	193
7.12.3	Computational Cost	193
7.12.4	Procedures for Users	195

7.13	Summary . . . . .	195
7.14	Exercises . . . . .	197
7.15	Computational Labs . . . . .	197
7.16	Appendix: Defining the Dual Problem with a Lagrangian . . . . .	197
7.16.1	Simple Example of a Dual Solution . . . . .	200
7.16.2	Concavity and Maximization of the Reduced Lagrangian . . . . .	202
7.17	Appendix: Karush-Kuhn-Tucker Conditions . . . . .	203
<b>III</b>	<b>Convolutional Neural Networks</b>	<b>205</b>
<b>8</b>	<b>Convolution and Correlation</b>	<b>207</b>
8.1	Convolution . . . . .	207
8.1.1	Time or Shift Invariance . . . . .	208
8.1.2	1D Convolution . . . . .	210
8.1.3	2D Convolution . . . . .	212
8.1.4	3D Convolution . . . . .	213
8.2	Correlation and Matched Filters . . . . .	213
8.2.1	Multilayer and Multifilter Convolution . . . . .	217
8.3	Summary . . . . .	220
8.4	Exercises . . . . .	220
<b>9</b>	<b>Convolutional Neural Networks</b>	<b>223</b>
9.1	Introduction . . . . .	223
9.1.1	Brief History of CNN . . . . .	224
9.2	Building Blocks of CNN . . . . .	227
9.2.1	CNN Forward Modeling . . . . .	228
9.2.2	Model Update by Back-propagation . . . . .	233
9.2.3	Keras Code . . . . .	235
9.2.4	Visualizing the Filters as Correlation Functions . . . . .	238
9.2.5	Visualizing the Images Associated with Feature Maps . . . . .	240
9.2.6	Regularization . . . . .	242
9.2.7	Mini-Batch, Step Length, Training and Data Augmentation . . . . .	246
9.3	Architectures of CNN . . . . .	247
9.3.1	AlexNet . . . . .	247
9.3.2	VGGNet . . . . .	248
9.3.3	ResNet . . . . .	248
9.4	Deep Learning Software . . . . .	250
9.5	Seismic Fault Detection by CNN . . . . .	251
9.6	Summary . . . . .	253
9.7	Exercises . . . . .	254
<b>10</b>	<b>Object Identification by CNN</b>	<b>257</b>
10.1	Introduction . . . . .	257
10.2	Image Classification . . . . .	258
10.2.1	Global Picture-size Classification . . . . .	258

10.2.2	Localize and Classify a Single Object . . . . .	258
10.2.3	Judging the Accuracy of the Predicted BB: IOU . . . . .	262
10.2.4	Localize and Classify Multiple Objects . . . . .	262
10.2.5	Pixel-size Classification: Semantic Segmentation . . . . .	270
10.3	Labeling Tools . . . . .	270
10.4	Salt Body Detection by AlexNet . . . . .	272
10.4.1	AlexNet CNN Architecture . . . . .	273
10.4.2	Numerical Results . . . . .	273
10.4.3	Monoparameter CNN . . . . .	273
10.4.4	Multiparameter CNN . . . . .	274
10.5	Detection of Rock Cracks by AlexNet . . . . .	274
10.5.1	Unbalanced Crack Data . . . . .	276
10.5.2	Numerical Results . . . . .	276
10.6	Detection of Basalt in GPR Images . . . . .	278
10.6.1	Numerical Results . . . . .	281
10.7	Fault Detection and Deconvolution . . . . .	283
10.8	Salt Boundary Detection by U-Net . . . . .	286
10.8.1	Keras U-Net Code . . . . .	290
10.8.2	Numerical Results . . . . .	293
10.9	Fault Detection by U-Net . . . . .	298
10.10	R-CNN Labeling of Blood Cells . . . . .	298
10.11	Small Object Detection by Superpixel-based CNN . . . . .	301
10.11.1	Superpixel Convolution Method . . . . .	302
10.11.2	Numerical Results . . . . .	303
10.12	Super-resolution Image Enhancement by CNN . . . . .	307
10.13	Identification of Bird Types by CNN . . . . .	308
10.13.1	Data Description and AlexNet model . . . . .	309
10.13.2	Numerical Results . . . . .	309
10.14	Summary . . . . .	310
10.15	Exercises . . . . .	310
10.16	Computational Labs . . . . .	312
10.17	Appendix: Selective Search . . . . .	312
10.18	Appendix: MATLAB Code for AlexNet Crack Detection . . . . .	314
10.19	Appendix: Simple Linear Iterative Clustering (SLIC) . . . . .	316
<b>11</b>	<b>Semantic Segmentation of Cracks in Photos</b>	<b>323</b>
11.1	Introduction . . . . .	323
11.2	Detection and Labeling of Cracks by U-Net . . . . .	323
11.2.1	Labeling Cracks . . . . .	324
11.2.2	Phase I Training and Validation . . . . .	325
11.2.3	Phase I Testing . . . . .	327
11.2.4	Phase II Training and Validation . . . . .	328
11.2.5	Phase II Testing . . . . .	328
11.3	Transfer Training . . . . .	332
11.4	Summary . . . . .	333

11.5	Computational Labs . . . . .	333
<b>12</b>	<b>Recurrent Neural Networks</b>	<b>337</b>
12.1	Theory of Recurrent Neural Networks . . . . .	338
12.1.1	Vanilla RNN Workflow . . . . .	338
12.1.2	Vanishing Gradient of VRNN . . . . .	344
12.2	Gated Recurrent Unit Network . . . . .	348
12.3	Training RNNs . . . . .	351
12.3.1	Truncated Back-propagation . . . . .	351
12.3.2	Bidirectional RNN . . . . .	351
12.3.3	Deep RNN . . . . .	352
12.4	RNN Applications in Geoscience . . . . .	354
12.4.1	Predicting Missing Well Logs by BRNN . . . . .	354
12.4.2	Predicting Well Logs from Seismic Reflection Data . . . . .	354
12.5	Summary . . . . .	356
12.6	Exercises . . . . .	358
12.7	Computational Labs . . . . .	358
12.8	Appendix: Vanishing Gradients . . . . .	358
12.9	Appendix: Long Short-Term Memory Architecture . . . . .	361
<b>13</b>	<b>Self-attention and Transformers</b>	<b>367</b>
13.1	Attention . . . . .	367
13.2	Trained Attention . . . . .	370
13.2.1	Single- and Multi-head Attention Architectures . . . . .	374
13.3	Transformer Architecture and Training . . . . .	378
13.4	Advances in Transformer Models . . . . .	379
13.4.1	Transformer Models and Event Picking in Seismograms . . . . .	380
13.5	Summary . . . . .	383
13.6	Exercises . . . . .	384
<b>IV</b>	<b>Unsupervised Learning</b>	<b>385</b>
<b>14</b>	<b>Autoencoders</b>	<b>387</b>
14.1	Theory of Autoencoders . . . . .	390
14.1.1	Linear Autoencoder and PCA . . . . .	393
14.2	Problems with Autoencoders . . . . .	394
14.3	Regularization of Autoencoders . . . . .	395
14.3.1	Dropout . . . . .	397
14.3.2	Denosing Autoencoders . . . . .	397
14.3.3	Contractive Autoencoders . . . . .	398
14.3.4	Sparse Autoencoders . . . . .	399
14.3.5	Variational Autoencoder . . . . .	400
14.3.6	Adversarially Constrained Autoencoder Interpolation . . . . .	404
14.4	Numerical Examples . . . . .	406
14.4.1	Seismic Interpolation Example . . . . .	406

14.4.2	Seismic Tomography . . . . .	407
14.4.3	4D Seismic Image Monitoring and Forecasting . . . . .	411
14.5	Summary . . . . .	414
14.6	Exercises . . . . .	415
14.7	Appendix: Autoencoder in Keras . . . . .	416
14.8	Appendix: Dropout=Regularization . . . . .	419
14.8.1	Least Squares Minimization with Dropout . . . . .	420
14.9	Appendix: KL Regularization . . . . .	421
<b>15</b>	<b>Convolutional Sparse Coding</b>	<b>423</b>
15.1	Introduction . . . . .	423
15.2	Theory . . . . .	424
15.2.1	Generalization of the Objective Function . . . . .	425
15.2.2	Workflow . . . . .	426
15.3	Numerical Examples . . . . .	427
15.3.1	Synthetic Test . . . . .	427
15.3.2	Field Data Test . . . . .	428
15.4	Discussion . . . . .	430
15.5	Summary . . . . .	434
15.6	Computational Labs . . . . .	438
15.7	Appendix: ADMM . . . . .	438
<b>16</b>	<b>Principal Component Analysis</b>	<b>441</b>
16.1	Introduction . . . . .	441
16.1.1	Historical Background of PCA . . . . .	444
16.2	Theory of Principal Component Analysis . . . . .	446
16.2.1	Intuitive Dinosaur Example . . . . .	448
16.2.2	Simple Two-Point Example . . . . .	448
16.2.3	Filtering the PCA Components . . . . .	450
16.2.4	PCA Workflow . . . . .	451
16.2.5	Singular Value Decomposition . . . . .	453
16.3	Numerical Examples . . . . .	455
16.3.1	Yellowstone Eruption Data . . . . .	455
16.3.2	Biplots and Geochemical Data from a Gold Site . . . . .	458
16.3.3	Nashville Limestones . . . . .	458
16.3.4	PCA Applied to Hyperspectral Images . . . . .	460
16.4	Kernel PCA . . . . .	466
16.5	Computational Costs . . . . .	467
16.6	Summary . . . . .	468
16.7	Exercises . . . . .	469
16.8	Computational Labs . . . . .	472
16.9	Appendix: Mathematics of Kernel PCA . . . . .	473
16.10	Appendix: MATLAB Kernel PCA CoLab . . . . .	475

<b>17 Clustering Algorithms</b>	<b>479</b>
17.1 Introduction . . . . .	479
17.2 K-means Clustering . . . . .	482
17.2.1 K-means Cluster Lab with Colab . . . . .	485
17.3 Agglomerative Clustering . . . . .	486
17.4 Information Weighted Clustering . . . . .	491
17.5 Fuzzy Clustering . . . . .	492
17.5.1 Arrival Time Picking and Fuzzy Clustering . . . . .	493
17.6 Elbow Method for Determining the Number of Clusters . . . . .	497
17.7 Support Vector Clustering . . . . .	499
17.8 DBSCAN . . . . .	502
17.8.1 DBSCAN Lab with Colab . . . . .	506
17.9 K-Nearest Neighbors Algorithm . . . . .	507
17.10 Self-organizing Map . . . . .	508
17.10.1 MATLAB SOM Examples . . . . .	512
17.11 Numerical Examples . . . . .	516
17.11.1 Picking Stacking Velocities by K-means Cluster Analysis . . . . .	516
17.11.2 Generating Models with Sharp Impedance Contrasts . . . . .	522
17.11.3 DBSCAN Applied to Earthquakes . . . . .	524
17.11.4 Self-Organized Maps and Classification of Porosity in Rocks . . . . .	532
17.11.5 Self-Organized Maps and Classification of Seismic Horizons . . . . .	538
17.11.6 Self-Organized Maps and Fault Detection . . . . .	539
17.12 Summary . . . . .	539
17.13 Exercises . . . . .	540
17.14 Computational Labs . . . . .	541
17.15 Appendix: Derivation of the Fuzzy Clustering Weight . . . . .	541
<b>18 Generative Adversarial Networks</b>	<b>545</b>
18.1 Introduction . . . . .	545
18.2 Theory of GANs . . . . .	547
18.2.1 GAN Objective Function and Minimax Algorithm . . . . .	548
18.3 Pseudo-Code for a GAN . . . . .	550
18.4 Simple Fault Example . . . . .	555
18.5 GAN Examples in Geoscience . . . . .	557
18.5.1 Seismic-to-Attribute Sections by a GAN . . . . .	557
18.5.2 Seismic Resolution Enhancement by GANs . . . . .	559
18.6 Summary . . . . .	559
18.7 Exercises . . . . .	562
18.8 Appendix: GAN Examples . . . . .	563
<b>V Bayesian Analysis</b>	<b>567</b>
<b>19 Sampling a Probability Distribution</b>	<b>569</b>
19.1 Sampling a Probability Distribution . . . . .	569



19.2	Sampling a PDF by the Inverse-Transform . . . . .	572
19.3	Sampling a PDF by Acceptance-Rejection . . . . .	572
19.4	Markov Chain . . . . .	576
19.5	Metropolis Markov Chain Monte Carlo Sampling . . . . .	578
19.5.1	Detailed Balance Equation . . . . .	580
19.5.2	Metropolis Sampling Example . . . . .	581
19.6	Gibbs Sampling . . . . .	583
19.7	Summary . . . . .	587
19.8	Exercises . . . . .	588
19.9	Appendix: MATLAB Functions for Metropolis Code . . . . .	588
<b>20</b>	<b>Bayes' Theorem</b>	<b>591</b>
20.1	Introduction . . . . .	591
20.2	Intuitive Interpretation of Bayes' Theorem . . . . .	594
20.2.1	Geometrical Interpretation of $P(C), P(T), \frac{P(C)}{P(T)}$ . . . . .	595
20.2.2	Medical Interpretation of $P(C), P(T), \frac{P(C)}{P(T)}$ . . . . .	596
20.2.3	Updating Predictions with New Information . . . . .	598
20.3	Naive Bayes' Theorem . . . . .	599
20.4	Motivation for Bayes' Theorem . . . . .	599
20.5	BI and Regularized Least Squares Inversion . . . . .	600
20.6	Geophysical Examples of Bayesian Inversion . . . . .	602
20.6.1	Prediction of an Oil-Spill Source . . . . .	602
20.6.2	VSP Traveltime Inversion . . . . .	605
20.6.3	Imaging Point Sources in a Hydrofrac Experiment . . . . .	613
20.6.4	Rock Types and Uncertainty from Well Log Data . . . . .	617
20.6.5	Rock Types from Well Logs and Seismic Data . . . . .	624
20.7	Summary . . . . .	624
20.8	Exercises . . . . .	628
20.9	Appendix: Bayes' Theorem Examples . . . . .	630
20.9.1	Example: Rainy Weather . . . . .	630
20.9.2	Example: Biking and Weather . . . . .	632
20.10	Appendix: Naive Bayes' Theorem, Weather and Biking . . . . .	634
20.11	Appendix: Kullback-Leibler Regularizer . . . . .	634
20.12	Appendix: Multisource VSP and $[\mathbf{L}^T \mathbf{L}]^{-1}$ . . . . .	637
20.13	Appendix: Conditional Covariance Matrix . . . . .	638
20.14	Appendix: Bayes and Conditional Covariance Matrix . . . . .	640
20.15	Appendix: Estimation of Statistical Parameters for VSP Data . . . . .	643
20.15.1	Estimating $p(\mathbf{d})$ and $p(\mathbf{m})$ . . . . .	643
20.15.2	Estimating $p(\mathbf{d} \mathbf{m})$ . . . . .	644
<b>21</b>	<b>Discriminant Analysis and Gaussian Mixture Model</b>	<b>647</b>
21.1	Introduction . . . . .	647
21.2	Generative and Discriminative Models . . . . .	649
21.3	Linear and Gaussian Discriminant Analysis . . . . .	650
21.3.1	Linear Discriminant Analysis . . . . .	651

21.3.2	Gaussian Discriminant Analysis . . . . .	653
21.4	Theory of the Gaussian Mixture Model . . . . .	657
21.4.1	Responsibility Function $p(k \mathbf{x}^{(n)})$ . . . . .	659
21.4.2	Expectation-Maximization Method . . . . .	660
21.4.3	1D Example of a Responsibility Function . . . . .	661
21.4.4	Geophysical Inversion . . . . .	663
21.4.5	Example of Generative and Discriminative Modeling . . . . .	663
21.4.6	Fuzzy Clustering and GMM . . . . .	665
21.5	Numerical Examples of GMM in Geoscience . . . . .	665
21.5.1	Turbidite Identification from Seismic Data . . . . .	665
21.5.2	Gravity and Magnetic Inversion with GMM Constraints . . . . .	667
21.5.3	Joint Inversion with Updated GMM Penalty Terms . . . . .	671
21.6	Summary . . . . .	672
21.7	Exercises . . . . .	674
21.8	Computational Labs . . . . .	676
21.9	Appendix: Gradients of the GDA Log-likelihood Function . . . . .	676
21.9.1	Solving for $\phi$ . . . . .	677
21.9.2	Solving for $\boldsymbol{\mu}$ . . . . .	678
21.9.3	Solving for $\boldsymbol{\Sigma}$ . . . . .	679
21.9.4	Gradient of the GMM Log-likelihood Function . . . . .	679
<b>VI</b>	<b>Seismic Inversion and Machine Learning</b>	<b>681</b>
<b>22</b>	<b>Physics-Informed Machine Learning Inversion</b>	<b>683</b>
22.1	Deterministic and Stochastic Inversion . . . . .	683
22.2	Incomplete Training and Physics-Informed ML . . . . .	685
22.2.1	Data Skeletonization and ML . . . . .	685
22.3	Physics-Informed Machine Learning . . . . .	686
22.3.1	Sequential ML+Physics Inversion . . . . .	687
22.3.2	Simultaneous ML+Physics Inversion . . . . .	687
22.4	Automatic Differentiation . . . . .	687
22.5	Summary . . . . .	687
<b>23</b>	<b>Tomographic Deconvolution</b>	<b>689</b>
23.1	Introduction . . . . .	689
23.1.1	Previous Work . . . . .	690
23.2	Theory of Tomographic Deconvolution . . . . .	691
23.2.1	Migration Deconvolution versus Tomographic Deconvolution . . . . .	692
23.2.2	Tomographic Deconvolution Workflow . . . . .	692
23.2.3	CNN Architecture . . . . .	693
23.3	Numerical Results . . . . .	694
23.3.1	Reflections: Single Anomaly Models . . . . .	694
23.3.2	Reflections: Multiple Anomaly Models . . . . .	695
23.3.3	Models that Break the Deconvolution Filter . . . . .	695

23.3.4	Refractions: Multiple Anomaly Models . . . . .	699
23.3.5	Refractions: Fault Anomaly Models . . . . .	699
23.4	Summary . . . . .	700
<b>24</b>	<b>Neural Network Least Squares Migration</b>	<b>703</b>
24.1	Introduction . . . . .	703
24.2	Theory of Neural Network Least Squares Migration . . . . .	705
24.2.1	Least Squares Migration . . . . .	705
24.2.2	Sparse Least Squares Migration . . . . .	706
24.2.3	Neural Network Least Squares Migration . . . . .	707
24.2.4	Multilayer Neural Network LSM . . . . .	709
24.3	Numerical Results . . . . .	709
24.3.1	Three-layer Velocity Model . . . . .	710
24.3.2	SEG/EAGE Salt Model . . . . .	712
24.3.3	North Sea Data . . . . .	714
24.4	Discussion . . . . .	718
24.5	Conclusions . . . . .	721
24.6	Appendix: Migration Green's Function . . . . .	721
24.7	Appendix: Soft Thresholding Function . . . . .	722
24.8	Appendix: Neural Network Least Squares Migration . . . . .	723
24.9	Appendix: Alignment of the Filters . . . . .	724
<b>25</b>	<b>Wave Equation Inversion and Neural Networks</b>	<b>729</b>
25.1	Introduction . . . . .	729
25.2	Theory for Wave Equation Inversion of Skeletonized Data . . . . .	731
25.2.1	Feature Extraction . . . . .	732
25.3	NML Theory . . . . .	733
25.4	NML Inversion of Crosswell Data . . . . .	735
25.4.1	Assessing the Optimal Dimension of the Latent Space . . . . .	736
25.5	NML Inversion of Refraction Data . . . . .	737
25.6	Summary . . . . .	742
25.7	Exercises . . . . .	743
25.8	Connective Function . . . . .	744
25.8.1	Gradient . . . . .	745
<b>26</b>	<b>Automatic Differentiation</b>	<b>747</b>
26.0.1	Automatic Differentiation Algorithm . . . . .	747
26.1	Modeling, Automatic Differentiation and Inversion . . . . .	753
26.1.1	Automatic Differentiation for the One-way Wave Equation . . . . .	753
26.1.2	Automatic Differentiation and Waveform Inversion . . . . .	754
26.2	Summary . . . . .	756
26.3	Exercises . . . . .	756
26.4	FD Solution to the Wave Equation . . . . .	757

<b>VII Background Appendices</b>	<b>763</b>
<b>27 Appendix: Geophysics Background</b>	<b>765</b>
27.1 Seismic Data Recording and Processing . . . . .	765
27.2 Diffraction Stack Migration . . . . .	772
27.3 Surface Waves . . . . .	776
27.3.1 Estimating Depth vs Frequency . . . . .	777
27.4 Seismic Reflection Data as an LTI System: $s(t) = r(t) \star w(t)$ . . . . .	777
<b>28 Probability Theory for Machine Learning</b>	<b>783</b>
28.1 Random Variables, Sample Space, Events and Probability . . . . .	783
28.1.1 Types of Events . . . . .	784
28.1.2 Probability . . . . .	785
28.1.3 Law of Total Probability . . . . .	790
28.2 Bayes' Theorem . . . . .	793
28.2.1 Bayes' Theorem and Cancer Example . . . . .	793
28.2.2 Bayes Theorem for $K$ Disjoint Events . . . . .	794
28.2.3 Bayesian Terminology . . . . .	794
28.3 Gaussian Probability Distribution . . . . .	795
28.3.1 Central Limit Theorem . . . . .	796
28.3.2 Multivariate Normal Distribution . . . . .	796
28.4 Sampling a Probability Distribution . . . . .	798
28.5 Maximum Likelihood Estimate and Neural Network Models . . . . .	798
28.6 Summary . . . . .	800
28.7 Exercises . . . . .	800

# Preface

This book presents the theory of machine learning (ML) algorithms and their applications to geoscience problems. More than half of the described algorithms fall under the class of neural network methods. Their description is at a level that can be understood by anyone with a modest background in linear algebra, calculus and probability. An elementary working knowledge of MATLAB is assumed and almost every chapter is accompanied by lab exercises to reinforce the ML principles. If you don't know MATLAB or Python, many of the labs use the CoLab, aka Colaboratory, notebook which is an easy-to-use product from Google that allows for the execution of Keras-based ML computer codes. It also provides for the free use of a cloud-based GPU processor<sup>1</sup> that can be used as soon as you login. There is no need for the troublesome installation of notebook software.

There are many definitions of *machine learning* (ML), but one of the best is attributed to Arthur Samuel<sup>2</sup>:

*Machine learning is the field of study that gives computers the ability to learn without being explicitly programmed.*

This definition doesn't mean we don't have to write the programs, it means that the ML algorithm finds the best model by learning from the data without relying on rules-based programming. For example, finding the best fit model  $\mathbf{y} = f(\mathbf{x})$  by a typical least squares procedure typically assumes a linear mathematical model  $f(\mathbf{x})$  that predicts the output  $\mathbf{y}$  from the input  $\mathbf{x}$ . For example,  $\mathbf{W}\mathbf{x} = \mathbf{y}$ , where  $\mathbf{W}$  is a matrix. A ML algorithm such as a neural network avoids this assumption by devising the best non-linear model learned from the data  $\mathbf{y}$  for the given architecture.

**Neural Network Example.** How does the neural network learn the model<sup>3</sup>  $f(\mathbf{x}) = \mathbf{y}$  from the data? Learning the model  $f(\mathbf{x})$  is accomplished by using a large training set of data  $(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})$  for  $n \in \{1, 2, \dots, N\}$ , and adjusting the parameters of the model until  $f(\mathbf{x}^{(n)})$  can accurately predict  $\mathbf{y}^{(n)}$  for any training pair. Once the model is learned, then it can accurately predict the output  $f(\mathbf{x}^{new}) = \mathbf{y}^{new}$  from a new input  $\mathbf{x}^{new}$ . This can be important

---

<sup>1</sup>Colab is a hosted Jupyter notebook service that requires no setup to use, while giving free access to computing resources including GPUs. See <https://colab.research.google.com/notebooks/intro.ipynb>

<sup>2</sup>This quote is often attributed to Arthur Samuel, but it cannot be found in Samuel (1959, 1967). A similar quote can be found at <https://www.techemergence.com/what-is-machine-learning/>. See <http://infolab.stanford.edu/pub/voy/museum/samuel.html> for a brief bio of this pioneering engineer.

<sup>3</sup>"Learning the model" is equivalent to computing the parameters of the non-linear function  $f(\mathbf{x})$  that allow for an accurate prediction of  $f(\mathbf{x}) = \mathbf{y}^{true}$  for all the training pairs  $(\mathbf{x}, \mathbf{y}^{true})$ .

for many applications, such as predicting the presence of underground contaminants  $\mathbf{y}^{new}$  from processed geoelectrical and/or seismic data  $\mathbf{x}^{new}$ , accurately detecting the presence of breast cancer  $\mathbf{y}^{new}$  recorded on a fusion of CAT scan and MRI images  $\mathbf{x}^{new}$ , or delineating the faults in  $\mathbf{y}^{new}$  on the right-hand side of Figure 1 from the input seismic section  $\mathbf{x}^{new}$  on the left-hand side.

The mathematical operations of a convolutional neural network model are represented by the block diagrams in Figure 1, where the input image  $\mathbf{x}$  of a seismic section is decomposed into component images contained in the FM1 block (or shallowest layer). Here, a block represents a set of mathematical operations that produce a stack of 2D images known as feature maps (FMs). Each of the 32 feature maps in FM1 is computed by convolving a small filter ( $3 \times 3$  filter in this example) with the input image, and then each pixel value in the FM is thresholded to be between 0 and 1 by a non-linear squashing function<sup>4</sup>. The FM1 feature maps are then subsampled by a factor of four (aka known as maxpooling), and the resulting images are filtered and thresholded by the FM2  $3 \times 3$  filters to get the 64 FMs for block 2. The subsampling acts as a low-pass spatial filter so that only coarser features of the seismic section are seen in, for example, FM3. In the end, the FMs represent the decomposition of a complicated object into high-wavenumber images on the left and low-wavenumber images on the far right. The FMs are weighted, upsampled, summed together and squashed to make a single decision, i.e. classification, at each pixel associated with the input image (Long et al., 2015). Does the pixel value represent a fault, then the answer is yes  $y = 1$  for its class. If not, then the answer is no  $y = 0$  for the output class. The displayed FMs are upsampled by a deconvnet procedure to transform them to the same size as the input image.

**Neural Networks and Deeper Connections.** Neural network models lack a rigorous theoretical foundation that explains their formidable number of commercial successes. This has motivated physicists and mathematicians to search for its mathematical and physical foundations. For example, they have recently discovered that successful CNN modeling can have many more tunable parameters than training data, i.e. equations of constraint, without suffering from overfitting. Arora et al. (2018) proves that a large number of FMs can be redundant and allows for simple compression of the model.

Another insight says that the CNN decomposition is similar to a dictionary-learning algorithm (Papayan et al., 2016, 2017a, 2017b; Sulam et al., 2018), where the different FMs can represent the multiscale components of a complicated object, such as its atoms or molecules. As an example, Figure 2 illustrates how two convolutional FMs, denoted as elementary "atoms" in the bottom row, are combined and filtered to create slightly more complex structures, i.e. molecules, at the second level. These "molecules" are then combined by a CNN model to create the global atom representing, in this case, a digit.

However, the basic operations of the neuron, i.e. atom, has much more complexity than previously thought, where the atom itself is composed of even finer structures. Work by Beniaguev et al. (2021) suggests that the cortical neuron mathematically performs the

---

<sup>4</sup>A squashing function  $\sigma(z)$  *squashes* a wide range of input values  $-\infty < z < \infty$  to be between a small range of output values such as  $0 \leq \sigma(z) \leq 1$ . The squashing operation sparsifies essential+extraneous information into its essential components (Papayan et al., 2016, 2017a, 2017b; Sulam et al., 2018; Chen et al., 2020b).

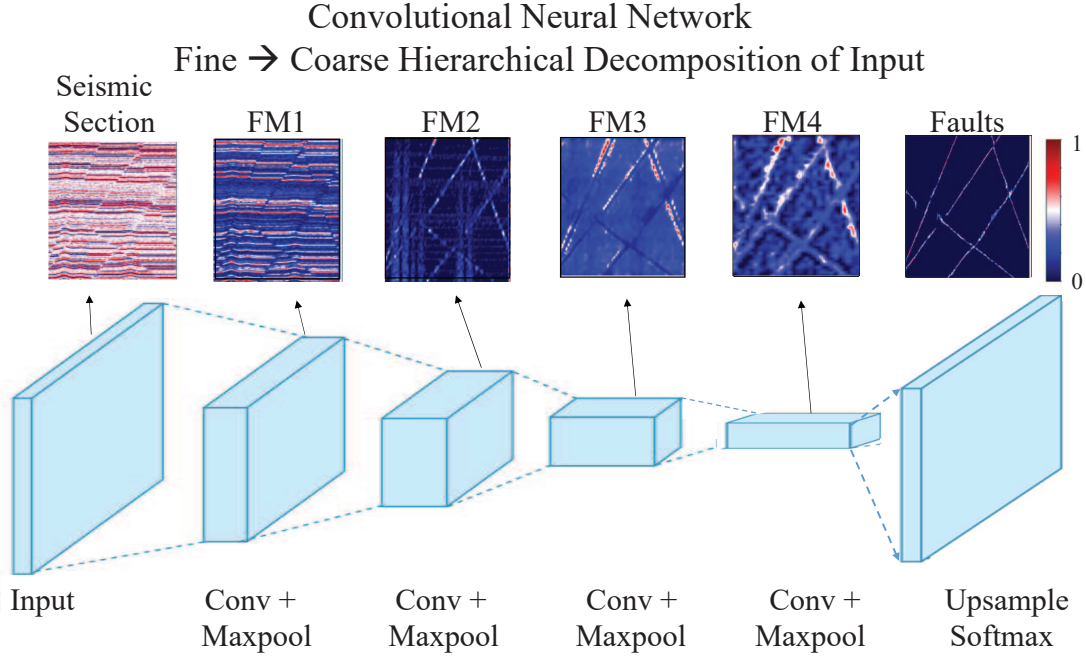


Figure 1: Block diagram of the convolutional neural network (CNN) architecture that decomposes the input picture of a seismic section into fine-scale feature maps (FM1 and FM2) and coarser-scale feature maps (FM3 and FM4). To provide a more interpretable image, the FMs are *deconvolved* for the display of interpretable images (Shi et al., 2018). The pixels in the far-right image are labeled  $0 < y < 1$ , where  $y = 1$  indicates that the pixel is a fault and  $0 \leq y \leq 1$  indicate the probability of being a fault. Typically, the shallow  $3 \times 3$  filters that compute the 32 FM1 images from the input image detect the sharp edges in the input, and the filters associated with FM2 to FM3 detect larger scale features such as the general shape of the fault. These *successive decompositions derive the building blocks at successive levels from specific combinations of building blocks at the previous level* (Holland, 1995). Figure partially adapted from Dertat (2017) and Shi et al. (2018).

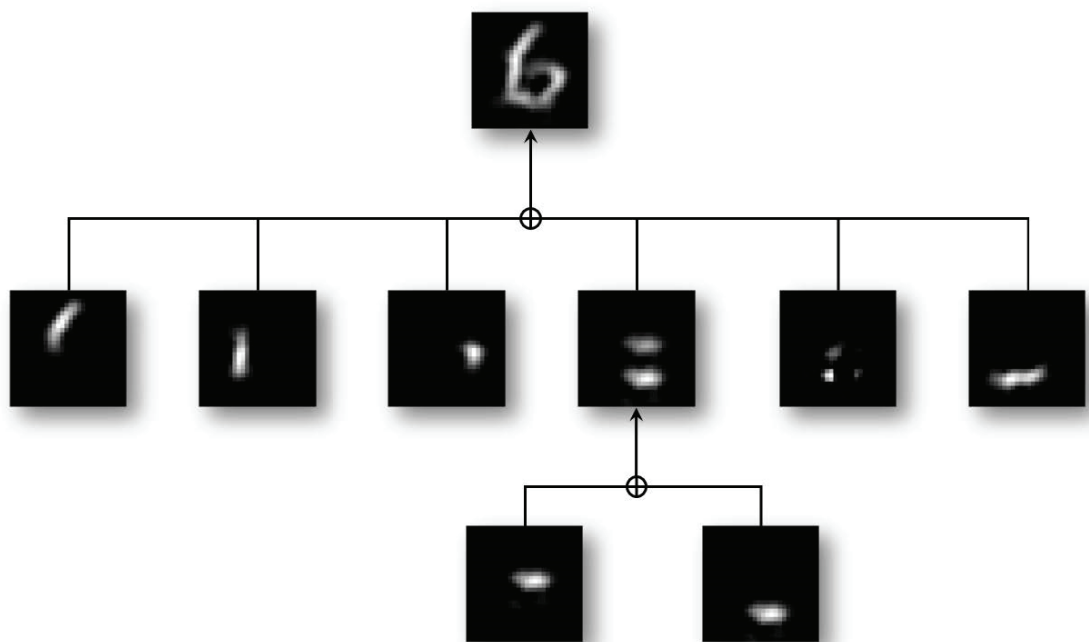


Figure 2: Elementary FMs in the bottom row are combined to form the molecules in the second row, which in turn are used to form the digit 6 in the top row. Figure from Sulam et al. (2018).

same functions as a deep neural network with 5-8 layers. This depth is associated with the neuron's dendritic branches that act as spatial pattern detectors.

Decomposing an input image into a sequence of simpler-to-more complicated parts is also used to describe the process of adaptive evolution illustrated in Figure 3. John Holland (1995) describes the fundamental set of procedures for adaptive evolution in nature:

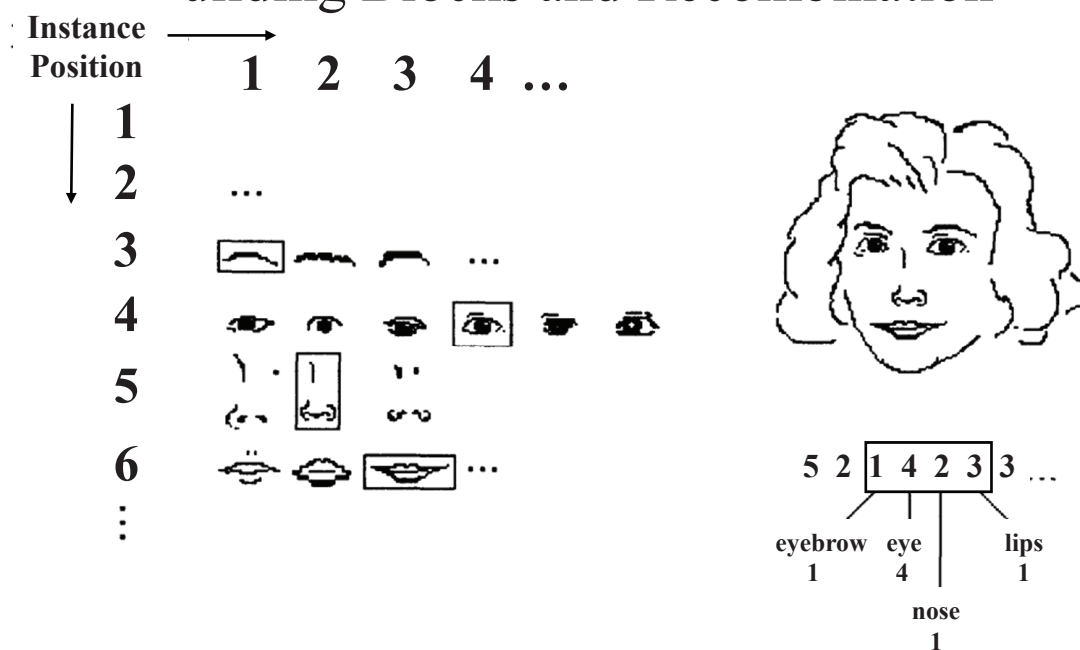
*'If model making, broadly interpreted, encompasses most of scientific activity, then the search for building blocks becomes the technique for advancing that activity. At a fundamental level, we have the quarks of Gell-Mann. Quarks can be combined to yield nucleons, the building blocks at the next level. The process can be iterated, deriving the building blocks at successive levels from specific combinations of building blocks at the previous level. The result is the quark/ nucleon/ atom/ molecule/ organelle/cell/ underpins much of physical science.'*

**Artificial Intelligence and CNN.** More generally, ML methods belong to the larger class of artificial intelligence<sup>5</sup> algorithms (see Figure 4), but the one that is attracting the most interest is that of deep learning with CNNs. Convolutional neural networks are now playing leading roles in the software for self-driving cars, language translation, speech recognition, computer vision, fusion and diagnosis of medical images, analysis of satellite and aerial

<sup>5</sup><https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/>



## Building Blocks and Recombination



A face can be described by stringing together the numbers that index its different component parts.

Figure 3: Neural networks is similar to John Holland's (1995) description of how nature evolves models: '...building blocks at successive levels from specific combinations of building blocks at the previous level.'. The important features of the face are denoted by the *feature* rectangles on the left. Figure adapted from Holland (1995).

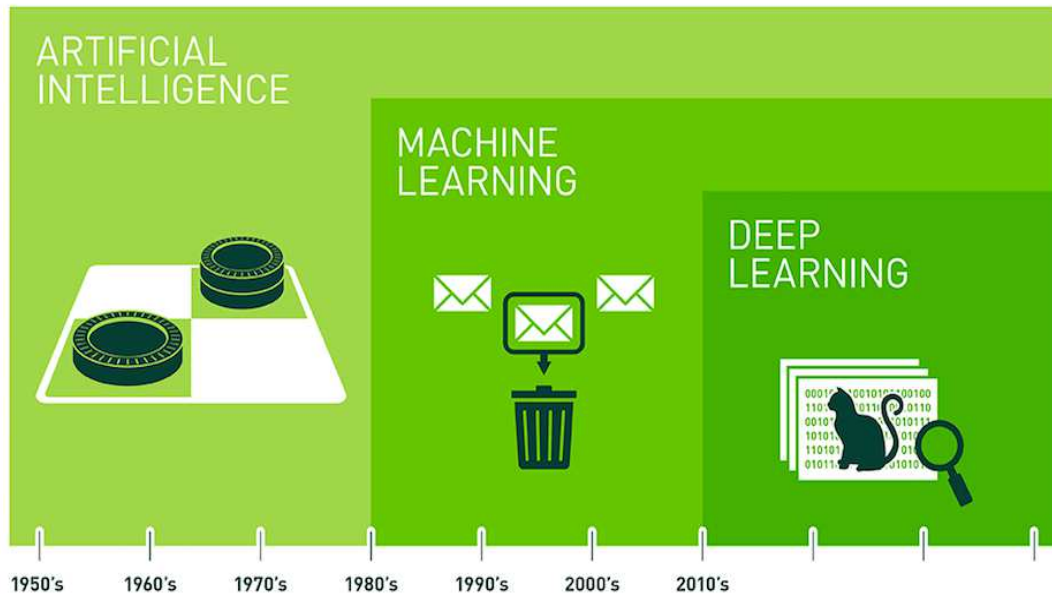


Figure 4: Artificial intelligence (AI) and the sub-classes of machine learning and deep learning, where AI involves a machine doing something that only a human would be able to do. Source of picture is Nvidia.

images for optimizing agricultural practices, and many other fields. Significantly, it is now being recognized as an important interpretation tool for analyzing lithology from seismic and well-log data, detecting faults in large 3D volumes of seismic data, and searching for patterns in earthquake data.

**Organization of the Book.** This book is divided into six main sections:

- **Mathematical Background and Optimization Theory.** Optimization theory and gradient descent methods for ML are reviewed, with an emphasis on the choice of step length methods. Readers with a familiarity of these algorithms can skip to the next sections.
- **Supervised Learning Algorithms.** Introduction to neural networks, fully-connected neural networks, applications to geoscience problems, and support vector machines.
- **Convolutional Neural Networks.** Basics of convolution and correlation, CNN, object identification, semantic segmentation, recurrent neural networks, Transformers and self-attention.
- **Unsupervised Learning Algorithms:** Autoencoders, convolutional sparse coding, principal component analysis, clustering methods, generative adversarial networks.
- **Bayesian Analysis.** Sampling, Bayes' theorem, and Gaussian mixture models.

- Seismic Inversion and ML. Physics-informed ML inversion, tomographic deconvolution, neural network LSM, wave equation inversion and neural networks, automatic differentiation.

Most chapters are accompanied by executable code (MATLAB and/or Keras with CoLab) that can be implemented by the reader with a modest level of programming knowledge. The understanding of the algorithms is deepened by parameter tuning and by examining the details of the code. Most of these codes have been written by my co-authors, without which this book would not have been written.

Case histories are used to demonstrate the practical use of ML in solving geoscience problems. These problems include fracture identification in photos, detection of rock cracks in drone photos, fossil and lithology detection in thin sections, prediction of permeability in rock samples, geochemical analysis, well-log analysis and identification of salt boundaries in seismic data, seismic arrival-time picking by a clustering method, least squares migration, velocity analysis, demultiple, noise reduction in seismic data and migration images, and some interpretation examples for seismic and radar data. The reader who diligently goes through the chapters and labs will have a thorough grounding in some of the fundamental ML methods used in geoscience.

**WWW Software Sites for Computational Labs.** The computational labs for the book *Machine Learning in Geoscience* are located at the following sites:

- *Machine Learning in Geoscience* Labs: <http://csim.kaust.edu.sa/files/ErSE394/LAB1/>
- *Machine Learning in Geoscience* Labs: <http://utam.gg.utah.edu/books/ML/index.html>
- *Machine Learning in Geoscience* Labs: <http://seg.org/books/Schuster.ML/index.html>
- *Machine Learning in Geoscience* Labs: [repository.kaust.edu.sa/bitstream/handle/10754/674007/GeoscienceMachineLearning.html](http://repository.kaust.edu.sa/bitstream/handle/10754/674007/GeoscienceMachineLearning.html)
- *Machine Learning in Geoscience* Labs: <https://earth.utah.edu/books/Schuster.ML/index.html>

The computational labs for the books *Seismic Interferometry* and *Seismic Inversion* are located at the following sites:

- *Seismic Interferometry* Labs: <http://utam.gg.utah.edu/Inter.LAB1/>
- *Seismic Inversion* Labs: [repository.kaust.edu.sa/bitstream/handle/10754/674016/SeismicInversion.html](http://repository.kaust.edu.sa/bitstream/handle/10754/674016/SeismicInversion.html)
- *Seismic Inversion* Labs: <https://earth.utah.edu/books/Schuster.SeismicInversion/index.html>

## Acknowledgments

GTS is very thankful for the financial and computer support provided by King Abdullah University of Science and Technology and the KSL facilities (<https://www.hpc.kaust.edu.sa>)

/). The assistance of Professor Xiangliang Zhang (KAUST and Notre Dame University), Dr. David Pugh (KSL) and Dr. Saber Feki (KSL) were very beneficial in bringing me to the frontiers of teaching ML. My loyal co-authors are to be acknowledged for writing many of the computer codes that accompany the chapters. I am very grateful to Professor Daniel Trad at Calgary University for his tireless editing of the manuscript. He provided many detailed comments and suggestions for improvement. I am deeply grateful to Dr. Yan Yang for discussions about modifications and her expert help in organizing and implementing corrections, and appreciate the faithful editing of the manuscript by my co-authors. This book benefitted from the inclusion of appendix 17.6.2 written by Yunsong Huang, the excellent geoscience examples provided by Eduardo Cano, Ahmad Ramdani and Tian Qiao, the concise summaries of many ML classes provided by Yuan Yang, and thorough lab instructions by Shi Yongxiang.

# Notation Convention

- $\mathbb{R}^N$  denotes the  $N$ -dimensional real vector space.
- $\mathbb{C}^N$  denotes the  $N$ -dimensional complex vector space.
- A column vector will be denoted by bold lower-case letters; e.g.,  $\mathbf{x} = [x_1, x_2, \dots, x_N]^T$  or  $\mathbf{x} = (x_1, x_2, \dots, x_N)^T$  represents the  $N \times 1$  vector where  $x_i$  is the  $i^{th}$  element of  $\mathbf{x}$ .
- A matrix will be denoted by bold upper-case letters; e.g.,  $\mathbf{A} \in \mathbb{R}^{M \times N}$  represents an  $M \times N$  real matrix whose  $ij^{th}$  element is denoted by  $A_{ij}$ .
- An order-of-magnitude estimate of a variable whose precise value is unknown is an estimate rounded to the nearest power of ten.
- A scalar will be denoted by lower-case letters.
- Subscripts are usually used to denote the element index of a vector or matrix.
- Superscripts with parentheses are used to denote an iterate of a vector or matrix; e.g.,  $\mathbf{x}^{(k)}$  denotes the  $k^{th}$  iterate of an iterative scheme.
- $\mathbf{x} \cdot \mathbf{y} = \mathbf{x}^T \mathbf{y} = \langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^N x_i^* y_i$  represents a dot product or an inner product between finite-dimensional vectors  $\mathbf{x}$  and  $\mathbf{y}$ .
- MATLAB syntax is sometimes used to represent vectors or matrices, e.g.,  $[a \ b; c \ d]$  denotes the matrix

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}.$$

- $\|\mathbf{x}\|_1$  denotes the 1-norm of the  $N \times 1$  vector  $\mathbf{x}$  which is equal to

$$\|\mathbf{x}\|_1 = \sum_{i=1}^N |x_i|.$$

- $\|\mathbf{x}\|_2$  denotes the 2-norm or Euclidean length of the  $N \times 1$  vector  $\mathbf{x}$  which is equal to

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^N x_i^2}.$$

If the subscript is missing then the 2-norm is indicated, the  $l^2$  norm is for a discrete vector with an infinite number of components (sequences) and  $L^2$  for a discrete vector with a finite number of components.

- The length of a vector  $\mathbf{x}$  will often be denoted as  $|\mathbf{x}|$  rather than  $\|\mathbf{x}\|_2$ .

- $\|\mathbf{x}\|_p$  denotes the  $L^p$ -norm of the  $N \times 1$  vector  $\mathbf{x}$  which is equal to

$$\|\mathbf{x}\|_p = \left( \sum_{i=1}^N x_i^p \right)^{\frac{1}{p}}.$$

- $\hat{\mathbf{x}} = \frac{\mathbf{x}}{|\mathbf{x}|}$  denotes the unit vector.

- $\mathbf{A}^*$  denotes the complex conjugate of the matrix  $\mathbf{A}$ .

- $\mathbf{A}^T$  denotes the transpose of matrix  $\mathbf{A}$ . We will often insist it also means the transpose and complex conjugated matrix  $\mathbf{A}$ .

- $\mathbf{A}^\dagger$  denotes the conjugated and transposed matrix  $\mathbf{A}$ .

- $\star$  denotes temporal convolution. For example, assuming  $f(t)$  and  $g(t)$  are real continuous functions of the scalar variable  $t$  and are square integrable then

$$f(t) \star g(t) = \int_{-\infty}^{\infty} f(t - \tau)g(\tau)d\tau = \int_{-\infty}^{\infty} f(\tau)g(t - \tau)d\tau. \quad (1)$$

- $\otimes$  denotes temporal correlation. For example, assuming  $f(t)$  and  $g(t)$  are real continuous functions of the scalar variable  $t$  and are square integrable then

$$f(t) \otimes g(t) = f(-t) \star g(t) = \int_{-\infty}^{\infty} f(\tau)g(t + \tau)d\tau. \quad (2)$$

- $\mathcal{F}[f(t)] = F(\omega)$  denotes the Fourier transform of  $f(t)$  and  $\mathcal{F}^{-1}[F(\omega)]$  is the inverse

Fourier transform of  $F(\omega)$ . For example,

$$\begin{aligned}
F(\omega) = \mathcal{F}[f(t)] &= \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt, \\
f(t) = \mathcal{F}^{-1}[F(\omega)] &= \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{i\omega t} d\omega, \\
\frac{d^n f(t)}{dt^n} &= \frac{1}{2\pi} \int_{-\infty}^{\infty} (i\omega)^n F(\omega) e^{i\omega t} d\omega, \\
f(-t) &= \mathcal{F}^{-1}[F(\omega)^*], \\
\mathcal{F}[f(t) \star g(t)] &= F(\omega) G(\omega), \\
\mathcal{F}[f(t) \otimes g(t)] = \mathcal{F}[f(-t) \star g(t)] &= F(\omega)^* G(\omega), \\
f(t) \star g(t) &= \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) G(\omega) e^{i\omega t} d\omega, \\
f(t) \otimes g(t) &= \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega)^* G(\omega) e^{i\omega t} d\omega, \\
f(t) \otimes g(t)|_{t=0} = \int_{-\infty}^{\infty} f(\tau) g(\tau) d\tau &= \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega)^* G(\omega) d\omega, \\
f(t) \otimes f(t)|_{t=0} = \int_{-\infty}^{\infty} f(\tau)^2 d\tau &= \frac{1}{2\pi} \int_{-\infty}^{\infty} |F(\omega)|^2 d\omega. \tag{3}
\end{aligned}$$

- The Dirac delta function  $\delta(t)$  is a generalized function that is zero everywhere on the real line, except at  $t = 0$ . The Dirac delta function has a broadband spectrum with the constant amplitude 1:

$$\delta(t - t') = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{i\omega(t-t')} d\omega. \tag{4}$$

For a smooth function  $f(\tau)$ , the delta function has the sifting property:

$$f(t) = \int_{-\infty}^{\infty} f(\tau) \delta(\tau - t) d\tau. \tag{5}$$

- The Kronecker delta function  $\delta_{kn} = 1$  if  $k = n$ , otherwise it is equal to zero for all integer values of  $k$  and  $n$ . It is equivalent to the *indicator* function  $\mathcal{I}\{k = n\}$ .





# Abbreviations

- ABC - Absorbing boundary condition
- Adagrad - Adaptive gradient
- Adam - Adaptive moment estimation
- ADCIG - Angle-domain common image gather
- AAE - Adversarial autoencoder
- AE - Autoencoder
- aka - Also known as
- ASSD - Average of the sum of squared distances
- BB - Bounding box
- BD - Blind deconvolution
- BI - Bayesian inversion
- BIC - Bayesian information criterion
- CAG - Common angle gather
- CG - Conjugate gradient
- CGAN - Conditional generative adversarial network
- CIG - Common image gather
- CMG - Common midpoint gather
- CNN - Convolutional neural network
- COG - Common offset gather
- CPT - Conditional probability table
- CSG - Common shot gather
- DBSCAN - Density-based spatial clustering

- DCGAN - Deep convolutional generative adversarial networks
- DM - Diffraction-stack migration
- DNN - Deep neural network
- EM - Expectation-maximization
- FC - Fully connected
- FCL - Fully-connected layer
- FCM - Fuzzy C-means Clustering method
- FCN - Fully-convolutional neural network
- FCNN - Fully-connected neural network
- FD - Finite difference
- FISTA - Fast iterative-soft-threshold-algorithm
- FM - Feature map
- FWI - Full waveform inversion
- GANS - Generative adversarial networks
- GOM - Gulf of Mexico
- GRU - Gated recurrent unit
- ISTA - Iterative-soft-threshold-algorithm
- IWC- -Information weighted clustering
- KM - Kirchhoff migration
- K-NN - K-nearest neighbors
- LAPGAN - Laplacian generative adversarial network
- LReLU - Leaky rectified linear unit
- LR - Logistic regression maps the linear function  $\mathbf{w} \cdot \mathbf{x} + b$  into the sigmoid function  $\frac{1}{1+e^{-\mathbf{w} \cdot \mathbf{x} - b}}$ .
- LReLU - Leaky rectified linear unit
- LSM - Least squares migration
- LSRTM - Least squares reverse time migration
- LSTM - Long short-term memory

- LSI - Linear shift invariant
- LTI - Linear time invariant
- MAP - Maximum a posterior
- MD - Migration deconvolution
- ML - Machine learning
- MLE - Maximum likelihood estimate
- MSE - Mean square error
- MVA - Migration velocity analysis
- NAG - Nesterov accelerated gradient
- NN - Neural network
- NML - Newtonian Machine Learning
- NMO - Normal moveout
- PCA - Principal component analysis
- PDE - Partial differential equation
- PDF - Probability density function
- PIML - Physics-informed Machine Learning
- PSTM - Prestack time migration
- QN - Quasi-Newton
- QP - Quadratic Programming
- RBF - Radial basis function is a real-valued function whose value depends only on the distance between the input and some fixed point. RBFs are also used as an exponential kernel in SVMs.
- R-CNN -Region CNN
- ReLU - Rectified linear unit
- RNN - Recurrent neural network
- RPN - Region proposal network
- RTM - Reverse time migration
- s.t. - Such that

- SD - Steepest descent
- SGD - Stochastic gradient descent
- SMO - Sequential minimal optimization
- SOM - Self-organizing map
- SRGANs = Super-resolution generative adversarial networks
- SPD - Symmetric positive definite
- SSD - Sum of squared distances
- SVM - Support vector machine
- SVR - Support vector regression
- SSP - Surface seismic profile
- TD = Tomographic deconvolution
- QP - Quadratic programming
- VAE - Variational autoencoder
- VRNN - Vanilla recurrent neural network
- VSP - Vertical seismic profile
- ZO - Zero offset
- VC Dimension - *"The Vapnik–Chervonenkis (VC) dimension is a measure of the capacity (complexity, expressive power, richness, or flexibility) of a space of functions that can be learned by a statistical classification algorithm. It is defined as the cardinality of the largest set of points that the algorithm can shatter."* ([https://en.wikipedia.org/wiki/Vapnik%E2%80%93Chervonenkis\\_dimension](https://en.wikipedia.org/wiki/Vapnik%E2%80%93Chervonenkis_dimension)). In other words, the classifier should be no more wigglier than the data deserve, otherwise the model will overfit the data. For a discriminative NN, the sample complexity for the NN is linear in the VC dimension or number of unknowns.
- wrt - With respect to
- WT - Wave equation traveltime tomography
- WTW - Wave equation traveltime and waveform tomography

# Chapter 1

## Introduction to Machine Learning in the Geosciences

The chapter provides a partial overview of machine learning methods and their historical development.

### 1.1 Introduction

Machine learning (ML) was defined by the IBM scientist Arthur Samuel in the 1950s:

*Machine learning is the field of study that gives computers the ability to learn without being explicitly programmed.*

As an example, many types of ML methods fit a non-linear relationship between input and output data. This is equivalent to a non-linear regression fitting of data. However, the input-output relationship is *not explicitly programmed* as a, for example, a weighted sum of polynomials with a specified order. Instead, the ML program *learns* the fitting function from the data.

There are dozens of ML algorithms today, such as support vector machines, principal component analysis, recurrent neural networks, clustering, object detection, convolutional networks and many others (Bishop, 2006). Some researchers use the phrase "deep learning", but this typically refers to convolutional neural networks with more than five layers (Goodfellow et al., 2016).

Over the last 75 years machine learning has undergone several metamorphisms. From the 1940s to early 1960s, cybernetics (Wiener, 1948), neural networks, convolutional neural networks, and deep learning as illustrated in Figure 1.1. Cybernetics was the hot research topic related to ML (Goodfellow et al., 2016) and is now used in a rather loose way to imply "control of any system using technology." (<https://en.wikipedia.org/wiki/Cybernetics>). Cybernetics is the scientific study of how humans, animals and machines control and communicate with each other. It was formulated around the same time as the fundamental development of a biological model for theories of learning (McCulloch and Pitts, 1943; Hebb, 1949). The neuron-function model of the brain led to the mathematical algorithm known

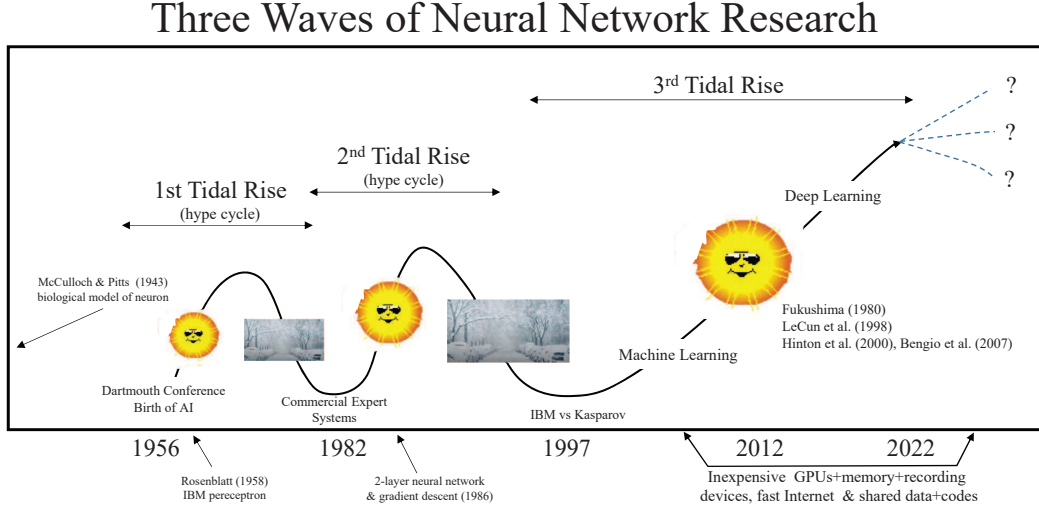


Figure 1.1: Three historical waves of artificial neural nets research (Goodfellow et al., 2016; Toosi et al., 2021), where a wave is characterized by periods of growth (sun symbol) and decline (winter scene). The third wave, "deep learning", is schematically described by a rapid rise whose future development is ahead of us.

as the perceptron introduced by Rosenblatt (1958). It later evolved to the more practical neural network model where, in its simplest form, the  $i^{th}$  input electrochemical signal  $x_i$  to a neuron is related to the output  $y$  by

$$y = \sigma\left(\sum_i w_i x_i + b\right), \quad (1.1)$$

where  $w_i$  is the weight to the input  $x_i$ ,  $b$  is a bias term and  $\sigma(z) = 1/(1 + e^{-z})$  is the sigmoid function shown in Figure 1.2. In the original perceptron, the *activation* function was the all-or-nothing<sup>1</sup>  $\sigma(z) = \text{sign}(z)$  function, but was later replaced by smoother almost-all-or-nothing functions such as the sigmoid  $\sigma(z) = 1/(1 + e^{-z})$  where the derivative exists everywhere.

According to Wikipedia (<https://en.wikipedia.org/wiki/Perceptron>): "The perceptron was intended to be a machine, rather than a program, and while its first implementation was in software for the IBM 704, it was subsequently implemented in custom-built hardware as the "Mark 1 perceptron" shown in Figure 1.3. This machine was designed for image recognition: it had an array of 400 photocells, randomly connected to the "neurons". Weights were encoded in potentiometers, and weight updates during learning were performed by electric motors. In a 1958 press conference organized by the US Navy, Rosenblatt made statements about the perceptron that caused a heated controversy among the fledgling AI community. The New York Times reported the perceptron to be *the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and*

<sup>1</sup>The *sign* function is also known as the Heavyside function.

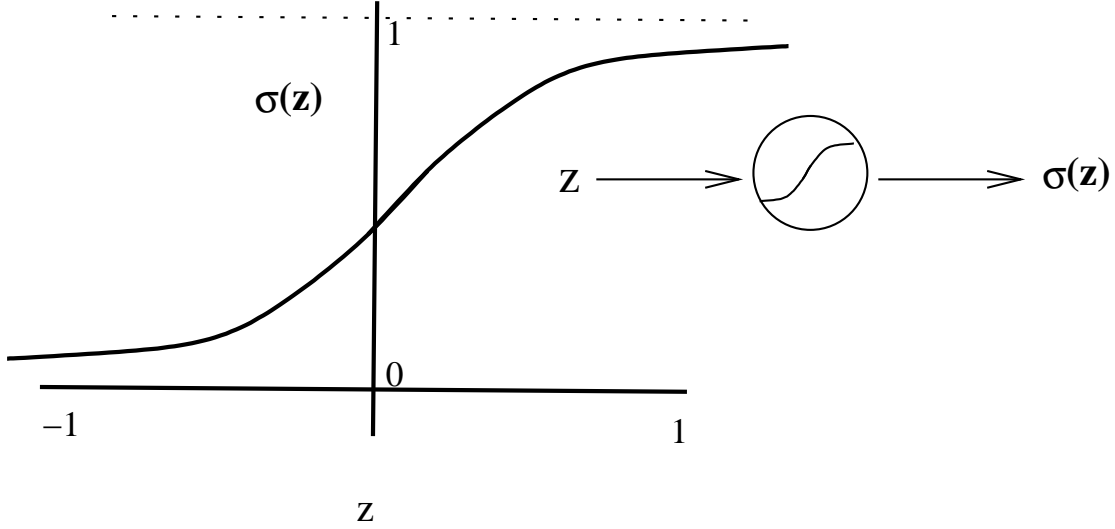


Figure 1.2: Sigmoid function plotted against its argument and its system diagram to the right. The sigmoid is also known as a *squashing* function because it squashes its output  $\sigma(z)$  to be between 0 and 1 for  $-\infty < z < \infty$ .

*be conscious of its existence.”.*

From a simple biological point of view the perceptron model suggests that an input electrical signal  $x$  will be transmitted by the neuron through the connecting synapse to the neighboring neuron if the input voltage exceeds a threshold value. The input values  $x_i$  are similar to the electrical impulses that travel through the many synapses that feed into the neuron.

A system of many perceptrons with sigmoid activation functions came to be known as a neural network in the 1980s (Figure 1.1), which could be used to perform practical functions such as classification of images. Instead of one perceptron, the  $N \times 1$  input signal  $\mathbf{x}$  feeds into  $M$  neurons so that equation 1.1 becomes

$$y_i = \sigma\left(\sum_{j=1}^N w_{ij}x_j + b_i\right), \quad (1.2)$$

where the  $M \times 1$  output vector is  $\mathbf{y}$ , the weights  $w_j \rightarrow w_{ij}$  become those associated with the  $M \times N$  weight matrix  $\mathbf{W}$ . The activation function  $\sigma$  acts on each element of the matrix-vector product  $z_i = \sum_{j=1}^N w_{ij}x_j + b_i$  to return the  $M \times 1$  output vector  $\mathbf{y}$ , which is also known as the activation vector  $\mathbf{a}$  if it is a *hidden* layer and not the last column of nodes. A single perceptron with multiple inputs is shown on the left side of Figure 1.4. According to Goodfellow et al. (2016), connectionism is another name similar to that of artificial neural networks (ANN) used for ML in the 1980s and 1960s. The word connectionist refers to the speculation in the 1950’s that “...the images of stimuli may never really be recorded at all, and that the central nervous system simply acts as an intricate switching network, where retention takes the form of new *connections*, or pathways, between centers of activity.” (Rosenblatt, 1958).



Figure 1.3: Mark 1 IBM computer for image recognition. Left photo: the camera system in which an input scene is recorded by a  $20 \times 20$  array of cadmium sulphide photocells to give an image with 400 pixels. Middle photo: the perceptron consisted of a patch board which allowed different configurations of input features to be tried. Right photo: racks of adaptive weights where the weight values were adjusted automatically by the learning algorithm. Images from Bishop (2006).

Weighting the input values and summing them together to give the input to the non-linear activation function  $\sigma$  defines the two basic building blocks of a neural network. The concatenation of each weighted sum followed by the application of the activation function forms one layer, and a sequence of such layers forms a multilayer neural network. An example is shown on the right side of Figure 1.4.

Interest in neural networks grew rapidly from the 1980s and peaked around the mid 1990s but started to fall off, partly because its limitations were revealed as being computationally too expensive to fulfill all of its early promises. It also suffered from the inability to theoretically predict its ability to perform well, including IBM's Deep Blue's success in beating a chess grandmaster<sup>2</sup> in 1997. Around the same time attention was being refocused to mathematically tractable methods such as Support Vector Machines (Bishop, 2006).

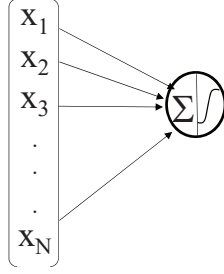
This decline in interest changed in 1998 with the development of the convolutional neural networks (CNNs) by LeCun et al. (1998), which was pioneered by related work from Fukusjima (1980) and Wang (1988, 1990). Instead of requiring massive memory and computations to perform neural network computations, CNN replaced fully connected layers and fully populated matrices with sparse convolutional matrices. For equation 1.2 this means that the fully-populated matrix-vector multiplication is replaced by a convolution of the input vector with a much smaller number of weights. The CNN algorithm allowed for the development of large neural networks with many deep layers, and gave rise to re-branding of *neural network research* as *deep learning research* (see Figure 1.1).

The level of deep learning research has exploded in a wide variety of fields, and has become publicly prominent with its success in popular media companies such as Facebook and Google and its use with self-driving cars. A force multiplier is the development of inexpensive GPUs that can expedite the convolutional operations.

<sup>2</sup>[https://en.wikipedia.org/wiki/Deep\\_Blue\\_versus\\_Garry\\_Kasparov](https://en.wikipedia.org/wiki/Deep_Blue_versus_Garry_Kasparov)



a) Single Perceptron



b) Neural Network

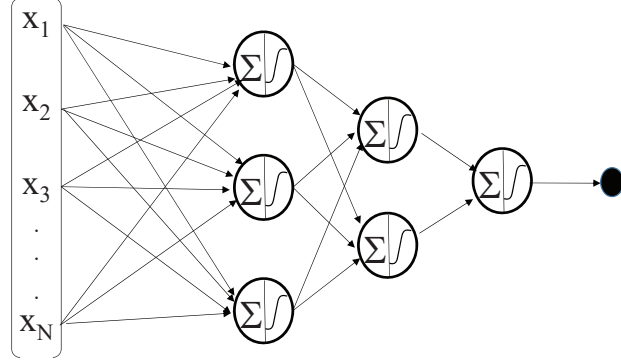


Figure 1.4: (Left) Single-layer perceptron and (right) a multilayer neural network. Recent work by Beniaguev et al. (2021) suggests that the new mathematical model for a cortical neuron performs the same functions as a deep neural network with 5-8 layers.

## 1.2 Three Classes of Machine Learning

Machine learning methods can be classified into the three classes shown in Figure 1.5: supervised learning, unsupervised learning, and reinforcement learning. Researchers are currently exploring their potential applications in three categories of seismic exploration: reservoir analysis, seismic data processing and seismic interpretation. The supervised ML method often used today in geosciences as well as in image recognition is that of convolutional neural networks, which is introduced in Chapter 9.

### 1.2.1 Supervised Learning

Supervised learning is a procedure for finding a model, for example finding the coefficients of  $\mathbf{W}$ , that predicts the output  $\mathbf{y}$  from the input data  $\mathbf{x}$ . The procedure is denoted as supervised learning because  $\mathbf{W}$  is found by using a large training set that consists of many training pairs  $(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})$  that are much more than the number of unknown weights in  $\mathbf{W}$ .

One use for supervised learning is for classifying input data into different categories. For example, the goal in Figure 1.5a is to create a neural network that can distinguish pictures of dogs from other types of animals. Here, the *training pairs* consist of dog images and the label  $y = 1$  for a dog and the label  $y = 0$  for not a dog. The  $N \times N$  dog image is *flattened* into an  $N^2 \times 1$  vector  $\mathbf{x}^{(n)}$ , and the training pair  $(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})$  consist of both the input  $\mathbf{x}^{(n)}$  and target output  $\mathbf{y}^{(n)}$  pairs. There are many training pairs. The label vector  $\mathbf{y}$  in this case is a scalar where  $y = 1$  for a dog and  $y = 0$  for not a dog.

Many pairs of training data are used to *train* the neural network so that an overdetermined system of equations is formed. The coefficients in  $\mathbf{W}$  are found by an iterative optimization method (see Chapters 4-6). An accurate estimate of  $\mathbf{W}$  will allow pictures of dogs that are outside the training set to be identified by the neural network.

Another example of supervised learning is shown in Figure 1.6 where thin sections

## Three Major Classes of Machine Learning

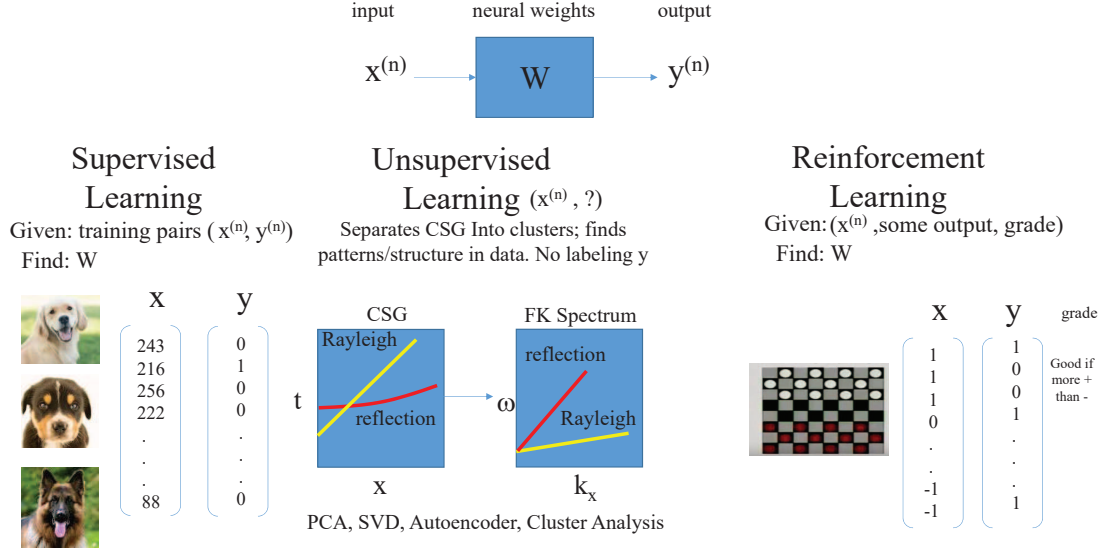


Figure 1.5: Three classes of machine learning.

of rocks are the input images (Patel and Chaterjee, 2016). The goal is to classify them according to the type of limestone. Instead of inputting the entire image of a thin section, they are skeletonized into smaller features, which is shown as a two part operation. Each colorized thin-section image is reduced to the histogram values of RGB colors. Then the second-order, third-order and fourth-order statistics of each of the three histograms are computed by calculating their variance, skewness, and kurtosis values. These are the values input into the neural network, and there are seven output classes. These classes correspond to the dominant type of geology in the image as determined by a geologist. A number of classified thin sections with class labels are used for training. The training estimates the optimal values of the coefficients in  $W$ . A new set of thin sections are then used as input into the neural network, which then classifies these out-of-the-training set data. The results are shown in the lower right side of Figure 1.6. Comparison of the predicted classes and the actual classes shows more than a 90% accuracy in labeling the data.

### Weakly Supervised and Semi-supervised Learning

A subset of supervised learning is semi-supervised learning where there are only a few labeled data but lots of unlabeled data (Patel et al., 2016). In this case it is too expensive to label more than a few examples of data.

There is also weakly-supervised learning where each image is given a label but the goal is to find labels at the pixel level. Each image can contain features from each class, but it is given a label that denotes the dominant feature. For example, small localized images of seismic sections might be labeled as predominantly salt dome or a turbidite sequence, but they also contain other geological structures. Labeling each localized image as just one class

## Deep Learning: Classification Limestones

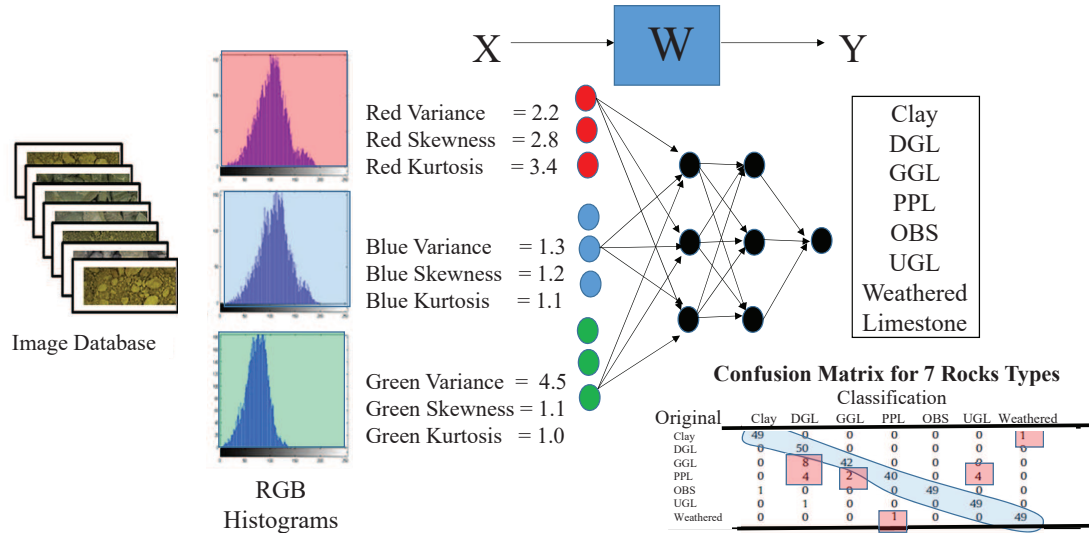


Figure 1.6: Neural network for classifying thin sections (Patel et al., 2016).

is quickly done by an interpreter, but it ignores the detailed geology. Using a collection of these labeled images, a dictionary can be built up and be used to train a weakly supervised neural network to label each image pixel as the type of geology belonging to one of the classes. Thus, detailed geology at the pixel level can be provided by a machine-learning algorithm. A key goal of representation learning is to disentangle the factors of variation that contribute to the appearance of an image (Patel et al., 2016).

### 1.2.2 Unsupervised Learning

Unsupervised learning is when the training data do not specify the target vector  $\mathbf{y}$  but recognizes that there are distinct patterns in the input data  $\mathbf{x}$ . The goal is to separate distinct patterns from one another. As an example, Figure 1.7a depicts a common shot gather of seismic data and its b) transform to phase-velocity and frequency space. The Rayleigh surface waves and P-wave guided waves (GW) are tangled together in the original data but are well separated in the  $C - f$  domain because they each propagate with very different propagation velocities; here,  $C$  is the phase velocity and  $f$  denotes frequency. Other examples include the separation of signal from noise by principle component analysis (PCA) in Chapter 16, clustering in Chapter 17, and discriminant analysis by a Gaussian Mixture Model in Chapter 21.

### 1.2.3 Reinforcement Learning

Reinforcement Learning is used to train a computer to play a game. The training consists of trial-and-error exercises with the starting conditions of the game, such as the

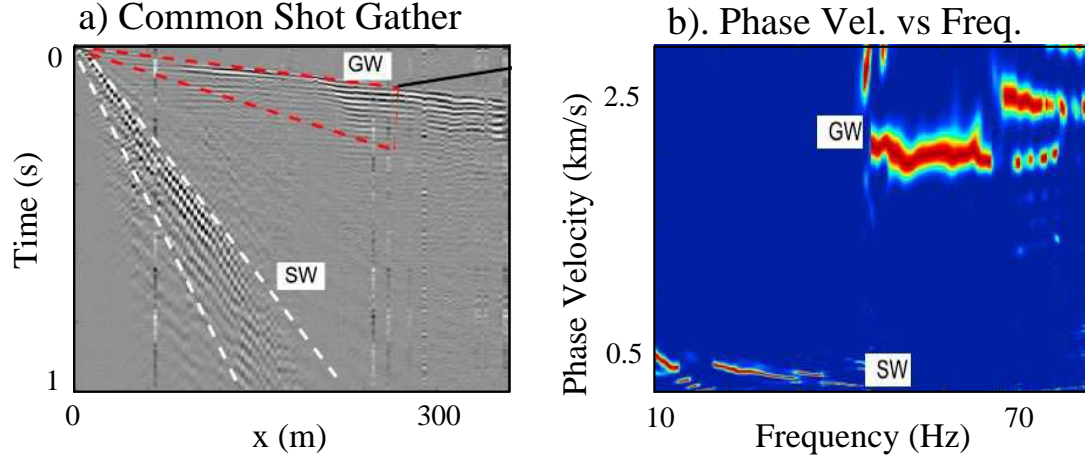


Figure 1.7: a). Common shot gather (CSG) of seismic data and b) phase velocity ( $\omega/k$ ) versus frequency ( $f = \omega/2\pi$ ) plot of the CSG. Here, the Rayleigh surface waves (SW) and P-wave guided waves (GW) are now well separated after a temporal Fourier transform and a Radon transform (Li et al., 2018).

positions of the white and red checker pieces, and the end point is the final desired outcome of, say, only white pieces remaining on the board. Here, the output is not a labeled class but it is the reward for winning the game. Rules for the game are learned by training from, e.g., chess games played by experts. A well-trained network can play against itself to increase its abilities (Sutton and Barto, 2018).

Is reinforcement learning (RL) supervised or unsupervised? Reinforcement learning is a middle ground, where it takes a small set of labeled data to begin the training, and then uses a much larger set of unlabeled data by learning the best behavior that maximizes a reward previous to the current state. It repeats this process in the next iteration to maximize the reward until no further progress can be made relative to the current state. Reinforcement Learning is also known as approximate dynamic programming, or neuro-dynamic programming in the fields of operation research and control theory. It is studied in economics under the topic of game theory ([https://en.wikipedia.org/wiki/Reinforcement\\_learning#Comparison\\_of\\_reinforcement\\_learning\\_algorithms](https://en.wikipedia.org/wiki/Reinforcement_learning#Comparison_of_reinforcement_learning_algorithms)).

A fundamental tool in RL is the Markov decision process with the following definitions ([https://en.wikipedia.org/wiki/Reinforcement\\_learning#Comparison\\_of\\_reinforcement\\_learning\\_algorithms](https://en.wikipedia.org/wiki/Reinforcement_learning#Comparison_of_reinforcement_learning_algorithms)). illustrated in Figure 1.8.

- A set  $S$  of environment and agent states is defined.
- A set  $A$  of actions of the agent is defined as

$$P(s, s') = \Pr(s_{t+1} = s' | s_t = s, a_t = a), \quad (1.3)$$

where  $\Pr(s_{t+1} = s' | s_t = s, a_t = a)$  is the probability of transition from state  $s$  to state  $s'$  after employing the action  $a$ . For example, moving the checker piece from one square to the one on the immediate right.

- $R_a(s, s')$  is the immediate reward after transition from  $s$  to  $s'$  with action  $a$ .

The goal is for the agent to learn a sequence of actions that maximize the reward function, where a neural network is often employed to learn the best actions.

Several examples illustrate the key ideas of RL.

- **Playing chess.** The novice chess player learns from past mistakes that led to being checkmated, e.g., he/she eventually learns not to sacrifice a bishop for a pawn. The player predicts a sequence of moves that might maximize the reward of winning the game, but also anticipates counter-moves by the opponent. In this way the environment of the chessboard is "sensed" at any one move, and different paths to winning are determined by both experience, intuition and logic.
- **Running.** A newly born puppy stands up on wobbly legs after being born. Trial and error experiments will soon lead to the ability to get nourishment from the mother and other food sources. The reward is to grow stronger and more adventurous.
- **Phil prepares for his exams by long hours of study.** However, poor grades force him to search for better ways to study. By trial-and-error he finds that working exercises in the back of each chapter of the course materials allows him to succeed in passing the course with flying colors. Listening and reading materials are insufficient for a deep understanding of the subject. Instead, he finds that solving problems allows one to thoroughly comprehend the topic.
- **Geophysical endeavors can also be cast into the framework of a game.** For the case of winning the game of accurately picking traveltimes in seismic data, Ma and Luo (2018) used a reinforcement learning algorithm to train the RL model. They created shot gathers in Figure 1.9a and computed energy-ratio values along a windowed section of the 1st-arrivals in Figure 1.9b; these energy ratios divided the energy along a short window of states at different transition steps by the energy of the states in the current and earlier transition steps. The energy ratios plotted over the entire shot gather is a reward map. The learning agent seeks the first-arrival onsets of the shot gather by maximizing the expected value of the cumulative discounted reward. After sufficient training the RL-picked first arrivals are depicted as the jittery red lines in Figure 1.9c-1.9d.

### 1.3 Summary

The key idea of machine learning is that it searches for patterns in data and learns from experience on how to accurately recognize such patterns in new data. A properly trained ML algorithm can often analyze a large volume of data many orders-of-magnitude faster and more thoroughly than teams of geoscientists. There are three types of ML methods: supervised learning, unsupervised learning and the combination of supervised and unsupervised methods known as reinforcement learning. The least expensive type is that of unsupervised learning, but it is also the least capable because it does not include the experience and knowledge from labeled training data. For that reason, the supervised methods are often

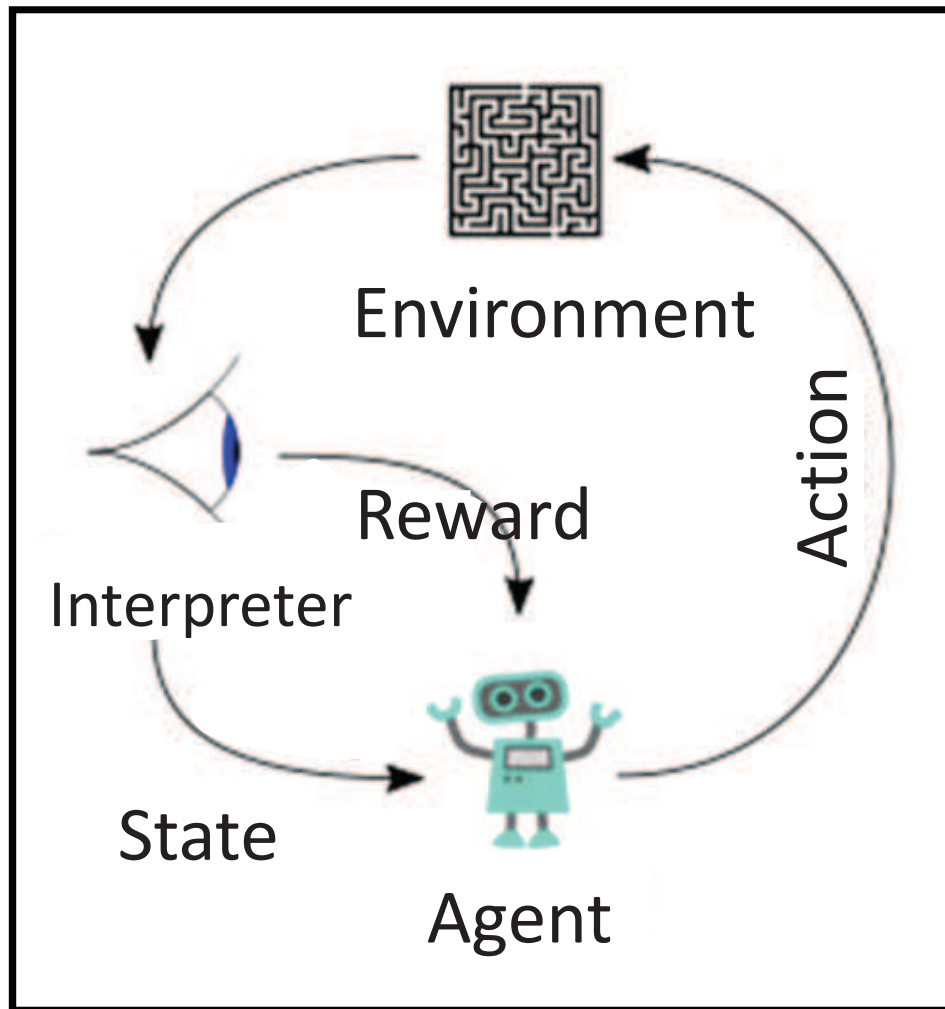


Figure 1.8: Reinforcement Learning diagram. The agent takes actions in an environment, such as the legal moves in a checkers game. These moves are accepted if they lead to a reward, such as fewer pieces of the opponent compared to your checker pieces. This process is repeated until the reward is maximized, such as the elimination of your opponent's pieces. Instead of training pairs of input-output examples, the RL algorithm learns from the games recorded from those played by human experts. Figure adapted from [https://en.wikipedia.org/wiki/Reinforcement\\_learning#Comparison\\_of\\_reinforcement\\_learning\\_algorithms](https://en.wikipedia.org/wiki/Reinforcement_learning#Comparison_of_reinforcement_learning_algorithms).

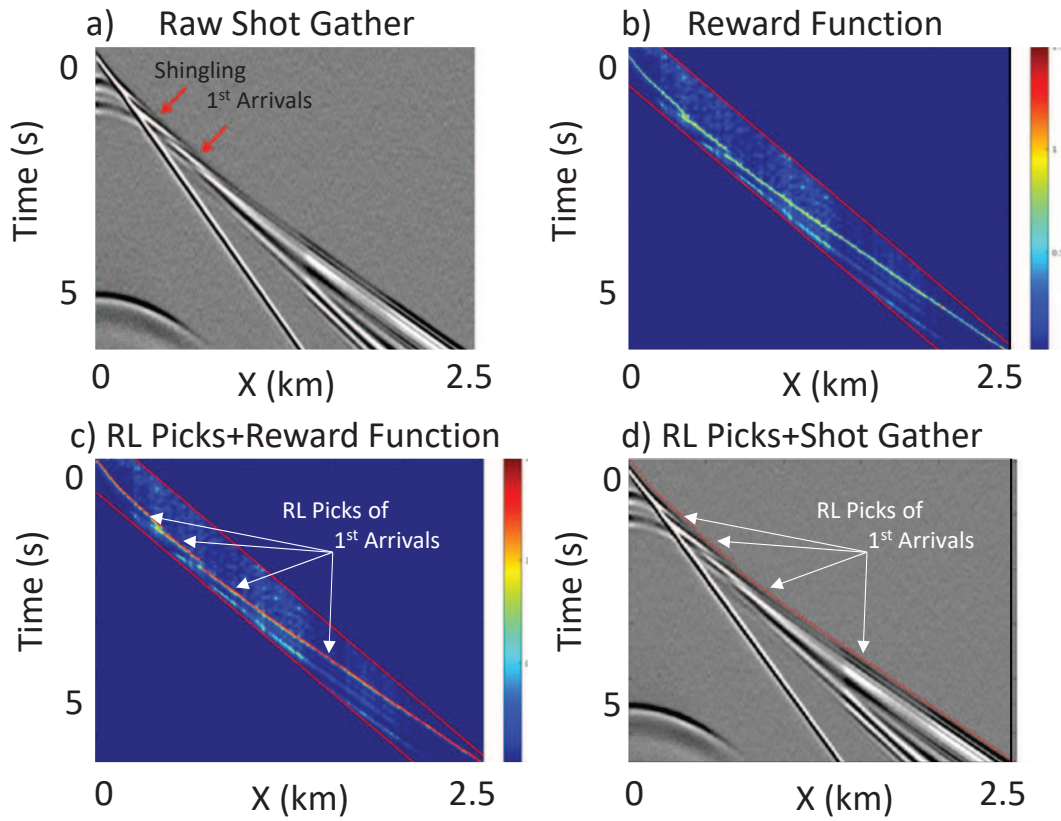


Figure 1.9: Synthetic data example showing a) a synthetic shot gather with bandlimited noise and shingling effects in the first arrivals, b) the reward function where hotter colors indicate greater rewards, c) the RL traveltimes picks overlaid with the reward function, and d) the RL picks overlaid on the shot gather. Shingled first arrivals are characterized by abrupt jumps in traveltimes of first arrivals.

preferred so that supervised learning is becoming a major force in expediting the analysis of geoscience data, including the interpretation of seismic volumes for oil exploration, analysis of satellite data, earthquake data, geochemical analysis, and potential fields data.

Many of the prominent achievements of deep learning seem magical, but also frustrating because they often appear to be theoretically impenetrable! Many of the neural-network based algorithms consist of a hedge podge of sub-architectures which, by trial-and-error, work better than any other approach in the applications of self-driving cars, language translation, speech recognition, computer vision, medical imaging and diagnosis, seismic interpretation, petrophysical analysis, etc.. Despite ongoing attempts (Papayan et al., 2017a-2017b; Sulam et al., 2018; Sharma, 2020; the Institute for Advanced Study at Princeton, <https://www.ias.edu/ideas/arora-machine-learning>; Rudolf Peierls Centre for Theoretical Physics, <https://saturdaytheory.physics.ox.ac.uk/events/ai-physics>; and many others), deep learning lacks a fully developed theoretical basis that allows for the general prediction of its performance as a function of design. But this seems to be a natural evolution of early scientific discoveries: establish an understanding of its principles by trial-and-error, intuition and examples from nature, then fully develop its theoretical principles. This is not too different than the technological developments of farming, mining, electricity and other practical tools, including full waveform inversion. For now, the geosciences are taking advantage of the currently developed ML tools and these "shut-up and compute" efforts will continue to accelerate for the next decade.



**Part I**

**Mathematical Background**



## Chapter 2

# Data Prediction by Least Squares Inversion

The goal of inversion is to find the *model*  $\mathbf{w}$  that *best predicts* the target data  $\mathbf{t}$  from input data  $\mathbf{X}$ . Here, the governing system of equations is represented by a system of equations  $\mathbf{X}\mathbf{w} = \mathbf{t}$  that are, typically, overdetermined, inconsistent and ill-conditioned. No exact solution exists so we seek the *optimal* model  $\mathbf{w}^*$  that gives the minimum prediction error  $\epsilon = \frac{1}{2} \|\mathbf{X}\mathbf{w} - \mathbf{t}\|^2$  under the  $L^2$  norm. We show how regularization and preconditioners can be used to alleviate problems with ill-conditioning, inconsistency and non-linearity. The inordinate expense of using a direct method for solving a large system of equations forces us to adopt the iterative solution method known as gradient optimization.

For general geophysical inversion, the elements of the target-data vector  $\mathbf{t}$  can take on any real-valued measurements, such as gravity readings, and the non-linear operator  $\mathbf{X}$  is characterized by the Newtonian physics of, e.g., a partial differential equation. In contrast, classification by a neural network restricts the element values of  $\mathbf{t}$  to be a restricted range of numbers<sup>1</sup> that indicate the class of the input data. For example, the input data might be photographs of dogs and cats, and the classification problem is to classify the input as either a cat where  $t_1 = 1$  and  $t_2 = 0$  or a dog where  $t_1 = 0$  and  $t_2 = 1$ . In this case the target vector is the  $2 \times 1$  vector  $\mathbf{t} = (t_1, t_2)^T$ . There is typically no governing equation based on physics, instead the model  $\mathbf{w}$  is extracted from patterns detected in a large amount of input samples  $(\mathbf{X}, \mathbf{t})$ .

### 2.1 Least Squares Inversion

The key ideas underlying least squares inversion will be explained by the simple example of determining the velocity of a motorcycle (see Figure 2.1a) as it travels down a gravel road of length 5 km. The procedure is to estimate the slowness  $w = t/x$ , i.e. the inverse speed, of the motorcycle by recording travel time  $t$  as a function of travel distance  $x$ . The timing measurements (in units of seconds) are recorded every 1 km on a gravel road next to the Bonneville Salt Flats ("[https://en.wikipedia.org/wiki/Bonneville\\_Salt\\_Flats](https://en.wikipedia.org/wiki/Bonneville_Salt_Flats)") in Utah

---

<sup>1</sup>If the output numbers of a non-linear model can be any real number then this is a non-linear regression problem.

(see Figure 2.1a) . Five travel times and distances are recorded and stored in the  $5 \times 1$  vectors  $\mathbf{t}$  and  $\mathbf{x}$ , respectively. The combined data set  $(\mathbf{x}, \mathbf{t})$  is known as a *sample* and contains errors in the traveltimes. We might also repeat the experiment at different sites (for example, see Figure 2.1c) to give us a new sample  $(\mathbf{x}', \mathbf{t}')$ ; the ensemble of samples is known as the *training data*.

### 2.1.1 Overdetermined, Inconsistent, and Ill-conditioned Equations

The relationship that links  $\mathbf{x}$  with  $\mathbf{t}$  will be hypothesized, for now, to be a linear model where

$$xw = t \rightarrow \underbrace{\begin{bmatrix} x_{11} \\ x_{21} \\ x_{31} \\ x_{41} \\ x_{51} \end{bmatrix}}_{\mathbf{X}} \underbrace{\begin{pmatrix} w_1 \end{pmatrix}}_{\mathbf{w}} = \underbrace{\begin{pmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \end{pmatrix}}_{\mathbf{t}=\text{observed times}} \rightarrow \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} \begin{pmatrix} w \end{pmatrix} = \begin{pmatrix} 0 \\ 0.102 \\ 0.33 \\ 0.50 \\ 0.80 \end{pmatrix}, \quad (2.1)$$

where  $\mathbf{X}$  is the  $5 \times 1$  input matrix of travel distances,  $\mathbf{w}$  is the  $1 \times 1$  slowness vector, and  $\mathbf{t}$  is the  $5 \times 1$  vector of recorded traveltimes. The element value  $x_{ij}$  is equal to the distance traveled to the  $i^{\text{th}}$  recording station with the motorcycle traveling at the  $j^{\text{th}}$  slowness value. The distances (units of km) and times (units of hours) associated with the above equation are plotted in Figure 2.2a where  $\mathbf{t}$  is assumed to be polluted with measurements errors, denoted as noise. The goal is to find the *best* estimate of the slowness  $w$ .

Equation 2.1 is an  $M \times N$  system of equations where  $M = 5$  is the number of equations and  $N = 1$  is the number of unknown slowness values. This is an *overdetermined* system of equations because  $M > N$ . It is also an *inconsistent* set of equations because there is no common solution. For example, the solution  $w = .102 \text{ hr/km}$  to the 2nd-row equation contradicts the solution to the 3rd-row solution  $w = .33/2 = .165 \text{ hr/km}$ .

If there are many solutions that can nearly satisfy the same equations then this is known as an *ill-conditioned system* of equations. For example, if the modeling equation is  $w_1 + 10^{-3}w_2 = t$ , then  $\delta t/\delta w_2 = 10^{-3} \rightarrow \delta w_2 = 10^3 \delta t$ , which says that a small change in the traveltime  $\delta t$  will lead to a large change in the model  $w_2$ . As an example, small traveltime errors in the data will lead to unrealistically large changes in the model. The term  $\delta t/\delta w_i$  is denoted as a *Fréchet derivative* that characterizes the sensitivity of the data  $t$  to changes in the  $i^{\text{th}}$  model parameter. More generally, inverting for the subsurface velocity model from the traveltimes of specific events picked from seismic data (see Chapter 14 and Appendix 27) is known as traveltime tomography.

### 2.1.2 Least Squares Solution

There is no exact solution to equation 2.1 so we need to define a criterion for *best solution*. The one we will discuss for now is the least squares solution<sup>2</sup> that minimizes the

---

<sup>2</sup>The least squares solution is sometimes known as regression because it the least squares solution is the one that *regresses* to the average.

a) Road next to Bonneville Salt Flats



b) Suggestion



c) Great Gravel Road to Great Salt Lake



Figure 2.1: Pictures taken on GTS motorcycle trip in Utah's West Desert with views of a) Bonneville Salt Flat region, b) wildlife sign, and c) the Great Salt Lake.

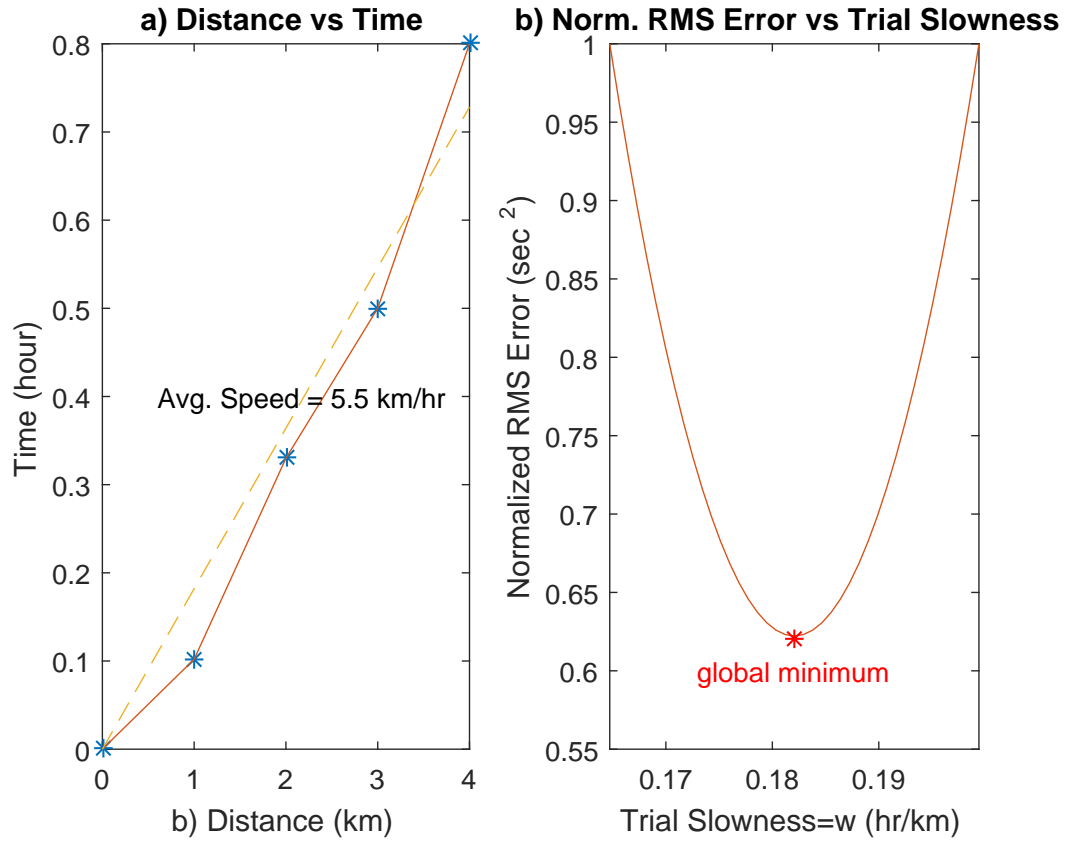


Figure 2.2: Plots for a) the observed  $t(x)$  (blue stars) and b) misfit function  $\epsilon$  for trial values of the slowness  $w$ . The dashed yellow line in a) represents the predicted times using the least squares estimate of the velocity and the red star in b) indicates the global minimum  $\epsilon(\mathbf{w}^*)$  where  $w^* = 1/5.5 = 0.182$  is the stationary point.

objective function<sup>3</sup>  $\epsilon$ , the sum of the squared residuals:

$$\begin{aligned}
 \epsilon &= \frac{1}{2} \overbrace{(\mathbf{X}\mathbf{w} - \mathbf{t})^T}^{\mathbf{r}^T} \overbrace{(\mathbf{X}\mathbf{w} - \mathbf{t})}^{\mathbf{r}}, \\
 &= \frac{1}{2} \mathbf{w}^T \mathbf{X}^T \mathbf{X} \mathbf{w} - \mathbf{t}^T \mathbf{X} \mathbf{w} + \frac{1}{2} \mathbf{t}^T \mathbf{t} \\
 &= \frac{1}{2} \sum_{i=1}^M \overbrace{((\mathbf{X}\mathbf{w})_i - t_i)^2}^{r_i^2},
 \end{aligned} \tag{2.2}$$

where  $\epsilon$  is also denoted as a data misfit function. Here,  $\mathbf{r}$  is the  $M \times 1$  residual vector which is the difference between the predicted times  $\mathbf{X}\mathbf{w}$  and measured times  $\mathbf{t}$ . For  $N = 1$ ,  $\epsilon$  plots out as the parabola in Figure 2.2b for the motorcycle data, for  $N = 2$  it describes the error bowl in Figure 2.3, and for  $N > 2$  it describes an  $N - 1$ -dimensional quadric surface.

The vertices of the parabola in Figure 2.2b and the error bowl in Figure 2.3 define the global minimum  $\mathbf{w}^*$  where the gradient  $\nabla \epsilon$  is zero. This zero-slope condition<sup>4</sup> is written as

$$\left. \frac{\partial \epsilon}{\partial w_k} \right|_{\mathbf{w}=\mathbf{w}^*} = 0, \quad \forall k, \tag{2.3}$$

where the  $k^{\text{th}}$  component of the  $N \times 1$  *misfit gradient* vector is explicitly written as

$$\begin{aligned}
 \nabla \epsilon_k = \frac{\partial \epsilon}{\partial w_k} &= \sum_{i=1}^{M=5} r_i \frac{\partial r_i}{\partial w_k}, \\
 &= \sum_{i=1}^5 r_i \underbrace{\frac{\partial r_i}{\partial w_k}}_{\substack{\partial r_i / \partial w_k = \sum_{n=1}^N x_{in} \partial w_n / \partial w_k = \sum_{n=1}^N x_{in} \delta_{nk} = x_{ik}}} \underbrace{\frac{\partial (\sum_{n=1}^N x_{in} w_n - t_i)}{\partial w_k}}_{\substack{(\mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{t}))_k}}, \\
 &= \sum_{i=1}^5 x_{ik} \left( \sum_{n=1}^N x_{in} w_n - t_i \right),
 \end{aligned} \tag{2.4}$$

and the *Kronecker delta function* is defined to be  $\delta_{ik} = 1$  if  $i = k$ , otherwise  $\delta_{ik} = 0$ . Notice that the outermost summation  $\sum_{i=1}^5 x_{ik} r_i$  in the last line is over the row index  $i$  of the element  $x_{ik}$ , so in matrix-vector notation this is equivalent to multiplying the transpose matrix  $\mathbf{X}^T$  by the residual vector  $\mathbf{r}$ .

Setting the derivative in equation 2.4 to be zero and rearranging gives the *normal equa-*

<sup>3</sup>The machine learning community often denotes  $\epsilon$  as the *loss function*, which can take many forms;  $\epsilon$  is also known as a data misfit function if there is no regularization.

<sup>4</sup>Equation 2.3 is considered to be a stationary condition because  $\epsilon$  does not change much over a small range of  $w$  at the bottom of the, for example, red curve in Figure 2.2.

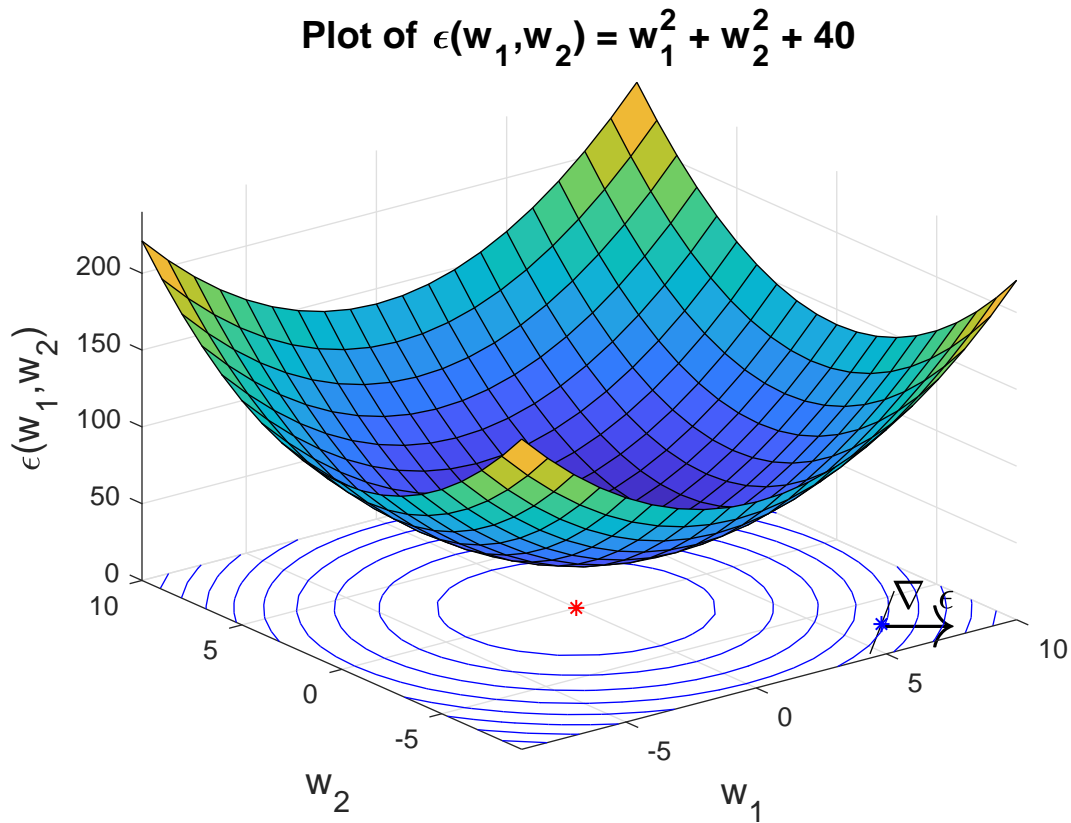


Figure 2.3: Misfit function and contours for the quadratic polynomial  $\epsilon = w_1^2 + w_2^2 + 40$ , where the red dot is directly below the global minimum at  $\mathbf{w}^* = (w_1^*, w_2^*)$ . The gradient  $\nabla\epsilon = (\frac{\partial\epsilon}{\partial w_1}, \frac{\partial\epsilon}{\partial w_2})$  points along the steepest uphill direction and is perpendicular to the contour tangent. See exercises 2.5.3-2.5.6.



tions<sup>5</sup>

$$[\mathbf{X}^T \mathbf{X}] \mathbf{w} = \mathbf{X}^T \mathbf{t}. \quad (2.5)$$

The least squares solution to the normal equations is

$$\mathbf{w} = [\mathbf{X}^T \mathbf{X}]^{-1} \mathbf{X}^T \mathbf{t}, \quad (2.6)$$

which minimizes the sum of the squared residuals. See Box 2.1.2 for the least squares solution to equation 2.1.

#### Example 2.1.1. Least Squares Solution

For equation 2.1, there is only one unknown in  $\mathbf{w} = (w)$  so that the stationary condition for equation 2.2 is

$$\frac{\partial \epsilon}{\partial w} = \sum_{i=1}^5 x_{i1}(x_{i1}w - t_i) = 0. \quad (2.7)$$

Rearranging and solving for  $w$  gives

$$w = \frac{\overbrace{[\mathbf{X}^T \mathbf{X}]^{-1}}^1 \overbrace{\mathbf{X}^T \mathbf{t}}^{\sum_{i=1}^5 x_{i1} t_i}}{\sum_{i=1}^5 x_{i1}^2} = 0.182. \quad (2.8)$$

The stationary value  $w^* = 0.182 \text{ hr/km}$  is at the global minimum indicated by the red star in Figure 2.2b.

### 2.1.3 Regularized Least Squares Solution

If there is a timing error  $\delta t_i$  at each of the  $i^{\text{th}}$  recording stations so that  $t_i \rightarrow t_i^0 + \delta t_i$ , then equation 2.8 becomes

$$w = \frac{\overbrace{\sum_{i=1}^5 x_{i1} t_i^0}^{w_o}}{\sum_{i=1}^5 x_{i1}^2} + \frac{\overbrace{\sum_{i=1}^5 x_{i1} \delta t_i}^{\delta w}}{\sum_{i=1}^5 x_{i1}^2}, \quad (2.9)$$

<sup>5</sup>These are called normal equations because they can be multiplied by  $\mathbf{w}^T$  and rearranged into the form  $\mathbf{w}^T \mathbf{X}^T (\mathbf{X} \mathbf{w} - \mathbf{t}) = (\mathbf{X} \mathbf{w}, \mathbf{X} \mathbf{w} - \mathbf{t}) = 0$ . This says that  $\mathbf{w}^*$  gives the predicted traveltime vector  $\mathbf{X} \mathbf{w}^*$  that is perpendicular to the residual vector  $\mathbf{X} \mathbf{w}^* - \mathbf{t}$ . Here, the parentheses indicate the inner product between two vectors.

where  $w_o$  is the true slowness and  $\delta w$  is the slowness error caused by timing errors. If the distances  $x_{i1}$  are very small<sup>6</sup> then the denominator in  $\frac{1}{\sum x_{i1}^2}$  can be close to zero and so amplify timing errors  $\delta t_i > x_{i1}$  into a large slowness error  $|\delta w| = \left| \frac{\sum_{i=1}^5 x_{i1} \delta t_i}{\sum_{i=1}^5 x_{i1}^2} \right| \gg 0$ . To avoid this *near-zero-divide* instability we add a positive *regularization* term  $\eta > 0$  to the denominator so that equation 2.8 becomes the regularized solution:

$$w = \frac{\sum_{i=1}^5 x_{i1} t_i}{\sum_{i=1}^5 x_{i1}^2 + \eta}, \quad (2.10)$$

where  $\eta$  is typically set to be between 1% and 5% of the largest diagonal value of the matrix  $\mathbf{X}^T \mathbf{X}$ . This *regularization parameter*  $\eta > 0$ , also known as the *damping term*, increases the value of the denominator and thereby prevents an unstable near-zero divide and the strong amplification of data errors (Menke, 1984). The cost, however, is a loss of accuracy in the final solution. As discussed in Box 2.1.3, an unstable system of equations can be characterized by several properties: gentle slopes in the objective function  $\frac{\partial \epsilon}{\partial w} \approx 0$  and large condition numbers associated with  $\mathbf{X}^T \mathbf{X}$ . See exercises 2.5.1-2.5.2.

---

<sup>6</sup>For example, assume that the five timing stations are spaced at 0.001 km intervals. In this case the timing errors can be very large compared to  $x_{i1}$  and so generate enormous slowness errors.

**Key Idea 2.1.1. Small Curvatures and Large Condition Numbers of  $\mathbf{X}^T \mathbf{X} \rightarrow$  Unstable Solutions**

The green curves in Figure 2.4 depict a) stable and b) less-stable misfit functions based on the magnitude of  $x_{i1}$  in  $\mathbf{X}$ . The larger value of  $x_{i1} = 0.001$  in Figure 2.4a leads to the parabola with steeper sides (green line) compared to the gently dipping one in Figure 2.4b where  $x_{i1} = 0.00011$ . The flatter green parabola in b) means that there is a wider range of  $w$  values, i.e. a more unstable solution, that can give almost the same misfit error as that at the global minimum (green star). Equivalently, this means that a small amount of data noise can lead to a large change in the model. This is an example of an ill-conditioned system of equations that forms the objective function.

If the curvature  $\frac{\partial^2 \epsilon}{\partial w^2}$  is close to zero at  $\mathbf{w}^*$ :

$$\frac{\partial^2 \epsilon}{\partial w^2} = \lim_{\delta w \rightarrow 0} \left[ \frac{\epsilon}{\partial w} \Big|_{w=w^*+\delta w} - \frac{\epsilon}{\partial w} \Big|_{w=w^*} \right] \approx 0, \quad (2.11)$$

then the slope  $\frac{\partial \epsilon}{\partial w}$  hardly changes around the zero-slope point  $w^*$ . This means that there are many solutions that almost minimize the sum of squared residuals. For equation 2.7 the curvature is

$$\frac{\partial^2 \epsilon}{\partial w^2} = \sum_{M=1}^5 x_{i1}^2, \quad (2.12)$$

which confirms that  $\frac{\partial^2 \epsilon}{\partial w^2} \approx 0$  when  $|x_{i1}| \approx 0 \forall i$ .

A general indicator of an unstable system of equations is that the condition number  $|\lambda_{max}/\lambda_{min}|$  of the matrix  $\mathbf{X}^T \mathbf{X}$  is large, where  $\lambda_{max}$  ( $\lambda_{min}$ ) is the maximum (minimum) eigenvalue of the matrix. Recall that the  $i^{th}$  orthogonal eigenvector  $\mathbf{x}_i$  of the symmetric positive definite matrix<sup>a</sup>  $\mathbf{X}^T \mathbf{X}$  is defined as

$$\mathbf{X}^T \mathbf{X} \mathbf{x}_i = \lambda_i \mathbf{x}_i, \quad (2.13)$$

where  $\lambda_i$  is the eigenvalue for the eigenvector  $\mathbf{x}_i$ . This means that the solution to the  $N \times N$  normal equations 2.5 can be expanded into a sum of  $N$  weighted eigenvectors

$$\mathbf{w} = \sum_{i=1}^N \beta_i \mathbf{x}_i. \quad (2.14)$$

The unknown coefficients  $\beta_i$  can be found by inserting equation 2.14 into the normal equations  $\mathbf{X}^T \mathbf{X} \mathbf{w} = \mathbf{X}^T \mathbf{t}$ , taking its dot product with the  $i^{th}$  eigenvector  $\mathbf{x}_i$ , and solving for  $\beta_i$  to get

$$\beta_i = \frac{(\mathbf{x}_i, \mathbf{X}^T \mathbf{t})}{\lambda_i}. \quad (2.15)$$

This says that if there are small timing errors  $\delta \mathbf{t}$  in  $(\mathbf{x}_i, \mathbf{X}^T(\mathbf{t} + \delta \mathbf{t}))$  then they will strongly amplified if the  $i^{th}$  eigenvalue  $\lambda_i$  is small compared to errors associated with much larger eigenvalues. Thus, normal equations with condition numbers  $|\lambda_{max}|/|\lambda_{min}|$  larger than about  $10^4$  should be regularized.

<sup>a</sup>A symmetric positive definite (SPD) matrix is one that is symmetric and all of its eigenvalues  $\lambda_i$   $i \in [1, 2 \dots N]$  are larger than 0, and the  $N$  orthonormal eigenvectors  $\mathbf{x}_i$   $i \in [1, 2 \dots N]$  span the space of solutions.

The damping term  $\eta$  in equation 2.10 was introduced in an ad hoc fashion to mitigate large errors in  $w$  due to an unstable system of equations. A more rigorous derivation is obtained by defining the objective function to be the weighted sum of data misfit  $\mathbf{r}^T \mathbf{r} = (\mathbf{X}\mathbf{w} - \mathbf{t})^T (\mathbf{X}\mathbf{w} - \mathbf{t})$  (green curves in Figure 2.4) and the model penalty  $\mathbf{w}^T \mathbf{w}$  (red curves in Figure 2.4) functions:

$$\begin{aligned} \epsilon &= \overbrace{\frac{1}{2}(\mathbf{X}\mathbf{w} - \mathbf{t})^T (\mathbf{X}\mathbf{w} - \mathbf{t})}^{\text{data misfit}=\frac{1}{2}\sum_i r_i^2} + \overbrace{\frac{1}{2}\eta\mathbf{w}^T \mathbf{w}}^{\text{model penalty}=\frac{1}{2}\eta\sum_i w_i^2}, \\ &= \frac{1}{2}\mathbf{w}^T [\mathbf{X}^T \mathbf{X} + \eta \mathbf{I}] \mathbf{w} - \mathbf{t}^T \mathbf{X} \mathbf{w} + \frac{1}{2} \mathbf{t}^T \mathbf{t}, \end{aligned} \quad (2.16)$$

which plots out as the blue curves in Figure 2.4. The derivative of equation 2.16 with respect to  $w_k$  yields the  $k^{th}$  component of the *misfit gradient*:

$$\begin{aligned} \frac{\partial \epsilon}{\partial w_k} &= \frac{1}{2} \frac{\partial \sum_i r_i^2}{\partial w_k} + \frac{\eta}{2} \sum_i \frac{\partial w_i^2}{\partial w_k}, \\ &= \underbrace{\sum_i x_{ik} r_i}_{(\mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{t}))_k} + \underbrace{\eta w_k}_{(\eta \mathbf{I} \mathbf{w})_k}, \end{aligned} \quad (2.17)$$

where  $\mathbf{I}$  is the  $N \times N$  identity matrix. Setting this gradient component to zero  $\forall k \in [1, 2 \dots N]$  and rearranging yields the *regularized normal equations*

$$[\mathbf{X}^T \mathbf{X} + \eta \mathbf{I}] \mathbf{w} = \mathbf{X}^T \mathbf{t}, \quad (2.18)$$

which is solved by the regularized least squares solution:

$$\mathbf{w} = [\mathbf{X}^T \mathbf{X} + \eta \mathbf{I}]^{-1} \mathbf{X}^T \mathbf{t}. \quad (2.19)$$

As a special case, setting the derivative in equation 2.17 to zero, renaming  $w_k \rightarrow w$  and solving for  $w$  gives equation 2.10.

The interpretation of equation 2.16 is that large values of  $\eta$  will favor solutions  $\mathbf{w}$  that are near the origin, but at the expense of a larger data misfit function  $\|\mathbf{r}\|^2$ . As an example, the global minimum  $w^*$  (blue star) for the blue curve in Figure 2.4b is closer to the origin than that in Figure 2.4a because  $\eta$  is 20% for b) and only 10% for a). However, if  $\eta$  is not too large then the damped solution (blue star) is close enough to the undamped solution (green star) with an acceptable loss of precision.

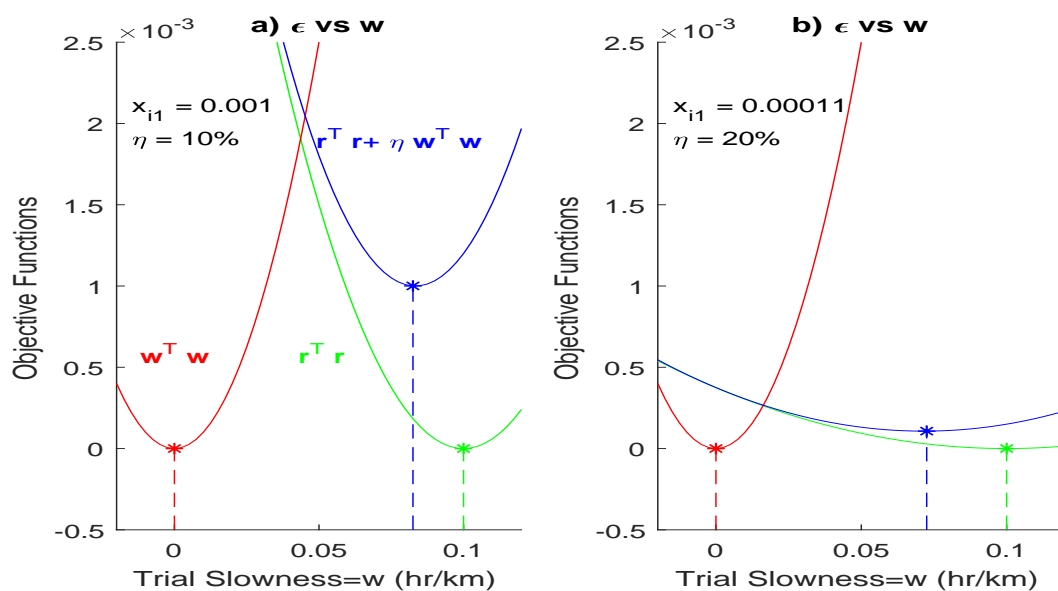


Figure 2.4: Values of the objective functions  $\mathbf{r}^T \mathbf{r}$  (green curves),  $\mathbf{w}^T \mathbf{w}$  (red curves) and  $\mathbf{r}^T \mathbf{r} + \eta \mathbf{w}^T \mathbf{w}$  (blue curves) plotted against trial values of  $w$  for the motorcycle equations 2.1. Here, there is no noise in the data so that  $(\mathbf{X}^T \mathbf{w} - \mathbf{t})^T (\mathbf{X}^T \mathbf{w} - \mathbf{t}) = 0$  at the global minimum denoted by the green stars. In contrast, the minimizer  $w^*$  for  $(\mathbf{X} \mathbf{w} - \mathbf{t})^T (\mathbf{X} \mathbf{w} - \mathbf{t}) + \eta \mathbf{w}^T \mathbf{w}$  is non-zero and is closer to  $w = 0$  for b)  $\eta = 20\%$  compared to a)  $\eta = 10\%$ . Larger values of  $\eta$  reward solutions with smaller length  $\mathbf{w}^T \mathbf{w} \approx 0$ . In addition, the slope of the green parabola is gentler for b) smaller values of  $x_{i1} = .00011$  compared to larger values in a)  $x_{i1} = .001$  in a).

The curvature  $\partial^2\epsilon/\partial w_i\partial w_j$  tensor of the misfit function in equation 2.2 is given as

$$\begin{aligned}\frac{\partial^2\epsilon}{\partial w_i\partial w_j} &= 1/2 \frac{\partial}{\partial w_i} \left[ \frac{\partial \mathbf{w}^T \mathbf{H} \mathbf{w}}{\partial w_j} \right], \\ &= \frac{\partial}{\partial w_i} [\mathbf{H} \mathbf{w}]_j, \\ &= H_{ij},\end{aligned}\tag{2.20}$$

Here,  $\mathbf{H} = \mathbf{X}^T \mathbf{X}$  is the Hessian matrix. Therefore,  $H_{ij}$  is the component of the curvature tensor which determines how quickly the  $j^{\text{th}}$  component of slope changes along the  $i$  direction. The Hessian matrix is symmetric so that  $H_{ij} = H_{ji}$ . Large values of  $|H_{ii}| \gg |H_{ij}| > 0$  indicate a steeply dipping topographically of the misfit function in the  $i^{\text{th}}$  direction.

#### 2.1.4 Geometrical Interpretation of Gradient $\nabla\epsilon$

The gradient vector  $\nabla\epsilon$  in Figure 2.3 points uphill along the steepest *ascent* direction. To show this, define the vector

$$\mathbf{w} = \mathbf{w}_o + \Delta\mathbf{w},\tag{2.21}$$

where  $\Delta\mathbf{w}$  is a specified vector with very small magnitude. Expanding  $\epsilon(\mathbf{w})$  in a Taylor series about  $\mathbf{w}_o$ , truncating after the first-order term in  $\Delta\mathbf{w}$  and rearranging gives:

$$\epsilon(\mathbf{w}) - \epsilon(\mathbf{w}_o) = \nabla\epsilon(\mathbf{w})^T \Big|_{\mathbf{w}=\mathbf{w}_o} \Delta\mathbf{w},\tag{2.22}$$

where  $\nabla\epsilon = (\frac{\partial\epsilon}{\partial w_1}, \frac{\partial\epsilon}{\partial w_2} \dots \frac{\partial\epsilon}{\partial w_N})^T$  and  $\Delta\mathbf{w} = (w_1, w_2 \dots w_N)^T$ . If  $\Delta\mathbf{w}$  is pointing to a close neighboring point on the same contour that passes through  $\mathbf{w}_o$  then  $\epsilon(\mathbf{w}_o + \Delta\mathbf{w}) = \epsilon(\mathbf{w}_o)$ , which implies  $(\nabla\epsilon(\mathbf{w}_o + \Delta\mathbf{w}), \Delta\mathbf{w}) = 0$ . This means that  $\nabla\mathbf{w}$  is perpendicular to the contour's tangent vector at  $\mathbf{w}_o$ . Furthermore, if we redefine  $\Delta\mathbf{w}$  to be a unit vector now pointing uphill and perpendicular to the contour at  $\mathbf{w}_o$  then  $\epsilon(\mathbf{w}) - \epsilon(\mathbf{w}_o) = \nabla\epsilon^T \Delta\mathbf{w} = |\nabla\epsilon| > 0$ . Thus,  $\mathbf{w}_o$  points uphill along the steepest *ascent* direction.

As illustrated in Figure 2.5, the gradient vector points uphill along the steepest ascent direction. Therefore,  $-\nabla\epsilon$  points downhill along the steepest descent direction as shown in Figure 2.5. This compares to the regularized gradient in equation 2.17 which points more towards the origin in Figure 2.5 because it is a weighted sum of the misfit gradient and the gradient of the penalty function  $1/2\nabla\mathbf{w}^T \mathbf{w} = \mathbf{w}$ . In fact, if  $\eta \gg 0$  then the penalty term dominates and  $\nabla\epsilon$  points at the origin.

The Gauss-Newton arrow  $\Delta\mathbf{w}$  in Figure 2.5 points toward the global minimum at  $\mathbf{w}^*$  with

$$\Delta\mathbf{w} = [\mathbf{X}^T \mathbf{X}]^{-1} \mathbf{X}^T \Delta\mathbf{t},\tag{2.23}$$

where  $\Delta\mathbf{t} = \mathbf{t} - \mathbf{t}'$ ,  $\Delta\mathbf{w} = \mathbf{w}^* - \mathbf{w}'$  and

$$\mathbf{X}\mathbf{w}' = \mathbf{t}'.\tag{2.24}$$

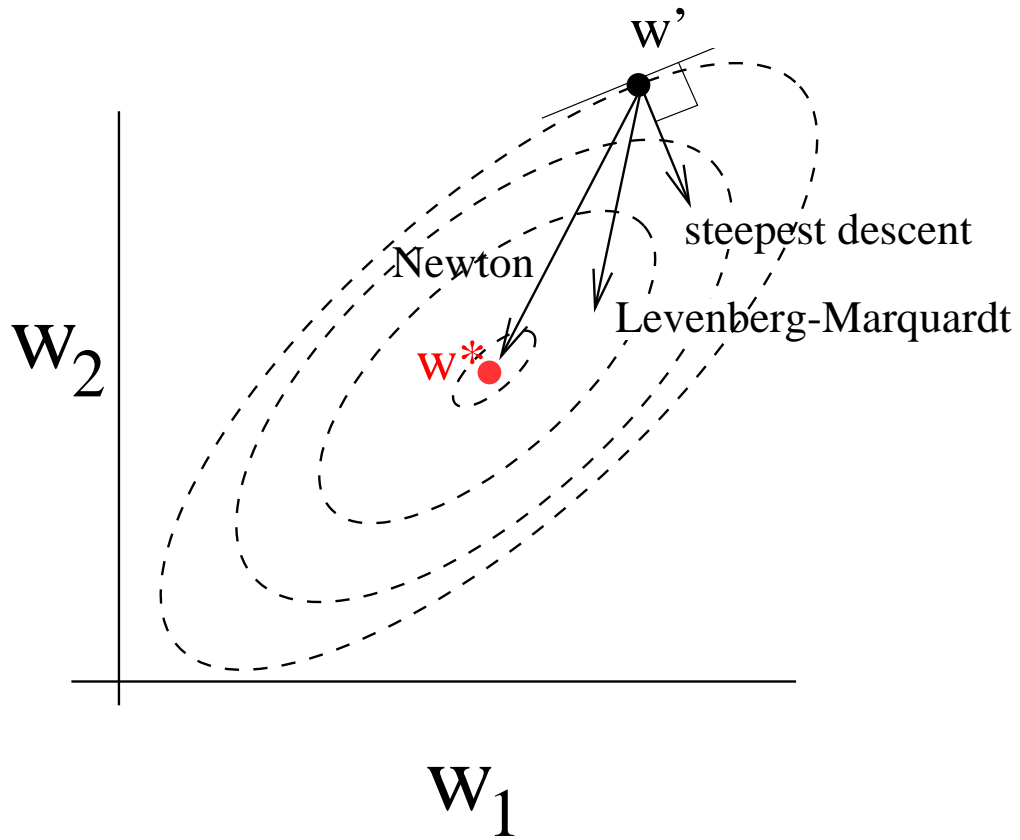


Figure 2.5: Elliptical contours associated with  $\epsilon(\mathbf{w})$  and the gradient vectors associated with the Gauss-Newton, Levenberg-Marquardt, and steepest descent directions. Here, the Levenberg-Marquardt gradient is also known as the regularized gradient (see equation 2.19) and the red star denotes the smallest value of  $\epsilon(\mathbf{w})$ .

Equation 2.23 can be proven by recalling that the optimal  $\mathbf{w}^*$  for the given traveltimes  $\mathbf{t}$  satisfies  $\mathbf{w}^* = [\mathbf{X}^T \mathbf{X}]^{-1} \mathbf{X}^T \mathbf{t}$ . The sub-optimal model  $\mathbf{w}'$  predicts the traveltimes  $\mathbf{t}'$  that exactly satisfy equation 2.24 but these traveltimes are not the same as the recorded ones  $\mathbf{t}$ . Multiplying equation 2.24 by  $\mathbf{X}^T$  and inverting gives

$$\mathbf{w}' = [\mathbf{X}^T \mathbf{X}]^{-1} \mathbf{X}^T \mathbf{t}'. \quad (2.25)$$

Subtracting  $\mathbf{w}'$  from  $\mathbf{w}^* = [\mathbf{X}^T \mathbf{X}]^{-1} \mathbf{X}^T \mathbf{t}$  gives the Gauss-Newton direction  $\Delta \mathbf{w} = \mathbf{w}^* - \mathbf{w}'$  where

$$\begin{aligned} \Delta \mathbf{w} &= [\mathbf{X}^T \mathbf{X}]^{-1} \mathbf{X}^T \mathbf{t} - [\mathbf{X}^T \mathbf{X}]^{-1} \mathbf{X}^T \mathbf{t}', \\ &= [\mathbf{X}^T \mathbf{X}]^{-1} \mathbf{X}^T \overbrace{(\mathbf{t} - \mathbf{t}')}^{\Delta \mathbf{t}}. \end{aligned} \quad (2.26)$$

Substituting  $\Delta \mathbf{w} = \mathbf{w}^* - \mathbf{w}'$  into the above equation and rearranging gives the Gauss-Newton update formula for non-linear inversion:

$$\mathbf{w}^* = \mathbf{w}' + [\mathbf{X}^T \mathbf{X}]^{-1} \mathbf{X}^T \Delta \mathbf{t}. \quad (2.27)$$

This formula is the starting point for many gradient descent methods that iteratively search in downhill directions for the global minimum. See section 2.2.

### 2.1.5 Preconditioning

If the number of unknowns  $N = 2$  so that  $\mathbf{w} = (w_1, w_2)$ , then the objective function  $\epsilon = 1/2 \|\mathbf{X}\mathbf{w} - \mathbf{t}\|^2$  can be described by the second-order polynomial

$$\epsilon = aw_1^2 + bw_2^2 + cw_1w_2 + dw_1 + ew_2 + f, \quad (2.28)$$

where  $(a, b, c, d, e, f)$  are functions of the input data. As a simple example, set  $c = d = e = f = 0$  so equation 2.28 can be rewritten as

$$\epsilon = \begin{pmatrix} w_1 & w_2 \end{pmatrix} \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}, \quad (2.29)$$

so that  $\epsilon$  plots out as the concentric circles in Figure 2.6a for  $a = b$ , or as the ellipses in Figure 2.6b for  $(a, b) = (1, .01)$ . If  $a \gg b$  then the ellipse become much taller than it is wide and leads to unstable solutions as discussed in Box 2.1.3. For example, a small value of  $b$  says that  $|\frac{\partial \epsilon}{\partial w_2}|, |\frac{\partial^2 \epsilon}{\partial w_2^2}| \approx 0$  so that large changes in  $w_2$  lead to small changes in  $\epsilon$ . Another way of saying this is that small changes in  $\delta t$  and, consequently,  $\epsilon$  can lead to large changes in  $w_2$ . In contrast, the steep dip of the ellipse along the  $w_1$  axis says that  $|\partial \epsilon / \partial w_1| \gg 0$  so that  $w_1$  is not affected too much by small timing errors.

The  $2 \times 2$  matrix  $\mathbf{X}$  associated with equation 2.29 is given by

$$\mathbf{X} = \begin{bmatrix} a^{1/2} & 0 \\ 0 & b^{1/2} \end{bmatrix}. \quad (2.30)$$



Here,  $\mathbf{X}$  is severely ill-conditioned if  $|a^{1/2}| \gg |b^{1/2}|$  so that the associated ellipses of  $\epsilon$  appear as long-narrow valleys. To transform these narrow valleys into rounder contours a diagonal *preconditioner matrix*  $\mathbf{C}$

$$\mathbf{C} = \begin{bmatrix} a^{-1/2} & 0 \\ 0 & b^{-1/2} \end{bmatrix}. \quad (2.31)$$

can be multiplied by the original equations to give

$$\mathbf{C}\mathbf{X}\mathbf{w} = \mathbf{C}\mathbf{t}, \quad (2.32)$$

where  $[\mathbf{C}]_{ij} = \frac{1}{[\mathbf{X}]_{ii}}\delta_{ij}$ . In this case,  $\mathbf{C}\mathbf{X} = \mathbf{I}$  and has unit-valued eigenvalues  $\lambda_1 = \lambda_2 = 1$  where the condition number  $|\lambda_2|/|\lambda_1| = 1$ . Thus, the associated objective function of  $\frac{1}{2}\|\mathbf{C}(\mathbf{X}\mathbf{w} - \mathbf{t})\|^2$  plots out as round circles.

Normalizing each row of  $\mathbf{X}$  by  $\mathbf{C}\mathbf{X}$  so it has about the same strength as the other ones is one type of preconditioning method<sup>7</sup> for accelerating convergence of iterative gradient methods. Preconditioning the data is also used in neural networks (see Chapter 9) where it is recommended that there should be the same number of data examples for any one type. Moreover, different types of input data will have different units, so normalizing one type of data (e.g. with units of km/s) by its variance will insure that the range of values will be about the same as those from a different type of data (e.g., units of kilograms). A similar preconditioning is used for illumination compensation of seismic data (Rickett, 2003) where the weak strength of deep reflections is compensated to balance out the amplitudes of much stronger early arrivals recorded near the source.

If the system of equations is that for the normal equations then, in general,  $\mathbf{C}$  is selected so that

$$\mathbf{C}[\mathbf{X}^T\mathbf{X}]\mathbf{C}^{-1}\mathbf{C}\mathbf{w} = \mathbf{C}\mathbf{X}^T\mathbf{t}, \quad (2.33)$$

where the eigenvalues of  $\mathbf{C}[\mathbf{X}^T\mathbf{X}]\mathbf{C}^{-1}$  are more favorable for faster convergence. For example, construct  $\mathbf{C}$  so that the condition number of  $\mathbf{C}[\mathbf{X}^T\mathbf{X}]\mathbf{C}^{-1}$  is much smaller than that for  $[\mathbf{X}^T\mathbf{X}]$ . The condition number can also be lowered by clustering the eigenvalues of  $\mathbf{C}[\mathbf{X}^T\mathbf{X}]\mathbf{C}^{-1}$  (Nocedal and Wright, 1999).

### 2.1.6 Overfitting Data

Equation 2.1 assumes that the simple one-dimensional model  $\mathbf{w} = (w)$  can explain the data. The result is a system of equations that is inconsistent because of, presumably, timing errors. However, the timing expert might disagree and say the data errors are too small to account for deviations from the dashed line in Figure 2.2a: we must have used the wrong physics to explain the data! Instead of the linear model  $t = x/v$ , a quadratic model with polynomial order  $P = 2$  in  $x$  might be a better fit where

$$t = w_1x + w_2x^2. \quad (2.34)$$

---

<sup>7</sup>Using a diagonal preconditioner is sometimes known as scaling (Nocedal and Wright, 1999)

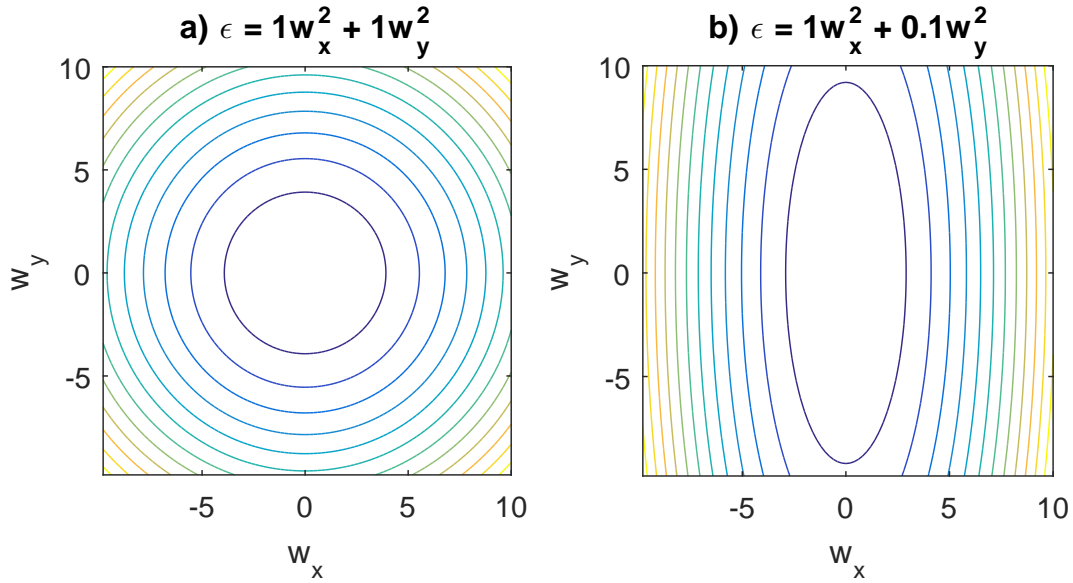


Figure 2.6: Contours of objective function  $aw_1^2 + bw_2^2$  for a)  $(a,b) = (1,1)$  and b)  $(a,b) = (1,0.1)$ .

In this case, the corresponding system of equations for the motorcycle data becomes

$$\underbrace{\begin{bmatrix} x_{11} & x_{11}^2 \\ x_{21} & x_{21}^2 \\ x_{31} & x_{31}^2 \\ x_{41} & x_{41}^2 \\ x_{51} & x_{51}^2 \end{bmatrix}}_{\mathbf{X}} \underbrace{\begin{pmatrix} w_1 \\ w_2 \end{pmatrix}}_{\mathbf{w}} = \underbrace{\begin{pmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \end{pmatrix}}_{\mathbf{t}=\text{observed times}} \rightarrow \begin{bmatrix} 0 & 0 \\ 1 & 1^2 \\ 2 & 2^2 \\ 3 & 3^2 \\ 4 & 4^2 \end{bmatrix} \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0.102 \\ 0.33 \\ 0.50 \\ 0.80 \end{pmatrix}. \quad (2.35)$$

The intuitive motivation for this quadratic model is that the motorcycle speed increases towards the end of the road as it becomes more paved. Thus, the quadratic term  $w_2x^2$  dominates when  $x$  is large near the end of the road.

As an example, Figure 2.7 depicts the results from fitting the quadratic equation 2.34 to the motorcycle data. Comparing Figures 2.2 and 2.7, the quadratic model gives the closest fit to the data in a) and has about 1/7 the misfit error of the linear model. This result suggests that the misfit error can be further decreased by increasing the order  $P$  of the polynomial model. Unfortunately, increasing  $P > M$  will yield an *underdetermined* system of equations where there are more unknowns  $P$  than the number  $M$  of equations. The least squares solution will then give a misfit error of nearly zero, but at the cost of *overfitting* the data where the complexity of the model exceeds that of the noise-free data. In this case the higher-order polynomials are being fitted to the rapidly varying noise, not the signal, in the data. Such a model will not easily be able to predict new data for experiments with much different noise. Another way of saying this in the machine learning community is that this *overfitted model does not generalize very well*.

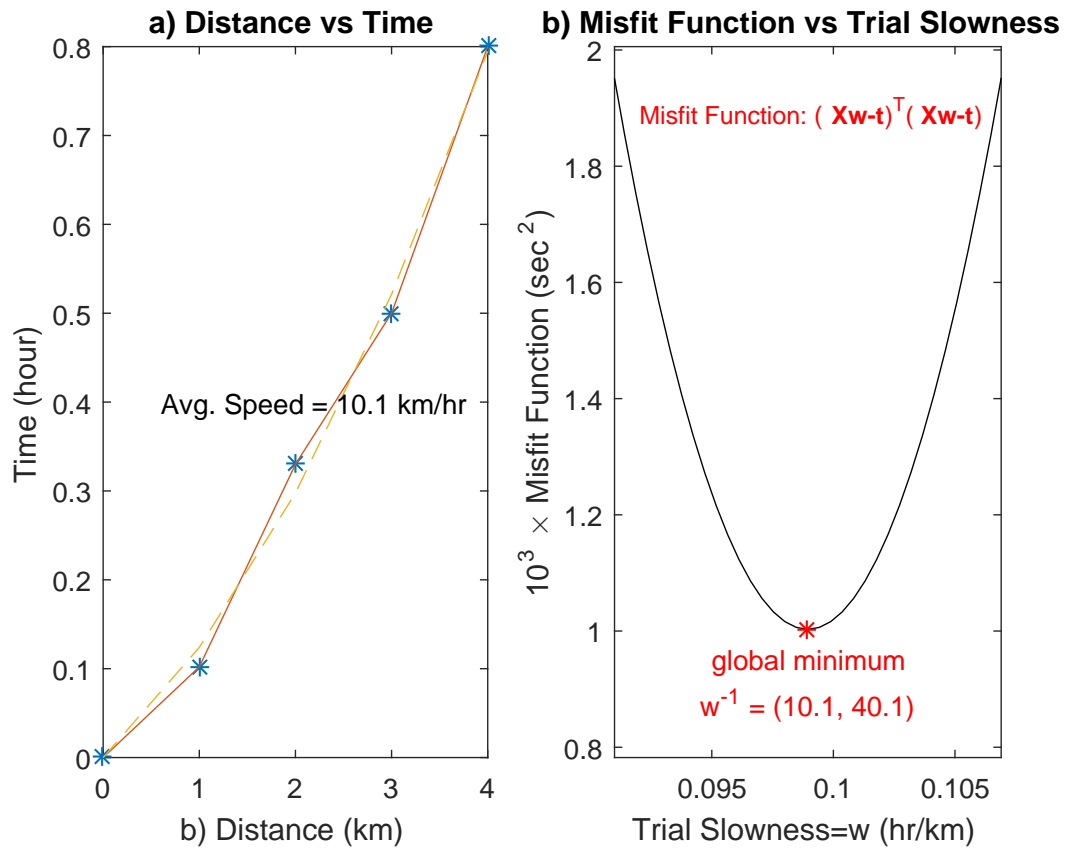


Figure 2.7: Same as Figure 2.2 except the quadratic equation  $t = w_1x + w_2x^2$  is used to fit (dashed line) the motorcycle data. In this case,  $\mathbf{w}^* = (0.099, 0.025) = (1/10.1, 1/40.1)$  and  $\epsilon$  is only plotted along the slowness component  $w_1$ , with  $w_2 = 0.025$ .

Instead of a polynomial, one might suggest a high  $N$ -dimensional linear model:

$$t_i = \sum_{j=1}^N x_{ij} w_j, \quad (2.36)$$

where  $\mathbf{w}$  is the  $N \times 1$  model vector and  $x_{ij}$  are unphysical weights specified by the mechanic. If these weights are specified to give rise to an  $\mathbf{X}^T \mathbf{X}$  matrix that is invertible then this model will also overfit the data. For example, the two equations  $w_1 = 1$  and  $w_1 = 2$  are inconsistent and generate non-intersecting lines in the coordinate-space  $(w_1, w_2)$ . Increasing the number of unknowns to include  $w_2$  so that the model equations are  $w_1 = 1$  and  $w_1 + .4w_2 = 2$  gives a consistent set of equations where the associated lines have a common intersection in  $(w_1, w_2)$ .

There are several choices for increasing the complexity of the model: increase the complexity of the input data by using higher-order polynomials as in equation 2.35, or by creating a non-linear relationship between the model parameters  $\mathbf{w}$  and the specified output. A simple example is to specify the model  $y = \sigma(wx)$ , where  $\sigma(z) = 1/(1 + e^{-wz})$  is the sigmoid function plotted in Figure 4.2. In this case, both the input parameters  $x$  and model parameters have a non-linear relationship with the output  $y$ . This is exactly the type of relationship all neural networks have between the input data  $\mathbf{x}$  and output data  $\mathbf{y}$ .

### Sanity Test for Overfitting

How can we determine if the complexity parameter  $P$  of the proposed model is higher than that in the noiseless data? The machine learning community attempts to answer this question by randomly dividing the entire training data set into several parts<sup>8</sup> (see the horizontal bar at the top of Figure 2.8): *training data* (60%), *validation data* (20%) and *test data* (20%). The goal is to find the optimal parameter values of  $\mathbf{w}^P$  for different orders  $P$  of polynomials in equation 2.34; for neural networks this might correspond to different numbers of nodes in the neural network.

1. The optimal models  $\mathbf{w}^P$  are computed from the training data (gold bar) for different values of  $P$ .
2. The validation dataset is a sample of data held back from training your model that is used to give an estimate of model skill while tuning the model's hyperparameters (Brownlee, 2017). The validation data (green bar) are then used to compute the misfit errors plotted in Figure 2.8. The model that is neither overfitted or underfitted is indicated by the black dot for  $P = 6$ . For the motorcycle example, the training set might be obtained by repeating the time trials many times to get a large set of training data. Higher-order polynomials  $P > 6$  have a larger value of the misfit function and indicate that these higher-order models were fitted to the noise in the training data; this resulted in large validation residuals because the noise in the validation data is quite different. These large residuals could also indicate that the types of training data were not diverse enough to represent the patterns in the validation data. Sometimes

---

<sup>8</sup>See Andrew Ng's recommendation at <https://www.youtube.com/watch?v=MyBSkmUeIEs>

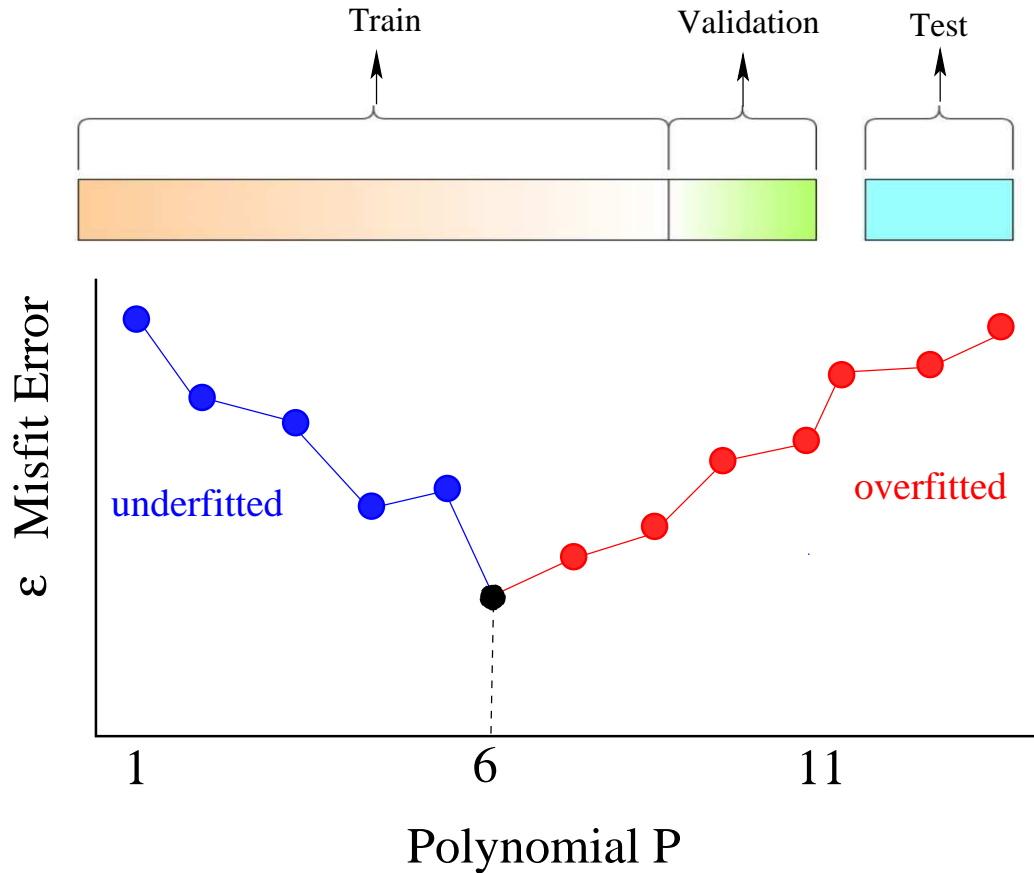


Figure 2.8: Misfit error  $\epsilon$  plotted against the order  $P$  of the polynomial. The misfit values  $\epsilon$  are for the validation (green bar) data using the optimal  $\mathbf{w}$  parameters obtained from the training (gold bar) dataset.

there are local minima in the Figure 2.8 curve, and Prechelt and Orr (2012) describe different criteria for stopping the iterations.

3. Finally, the test data (blue bar in Figure 2.8) is used to provide an unbiased evaluation of the final model  $P = 6$  selected from Figure 2.8 (Brownlee, 2017). The test data for the  $P = 6$  model can be evaluated for a misfit error and if it is close to that for the validation data and model  $P = 6$  then this increases the confidence that this model generalizes well to other holdout data. The literature often interchanges term validation dataset with test data, which introduces confusion in the meaning of these terms (Brownlee, 2017).

If the test dataset has never been used in training, the test dataset is also called a holdout dataset. A strategy for dividing up the data into different sets is also used for cross-validation studies<sup>9</sup> (Golub and von Matt, 1997) discussed in section 4.1.5.

<sup>9</sup>See [https://en.wikipedia.org/wiki/Training\\_test\\_and\\_validation\\_sets](https://en.wikipedia.org/wiki/Training_test_and_validation_sets) for further details.

### 2.1.7 Inclusion of Bias Factor

Assume that the timing clock was not properly calibrated before the motorcycle trial so that the registered starting time consisted of an error of  $\Delta\tau = 0.1$  hour for the  $x = 0$  position. This time-shift bias will be inherited by the times recorded at the other 4 stations. Unless this bias is corrected, the estimated velocity errors will be large for the linear model  $t = wx$ .

To account for this timing *bias* the linear model in equation 2.1 should be changed to

$$t = w_0 + w_1x, \quad (2.37)$$

where  $w_0$  is the bias term that can model a line that does not intercept the origin. In this case the system of equations in equation 2.1 becomes

$$\overbrace{\begin{bmatrix} 1 & x_{11} \\ 1 & x_{21} \\ 1 & x_{31} \\ 1 & x_{41} \\ 1 & x_{51} \end{bmatrix}}^{\mathbf{X}} \overbrace{\begin{pmatrix} w_0 \\ w_1 \end{pmatrix}}^{\mathbf{w}} = \begin{pmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \end{pmatrix}, \quad (2.38)$$

where the least squares solution  $(w_0^*, w_1^*)$  can now be computed to accurately predict the data.

The importance of accurately estimating the bias term  $w_0$  is similar to estimating an accurate starting model for seismic inversion. That is, cycle-skipping problems can be mitigated if an accurate low-wavenumber model is used to account for the bulk of an event's traveltime. As we will discuss in sections 4.1 and 4.5, neural network models include a bias term to shift the input to their activation functions in order to best fit the input points.

## 2.2 Steepest Descent Optimization

Defining  $\mathbf{w}^* \rightarrow \mathbf{w}^{(k+1)}$ ,  $\mathbf{w}' \rightarrow \mathbf{w}^{(k)}$ ,  $\mathbf{t}' = \mathbf{t}^{(k)}$ , and  $\Delta\boldsymbol{\tau}^{(k)} = \mathbf{t}^{(k)} - \mathbf{t}$  in equation 2.27 gives the Gauss-Newton iteration formula (Nocedal and Wright, 1999)

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - [\mathbf{X}^T \mathbf{X}]^{-1} \mathbf{X}^T \Delta\boldsymbol{\tau}^{(k)}, \quad (2.39)$$

where  $\Delta\boldsymbol{\tau}^{(k)} = \mathbf{t}^{(k)} - \mathbf{t}$  is the data-residual vector at the  $k^{th}$  iteration. For a linear system of equations with a well-conditioned  $[\mathbf{X}^T \mathbf{X}]$ , the global minimum should be reached in one iteration. However, the inverse to  $[\mathbf{X}^T \mathbf{X}]$  is too expensive to compute for large data sets so its inverse is approximated by the diagonal matrix  $[\mathbf{X}^T \mathbf{X}]_{ij}^{-1} \approx \frac{1}{[\mathbf{X}^T \mathbf{X}]_{ii}^2} \delta_{ij}$  to give the preconditioned steepest descent formula:

$$w_i^{(k+1)} = w_i^{(k)} - \alpha \frac{1}{[\mathbf{X}^T \mathbf{X}]_{ii}^2} (\mathbf{X}^T \Delta\boldsymbol{\tau}^{(k)})_i, \quad (2.40)$$

where  $\alpha$  is a step length that can be numerically determined, for example, by a numerical line search (Gill, 1981) that  $\mathbf{w}^{(k+1)}$  reduces the misfit error as much as possible<sup>10</sup>. See section 3.2. Unlike the Gauss-Newton formula for linear systems, equation 2.40 does not typically give the right answer for one *iteration*. Therefore this formula is used repeatedly to give updated solutions until convergence. The sequence of steps is as follows.

1. Specify  $k = 0$  and a starting *guessed* solution  $\mathbf{w}^{(0)}$ . The starting solution is used to give the predicted traveltimes  $\mathbf{X}\mathbf{w}^{(0)} = \mathbf{t}^{(0)}$  and the residual  $\Delta\boldsymbol{\tau}^{(0)} = \mathbf{t}^{(0)} - \mathbf{t}$ .
2. Equation 2.40 is used to estimate  $\mathbf{w}^{(k+1)}$  after a suitable step length is computed. We will discuss adaptive step lengths in the next chapter, but for now conveniently set  $\alpha = .01$  or set it equal to  $\alpha = \alpha_0 / \sqrt{(\sum_{\kappa=1}^k (g_i^{(\kappa)})^2 + \eta)}$ , where  $g_i^{(\kappa)}$  is the magnitude of the  $i^{th}$  component of the misfit gradient at the  $\kappa^{th}$  iteration,  $\eta > 0$  is a small damping parameter for stability and  $\alpha_0$  is the starting step length. This is known as the Adagrad step length that decreases with iteration number, and where smaller derivatives lead to larger step sizes. The problem is that this step size becomes too small for a large number of iterations so progress is stalled. Modifications can be made such as the Adadelta (Ruder, 2017) which restricts the summation in the denominator to a small range of previous iterations. See section 3.1.
3. Compute the new predicted traveltime vector  $\mathbf{X}\mathbf{w}^{(k+1)} = \mathbf{t}^{(k+1)}$  and the residual  $\Delta\boldsymbol{\tau}^{(k+1)} = \mathbf{t}^{(k+1)} - \mathbf{t}$ . If the length  $\|\mathbf{r}^{(k+1)}\|$  of the residual vector falls below some specified limit, stop. Otherwise redefine  $k := k + 1$  and repeat steps 2-3 until convergence.

If regularization and preconditioning are used then the preconditioned-regularized steepest descent formula is (see exercise 2.5.10):

$$w_i^{(k+1)} = w_i^{(k)} - \alpha \frac{1}{[\mathbf{X}^T \mathbf{X}]_{ii}^2} (\mathbf{X}^T \Delta\boldsymbol{\tau}^{(k)} + \eta \mathbf{w}^{(k)})_i, \quad (2.41)$$

where  $\eta$  is the regularization parameter. The next chapter will examine the properties of the steepest descent method applied to a non-linear system of equations, which is the optimization technique for the neural network method.

Figure 2.9 presents the results of using both the steepest descent and preconditioned steepest descent methods in solving an overdetermined system of equations. In this case the system of equations is given by

$$\begin{bmatrix} 4 & 6 \\ 2 & 5 \\ 0 & 3 \\ 1 & 4 \end{bmatrix} \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} 10 \\ 7 \\ 3 \\ 5 \end{pmatrix}, \quad (2.42)$$

and it is obvious that the preconditioned steepest descent is more than 5 times faster than the steepest descent method. A fragment of a MATLAB steepest descent code is below.

---

<sup>10</sup>For a linear system of equations, there is an analytic formula for the step length, but it is typically not used for the non-linear problems addressed by the neural network method.

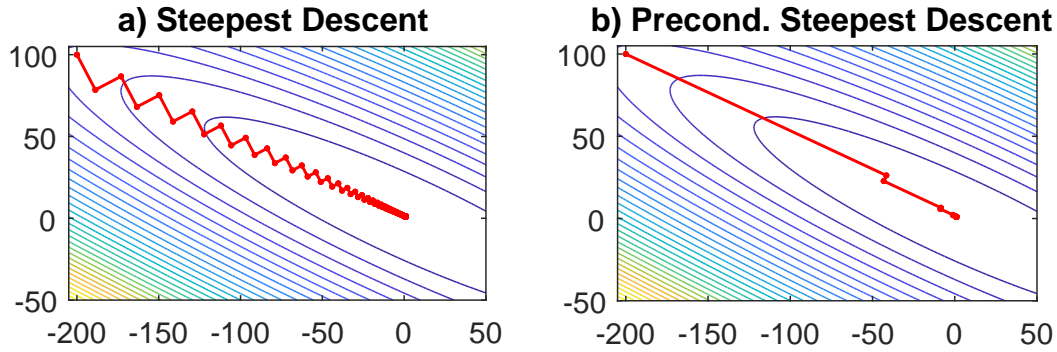


Figure 2.9: Iterative solutions to equation 2.42 by the a) steepest descent and b) preconditioned steepest descent methods.

**Code 2.2.1.** *MATLAB Steepest Descent*

```
H=X'*X;
for i=1:100
g=X'*(X*w-t); % gradient
alpha=g'*g/(g'*H*g); % step length
w=w-alpha*g; % model update
r=X*w-t; % residual
if r'*r <=.0001;i=100;end
end
```

where  $\mathbf{H} = \mathbf{X}^T \mathbf{X}$  is the Hessian matrix and  $\alpha = \mathbf{g}^T \mathbf{g} / (\mathbf{g}^T \mathbf{H} \mathbf{g})$  is the exact step length for an objective function that is exactly quadratic in the model parameters. This step-length formula is derived in Appendix 2.6 but is typically not used if the modeling parameters are not linearly related to the data, which is almost always the case for neural networks or seismic inversion. In addition, computing the Hessian matrix is often too expensive for practical neural network problems because the number of parameters is too large.

## 2.3 Non-Linear Optimization

An example of a linear relation between input data  $\mathbf{X}$ , output data  $\mathbf{t}$  and model  $\mathbf{w}$  is given in equations 2.37-2.38. In general a linear operator<sup>11</sup> denoted by  $f(\mathbf{w})$  must satisfy the superposition  $f(\mathbf{w}_1 + \mathbf{w}_2) = f(\mathbf{w}_1) + f(\mathbf{w}_2)$  and scaling  $f(\alpha \mathbf{w}) = \alpha f(\mathbf{w})$  properties, where  $\alpha$  is a constant. These two properties are both satisfied for any linear system of equations, but the superposition property is not satisfied for non-linear equations such as  $f(w) = w_1^4 - 10w_2 + 4x$ . For this quartic polynomial there can be two minima, where an iterative gradient method can miss the global minimum and get stuck in the local minimum.

<sup>11</sup>An operator is a set of rules that are applied to an input to give an output. For example, the rules for equation 2.37 operator say that the input value of  $x$  should be multiplied by  $w_1$  and  $w_o$  is added to the result to give  $t$ .



To find the optimal  $\mathbf{w}^*$  that minimizes  $f(\mathbf{w})$  we can use the non-linear steepest descent formula, which is the same as equation 2.39 except  $\mathbf{X}^T$  and  $[\mathbf{X}^T \mathbf{X}]^{-1}$  will have a superscript  $(k)$ . This superscript is needed because  $\mathbf{X}$  will be a function of the unknown model parameters in  $\mathbf{w}$ , which will change after each iteration. Let us now present a few examples that solve non-linear equations by an iterative steepest descent solution.

### 2.3.1 MATLAB Examples of Newton's Method

A Newton method is now used to solve for the points  $\mathbf{w}^*$  that minimize 1D and 2D non-linear functions. The MATLAB codes are provided so the reader can explore finding optimal points for different non-linear functions.

#### Example 2.3.1. 1D function

The smooth function  $f(w)$  can be expanded about  $w_o$  in a Taylor series to give

$$\begin{aligned} f(w) &= f(w_o) + \frac{\partial f(w)}{\partial w} \Big|_{w=w_o} \Delta w + 1/2 \frac{\partial^2 f(w)}{\partial w^2} \Big|_{w=w_o} \Delta w^2 + O(\Delta w^3), \\ &\approx f(w_o) + \frac{\partial f(w_o)}{\partial w} + 1/2 \frac{\partial^2 f(w_o)}{\partial w^2}, \end{aligned} \quad (2.43)$$

where  $\Delta w$  is small,  $\frac{\partial f(w_o)}{\partial w} := \frac{\partial f(w)}{\partial w} \Big|_{w=w_o}$  and  $\Delta w = w - w_o$ .

Taking the derivative of equation 2.43 with respect to  $w$ , evaluating this derivative at the stationary point  $w$  where  $\partial f(w)/\partial w = 0$  and rearranging gives the finite-difference approximation of the second derivative<sup>12</sup>

$$\frac{\partial f(w_o)}{\partial w} = -\frac{\partial^2 f(w_o)}{\partial w^2} \Delta w. \quad (2.44)$$

This equation can be solved for  $\Delta w$  to give

$$\Delta w = w - w_o = -\frac{\partial f(w_o)}{\partial w} / \frac{\partial^2 f(w_o)}{\partial w^2}. \quad (2.45)$$

This is only an approximation, so successive substitution leads to the Newton iteration formula

$$w^{(k+1)} = w^{(k)} - \frac{\partial f(w^{(k)})}{\partial w} / \frac{\partial^2 f(w^{(k)})}{\partial w^2}. \quad (2.46)$$

In general, the convergence rate of Newton's method depends on the variable character of  $f(x)$ , as determined by the values of the curvature  $\frac{\partial^2 f(w^{(k)})}{\partial w^2}$  and slope  $\frac{\partial f(w^{(k)})}{\partial w}$  terms. In addition, the speed of convergence can also be determined by how far the starting point is from the global minimum.

The MATLAB script which implements Newton's method for the 1D non-quadratic function  $f(w) = aw^4 + w^2 - 2w$  is given below,

---

<sup>12</sup>The second derivative  $f''(w)$  can be approximated by the difference between two first-order derivatives  $f''(w_o) \approx [f(w_o + dw)' - f(w_o)'] / dw$  for small  $dw$ .

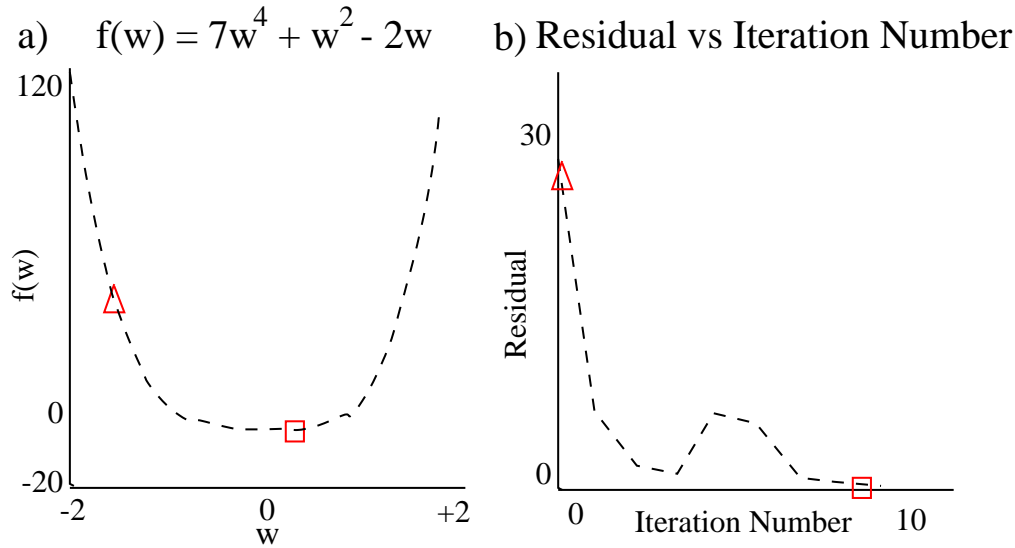


Figure 2.10: Plot of a) quartic function and b) residual vs iteration curve. The diamond (square) represents the starting (ending) model.

### Code 2.3.1. MATLAB 1D Newton Method

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 1D Newton method to find zeros of a quartic function
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all;
a = 7;
subplot(121); w = [-2:1:2];
plot(w, a*w.^4 + w.^2 - 2*w);
%
w = -1.5; % starting point
f(1) = (a*w.^4 + w.^2 - 2*w); % Quartic Function
ww(1) = w;
for it = 2:10 % Start iterations
    f1prime = a*4*w.^3 + 2*w - 2;
    f2prime = a*12*w.^2 + 2;
    w = w - f1prime/f2prime; % Newton formula
    ww(it) = w;
    f(it) = (a*w.^4 + w.^2 - 2*w);
    residual(it-1) = abs(f(it) - f(it-1));
end
```

where the values of  $f(w^{(k)})$  are plotted against  $w^{(k)}$  in Figure 2.10. Unlike the Hessian for the quadratic objective function, the second-order derivative, i.e. curvature, of the quartic function depends on the value of the evaluation point  $w$ .

### Example 2.3.2. 2D Function

The Rosenbrock function is plotted at the top of Figure 2.11 and the iterates associated with Newton optimization are shown beneath it. The MATLAB code for finding its minimum is described in MATLAB code 2.3.2. Just like the quartic function, the curvature value of  $\epsilon$  depends on the location of the evaluation point  $\mathbf{w} = (w_1, w_2)$ .

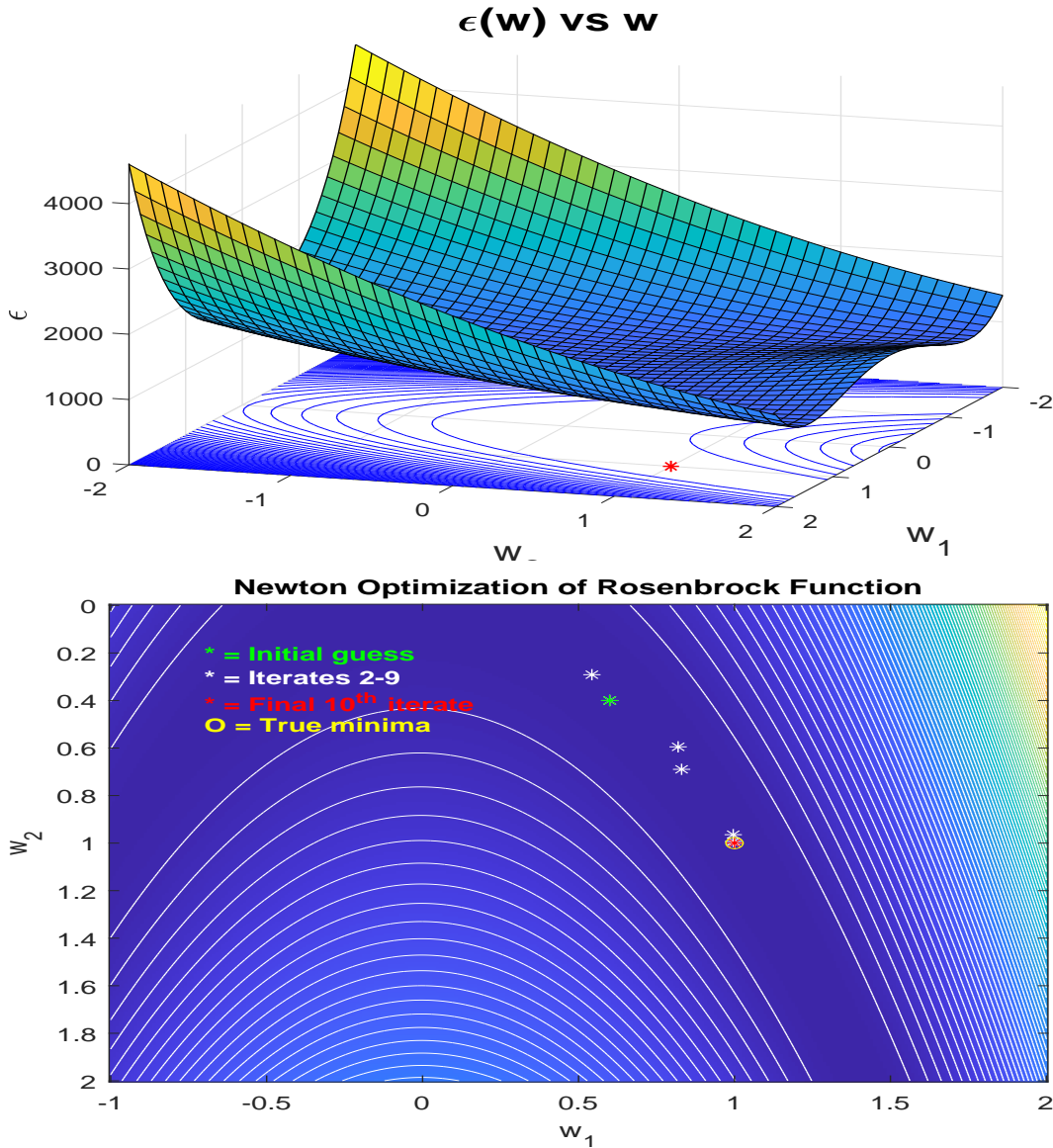


Figure 2.11: (Top) Contours of the Rosenbrock function, where the global minimum at  $(w_1, w_2) = (1, 1)$  is indicated by the red star. Lower plot depicts Newton iterates converging to the true minimum within 10 iterations; results computed by the MATLAB code 2.3.2.

**Code 2.3.2.** *MATLAB Newton Method for Minimizing Rosenbrock Function*

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Iterative Newton method for finding minimizer of Rosenbrock function
% f=100.*(w2-w1.^2).^2+(1-w1).^2. The minimizer point is (1,1)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
alpha=.5;
for itt=1:2

w1 =.6;
w2 =.4;
w = [w1;w2]; % Define starting point (w1,w2)
%
iter = 10;
ww = zeros(iter,2);
ww(1,1) = w1;
ww(1,2) = w2;
%
for it = 2:iter % Begin non-linear iterations
g=[-400*w1*(w2-w1^2)-2*(1-w1); 200*(w2-w1^2)]; % 2x1 gradient vector
H=[1200*w1^2-400*w2+2 -400*w1; -400*w1 200]; % 2x2 Hessian matrix
if itt==1;H(1,2)=0;H(2,1)=0;end
w = w-inv(H)*g; % Non-linear Newton formula
ww(it,:) = w; % Update iterative solution
w1 = w(1);x1=w1;
w2 = w(2);x2=w2;
end
plotrosen(ww,iter,x1,x2)
end

```

The plotting function for the Rosenbrock minimization code is below.

**Code 2.3.3.** *MATLAB Plot Subroutine for Minimizing Rosenbrock Function*

```

function plotrosen(ww,iter,x1,x2)
xx=ww;
xmx = 2; % Define plotting parameters
xmin = -1;
ymin0 = 0;
ymx = 2;
x = [xmin:.01:xmx];
y = [ymin0:.01:ymx];
[X,Y] = meshgrid(xmin:.01:xmx,ymin0:.01:ymx );
f = 100.*(Y-X.^2).^2+(1-X).^2; % Rosenbrock Function
imagesc(x,y,f);
hold on;
contour(X,Y,f,80,'w'); % Plot Rosenbrock function ith contours
hold off
hold on;
for i = 1:iter;
plot(xx(i,1),xx(i,2),'w*');
end
plot(x1,x2,'*r');
plot(ww(1,1),ww(1,2),'*g');
hold off;
xlabel('w_1');
ylabel('w_2');
title('Newton Optimization of Rosenbrock Function')

```

## 2.4 Summary

Systems of equations  $\mathbf{X}\mathbf{w} = \mathbf{t}$  can be ill-conditioned, inconsistent, and overdetermined. In these cases the regularized least squares solution  $\mathbf{w}^*$  minimizes a weighted combination of data misfit  $\|\mathbf{X}\mathbf{w} - \mathbf{t}\|^2$  and penalty  $\eta\|\mathbf{w}\|^2$  terms, where  $\eta$  is the regularization parameter. A *bias column vector* is often appended to  $\mathbf{X}$  to account for bulk shifts in the input data  $\mathbf{t}$ . The  $\mathbf{w}^*$  solves an  $N \times N$  regularized normal system of equations in equation 2.18, which is now a *consistent set of equations*. The solution avoids unstable models far from the origin, but at the loss of some precision for noiseless data. Another way to reduce the condition number is by applying the precondition matrix  $\mathbf{C}$  to  $\mathbf{X}^T\mathbf{X}$  such that  $\mathbf{C}[\mathbf{X}^T\mathbf{X}\mathbf{C}^T]$  is more well conditioned. For practical reasons, the scaling preconditioner matrix is used which is a diagonal matrix with the  $i^{th}$  diagonal element equal to the reciprocal of  $[\mathbf{X}^T\mathbf{X}]_{ii}$ .

Overfitting can be an issue when the complexity of the hypothetical model becomes greater than that of the theoretical model. This can lead to the problem of the theoretical model explaining both the signal and the noise. An ad hoc remedy is to divide the training data into several data sets, and once the optimal parameters of the model are found on the training data, determine their effectiveness of the validation data.

The penalty function can be the misfit between an a priori model  $\mathbf{w}_0$  and the estimated one, in which case it takes the form  $\|\mathbf{w} - \mathbf{w}_0\|^2$ . This is known as a model misfit function. Other penalty functions can be used such as a smoothness constraint where the penalty term is the magnitude of the gradient  $\|\nabla\epsilon\|$  w/r to the model parameters (see Chapter 14). The tradeoff between honoring the data misfit and penalty function is determined by the value of the regularization parameter  $\eta$ . The penalty function can be defined to enforce certain characteristics of the model, such as the final model should smoothly vary in the spatial coordinates. In this case the penalty function can be, e.g.,  $\sum_i (\frac{\partial^n \mathbf{w}}{\partial x_i^n})^2$ , where  $n$  is the specified order of the spatial derivative (Schuster, 2017).

We learned the meaning of the following terms used in the machine learning (ML) community.

- Ill-conditioned, inconsistent, and overdetermined system of equations.
- Objective function, misfit function, step length and penalty function. The objective function is often denoted as the *loss* function in the ML community. For the batch gradient descent, gradients of all the training examples can be computed simultaneously, and then added together to get the total gradient. This is impractical because of physical memory limitations and large data sets. Instead, stochastic gradient descent is used with small mini-batches of data. Each mini-batch ranges from 64 to 256 examples.
- Training data, validation data, and testing data. The training data are divided into several data sets and used to decide which is the best model  $\mathbf{w}$  and regularization parameters should be used without overfitting the data. Details for implementing some anti-overfitting strategies will be described in the cross-validation section 4.1.5 and Chapter 14.
- Normal equations, least squares solution, preconditioning and regularized least squares solutions. Regularized least squares is sometimes denoted as damped least squares.

- The bias term is used to account for bulk shifts in the data and is almost always used for neural network methods.
- Iterative optimization methods: Gauss-Newton method for a linear system of equations, the steepest descent method, and the regularized and preconditioned steepest descent method. The regularized and preconditioned gradient descent methods are the most popular and practical optimization methods.

For most geoscience problems, the number  $N$  of unknowns is too large for the computation of a direct inverse matrix  $[\mathbf{X}^T \mathbf{X} + \eta \mathbf{I}]^{-1}$ , which will cost  $O(N^3)$  algebraic operations. The alternative is an iterative solution method that costs  $O(KN^2)$ , where the number of iterations is  $K \ll N$ , we hope!

## 2.5 Exercises

1. Why don't we allow the regularization parameter  $\eta$  in equations 2.9 or 2.10 to be less than zero?
2. For noiseless data where  $\delta t_i = 0$ , derive the formula for the error  $\delta w$  introduced by the damping parameter  $\eta = .01x_{i1}$  in equation 2.10. Hint: Recall the approximation  $\frac{1}{x^2 + \eta} \approx \frac{1}{x^2} (1 - \frac{\eta}{x^2})$  for  $\eta \ll x^2$ .
3. Define  $\mathbf{w} = \mathbf{w}_o + \alpha \hat{\mathbf{s}}$  where  $\alpha$  is a scalar and  $\hat{\mathbf{s}}$  is a specified unit vector. Show that  $\frac{\partial f(\mathbf{w})}{\partial \alpha} = \hat{\mathbf{s}}^T \nabla f(\mathbf{w})$ , which is known as a directional derivative. Explain why  $\frac{\partial f(\mathbf{w})}{\partial \alpha}$  gives the slope of  $f(\mathbf{w})$  in the direction  $\hat{\mathbf{s}}$ .
4. Assume a function  $f(\mathbf{w})$  that can be approximated by a first-order Taylor series expanded about the point  $\mathbf{w}_o$ :

$$\begin{aligned} f(\mathbf{w}) - f(\mathbf{w}_o) &= \frac{\partial f(\mathbf{w}_o)}{\partial w_1} \delta w_1 + \frac{\partial f(\mathbf{w}_o)}{\partial w_2} \delta w_2, \\ &= \delta \mathbf{w}^T \nabla f(\mathbf{w}_o), \end{aligned} \tag{2.47}$$

where  $\mathbf{w} = (w_1, w_2)^T$ ,  $\delta \mathbf{w}^T = (\delta w_1, \delta w_2)$  and the magnitude  $|\delta \mathbf{w}|$  is very small. If  $\mathbf{w}_o$  is on the contour where  $f(\mathbf{w}_o) = \text{const}$  and  $\mathbf{w} = \mathbf{w}_o + \delta \mathbf{w}$  is on the same contour then  $f(\mathbf{w}) = f(\mathbf{w}_o)$  and  $\delta \mathbf{w}$  is parallel to the tangent of the contour at  $\mathbf{w}_o$ . Therefore, the leftside of the above equation is zero if  $\mathbf{w}$  and  $\mathbf{w}_o$  are on the same contour. Explain why this also says that the gradient  $\nabla f(\mathbf{w}_o) = (\frac{\partial f(\mathbf{w}_o)}{\partial w_1}, \frac{\partial f(\mathbf{w}_o)}{\partial w_2})^T$  is perpendicular to the tangent of this contour.

5. Prove that  $\nabla f(\mathbf{w}_o)$  points toward increasing values of  $f(\mathbf{w})$  by defining  $\delta \mathbf{w}$  in equation 2.47 to be pointing uphill where  $f(\mathbf{w} = \mathbf{w}_o + \delta \mathbf{w}) > f(\mathbf{w}_o)$ . Hint: recall that two vectors point in similar directions if  $(\mathbf{x}, \mathbf{y}) = |\mathbf{x}||\mathbf{y}| \cos \theta > 0$ .
6. Prove that  $\nabla f(\mathbf{w})$  is parallel to the direction of maximum increase in  $f(\mathbf{w})$  at  $\mathbf{w}$ .

7. Define the second-order Taylor expansion of  $f(x)$  about  $\mathbf{w}_o$  as

$$f(\mathbf{w}) = f(\mathbf{w}_o) + \sum_{i=1}^2 \frac{\partial f(\mathbf{w}_o)}{\partial w_i} \delta w_i + \frac{1}{2} \sum_{j=1}^2 \sum_{i=1}^2 \frac{\partial^2 f(\mathbf{w}_o)}{\partial w_i \partial w_j} \delta w_i \delta w_j, \quad (2.48)$$

and define  $\nabla f(\mathbf{w}_o) = (\frac{\partial f(\mathbf{w}_o)}{\partial w_1}, \frac{\partial f(\mathbf{w}_o)}{\partial w_2})^T$ . Show that equation 2.48 can be written as

$$f(\mathbf{w}) = f(\mathbf{w}_o) + \delta \mathbf{w}^T \nabla f(\mathbf{w}_o) + \frac{1}{2} \delta \mathbf{w}^T \mathbf{H} \delta \mathbf{w}, \quad (2.49)$$

where

$$\mathbf{H} = [\nabla \nabla^T] f(\mathbf{w}_o) = \begin{bmatrix} \frac{\partial^2 f}{\partial w_1^2} & \frac{\partial^2 f}{\partial w_1 \partial w_2} \\ \frac{\partial^2 f}{\partial w_1 \partial w_2} & \frac{\partial^2 f}{\partial w_2^2} \end{bmatrix}. \quad (2.50)$$

Show that  $\hat{\mathbf{s}}^T \mathbf{H} \hat{\mathbf{s}}$  gives the curvature of  $f(\mathbf{w})$  along the direction specified by  $\hat{\mathbf{s}}$ . Hint: Define  $\mathbf{w} = \mathbf{w}_o + \alpha \hat{\mathbf{s}}$  so that

$$\frac{d^2 f}{d\alpha^2} = \frac{d}{d\alpha} \frac{df}{d\alpha} = \frac{d}{d\alpha} [(\nabla f)^T \hat{\mathbf{s}}] = \hat{\mathbf{s}}^T [\nabla (\nabla f)^T] \hat{\mathbf{s}} = \hat{\mathbf{s}}^T \overbrace{[\nabla \nabla^T f(\mathbf{w})]}^{\text{Hessian}} \hat{\mathbf{s}}. \quad (2.51)$$

8. Can there ever be more than one isolated minimum for a quadratic function? Prove your answer. Can there ever be more than one isolated minimum for a non-quadratic function?
9. Same question as the previous one except find the maximum and minimum curvatures of the Rosenbrock function instead of the slope. What is the curvature along the slope direction?
10. Starting from the regularized objective function, derive equation 2.41.
11. The sign and magnitude of the eigenvalue  $\lambda_i$  of  $\mathbf{H}$  determine the shape of  $f(\mathbf{w})$  along the  $i^{th}$  coordinate direction. For a 2D geometry this can be shown by setting  $\mathbf{w} = \mathbf{w}^* + \alpha \mathbf{e}_1 + \beta \mathbf{e}_2$ , where  $\mathbf{e}_i$  is the  $i^{th}$  orthonormal eigenvector of the symmetric matrix  $\mathbf{H}$ ,  $\mathbf{w}^*$  is the point where  $f(\mathbf{w})$  is a minimum (i.e.,  $\mathbf{g}(\mathbf{w}^*) = 0$ ), and  $\alpha$  and  $\beta$  are scalars. Expanding  $f(\mathbf{w})$  about the minimum point  $\mathbf{w}^*$  so that equation 2.49 becomes

$$\begin{aligned} f(\mathbf{w}^* + \alpha \mathbf{e}_1 + \beta \mathbf{e}_2) &= f(\mathbf{w}^*) + [\alpha^2 \mathbf{e}_1^T \mathbf{H} \mathbf{e}_1 + \beta^2 \mathbf{e}_2^T \mathbf{H} \mathbf{e}_2]/2, \\ &= f(\mathbf{w}^*) + [\lambda_1 \alpha^2 \mathbf{e}_1^T \mathbf{e}_1 + \lambda_2 \beta^2 \mathbf{e}_2^T \mathbf{e}_2]/2, \\ &= f(\mathbf{x}^*) + [\lambda_1 \alpha^2 + \lambda_2 \beta^2]/2, \end{aligned} \quad (2.52)$$

where the gradient term  $\mathbf{g}^T \Delta \mathbf{w}$  is zero at the minimum point  $\mathbf{w}^*$ . The eigenvalues are positive if  $\mathbf{H}$  is a SPD matrix, so any move along an eigenvector direction from  $\mathbf{w}^*$  will increase the value of the function. Hence,  $f(\mathbf{w})$  describes a bowl-like surface

around the minimum point  $\mathbf{w}^*$  (see left column of plots in Figure 2.12). Large positive eigenvalues suggest that small changes in position lead to large changes in  $f(\mathbf{w})$  so that the bowl has steeply curving sides; conversely, small positive eigenvalues suggest a bowl with gently curving sides.

If the Hessian is negative definite (i.e.,  $\mathbf{H}$  has only negative eigenvalues) then equation 2.52 says that any move from  $\mathbf{w}^*$  along an eigenvector direction will decrease the function, i.e.,  $f(\mathbf{w})$  describes an inverted bowl about the *maximal* point  $\mathbf{w}^*$ . Show that an indefinite Hessian (both positive and negative eigenvalues) has the property that a move along one eigenvector direction will decrease the function value while a move along the other eigenvector direction will increase the function value. This latter surface describes the saddle depicted in 2.12f.

12. Plot the Rosenbrock function with the MATLAB commands:

**Code 2.5.1. MATLAB Rosenbrock Function**

```
x1=-1.25;xr=1.05;dx=.05; y1=-.3;yr=1.2;dy=.05; facx=1;facy=1;
[xx,yy]=meshgrid([x1:dx:xr]*facx,[y1:dy:yr]*facy);
%[xx,yy]=meshgrid([-0.05:0.005:0.45],[-0.06:0.005:0.12]);
mis=100*(yy - xx.^2).^2 + (1-xx).^2; contour(xx,yy,mis,252)
```

Now plot the contours for  $\partial f/\partial w_1$  and  $\partial f/\partial w_2$ . Is  $f(\mathbf{w})$  a quadratic or non-quadratic function? Are the elements of the Hessian matrix, the components of the gradient, or the curvature along the  $w_1$  direction functions of  $(w_1, w_2)$ ? For an objective function that is strictly quadratic, are the elements of the Hessian and gradient a function of spatial position? Same question, except the objective function is a cubic polynomial.

13. Assume  $f(\mathbf{w}) = w_1^2 + w_2^2 - 2w_1w_2$ . Plot the contours of  $f(\mathbf{w})$ . Is  $f(\mathbf{w})$  a quadratic or non-quadratic function? Are the elements of the Hessian matrix, the components of the gradient, or the curvature along the  $w_1$  direction functions of  $(w_1, w_2)$ ? For the above quadratic function, how many minima are there in the objective function? Identify the local and global minimum points for  $f(\mathbf{w})$ .
14. Can there ever be more than one isolated minimum for a quadratic function? Prove your answer. Can there ever be more than one isolated minimum for a non-quadratic function?
15. Find the slope of the Rosenbrock function  $f(\mathbf{w})$  at  $\mathbf{w} = (2, 3)$  along the line direction parallel to the vector  $(1, 1)$ . Find the slope of  $f(\mathbf{w})$  at  $(2, 3)$  that is parallel to the vector orthogonal to the vector  $(1, 1)$ . Show work using the  $2 \times 1$  gradient vector.
16. Same question as the previous one except find the maximum and minimum curvatures of the Rosenbrock function instead of the slope. What is the curvature along the slope direction? Is the greatest curvature always along the gradient direction?
17. Show that a general quadratic function  $f(\mathbf{w}) = (1/2)\mathbf{w}^T \mathbf{H} \mathbf{w} + \mathbf{g}^T \mathbf{w} + c$  where  $\mathbf{H}$  is symmetric, can also be described as  $f(\mathbf{w}) = (1/2)(\mathbf{w} - \mathbf{w}')^T \mathbf{H}(\mathbf{w} - \mathbf{w}') + c'$  where  $\mathbf{H} \mathbf{w}' = -\mathbf{g}$  and  $c' = c - (1/2)\mathbf{w}'^T \mathbf{H} \mathbf{w}'$ . What is the geometrical meaning of this transformation?



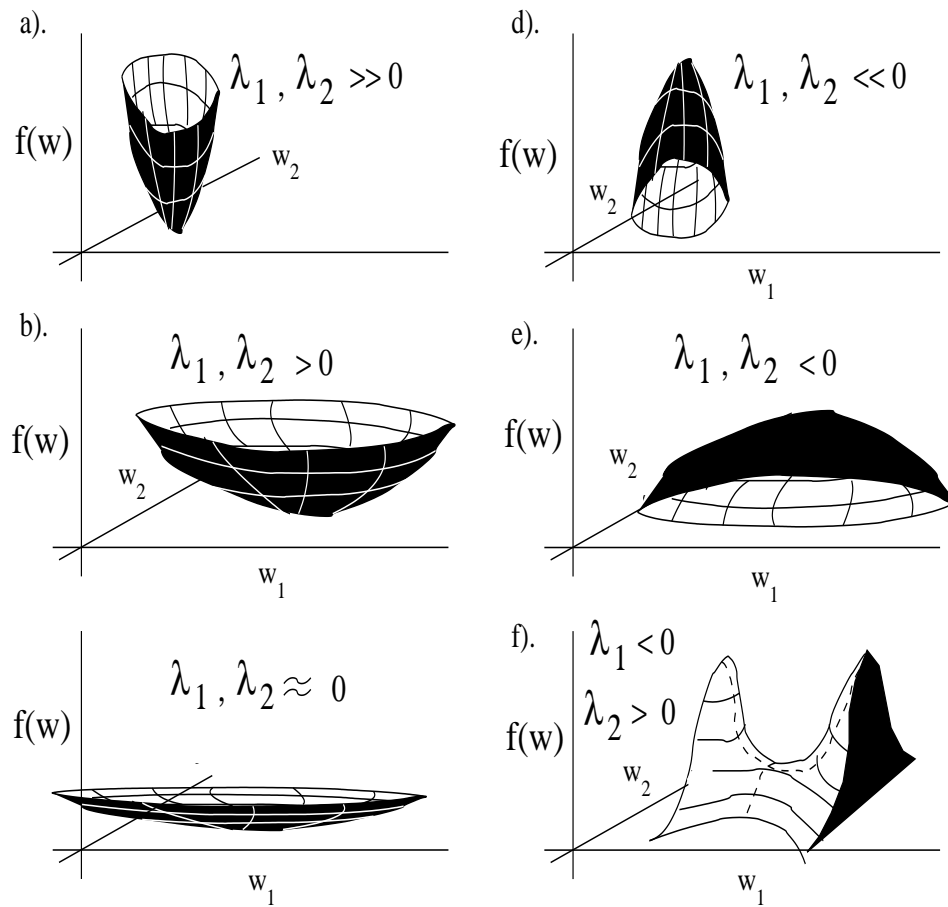


Figure 2.12: Plots of functions with different eigenvalues  $\lambda_1$  and  $\lambda_2$  for the  $2 \times 2$  Hessian matrix. Left column of figures are associated  $\lambda_1, \lambda_2 > 0$  while the right column corresponds to examples where at least one of the eigenvalues is negative.

18. Design the elements of a  $2 \times 2$  Hessian matrix so all of the shapes shown in Figure 2.12 are obtained. Use a MATLAB plotting code similar to

```
[X,Y,Z] = peaks(30);
figure
surfc(X,Y,Z)
```

19. Starting from an objective function with multiple minima, devise a gradient descent method and multiscale strategy to mitigate getting stuck in a local minima. A frequency-domain multiscale strategy applies a low-pass filter to the function, iterates until the residual decrease stalls. This last point defines the starting point for a gradient descent method which uses the function that is filtered by an intermediate-pass filter. This procedure is repeated for higher-pass versions of  $f(w)$  until convergence to the global minimum.

An example of an objective function with local minima is shown below.

**Code 2.5.2.** *MATLAB 3D Plot of Objective Function*

```
[X Y Z]=peaks(100);
s=sqrt(X.^2+Y.^2);
surfc(cos(2*pi*s*2)+s);
```

One-dimensional and 2D examples are shown in Figure 2.13 along with the MATLAB script. To compute the element values of the gradient and Hessian you will have to use finite-difference approximations.

**Code 2.5.3.** *MATLAB 3D Plot of Figure 2.13.*

```
clear all
x = -1:.05:2;
y = -humps(x);n=length(y);
subplot(121)
plot(x,y)
xlabel('x')
ylabel('humps(x)')
grid on;subplot(122)
[X Y Z]=peaks(n);
yy=y'*y+10;
yy=real((yy).^(.2));
surfc(X , Y, -yy)
```

## 2.6 Appendix: Exact Step Length

The derivation of the step length for the exact line search is now given. The 1D line search problem is defined as finding the optimal value of the scalar  $\alpha$  that minimizes  $\epsilon(\mathbf{w} +$

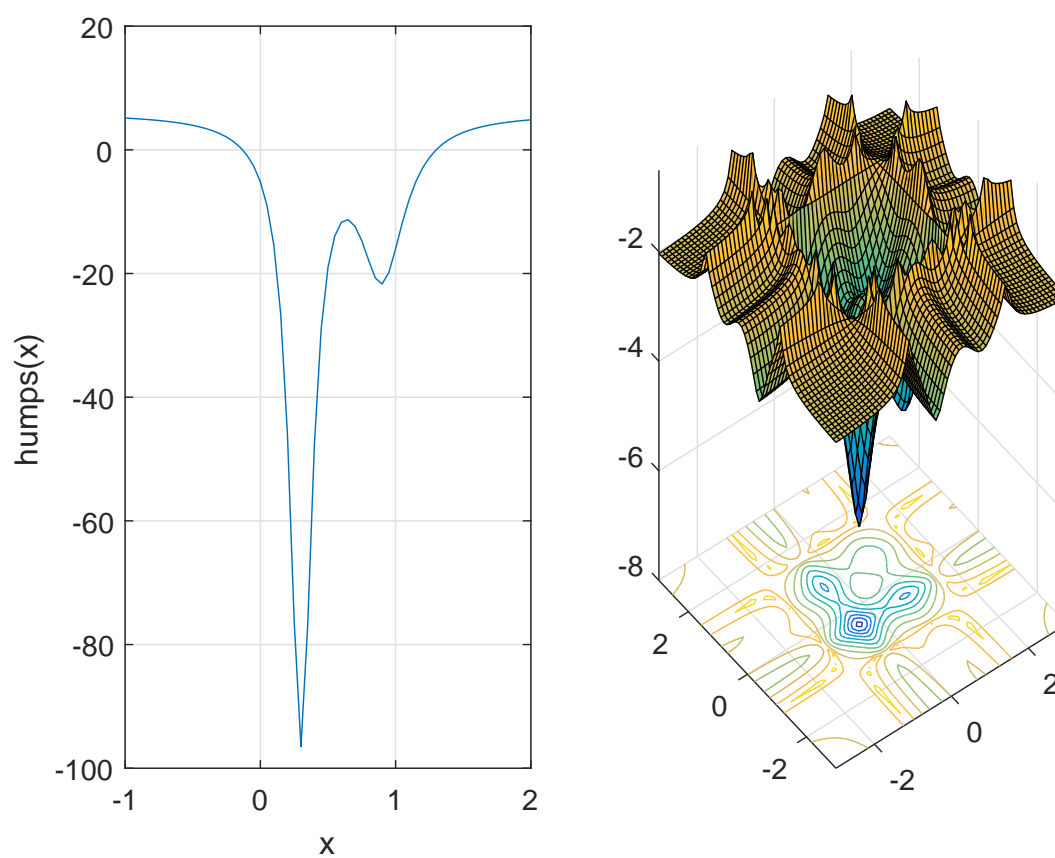


Figure 2.13: Objective function with local minima.

$\alpha\Delta\mathbf{w}$ ) for a fixed  $\mathbf{w}$  and  $\Delta\mathbf{w}$ . Without loss of generality, we can set  $\mathbf{w} = 0 + \alpha\Delta\mathbf{w}$  and  $\eta = 0$  in equation 2.16 to get

$$\begin{aligned}\epsilon(\mathbf{w} + \alpha\Delta\mathbf{w}) &= \frac{1}{2}\alpha^2(\mathbf{w} + \Delta\mathbf{w})^T[\mathbf{X}^T\mathbf{X}](\mathbf{w} + \Delta\mathbf{w}) - \alpha\mathbf{t}^T\mathbf{X}(\mathbf{w} + \Delta\mathbf{w}) + \frac{1}{2}\mathbf{t}^T\mathbf{t}, \\ &= \frac{1}{2}\alpha^2\Delta\mathbf{w}^T[\mathbf{X}^T\mathbf{X}]\Delta\mathbf{w} + \frac{1}{2}\alpha^2\mathbf{w}^T[\mathbf{X}^T\mathbf{X}]\mathbf{w} + \alpha^2\Delta\mathbf{w}^T[\mathbf{X}^T\mathbf{X}]\mathbf{w} \\ &\quad - \mathbf{t}^T\mathbf{X}(\mathbf{w} + \Delta\mathbf{w}) + \frac{1}{2}\mathbf{t}^T\mathbf{t}.\end{aligned}\tag{2.53}$$

Setting  $\eta = 0$  in equation 2.16 and subtracting it from equation 2.53 gives

$$\epsilon(\mathbf{w} + \alpha\Delta\mathbf{w}) - \epsilon(\mathbf{w}) = \frac{1}{2}\alpha^2\Delta\mathbf{w}^T[\mathbf{X}^T\mathbf{X}]\Delta\mathbf{w} + \alpha\Delta\mathbf{w}^T\mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{t}).\tag{2.54}$$

Differentiating the above equation with respect to  $\alpha$  and setting the result equal to zero yields the stationary condition where  $\epsilon(\mathbf{w} + \alpha\Delta\mathbf{w})$  achieves a minimum along the steepest descent direction  $\Delta\mathbf{w}$ :

$$\begin{aligned}\frac{\partial\epsilon(\mathbf{w} + \alpha\Delta\mathbf{w})}{\partial\alpha} &= \alpha\Delta\mathbf{w}^T\mathbf{H}\Delta\mathbf{w} + \Delta\mathbf{w}^T\mathbf{g}, \\ &= 0,\end{aligned}\tag{2.55}$$

where the  $N \times N$  Hessian matrix is defined as  $\mathbf{H} = \mathbf{X}^T\mathbf{X}$  and the gradient is  $\mathbf{g} = \mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{t})$ . Solving for  $\alpha$  gives the exact step length:

$$\alpha = \frac{-\mathbf{g}^T\Delta\mathbf{w}}{\Delta\mathbf{w}^T\mathbf{H}\Delta\mathbf{w}}.\tag{2.56}$$

This step-length calculation is exact if  $\epsilon(\mathbf{w})$  is a quadratic functional. For the steepest descent method  $\Delta\mathbf{w} = -\mathbf{g}$  so that

$$\alpha = \frac{\mathbf{g}^T\mathbf{g}}{\mathbf{g}^T\mathbf{H}\mathbf{g}}.\tag{2.57}$$

For non-linear problems, such as neural networks or full waveform inversion, this exact line search formula does not work well because it is suited only for linear problems where the data and model are linearly related. For non-linear problems a numerical line search method is used to find  $\alpha$ .

## Chapter 3

# Specialized Gradient Descent Methods

The previous chapter introduced iterative gradient-descent methods as an affordable means for estimating the solution to a large system of equations. We now introduce a few more such methods, such as the conjugate gradient method and some ad hoc methods developed by the machine learning community. These methods have one thing in common: they use gradient information from previous iterations (and in some cases a future iteration) to better estimate the next best move, similar to the skier in Figure 3.1.

### 3.1 Ad Hoc Gradient Descent Methods

Objective functions are often characterized by contours that are distorted along certain coordinate directions, such as the ones in Figure 2.9. Ellipses with large aspect ratios are a symptom of an ill-posed problem, which typically leads to slow convergence by a steepest descent method. To partly remedy this difficulty, ad hoc variations of steepest descent have been developed (Ruder, 2017), including the methods of momentum, Adagrad (adaptive gradient), Adadelata, RMSprop, and Adam (adaptive moment estimation). These methods are described below.

#### 3.1.1 Momentum Gradient Descent

For ill-conditioned equations, the  $L^2$  objective function is characterized by elongated contours that resemble long narrow valleys, e.g. Figure 2.9. Consequently, the steepest descent path will oscillate between ascending and descending the steep walls of the valley in Figure 2.9a. In this case the  $k^{th}$  gradient can be decomposed into a sum of two orthogonal components  $\nabla\epsilon^{(k)} = \nabla\epsilon_{\parallel}^{(k)} + \nabla\epsilon_{\perp}^{(k)}$ , where  $\nabla\epsilon_{\perp}^{(k)}$  points up the steep sides of the valley while  $\nabla\epsilon_{\parallel}^{(k)}$  is the direction along the long valley floor. Assuming that the steep gradient components  $\nabla\epsilon_{\perp}^{(k)} \approx -\nabla\epsilon_{\perp}^{(k-1)}$  of two successive moves have nearly the same magnitude



Figure 3.1: Skiers looking ahead for best track that makes significant downhill progress without hitting a local maxima. Photo by Courtney McNiff.

and opposite polarity then

$$\begin{aligned} \overbrace{\nabla \epsilon^{(k)}}^{\text{current gradient}} + \overbrace{\nabla \epsilon^{(k-1)}}^{\text{previous gradient}} &= \cancel{\nabla \epsilon_{\perp}^{(k)}} + \cancel{\nabla \epsilon_{\perp}^{(k-1)}} + \nabla \epsilon_{\parallel}^{(k)} + \nabla \epsilon_{\parallel}^{(k-1)}, \\ &\approx 2\nabla \epsilon_{\parallel}^{(k)}. \end{aligned} \quad (3.1)$$

Adding two successive gradients will result in an update direction  $2\nabla \epsilon_{\parallel}^{(k)}$  that is largely along the valley floor and avoids the unproductive oscillations  $\nabla \epsilon_{\perp}^{(k)}$  perpendicular to the long axis of the valley.

Therefore, the momentum method (Sutton, 1986; Ruder, 2017) computes a new move (aka update) direction that is parallel to a weighted sum of the current-gradient and previous momentum-gradient directions:

$$\overbrace{\mathbf{v}^{(k)}}^{\text{current moment. grad.}} = \beta \overbrace{\mathbf{v}^{(k-1)}}^{\text{prev. moment. grad.}} + \alpha \overbrace{\nabla \epsilon^{(k)}}^{\text{current grad.}}, \quad (3.2)$$

so that the momentum update formula is

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \mathbf{v}^{(k)}. \quad (3.3)$$

Here,  $0 < \beta < 1$  should be set to be around 0.9 (Ruder, 2017) and  $\alpha$  is determined by a trial-and-error procedure. Empirical tests suggest that the convergence rate of the momentum method is typically faster than that of steepest descent when the ellipse-like contours have a large aspect ratio.

The momentum method is now used to find the minimizer to the non-linear Rosenbrock function  $\epsilon = 100(w_2 - w_1^2)^2 + (1 - w_1)^2$ . Figure 3.2c plots the results after 10 iterations of equation 3.3. At the step size of  $\alpha = 0.0045$ , the iterated points (white stars) show little progress in getting to the stationary point (yellow circle) at  $(1, 1)$ . When the step size was increased, the updated iterates wandered far from the stationary point at  $(1, 1)$ . In comparison, the Newton method achieved convergence within 4 iterations in Figure 3.2d. This is superior to the divergent results of the preconditioned SD and standard SD methods plotted in Figure 3.2a- 3.2b. Similar to the momentum method, changing the step sizes for the SD methods made the results even worse. Instead of a fixed step length, the exact step length computed by equation 2.57 was also tested for the standard SD. The results (not shown) showed that the iterative solutions stayed close to the starting point.

Why did the Newton method succeed while the other methods fail? The likely answer is that the Newton method also computed the curvature information at each iterate, which allowed it to move to the global minimum within a few iterations. It also was able to adjust the step size at each iteration. Unlike a quadratic functional, the curvature of a highly non-linear function changes from one point to the next and so it is important to know this information at each step to estimate the best step size. The SD and momentum methods did not have this curvature information and just used a fixed step length that was sub-optimal.

As a remedy to the divergence problem, the step sizes were significantly reduced and the number of iterations was increased. In addition, the step size was finely tuned by trial and

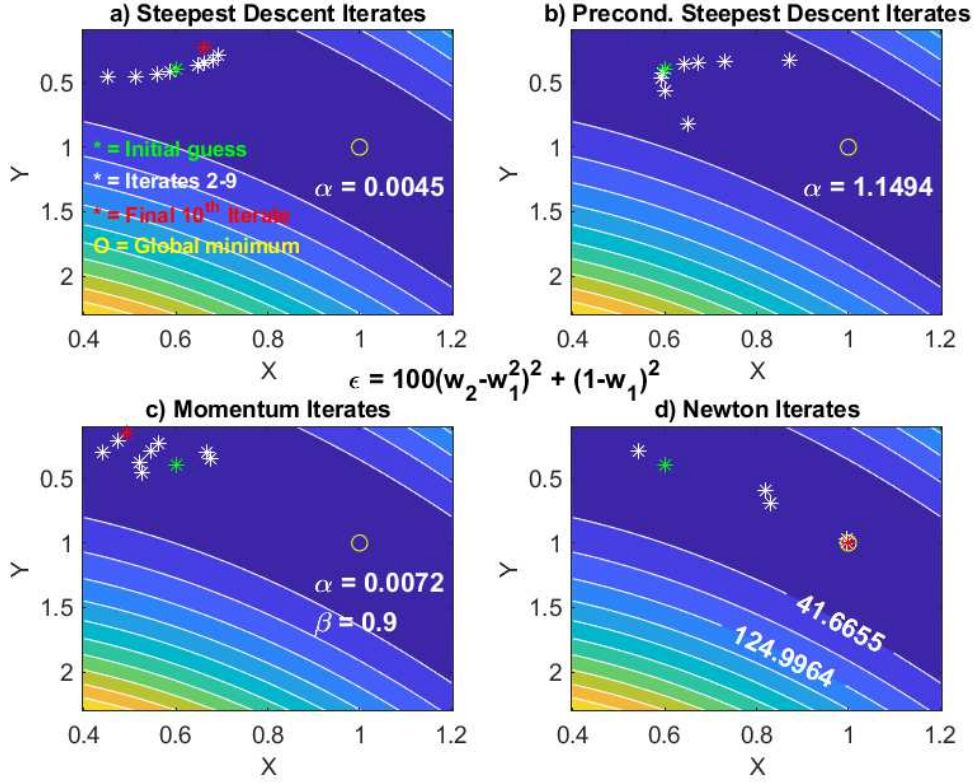


Figure 3.2: Plot of ten iterates for minimizing the Rosenbrock function by the a) steepest descent , b) preconditioned steepest descent, c) momentum and d) Newton methods. Here,  $\alpha$  corresponds to the fixed step size and  $\beta$  is in equation 3.3. Results computed by the MATLAB code 3.1.1.

error to achieve much better convergence towards the minimum, as shown by the results in Figure 3.3. The lesson to be learned here is that a static step size can be inefficient and lead to divergence. Thus fine tuning of the step size should be performed and it should vary from location to location. A good strategy might be to check if the objective function has decreased  $\epsilon(\mathbf{w}^{(k)}) > \epsilon(\mathbf{w}^{(k)} + \alpha \Delta \mathbf{w}^{(k)})$  after the update  $\mathbf{w}^{(k)} + \alpha \Delta \mathbf{w}^{(k)}$ , and if so keep iterating with the same step size until epsilon increases. In this case, halve the step length  $\alpha := \alpha/2$  and check for the validity of  $\epsilon(\mathbf{w}^{(k)}) > \epsilon(\mathbf{w}^{(k)} + 0.5\alpha \Delta \mathbf{w}^{(k)})$ . If not true, then halve again until  $\epsilon$  has decreased. We denote this as backtracking by the bisection method. A bisection backtracking MATLAB code is described in section 4.3.3.

However, even if both the gradient and the curvature is known at each point the Newton method has no guarantee of convergence to a global minimum for highly non-linear objective functions.

The MATLAB codes for the gradient descent methods of Newton, momentum, SD and preconditioned SD are given in Code 3.1.1. The example they use is for finding the global minimum of the Rosenbrock objective function, and the associated plotting code is in Appendix 3.6.



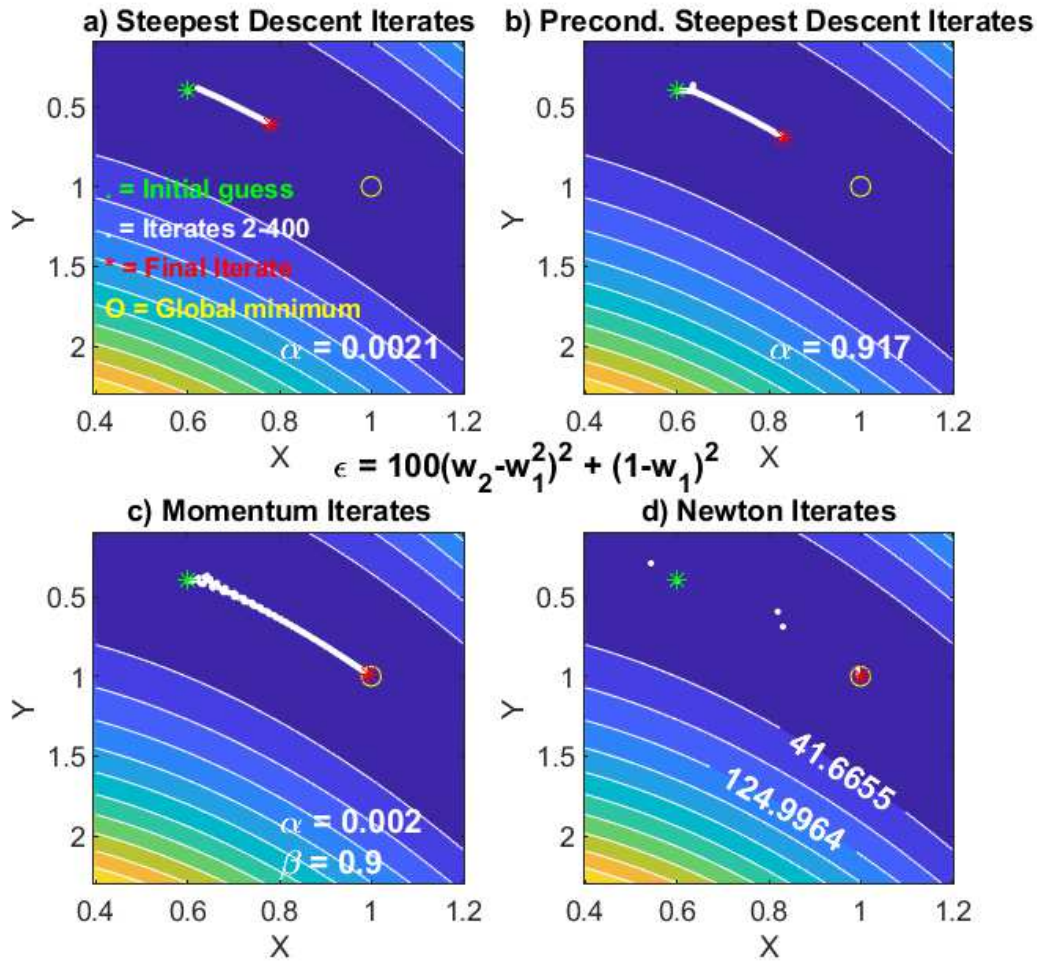


Figure 3.3: Same as Figure 3.2 except there are 400 iterations and the step sizes for a), b) and c) have been reduced and finely tuned to achieve much better progress in finding the minimum at  $(1, 1)$ .

**Code 3.1.1.** *MATLAB Gradient Code for Minimizing Rosenbrock Function in Figure 3.2: SD, Preconditioned SD, momentum and Newton. Plotting subroutine is in Appendix 3.6*

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Iterative gradient methods for finding minimizer of Rosenbrock function
% f=100.*(w2-w1.^2).^2+(1-w1).^2 and Minimizer point=(1,1)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all
nitt=4; iter= 10; g0=0; gg=zeros(2, iter);
www=zeros(iter, 2, nitt);

for itt=1:nitt
    w1 =.6;
    w2 =.4;
    w = [w1;w2]; % Initial point (w1,w2)
    %
    ww = zeros(iter, 2);
    ww(1,1) = w1;
    ww(1,2) = w2;
    %
    for it = 2:iter % Begin iterations
        g=[-400*w1*(w2-w1^2)-2*(1-w1); 200*(w2-w1^2)]; % 2x1 gradient vector
        H=[1200*w1^2-400*w2+2 -400*w1; -400*w1 200]; % 2x2 Hessian matrix
        if itt==1; alpha=.0045; H=eye(2,2); % Steepest Descent
            %alpha=g'*g/(g'*H*g)
            g=alpha*g; gg(:, it)=g; end
        if itt==2; alpha1=.88; H(1,2)=0; % Precond. Steepest Descent
            H(2,1)=0; H=alpha1*H; end;
        if itt==3; alpha2=.007; beta=.9; % Momentum
            g=alpha2*g-beta*g0; H=eye(2,2); g0=-g; end
        w = w-inv(H)*g; % Newton Update
        ww(it,:) = w;
        w1 = w(1); x1=w1;
        w2 = w(2); x2=w2;
    end
    www(:, :, itt)=ww;
end

subplot(221); ww=www(:, :, 1); plotrosen2(ww, iter, x1, x2, 1);
text(.9, 1.3, ['\alpha = ', num2str(alpha)], 'Color', 'white', 'Fontweight', 'bold')
subplot(222); ww=www(:, :, 2); plotrosen2(ww, iter, x1, x2, 2);
text(.9, 1.3, ['\alpha = ', num2str(alpha1)], 'Color', 'white', 'Fontweight', 'bold')
subplot(224); ww=www(:, :, 4); plotrosen2(ww, iter, x1, x2, 4);
subplot(223); ww=www(:, :, 3); plotrosen2(ww, iter, x1, x2, 3);
text(.9, 1.3, ['\alpha = ', num2str(alpha2)], 'Color', 'white', 'Fontweight', 'bold')
text(.9, 1.6, ['\beta = ', num2str(beta)], 'Color', 'white', 'Fontweight', 'bold')

```

Instead of the highly non-linear Rosenbrock function, the gradient methods are used to iteratively find the minimizer to the quadratic function  $\epsilon = w_1^2 + 0.001w_2^2 + w_1w_2$ . The corresponding Hessian matrix is

$$\mathbf{H} = \begin{bmatrix} 2 & 1 \\ 1 & 0.002 \end{bmatrix}, \quad (3.4)$$

where  $\epsilon = \frac{1}{2}\mathbf{w}^T\mathbf{H}\mathbf{w} = w_1^2 + 0.001w_2^2 + w_1w_2$  if  $\mathbf{H}$  is defined as in the above equation.

Figure 3.4 shows that the contours of  $\epsilon$  describe a saddle-like topography, where the saddle point is at  $\mathbf{w} = (0, 0)$ . This is not surprising because the eigenvalues of  $\mathbf{H}$  are  $\lambda_1 = -0.41$  and  $\lambda_2 = 2.41$  (see Figure 2.12f in exercise 2.5.11). This means that  $\mathbf{H}$  is not SPD. Nevertheless, the second-order Newton method finds the stationary point within one iteration because the functional is a quadratic and the condition for the existence of a unique solution is that  $\lambda_i \neq 0 \forall i$ . In contrast, the first-order gradient descent methods hopelessly wander around, each assuming that  $\mathbf{H}$  is diagonally dominant so its inverse can

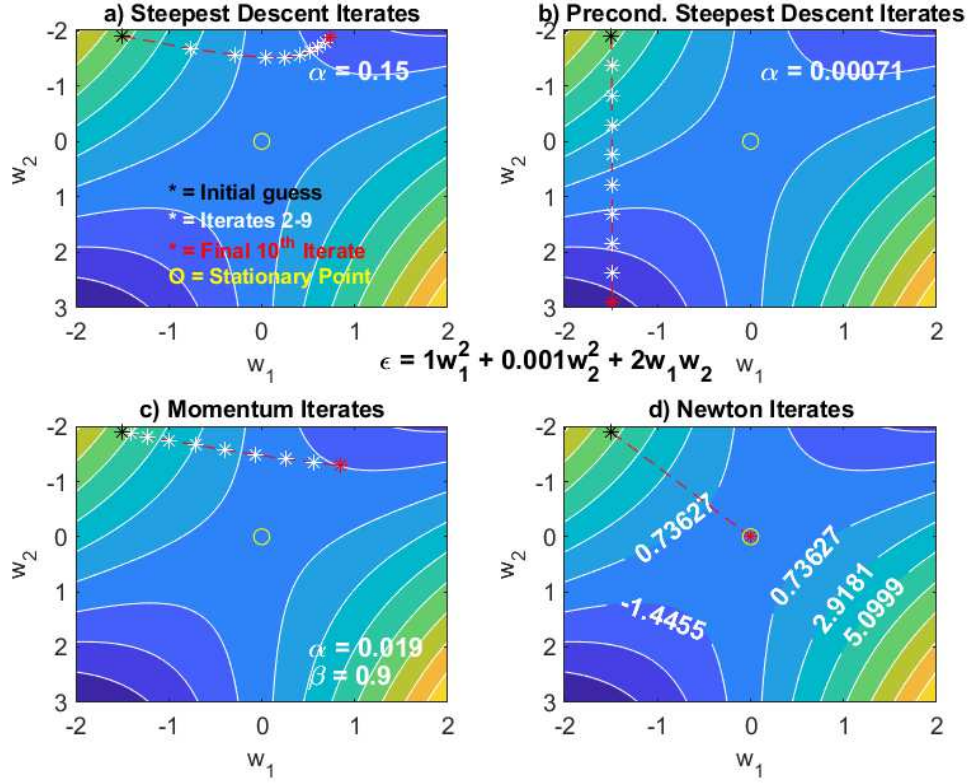


Figure 3.4: Same as Figure 3.2 except  $\epsilon = w_1^2 + 0.001w_2^2 + w_1w_2$ . Here the contours describe a saddle with a stationary point at  $\mathbf{w} = (0,0)$ . All the methods fail to get to the stationary point except the Newton method.

be approximated by a diagonal matrix<sup>1</sup> or a scaled identity matrix. This is wishful machine thinking because neither  $\mathbf{H}$  or its inverse  $\mathbf{H}^{-1}$  are diagonally dominant:

$$\mathbf{H}^{-1} = \begin{bmatrix} -0.002 & 1.004 \\ 1.004 & -2.0080 \end{bmatrix}. \quad (3.5)$$

Such wishful thinking mislead the preconditioned SD down the wrong path, as discussed in exercise 3.5.1.

A friendlier objective function is the SPD quadratic

$$\epsilon = w_1^2 + 0.06w_2^2, \quad (3.6)$$

<sup>1</sup>A square matrix is diagonally dominant if, for every row of the matrix, the magnitude of the diagonal entry in a row is larger than or equal to the sum of the magnitudes of all the other entries in that row.

where the elliptical contours of  $\epsilon$  in Figure 3.5 surround the minimum at  $\mathbf{w} = (0, 0)$ . Here,

$$\mathbf{H} = \begin{bmatrix} 2 & 0 \\ 0 & 0.12 \end{bmatrix}, \quad (3.7)$$

which is diagonally dominant but the aspect ratio of the ellipses is greater than 7 : 1. Here,  $\mathbf{H}$  is a scaled diagonal matrix so the preconditioned steepest descent uses the preconditioner  $[\mathbf{H}^{-1}]_{ij} = \delta_{ij}/H_{ii}$ , which is also the exact Hessian inverse. Therefore preconditioned steepest descent (SD) should perform as well as the Newton method.

Four gradient descent methods are now used to find the minimum of  $\epsilon$  in equation 3.6, and the results are plotted in Figure 3.5. The preconditioned SD method converged within one iteration, which is expected because the preconditioned matrix is the same as the exact inverse  $\mathbf{H}^{-1}$ . The SD result in Figure 3.5a uses the exact step size  $\alpha = \mathbf{g}^T \mathbf{g} / \mathbf{g}^T \mathbf{H} \mathbf{g}$ , which adaptively changed at each step so it converged within about ten iterations. The momentum results in Figure 3.5c used a fixed step size and needed 10 iterations. Here, the step size of  $\alpha = 0.54$  was found by a trial-and-error process to converge in just 10 iterations, but fewer iterations are needed if the value of  $\alpha$  is increased. The Adagrad results in Figure 3.5d, which uses gradient information from the previous iterations, converged within about 4 iterations. The Adagrad method will now be discussed.

### 3.1.2 Adagrad Gradient Descent

The step sizes for steepest descent and momentum, are monolithic in that they apply the same multiplicative factor  $\alpha$  to all components of the gradient  $\nabla \epsilon$ . In contrast, the adaptive gradient (Adagrad) method applies a different *step size* to each component  $g_i = (\nabla \epsilon)_i$ . Each component of the step size depends on information from the current and past iterates.

The formula for the Adagrad method (Duchi et al., 2011) is

$$w_i^{(k+1)} = w_i^{(k)} - \frac{\overbrace{\alpha}^{\text{preconditioner}} \overbrace{\frac{g_i^{(k)}}{\partial \epsilon^{(k)}}}^{\text{gradient}}}{\sqrt{\sum_{k'=1}^k (g_i^{(k')})^2 + \eta}}, \quad (3.8)$$

where  $0 < \eta < 1$  is a small damping factor that is recommended by Ruder (2017) to be about  $10^{-8}$ . Here we have included the superscript  $k$  in  $\epsilon^{(k)}$  to indicate that it changes from one iteration to the next.

We denote the term in front of the gradient component  $g_i^{(k)}$  as a preconditioning term because it roughly resembles that for the diagonal preconditioner in equation 2.41. Preconditioning the gradient by dividing by the cumulative sum of squared gradient components from previous iterations can alleviate ill-conditioning problems when the ellipses of  $\epsilon$  are severely distorted. For example, assume  $\mathbf{g}_{||}^{(k)}$  is the component of the gradient along the gently dipping valley floor, where the floor is almost flat so  $\|\mathbf{g}_{||}^{(k)}\| \approx 0$ . In that case, the magnitudes of  $\mathbf{g}_{||}^{(k)}$  for current and past iterations  $k, k-1, k-2, \dots$  are small, so dividing  $\mathbf{g}_{||}^{(k)}$  by the square root term in equation 3.8 boosts the step size component along the valley

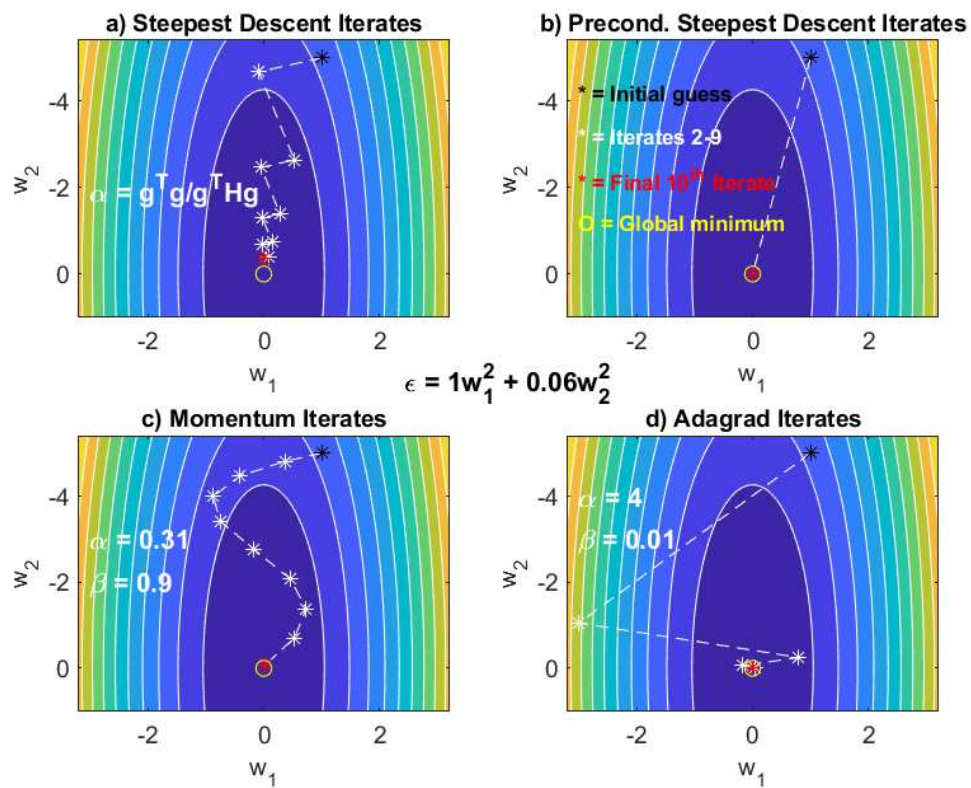


Figure 3.5: Same as Figure 3.4 except  $\epsilon = w_1^2 + 0.06w_2^2$  and the Adagrad result is plotted in d) instead of the Newton result. The Newton result (not shown) is the same as that in b) because the actual inverse  $\mathbf{H}^{-1}$  is the same as the preconditioned inverse  $\mathbf{H}^{-1} = \delta_{ij}/H_{ii}$  in this example. Here the eigenvalues of  $\mathbf{H}$  are  $\lambda_1 = 0.12$  and  $\lambda_2 = 2.0$  so  $\mathbf{H}$  is a SPD Hessian.

floor. In contrast, the step size for the component of the gradient  $\mathbf{g}_\perp^{(k)}$  pointing up the steep walls will be reduced because that component has a large magnitude so its squared reciprocal in the summation of equation 3.8 will be small. Perfect, Adagrad encourages moves along the valley floor towards the minimum and discourages them from climbing the walls.

As a test, the Adagrad method is used to find the minimizer of  $\epsilon = w_1^2 + 0.06w_2^2$ . The results are plotted in Figure 3.5d where no more than 4 iterations are needed for convergence. The results are not shown for minimizing the Rosenbrock function or the saddle-point functional because, similar to the 1st-order gradient methods momentum, SD and preconditioned SD, Adagrad did not perform well for those examples.

According to empirical tests (Ruder, 2017), the main benefit of Adagrad is that it eliminates the need to manually tune the step size as a function of iteration number. The  $\alpha$  value can be set to 0.01 and the iterations can automatically proceed<sup>2</sup>. However, with a large number of iterations the summation in the denominator gets too large so that convergence becomes stalled. To overcome this problem, the gradient method of Adadelata is developed.

### 3.1.3 Adadelata Gradient Descent

To prevent the denominator in equation 3.8 from getting too big when  $k$  is large, we restrict the summation in the square root to a small number of terms:

$$\tilde{E}((g_i^{(k)})^2) = \gamma \sum_{k'=k-\kappa}^{k-1} (g_i^{(k')})^2 + (1-\gamma)(g_i^{(k)})^2, \quad (3.9)$$

and add  $(1-\gamma)(g_i^{(k)})^2$  where  $\gamma = 0.9$  and  $g_i^{(k)} = 0$  for  $k \leq 0$ . Here,  $\kappa$  is the restricted number of terms in the running average so that the non-decaying version of equation 3.8 is

$$w_i^{(k+1)} = w_i^{(k)} - \frac{\alpha}{\sqrt{\tilde{E}((g_i^{(k)})^2) + \eta}} \frac{\partial \epsilon^{(k)}}{\partial w_i}. \quad (3.10)$$

The above formula is identical to the RMSprop formula developed by Hinton (Goodfellow and Hinton, 2016), who suggests  $\gamma = 0.9$  and  $\alpha = 0.001$ .

However, the term on the far right-side of equation 3.10 is unitless because the gradient  $\frac{\partial \epsilon^{(k)}}{\partial w_i}$  is divided by a squared gradient-like term in the square root. The ad hoc remedy is to replace  $\alpha$  with the damped square root of the running average  $\tilde{E}((\Delta w_i^{(k-1)})^2)$  (see equation 3.9) to give the Adadelata update formula:

$$w_i^{(k+1)} = w_i^{(k)} - \frac{\sqrt{\tilde{E}((\Delta w_i^{(k-1)})^2) + \eta}}{\sqrt{\tilde{E}((g_i^{(k)})^2) + \eta}} \frac{\partial \epsilon^{(k)}}{\partial w_i}. \quad (3.11)$$

Multiplying the numerator by a term with units of  $\Delta w_i^{(k)} = w_i^{(k)} - w_i^{(k-1)}$  now gives an update with the correct units. However, it can be argued that the step size  $\alpha$  carries the

---

<sup>2</sup>If the Hessian is not diagonally dominant, then the Hessian inverse can no longer be accurately approximated by  $\frac{1}{\mathbf{H}_{ii}}$ . Hence, the step size should be adjusted at such iterations.

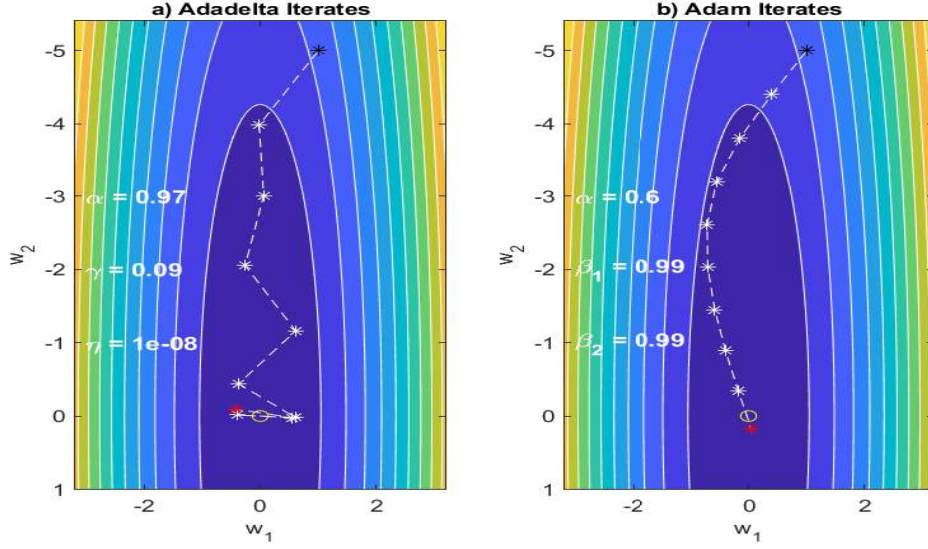


Figure 3.6: Results for finding the minimizer of  $\epsilon = w_1^2 + 0.06w_2^2$  by the a) Adadelta (equation 3.10) and b) Adam methods. Here the number of terms in the summation of equation 3.9 is 4.

correct units because the SD method implicitly assumes  $\mathbf{H} \approx \alpha \mathbf{I}$ .

Ruder (2017) claims that there is no need to set a value to  $\alpha$  here, it adapts the step size to the topographic changes in the objective function. An example of the Adadelta method using equation 3.10 is shown in Figure 3.6a. Similar to the steepest descent results in Figure 3.5a, it oscillates back and forth across the steep sides of the valley until it converges to the minimum point.

### 3.1.4 Nesterov Accelerated Gradient (NAG)

Replacing the current gradient  $\nabla\epsilon(\mathbf{w}^{(k)})$  in equation 3.2 with the peek-ahead gradient at  $\mathbf{w}^{(k)} - \beta\Delta\mathbf{w}^{(k-1)}$  gives the Nesterov accelerated gradient (NAG):

$$\mathbf{v}^{(k)} = \beta\mathbf{v}^{(k-1)} + \alpha \overbrace{\nabla\epsilon(\mathbf{w}^{(k)} - \beta\mathbf{v}^{(k-1)})}^{\text{peek-ahead gradient}} \quad (3.12)$$

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \mathbf{v}^{(k)}. \quad (3.13)$$

Peeking ahead to the approximate gradient at the next iteration provides a more accurate estimate of the slopes between  $\mathbf{w}^{(k-1)}$  and  $\mathbf{w}^{(k+1)}$ . This can provide more productive update directions and avoid local maxima like the rock walls in Figure 3.1. NAG is reported by Bengio et al. (2013) to give faster convergence for recurrent neural networks (see Chapter 12) compared to the momentum method.



### 3.1.5 Adam Gradient Descent

Similar to Adadelta and Adagrad, Kingma and Ba (2014) developed the adaptive moment estimation (Adam) method that applies a different step size to each model component  $w_i$ . These step sizes adapt to the changing topography of the objective function. Similar to momentum, Adam defines a weighted average of current and past gradients, denoted as the first moment:

$$m_i^{(k)} = \beta_1 m_i^{(k-1)} + (1 - \beta_1) g_i^{(k)}, \quad (3.14)$$

and the average of the second moments

$$\sigma_i^{2(k)} = \beta_2 \sigma_i^{2(k-1)} + (1 - \beta_2) (g_i^{(k)})^2, \quad (3.15)$$

of the gradients. Unlike momentum's simple averaging of the current gradient and previous update directions, equation 3.14 is an explicit cumulative weighted sum of all the previous gradients. This is known as exponential averaging (aka a running average) where gradients at the early iterations are downweighted, or decayed (exercises 3.6- 3.7), compared to the ones computed at later iterations (Kingma and Ba, 2014).

This decay strategy makes sense because the local geometry of the objective function is most accurately sensed at the points for the last few iterations of the gradient, not the ones far from the current point. The averaging of these gradients over the many previous iterations provides stabilization of the new gradient because *the last iterate is noisy due to stochastic approximation (of the mini-batch of data), better generalization performance is often achieved by averaging (Kingman and Ma, 2014).*

The initial values of  $m_i^{(k)}$  and  $\sigma_i^{2(k)}$  are set to zero for  $k = 0$  so it biases these terms to zero (see exercises 3.5.6- 3.5.7). This bias can be counteracted by defining the scaled first and second moments (Bushaev, 2018):

$$\begin{aligned} \hat{m}_i^{(k)} &= \frac{m_i^{(k)}}{1 - \beta_1^k}, \\ \hat{\sigma}_i^{2(k)} &= \frac{\sigma_i^{2(k)}}{1 - \beta_2^k}, \end{aligned} \quad (3.16)$$

so the Adam update formula is

$$w_i^{(k+1)} = w_i^{(k)} - \frac{\alpha}{\sqrt{\hat{\sigma}_i^{2(k)}} + \eta} \hat{m}_i^{(k)}, \quad (3.17)$$

where the values of  $\beta_1 = .9$ ,  $\beta_2 = 0.99$  and  $\eta = 10^{-8}$ . If  $L^2$  regularization is used then the above formula becomes (Bushaev, 2018)

$$w_i^{(k+1)} = w_i^{(k)} - \frac{\alpha}{\sqrt{\hat{\sigma}_i^{2(k)}} + \eta} \hat{m}_i^{(k)} - \alpha \lambda w_i^{(k)}, \quad (3.18)$$

Empirical results show it works well compared to other adaptive learning methods since



Adam can be viewed as the combination of RMSprop and SGD with momentum. Despite some problems, Bushaev (2018) claims that Adam is one of the best optimization algorithms for deep learning and its popularity is growing very fast.

An example of the Adam method using equation 3.17 is shown in Figure 3.6b. The ten iterations trace a white curve that is a much smoother track than that for the Adadelata method. The MATLAB code 3.1.2 for computing the results in Figure 3.6 is below, with the plotting subroutine *plotDelta* in Appendix 3.7.

**Code 3.1.2.** *MATLAB Code for Minimizing  $\epsilon = w_1^2 + 0.06w_2^2$  in Figure 3.6: Adadelata and Adam Methods. Plotting subroutine plotDelta is in Appendix 3.7*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Iterative Adadelata and Adam methods for finding minimizer of w_1^2 + 0.06w_2^2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all
nitt=2; iter= 10; g0=0; gg=zeros(2, iter);
www=zeros(iter, 2, nitt);
for itt=1:nitt

    w1 =1; w2 =-5; w0=[w1 w2];
    w = [w1; w2]; % Define starting point (w1, w2)
    %
    ww = zeros(iter, 2);
    ww(1, 1) = w1;
    ww(1, 2) = w2;
    a1=1; a2=.06;
    gsum2x=0; gsum2y=0; wp=w;
    %
    mx=0; my=0; sx=0; sy=0; gh=zeros(2, iter); kno=4;
    for it = 2:iter
        % Begin non-linear iterations
        g=[2*a1*w1; 2*a2*w2]; % 2x1 gradient vector
        H=2*[a1 0; 0 a2]; % 2x2 Hessian matrix
        if itt==1;
            [g, alphaAdm, beta1Adm, beta2Adm, H, mx, my, sx, sy]=...
                Adam(g, H, mx, my, sx, sy, it-1) % Adam
        end
        if itt==2; [g, H, alAdel, gam, lam, gh]=Adelta(g, H, it, gh, kno);
        end; % Adelta
        w = w-inv(H)*g; % Non-linear Newton formula
        ww(it, :) = w; wp=w; % Update iterative solution
        w1 = w(1); x1=w1;
        w2 = w(2); x2=w2;
    end
    www(:, :, itt)=ww;
end
xmax=3.2; xmin=-3.2; ymax=1; ymin=-5.4;
ww=www(:, :, 1);
subplot(122); io=1; plotDelta...
    (ww, iter, x1, x2, io, a1, a2, w0, xmax, ymax, xmin, ymin);
text(-3, -3, ['\alpha = ', num2str(alphaAdm)], 'Color', 'white', 'Fontweight', 'bold')
text(-3, -2, ['\beta_1 = ', num2str(beta1Adm)], 'Color', 'white', 'Fontweight', 'bold')
text(-3, -1, ['\beta_2 = ', num2str(beta2Adm)], 'Color', 'white', 'Fontweight', 'bold')
ww=www(:, :, 2);
subplot(121); io=2; plotDelta...
    (ww, iter, x1, x2, io, a1, a2, w0, xmax, ymax, xmin, ymin)
text(-3, -3, ['\alpha = ', num2str(alAdel)], 'Color', 'white', 'Fontweight', 'bold')
text(-3, -2, ['\gamma = ', num2str(gam)], 'Color', 'white', 'Fontweight', 'bold')
text(-3, -1, ['\eta = ', num2str(lam)], 'Color', 'white', 'Fontweight', 'bold')
print -depsc ch.RegressADelta.eps
```

**Code 3.1.3.** *MATLAB Adam Gradient Descent Subroutine in Code 3.1.2*

```

function [g,alpha,beta1,beta2,H,mx,my,sx,sy]=Adam(g,H,mx,my,sx,sy,it)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Adam Gradient Computation
% m^(k)=beta1*m^(k-1)+(1-beta1)*g^(k)
% s^(k)=beta2*s^(k-1)+(1-beta2)*g^(k)^2;
% mh^(k)=m^(k)/(1-beta_1^k)
% sh^(k)=s^(k)/(1-beta_2^k)
% w^(k+1)=w^(k+1) - alpha/sqrt(sh^(k)+lam) *mh^(k)
%
% (g,H) - in/output- gradient and Hessian input/output
% mh - in/output- 1st moment gradient
% sh - in/output- 2nd moment gradient
% it - input - iteration #
% lam,alpha - input - damping term, step size
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
alpha=.6;beta1=.9;beta2=.99;
gx=g(1);gy=g(2);lam=10^-8;
mx=beta1*mx+(1-beta1)*gx;
my=beta1*my+(1-beta1)*gy;
sx=beta2*sx+(1-beta2)*gx^2;
sy=beta2*sy+(1-beta2)*gy^2;
mxh=mx/(1-beta1^it);
myh=my/(1-beta1^it);
sxh=sx/(1-beta2^it);
syh=sy/(1-beta2^it);
H=eye(2,2);
g(1)=alpha*mxh/(sqrt(sxh)+lam);
g(2)=alpha*myh/(sqrt(syh)+lam);

```

and the MATLAB code for Adadelta is below.

**Code 3.1.4.** *MATLAB Adadelta Gradient Descent Subroutine in Code 3.1.2*

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Adadelta Gradient Descent Code
% E=gam/kno sum_{k=it-kno}^{it-1} (g^(k))^2+(1-gam)(g^(it))^2
% Preconditioner: P=alpha./sqrt(E+lam);
% Preconditioned gradient: g=P.*g
%
% (g,H) - in/output- gradient and Hessian input/output
% gh - in/output- iteration history of input gradient
% gh initial predefined as gh=zeros(2,nit)
% it - input - iteration #
% kno - input - # terms in above summation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
alpha=.97;lam=10^-8;gam=.09;
g2=0*g;
gh(:,it)=g;
kstart=it-kno;if kstart<1;kstart=1;end
icnt=0;
if it>1;
    for k=kstart:it-1
        g2=g2+gh(:,k).^2;
        icnt=icnt+1;
    end
end
E=gam*g2 +(1-gam)*g.^2;
P=alpha./sqrt(E+lam);
H=eye(2,2);
g=P.*g;

```

### 3.1.6 AdaMax Gradient Descent

The AdaMax gradient descent method replaces the first moment averaging of gradients in equation 3.14 with

$$\begin{aligned} u_i^{(k)} &= \beta_2^\infty m_i^{(k-1)} + (1 - \beta_2^\infty) |g_i^{(k)}|^\infty, \\ &= \max(\beta_2 m_i^{(k-1)}, |g_i^{(k)}|), \end{aligned} \quad (3.19)$$

where the  $\infty$  superscript indicates the  $L^\infty$  norm (Garcia, 2018). The  $L^\infty$  formula<sup>3</sup> above ignores a smaller component of the gradient in favor of a larger value, and the exponential averaging<sup>4</sup> provides some stability in the presence of noise. The AdaMax update formula is

$$w_i^{(k+1)} = w_i^{(k)} - \frac{\alpha}{\sqrt{u_i^{(k)}} + \eta} \hat{m}_i^{(k)}, \quad (3.20)$$

where useful default values of the hyperparameters (Ruder, 2017) are  $\alpha = 0.02$ ,  $\eta = 0.002$ ,  $\beta_1 = 0.9$  and  $\beta_2 = 0.99$ .

The Adamax Keras code is below.

**Code 3.1.5.** *Keras AdaMax Gradient Descent Subroutine (k-eras.io/api/optimizers/adamax/)*

```
tf.keras.optimizers.Adamax(
    learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-07, name="Adamax", **kwargs)

m = 0 # Initialize initial 1st moment vector
v = 0 # Initialize the exponentially weighted infinity norm
t = 0 # Initialize timestep

t += 1
m = beta1 * m + (1 - beta) * g
v = max(beta2 * v, abs(g))
current_lr = learning_rate / (1 - beta1 ** t)
w = w - current_lr * m / (v + epsilon)
```

### 3.1.7 Hybrid Step-length Strategy

Keskar and Socher (2017) report that *Adam outperforms SGD in both training and generalization metrics in the initial portion of the training, but then the performance stagnates*. Their recommendation is that a method such as Adam is used for the initial iterations, then switch to SGD when its convergence rate stalls. Here, the adaptive Adam method applies a non-uniform step length to each parameter while the SGD method applies the same step length to all the parameters. This hybrid strategy arises from the observation of Wilson et al. (2017) that non-uniform scaling of step lengths inhibits generalization.

<sup>3</sup>According to Ruder (2017), the  $L^\infty$  norm can sometimes provide more stable behavior with inverse problems.

<sup>4</sup>Exponential averaging in equation 3.19 adds the weighted gradient  $\mathbf{g}^{(k)}$  to the sum of the previous weighted gradients. These weights become smaller with the earlier iterations. See exercises 3.6- 3.7.

### 3.2 Numerical Line Search

The local quadratic assumption breaks down when the misfit functional is highly non-linear so that the exact line search can be inaccurate. An alternative is the numerical line search method which estimates the step size by evaluating the objective function at several points along a downhill direction. For example, a quadratic line search begins by defining the search direction  $\Delta \mathbf{w}$ , computing several reasonable values of the step size, e.g.,  $\alpha_1 = 0.5$  and  $\alpha_2 = 1$ , and the functional values  $\epsilon(\mathbf{w} + \alpha_1 \Delta \mathbf{w})$  and  $\epsilon(\mathbf{w} + \alpha_2 \Delta \mathbf{w})$  are tested to determine if they are less than  $\epsilon(\mathbf{w})$ , and if  $\epsilon(\mathbf{w} + \alpha_1 \Delta \mathbf{w}) \leq \epsilon(\mathbf{w} + \alpha_2 \Delta \mathbf{w})$ . Then, the following quadratic formula is used to determine the interpolated value of the functional:

$$\begin{aligned} \epsilon(\mathbf{w} + \alpha \Delta \mathbf{w}) = & \frac{(\alpha - \alpha_1)(\alpha - \alpha_2)}{\alpha_1 \alpha_2} \epsilon(\mathbf{w}) + \frac{\alpha(\alpha - \alpha_2)}{\alpha_1(\alpha_1 - \alpha_2)} \epsilon(\mathbf{w} + \alpha_1 \Delta \mathbf{w}) \\ & + \frac{\alpha(\alpha - \alpha_1)}{\alpha_2(\alpha_2 - \alpha_1)} \epsilon(\mathbf{w} + \alpha_2 \Delta \mathbf{w}) \quad 0 \leq \alpha \leq \alpha_2. \end{aligned} \quad (3.21)$$

The above function represents a quadratic polynomial in  $\alpha$  that gives  $\epsilon(\mathbf{w})$ ,  $\epsilon(\mathbf{w} + \alpha_1 \Delta \mathbf{w})$ , and  $\epsilon(\mathbf{w} + \alpha_2 \Delta \mathbf{w})$  for  $\alpha = 0, \alpha_1, \alpha_2$ , respectively. The scalar value of  $\alpha$  that minimizes the above formula is found by differentiating  $\epsilon(\alpha)$  with respect to  $\alpha$ , setting the result to zero, and solving for  $\alpha$ .

The above interpolation formula requires three functional evaluations of  $\epsilon(\mathbf{w})$ , so an alternative is to use the values of  $\epsilon(\mathbf{w})$ ,  $\epsilon(\mathbf{w} + \alpha_1 \Delta \mathbf{w})$ , and the existing directional derivative of  $\epsilon(\mathbf{w})'$  at  $\mathbf{x}$  to get the quadratic interpolation formula:

$$\epsilon(\mathbf{w} + \alpha \Delta \mathbf{w}) = \frac{\alpha^2}{\alpha_1^2} [\epsilon(\mathbf{w} + \alpha_1 \Delta \mathbf{w}) - \alpha_1 \epsilon(\mathbf{w})' - \epsilon(\mathbf{w})] + \alpha \epsilon(\mathbf{w})' + \epsilon(\mathbf{w}), \quad (3.22)$$

where  $\epsilon(\mathbf{w})' = \frac{d\epsilon(\mathbf{w} + \alpha \Delta \mathbf{w})}{d\alpha}|_{\alpha=0}$  represents the derivative of  $\epsilon(\mathbf{w})$  along the search direction. Note,  $\epsilon(\mathbf{w} + \alpha \Delta \mathbf{w})$  evaluated at  $\alpha = 0$  gives the value of  $\epsilon(\mathbf{w})$ , and  $d\epsilon(\mathbf{w} + \alpha \Delta \mathbf{w})/d\alpha$  gives the value  $\epsilon(\mathbf{w})'$  at  $\alpha = 0$ , and  $\epsilon(\mathbf{w} + \alpha \Delta \mathbf{w})|_{\alpha=\alpha_1}$  yields  $\epsilon(\mathbf{w} + \alpha_1 \Delta \mathbf{w})$ . A backtracking bisection code is in section 4.3.3.

### 3.3 Conjugate Gradient

Assume an  $N \times N$  symmetric real matrix  $\mathbf{H}$  where the eigenvalues are positive (symmetric positive definite, aka SPD) such that

$$\mathbf{H}\mathbf{w} = \mathbf{b}, \quad (3.23)$$

where  $\mathbf{b}$  is aspecified  $N \times 1$  vector. The goal is to find the solution  $\mathbf{w}$  by an iterative gradient-descent method that is much faster than the steepest descent algorithm. In this case the objective function is

$$\epsilon = \frac{1}{2} \mathbf{w}^T \mathbf{H} \mathbf{w} - \mathbf{w}^T \mathbf{b}, \quad (3.24)$$

where the gradient  $\mathbf{g}^{(1)}$  at  $\mathbf{w}^{(1)}$  is

$$\nabla \epsilon(\mathbf{w}) \Big|_{\mathbf{w}=\mathbf{w}^{(1)}} = \overbrace{\mathbf{H}\mathbf{w}^{(1)} - \mathbf{b}}^{\mathbf{g}^{(1)}}, \quad (3.25)$$

and the starting point is  $\mathbf{w}^{(0)}$  in Figure 3.7.

By definition, the gradient at the optimal point  $\mathbf{w}^*$  satisfies

$$\nabla \epsilon(\mathbf{w}) \Big|_{\mathbf{w}=\mathbf{w}^*} = \mathbf{H}\mathbf{w}^* - \mathbf{b} = 0 \rightarrow \mathbf{H}\mathbf{w}^* = \mathbf{b}. \quad (3.26)$$

Figure 3.7 illustrates that the update direction  $\Delta\mathbf{w}^{(0)} = \mathbf{w}^{(1)} - \mathbf{w}^{(0)}$  is perpendicular to the gradient  $\mathbf{g}^{(1)}$  at  $\mathbf{w}^{(1)}$ , i.e.,

$$(\Delta\mathbf{w}^{(0)}, \overbrace{\mathbf{H}\mathbf{w}^{(1)} - \mathbf{b}}^{\mathbf{g}^{(1)}}) = 0. \quad (3.27)$$

Plugging  $\mathbf{b} = \mathbf{H}\mathbf{w}^*$  from equation 3.26 into equation 3.27 gives

$$(\Delta\mathbf{w}^{(0)}, \overbrace{\mathbf{H}\{\mathbf{w}^* - \mathbf{w}^{(1)}\}}^{\Delta\mathbf{w}^*}) = 0. \quad (3.28)$$

This equation says that, for the  $2 \times 2$  matrix  $\mathbf{H}$ , the optimal update direction  $\Delta\mathbf{w}^{(1)} \rightarrow \Delta\mathbf{w}^*$  from  $\mathbf{w}^{(1)}$  is conjugate<sup>5</sup> to the previous move direction  $\Delta\mathbf{w}^{(0)}$ . Unlike the steepest descent method, which will waste many iterations oscillating back and forth between the opposite sides of the steep-valley walls, the conjugate gradient (CG) iterations will converge in only two iterations for the  $2 \times 2$  system of equations in equation 3.23.

Since  $\mathbf{w}^*$  is in the plane spanned by  $\Delta\mathbf{w}^{(0)}$  and  $\mathbf{g}^{(1)} = \frac{\partial \epsilon(\mathbf{w}^{(1)})}{\partial \mathbf{w}}$ , that is

$$\Delta\mathbf{w}^* = -\mathbf{g}^{(1)} + \beta \Delta\mathbf{w}^{(0)}, \quad (3.29)$$

then plugging equation 3.29 into equation 3.28 and rearranging gives the value for  $\beta$

$$\beta = \frac{-(\Delta\mathbf{w}^{(0)}, \mathbf{H}\mathbf{g}^{(1)})}{(\Delta\mathbf{w}^{(0)}, \mathbf{H}\Delta\mathbf{w}^{(0)})}. \quad (3.30)$$

More generally, if the system of equations is governed by an  $N \times N$  SPD matrix  $\mathbf{H}$ , then the successive updates  $\Delta\mathbf{w}^{(k)}$  can be made conjugate to the previous update  $\Delta\mathbf{w}^{(k-1)}$  using the update constraint on  $\beta$ :

$$\beta = \frac{(\Delta\mathbf{w}^{(k-1)}, \mathbf{H}\mathbf{g}^{(k)})}{(\Delta\mathbf{w}^{(k-1)}, \mathbf{H}\Delta\mathbf{w}^{(k-1)})}. \quad (3.31)$$

In this case the CG algorithm is guaranteed to converge in at most  $N$  iterations (Nocedal and Wright, 1999). In practice, far fewer iterations are typically needed to get a useful

---

<sup>5</sup>Two vectors  $\mathbf{x}$  and  $\mathbf{y}$  are defined to be conjugate to one another if  $(\mathbf{x}, \mathbf{H}\mathbf{y}) = 0$ .

The CG workflow is summarized by the following equations.

- $$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \alpha \Delta \mathbf{w}^{(k)}, \quad (3.32)$$

3. The gradient at  $\mathbf{w}^{(k+1)}$  is computed by

$$\mathbf{g}^{(k+1)} = \nabla \epsilon(\mathbf{w}^{(k+1)}), \quad (3.33)$$

so that  $\Delta \mathbf{w}^{(k+1)}$  is found by

$$\Delta \mathbf{w}^{(k+1)} = -\mathbf{g}^{(k+1)} + \beta \Delta \mathbf{w}^{(k)}. \quad (3.34)$$

Here,  $\beta$  is given in equation 3.31, but it can be replaced by the equivalent Polack-

Ribere formula (Nocedal and Wright, 1999)

$$\beta = \frac{(\mathbf{g}^{(k+1)} - \mathbf{g}^{(k)}, \mathbf{g}^{(k+1)})}{(\mathbf{g}^{(k)}, \mathbf{g}^{(k)})}. \quad (3.35)$$

This formulation resets the new update direction  $\Delta \mathbf{w}^{(k)}$  to be the steepest descent direction if progress stalls so that the descent directions are the same. This is necessary in non-linear CG because the conjugacy among the previous updates is lost when the objective function is highly non-linear.

4. Set  $k := k + 1$  and repeat steps 2-4 until convergence.

### 3.3.1 MATLAB Conjugate Gradient Code

The MATLAB code for solving a square SPD system of equations by a CG method is given below.

**Code 3.3.1.** *MATLAB Code for Conjugate Gradient for a Square System of Equations*

```
%
% CG Iterations to solve SPD eqn Lx=t
% where gradient gk = L*xk-t;
% and iterate soln xk1 = xk + alpha*dk1
%
clear all;
L = [400 1; 1 4];           % Simple 2x2
t = [5 2]';                 % problem
[m,n] = size(L);
nit = 3;
%
xk = zeros(n,1);            % Starting point at origin
gk = (L*xk-t); dk=-gk;
%
for q = 1:nit;
    alpha = -dk'*gk/(dk'*L*dk);
    xk1 = xk + alpha*dk;      % Compute new iterate
    gk1 = gk+alpha*L*dk;      % Find new gradient
    beta = gk1'*gk1/(gk'*gk); % Fletcher-Reeves
    beta = gk1'*(gk1-gk)/(gk'*gk); % Polak-Ribiere
    dk1 = -gk1+beta*dk;       % Find new conj. direction
    dk = dk1;                 % Refresh iterates
    gk = gk1;
    xk = xk1
end
%
```

If the system of equations is overdetermined then the problem is to find  $\mathbf{x}$  that satisfies the normal equations  $\mathbf{L}^T \mathbf{L} \mathbf{x} = \mathbf{L}^T \mathbf{t}$ . Similar to the steepest descent solution of an overdetermined system of equations, the solution  $\mathbf{x}$  is found that minimizes the misfit function given by the sum of the squared residuals:

$$f(\mathbf{x}) = \frac{1}{2}(\mathbf{L}\mathbf{x} - \mathbf{t})^T(\mathbf{L}\mathbf{x} - \mathbf{t}) = \frac{1}{2}\mathbf{x}^T \mathbf{L}^T \mathbf{L} \mathbf{x} - \mathbf{t}^T \mathbf{L} \mathbf{x} + \frac{1}{2}\mathbf{t}^T \mathbf{t}. \quad (3.36)$$

Here  $f(\mathbf{x})$  is a quadratic functional in  $\mathbf{x}$ , and the right hand side of equation 3.36 can be viewed as the first three terms of a Taylor series in  $\mathbf{x}$  expanded about the origin. The gradient is given by

$$\nabla f(\mathbf{x}) = \mathbf{L}^T(\mathbf{L}\mathbf{x} - \mathbf{t}), \quad (3.37)$$

and the solution can be found by iteratively minimizing along mutually conjugate directions.

The matrix product in the gradient should be computed by first calculating the residual  $\mathbf{r}$  (i.e.,  $\mathbf{r} = \mathbf{L}\mathbf{x} - \mathbf{t}$ ), and then multiplying it by  $\mathbf{L}^T$ . This minimizes the computational expense by only carrying out matrix-vector multiplication (rather than the matrix-matrix  $\mathbf{L}^T\mathbf{L}$  multiplication), and also minimizes round-off error. Scales (1987) applied the CG solver to some travelttime tomography problems and reports that the linear CG solver performs at least as well as a SIRT (simultaneous iterative reconstruction tomography) method.

Below is a conjugate gradient MATLAB code for solving an overdetermined system of linear equations. There is no need to show the results for finding the solution to the SPD quadratic  $\epsilon w_1^2 + 0.06w_2^2$  problem, it will converge to the minimum within two iterations no matter where the starting point is. The first move will be along the steepest descent direction, and then the next one will be to the point that minimizes  $\epsilon$ . This is because CG optimization is guaranteed to find the stationary  $N \times 1$  point  $\mathbf{x}^*$  within  $N$  iterations for any quadratic objective function as long as the  $N \times N$  Hessian is SPD (Nocedal and Wright, 1999).



**Code 3.3.2.** *MATLAB Conjugate Gradient Code for an Overdetermined System of Equations*

```

%
% CG Iterations to solve overdetermined eqns Lx=b
% where gradient gk = L'*(L*xk-b);
% and iterate soln xk1 = xk + alpha*dk1
%
clear all;
L = [400 1; 1 4; 0 1]; % Simple 3x2
b = [5 2 .5]'; % problem
[m,n] = size(L);
nit = 3;
%
xk = zeros(n,1); % Starting point at origin
gk = L'*(L*xk-b); dk=-gk;
%
for q = 1:nit;
    res = L*dk;
    res1 = L'*res;
    alpha = -dk'*gk/(res'*res);
    xk1 = xk + alpha*dk; % Compute new iterate
    gk1 = gk+alpha*res1; % Find new gradient
    beta = gk1'*gk1/(gk'*gk); % Fletcher-Reeves
    beta = gk1'*(gk1-gk)/(gk'*gk); % Polak-Ribiere
    dk1 = -gk1+beta*dk; % Find new conj. direction
    dk = dk1; % Refresh iterates
    gk = gk1;
    xk = xk1
end

```

If the  $N \times N$  matrix  $\mathbf{L}$  is characterized by eigenvalues  $\lambda_i$ , then  $\mathbf{L}^T \mathbf{L}$  has the eigenvalues  $\lambda_i^2$ . This means that the condition number for  $\mathbf{L}^T \mathbf{L}$  will be squared compared to that for  $\mathbf{L}$ , which can result in slower convergence; hence the need for preconditioning. An alternative is the iterative LSQR method which is a more reliable algorithm when the system of equations is ill-conditioned (Paige and Saunders, 1982).

For a quadratic functional, the CG direction should always be reset to the gradient direction at or before the  $N^{th}$  iteration, where  $N$  is the dimension of the vector  $\mathbf{w}$ . For a non-linear functional, the CG direction should be reset to the steepest descent direction every few iterations because the search directions quickly lose mutual conjugacy.

Preconditioning can also be introduced with the CG method to take advantage of information from past updates to assign separate weighting to each component of the new update direction. Gradient information from many previous moves can be used to sequentially update the preconditioner to be a more accurate approximation to the Hessian inverse. This type of algorithm defines the quasi-Newton (QN) method (Nocedal and Wright, 1999). Since these methods attempt to approximate the second-order derivatives in the Hessian, then they are called second-order gradient descent methods compared to the first-order ones such as SD. The QN method uses update and gradient information from a limited number of previous moves to expedite convergence. The penalty of QN is that it is more computationally expensive than first-order methods, typically  $O(m)$  times more expensive where  $m$  is the number of iterative histories it must store. In the case of the sampled QN methods (Berahas et al., 2019),  $m$  is the number of function evaluations sampled around each up-

dated solution. Surprisingly, the numerical tests (Berahas et al., 2019) for a neural network and the MNIST data set showed that the first-order gradient method Adam significantly outperformed the tested QN methods in quickly achieving convergence.

### 3.4 Summary

Nine gradient descent methods are described and tested: SD, preconditioned SD, Newton, momentum, Adadelta, Adagrad, Adam, and CG. The tests are for finding the minimizers for a non-linear functional, a quadratic functional which violated the SPD condition, and a SPD quadratic functional where the aspect ratio of the ellipses was greater than 7 : 1. The Newton method revealed the best performance because it sensed both the local gradient and curvature of the functional at any point, and therefore is able to achieve convergence for all three examples. However, the high computational cost of the Newton method<sup>6</sup> often prevents it from being used for realistic ML problems. The next best methods (SD, Adam, Adagrad and Adadelta) are those that weight each unknown  $w_i$  with a separate weighting scheme, which we called a preconditioner. Preconditioning acts as an approximation to the Hessian inverse, and tends to transform the ellipses of  $\epsilon$  into more rounded contours and therefore speed up the convergence rate.

The preconditioned gradient descent methods tended to have faster convergence rates than the methods that applied the same step size to the gradient vector. Only the Newton method showed acceptable results for the Rosenbrock function and the quadratic that violated the SPD condition.

Similar to momentum, Adagrad, Adadelta, and Adam, the CG algorithm uses gradient information from a previous iterate to expedite convergence. However, it is unique in that it uses the previous gradient  $\mathbf{g}^{(k)}$  and update vectors  $\Delta\mathbf{w}^{(k)}$  to enforce conjugacy ( $\Delta\mathbf{w}^{(k)}, \mathbf{H}\Delta\mathbf{w}^{(k+1)} = 0$  between the next update  $\Delta\mathbf{w}^{(k+1)}$  and the previous update  $\Delta\mathbf{w}^{(k)}$  vectors).

To summarize, all of the above gradient descent methods aim to achieve a single goal: get to the global minimum in a computationally efficient manner. Theoretically, the second-order methods should be the most effective gradient descent methods, but limited empirical results suggest that Adam can sometimes be the most effective gradient optimization method. This is consistent with Bushaev’s (2018) claims that Adam is one of the best optimization algorithms for deep learning and its popularity is growing very fast. Contrarily, Adam can sometimes be less effective at generalization so the cure, in some cases, is to use Adam for early iterations and then use SGD with a uniform step length to finish up the iterations (Keskar and Socher, 2017).

As a reminder, none of the gradient methods guarantee convergence to the global minima for highly non-linear problems. Efficiently avoiding local minima is still a major challenge in ML and optimization theory.

---

<sup>6</sup>For an  $N \times N$  Hessian  $\mathbf{H}$ , the computational cost for a direct inverse  $\mathbf{H}^{-1}$  is  $O(N^3)$  algebraic operations, which is prohibitive for typical ML problems with more than  $10^6$  unknowns.

### 3.5 Exercises

1. The preconditioned SD assumes a diagonally dominant Hessian  $\mathbf{H}$  in order to accurately approximate its inverse with the preconditioner  $[\mathbf{H}]_{ij}^{-1} \approx \delta_{ij}/H_{ii}$ . Explain why the preconditioned SD did so poorly in preferring to move downward at each iteration in Figure 3.4. Hint:

$$\begin{aligned}\Delta w_1 &= -\{\mathbf{H}^{-1}\mathbf{g}\}_1 = -H_{11}^{-1}g_1 - H_{12}^{-1}g_2. \\ \Delta w_2 &= -\{\mathbf{H}^{-1}\mathbf{g}\}_2 = -H_{21}^{-1}g_1 - H_{22}^{-1}g_2.\end{aligned}\tag{3.38}$$

Now look at values of the components  $H_{11}^{-1}$ ,  $H_{12}^{-1}$ ,  $H_{21}^{-1}$  and  $H_{22}^{-1}$  in equation 3.4 and interpret which way they are encouraging the update to go for  $\Delta w_1$ . Compare this to the direction of the preconditioned SD update

$$(\Delta w_1, \Delta w_2) = -(g_1/H_{11}, g_2/H_{22}).\tag{3.39}$$

That is, pick an iterate point on the saddle-like curve, estimate the gradient values for  $\mathbf{g} = (g_1, g_2)$ , and plug these values into equations 3.38-3.39.

2. Becker and LecCun (1988) proposed a diagonal approximation to the Hessian. This diagonal approximation can be computed with one additional forward and back-propagation through the model, effectively doubling the computation over standard gradient descent. Once the diagonal of the Hessian is computed,  $\text{diag}(\mathbf{H})$ , the update rule becomes:

$$\Delta w_i = \frac{1}{|H_{ii}| + \eta}.\tag{3.40}$$

For a rectangular system of equations  $\mathbf{L}\mathbf{w} = \mathbf{d}$ , we know that the least squares solution is  $\mathbf{w} = [\mathbf{L}^T\mathbf{L}]^{-1}\mathbf{L}^T\mathbf{d}$ . In this case the diagonal approximation to the inverse is  $\{[\mathbf{L}^T\mathbf{L}]^{-1}\}_{ij} \approx \delta_{ij}/[\mathbf{L}^T\mathbf{L}]_{ii}$ . This means that the  $i^{\text{th}}$  diagonal component  $[\mathbf{L}^T\mathbf{L}]_{ii}$  is computed by taking the dot product of the  $i^{\text{th}}$  row vector of  $\mathbf{H}$  with itself. In this case show that this dot product is always greater than zero so the absolute value in the above equation is unnecessary for this example. If  $\mathbf{L}$  is the modeling operator and  $\mathbf{L}^T$  is the back-propagation operator, show how these two operations can be efficiently used to get  $[\mathbf{L}^T\mathbf{L}]_{ii}$ . Hint: Applying  $\mathbf{H}$  to the  $N \times 1$  unit vector  $(1, 0, 0, \dots, 0)$  returns the first column vector of  $\mathbf{H}$ .

3. Test the sensitivity of convergence to the step size for the momentum, preconditioned SD and steepest descent methods applied to the Rosenbrock function. Use several different starting points. Plot the number of iterations for convergence versus step size.
4. Same as previous exercise except use a bisection backtracking algorithm to determine the step size.
5. Find the eigenvalues of  $\mathbf{H}$  for the Rosenbrock function at different points  $\mathbf{w}$ . Hint: *MATLAB* `eig(H)` returns the eigenvalues of a square matrix. Can one say that the

Hessian matrices associated with the Rosenbrock function at different points are SPD everywhere? Explain.

6. Show by successive substitution (see Bushaev (2018)) that

$$\begin{aligned} m_i^{(k)} &= \beta_2 m_i^{(k-1)} + (1 - \beta_2)(g_i^{(k)})^2, \\ &= (1 - \beta_2) \sum_{k'=1}^k \beta_2^{k-k'} (g_i^{(k')})^2. \end{aligned} \quad (3.41)$$

Does this *exponential moving average* downweight or upweight gradients computed at the early iterations if  $\beta_2 = .9$ ? Explain why this is useful for finding the stationary point for non-linear objective functions.

7. Consider that the  $k^{th}$  minibatch of data is noisy and can be considered as the  $k^{th}$  sample in an ensemble of random minibatches. The random character of the noisy data is inherited by the gradient  $\mathbf{g}^{(k)}$  at the  $k^{th}$  iteration. Therefore, the  $M \times 1$  vector  $\mathbf{g}^{(k)}$  is a multidimensional random vector, whose average value can be approximated as

$$\bar{\mathbf{g}} = E[\mathbf{g}^{(k)}] \approx \frac{1}{K} \sum_{k=1}^K \mathbf{g}^{(k)}, \quad (3.42)$$

where the summation is over  $K$  samples from the ensemble. Show that

$$\begin{aligned} E[m_i^{(k)}] &= E\left[(1 - \beta_2) \sum_{k'=1}^k \beta_2^{k-k'} (g_i^{(k')})^2\right], \\ &= E[(g_i^{(k)})^2] (1 - \beta_2) \sum_{k'=1}^k \beta_2^{k-k'} + \eta, \\ &= E[(g_i^{(k)})^2] (1 - \beta_2^k), \end{aligned} \quad (3.43)$$

where  $m_i^{(k)}$  is defined in equation 3.41. Hint: Recall the geometric series  $1/(1 - x) = 1 + x + x^2 + \dots$  for  $0 < x < 1$ . Explain why the average  $E[m_i^{(k)}]$  is biased to zero for a large number of iterations and  $\beta_2 = 0.99$ .

8. Use the CG code 3.3.1 to iteratively find the minimum for  $\epsilon = w_1^2 + \omega w_2^2$  and compare the results to those computed by Adam for different values of  $\omega \in \{.06, .006, .0006, .00006\}$ . What is the condition number  $\lambda_{max}/\lambda_{min}$  of  $\mathbf{H}$  in each case? How much does the convergence rate of Adam depend on the values of the hyperparameters  $(\alpha, \beta_1, \beta_2)$ ?
9. Compare the performances of the Adam and AdaMax methods for finding the stationary points of the Rosenbrock function for different starting points.
10. Derive the Polack-Ribiere formula in equation 3.35.

### 3.6 Appendix: Plotting Subroutine for Rosenbrock Minimizer Code 3.1.1

The plotting function for the Rosenbrock minimization code 3.1.1 is below.

#### Code 3.6.1. MATLAB Plot Subroutine for Code 3.1.1

```
function plotrosen1(ww,iter,x1,x2,io)
mxx=max(ww(:,1),1);mnx=min(ww(:,1),1);mxy=max(ww(:,2),1);mny=min(ww(:,2),1);
dxx=mxx-mnx; dyy=mxy-mny;

xx=ww;xmx = mxx+dxx/2;xmin = mnx-dxx/2;ymin0 = mny-dyy/2;ymx = 2*mxy+dyy/2;
x = [xmin:.01:xmx];y = [ymin0:.01:ymx];[X,Y] = meshgrid(xmin:.01:xmx,ymin0:.01:ymx );

f      = 100.*(Y-X.^2).^2+(1-X).^2;          % Rosenbrock Function
imagesc(x,y,f);
hold on;
[C,h]=contourf(X,Y,f,10,'w');    % Plot Rosenbrock function ith contours
if io==4;v=[41 124 208]; clabel(C,h,v,'color','white','Fontweight','bold');
end
hold off
hold on;
for i = 1:iter;
    plot(xx(i,1),xx(i,2),'w*');
end
plot(xx(iter,1),xx(iter,2),'*r');
plot(ww(1,1),ww(1,2),'*g');
plot(1,1,'oy')
hold off;
xlabel('X','FontSize',9);
ylabel('Y','FontSize',9);
if io==2;
    title(['b] Precond. Steepest Descent Iterates'],...
        'FontSize',9);end
if io==4;
    title(['d] Newton Iterates'],...
        'FontSize',9);end
if io==3;
    title(['c] Momentum Iterates'],...
        'FontSize',9);end
if io==1;
    title(['a] Steepest Descent Iterates'],...
        'FontSize',9);
text(.42,1.0,'* = Initial guess','FontSize',8,'color','green','Fontweight','bold');
text(.42,1.25,'* = Iterates 2-9','FontSize',8,'color','white','Fontweight','bold');
text(.42,1.5,'* = Final 10th Iterate','FontSize',8,'color','red','Fontweight','bold');
text(.42,1.75,'0 = True minima','FontSize',8,'color','yellow','Fontweight','bold');
;end
```

### 3.7 Appendix: Plotting Subroutine for Adadelata and Adam Minimizer Code 3.1.2

The MATLAB plotting function *plotDelta* for the quadratic minimization code 3.1.2 is below.

**Code 3.7.1.** *MATLAB Plot Subroutine for Code 3.1.2*

```

function plotDelta(ww,iter,x1,x2,io,a1,a2,w0,...
    xmax,ymax,xmin,ymin)
xx=ww;
x    = [xmin:.01:xmax];
y    = [ymin:.01:ymax];
[X,Y] = meshgrid(xmin:.01:xmax,ymin:.01:ymax );
f     = a1*X.^2+a2*Y.^2;          % quadratic Function
imagesc(x,y,f);
hold on;
[C,h]=contourf(X,Y,f,10,'w');    % Plot quadratic function ith contours
if io==3;v=[1 2.3 -.1];
%   clabel(C,h,'color','white','Fontweight','bold');
end
hold off
hold on;
%for i = 1:iter;
    plot(xx(1:iter,1),xx(1:iter,2),'w-*');
%end
plot(xx(iter,1),xx(iter,2),'*r');
plot(ww(1,1),ww(1,2),'*k');
plot(0,0,'oy')
hold off;
xlabel('w_1','FontSize',9);
ylabel('w_2','FontSize',9);

if io==1;
    title(['b] Adam Iterates'],...
        'FontSize',9);
;end

if io==2;
    title(['a] Adadelata Iterates'],...
        'FontSize',9);
end

end
end

```

# **Part II**

## **Supervised Learning**





## Chapter 4

# Introduction to Neural Networks

Chapter 2 showed how to compute the optimal coefficients of the model matrix  $\mathbf{W}$  that minimized the sum of squared errors between the predicted data  $\mathbf{W}\mathbf{x}$  and observed data  $\mathbf{t}$ . Here, the coefficients of the vector  $\mathbf{t} \in \Re$  can be any value on the real line. These optimal coefficients are known as the least squares solution to the regression problem. In contrast, the *classification* problem seeks the model that classifies the input data  $\mathbf{x}$  into a few categories. In this case the element values of  $\mathbf{t}$  are restricted to a small set of numbers, one for each category. For the example of the binary color classification of motorcycles, the  $1 \times 1$  target vector  $\mathbf{t} = t$  is a scalar label where  $t = 1$  indicates the class of red motorcycles and  $t = 0$  indicates the class for non-red motorcycles. Unlike the regression problem, the binary labels are restricted to just two numbers, 1 and 0.

As an example of comparing linear regression with non-linear classification, Figure 4.1a plots the linear relationship (black line fitted to the blue points) between the undergraduate majors of students and their GPA values  $y$  obtained in a rigorous geoscience masters program. The GPA inferred from the best-fit linear model  $y = mx + b$  predicts that a physics major, with the test point denoted by the star, will likely achieve a high GPA in graduate school<sup>1</sup>.

In contrast, Figure 4.1b plots the training data  $(x^{(k)}, y^{(k)})$   $k \in \{1, 2, \dots, K\}$  for binary classification of a student's happiness index in graduate school as a function of their undergraduate education. Here, the index  $y = 1$  (red points) denotes a happy experience, while  $y = 0$  (blue points) denotes an unhappy experience. These binary labels are "best" fitted by the S-shaped sigmoid function

$$y = g(mx + b) = \frac{1}{1 + e^{-(mx+b)}}, \quad (4.1)$$

that predicts the happiness of a test student (star) with a background in physics. The sigmoid function is plotted in Figure 4.2a and resembles that of a smoothed Heaviside function, which "squashes" output values to be nearly zero with an input below a specified threshold, otherwise it outputs a value close to unity. This type of squashing function is

---

<sup>1</sup>From now on, we replace the target variable  $\mathbf{t}$  with the variable  $\mathbf{y}$ .

ideally suited to the binary classification problem and represents the simplest form of a single-node neural network. The sigmoid function in equation 4.1 is also known as the *logistic sigmoid function* (Bishop, 2006), which is often used for non-linear classification of categorical data. This chapter now presents the theory and practice of solving classification and regression problems by a neural network using non-linear activation functions such as the sigmoid function.

Judea Pearl (2018 Turing award): “All the impressive achievements of deep learning amount to just curve fitting”.

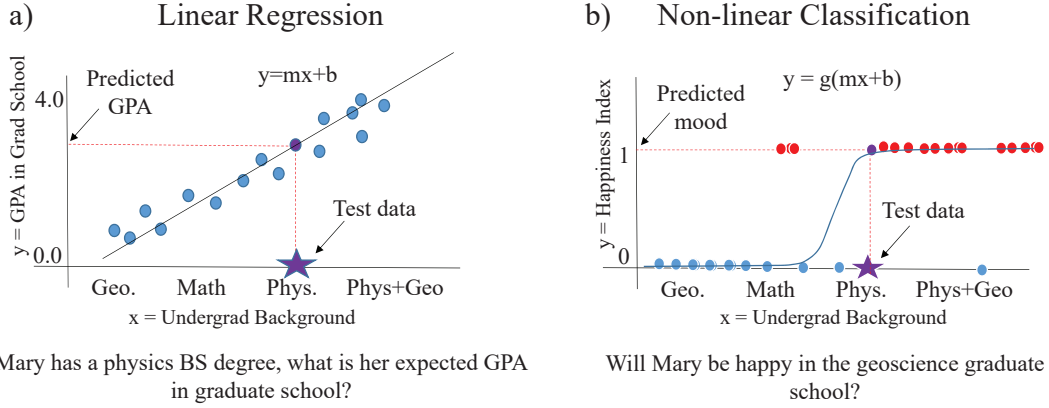


Figure 4.1: a) Examples of a line  $y = mx + b$  best fitted to  $(x, y)$  data and b) best fit of a sigmoid to categorical data  $(x, y)$ . Here, the GPA values in a) are  $0 \leq y \leq 4.0$  while in b) the categorical target data are restricted to the binary labels of happy  $y = 1$  or not happy  $y = 0$ .

## 4.1 Introduction

The problem of finding the optimal model parameters  $w$  and  $b$  that best explain the data  $y^{(k)}$  in Figure 4.1b is a simple neural network problem. Let’s solve this non-linear fitting problem (see Exercise 4.7.1) by an iterative steepest descent method. First, we define the *best solution* as  $m$  and  $b$  that minimize, e.g., the sum of squared errors:

$$\epsilon = \frac{1}{2} \sum_{k=1}^K \left( \overbrace{g(mx^{(k)} + b)}^{y^{pred}} - \overbrace{y^{(k)}}^{y^{obs}} \right)^2, \quad (4.2)$$

where the summation in  $k$  is over the indices associated with the training pairs  $(x^{(k)}, y^{(k)})$  and  $y^{pred.} = g(mx^{(k)} + b)$  is the predicted class.

In general, the input  $x$  in equation 4.2 can be a long vector  $\mathbf{x}$  that represents a flattened 2D or 3D image and the neural network can be a concatenation of matrix-vector products

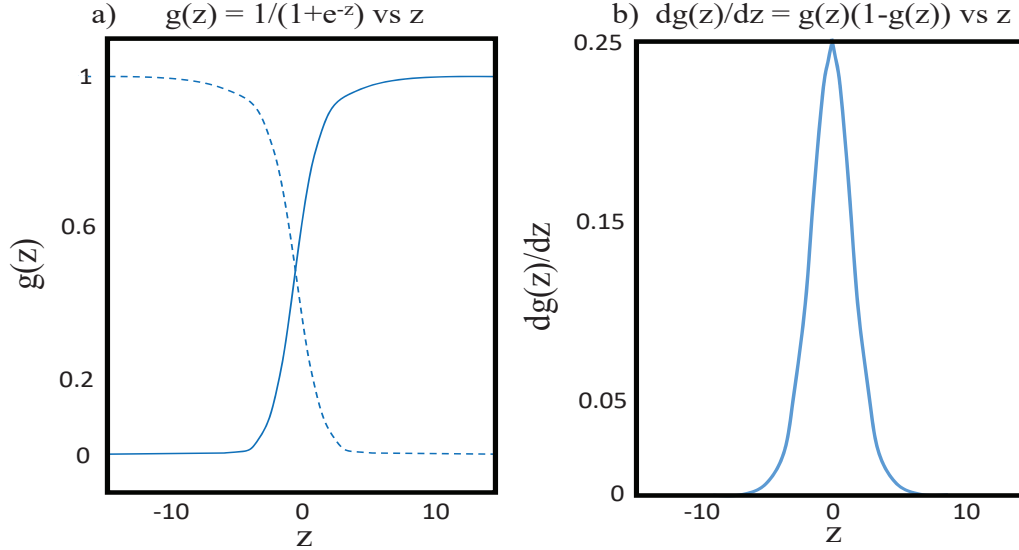


Figure 4.2: a) Sigmoid function  $\sigma(z) = g(z = wx + b) = \frac{1}{1+e^{-(wx+b)}}$  and b) its derivative for  $b = 0$  and  $w = 1$ . The dashed curve in a) is the function  $1 - g(z)$ , which is the mirror image of  $g(z)$  about  $z = 0$ . Here,  $w$  plays the role of the slope  $m$  in Figure 4.1.

and non-linear operators such as the sigmoid function. As an example, if the input  $\mathbf{x}$  is an  $(M + 1) \times 1$  vector and there is a single node (see Figure 4.3a) then the predicted output  $y^{pred.}$  in equation 4.2 is replaced by

$$y^{pred.} = g(\overbrace{\mathbf{w}^T \mathbf{x}}^z) = \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}}}, \quad (4.3)$$

where  $\mathbf{w} = (w_0, w_1, w_2, \dots, w_M)$ ,  $\mathbf{x} = (1, x_1, x_2, \dots, x_M)$ , the bias term  $b$  is  $w_0$  and

$$\frac{\partial g(\mathbf{w}^T \mathbf{x})}{\partial w_i} = g(\mathbf{w}^T \mathbf{x})(1 - g(\mathbf{w}^T \mathbf{x}))x_i \quad \text{for } i \in \{0, 1, 2, \dots, M + 1\}. \quad (4.4)$$

Here, the role of the bias term  $w_0 = b$  is to laterally shift the S-like curve to best fit the input points. We will often designate the sigmoid function in the above equations as  $\sigma(z) = g(z)$ .

More generally almost any bumpy curve  $f(x)$  with continuous derivatives can be represented by a weighted sum of sigmoids:

$$f(x) = \sum_{i=1}^I W_i \sigma(w_i x + b_i), \quad (4.5)$$

which is illustrated for  $I = 3$  in Figure 4.4a. Here, the green, blue and red sigmoids in Figure 4.4b are added together to get the bumpy curve in Figure 4.4a. The  $b_i$  terms shift the sigmoids along the x-axis, the weights  $W_i$  determine their strengths and polarities, and the larger values of  $w_i$  sharpen the *smooth jumps* in the sigmoids. The values of

$(b_i, w_i, W_i)$  are found that best fit the categorical data points denoted by brown asterisks in Figure 4.4a. This combination of parameters describes a two-layer network with three nodes in the hidden layer and one node in the last layer. The input layer is one node, sigmoid activation functions are used in the hidden layer, and the last layer is a linear layer. The next two sections present the biological inspiration and generalization of a neural network.

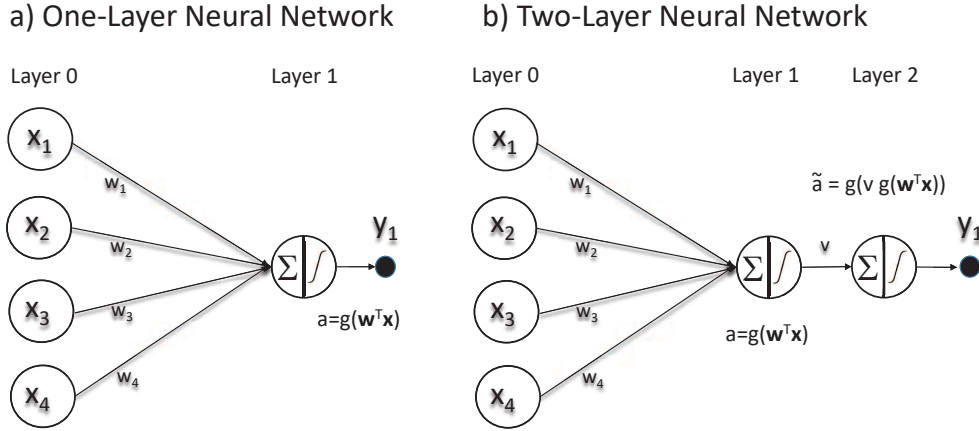


Figure 4.3: Diagrams of a) single-layer and b) two-layer neural networks, each with a single node per layer; the node for the bias terms are silent here. In a), the summation and S-like symbols in the nodes denote the matrix-vector multiplication  $\mathbf{w}^T \mathbf{x} = (w_0, w_1, w_2, w_3, w_4)^T (1, x_1, x_2, x_3, x_4)$  and the sigmoid thresholding operation  $\frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x})}$ , respectively.

#### 4.1.1 Biological Inspiration of a Neural Network

The mathematical model of the biological neuron was first described by McCulloch and Pitts (1943) as a weighted sum of input signals<sup>2</sup> inserted into a neuron. After summation of these signals, the neuron applies a suitable threshold function to give the final output signal at the axon in Figure 4.5. The thresholding function can be mathematically described by an S-shaped sigmoid function, but other threshold functions can be used as well.

There are three components to the overly simplified neuron model in Figure 4.5.

- **Dendrites.** Many different dendrite "wires" feed into a neuron, and the weight of an input signal is increased by increasing the ratio of synaptic neurotransmitters to

<sup>2</sup>These signal represent excitation or inhibitory electrochemical signals at neural dendrites ([https://en.wikipedia.org/wiki/Artificial\\_neuron](https://en.wikipedia.org/wiki/Artificial_neuron)).

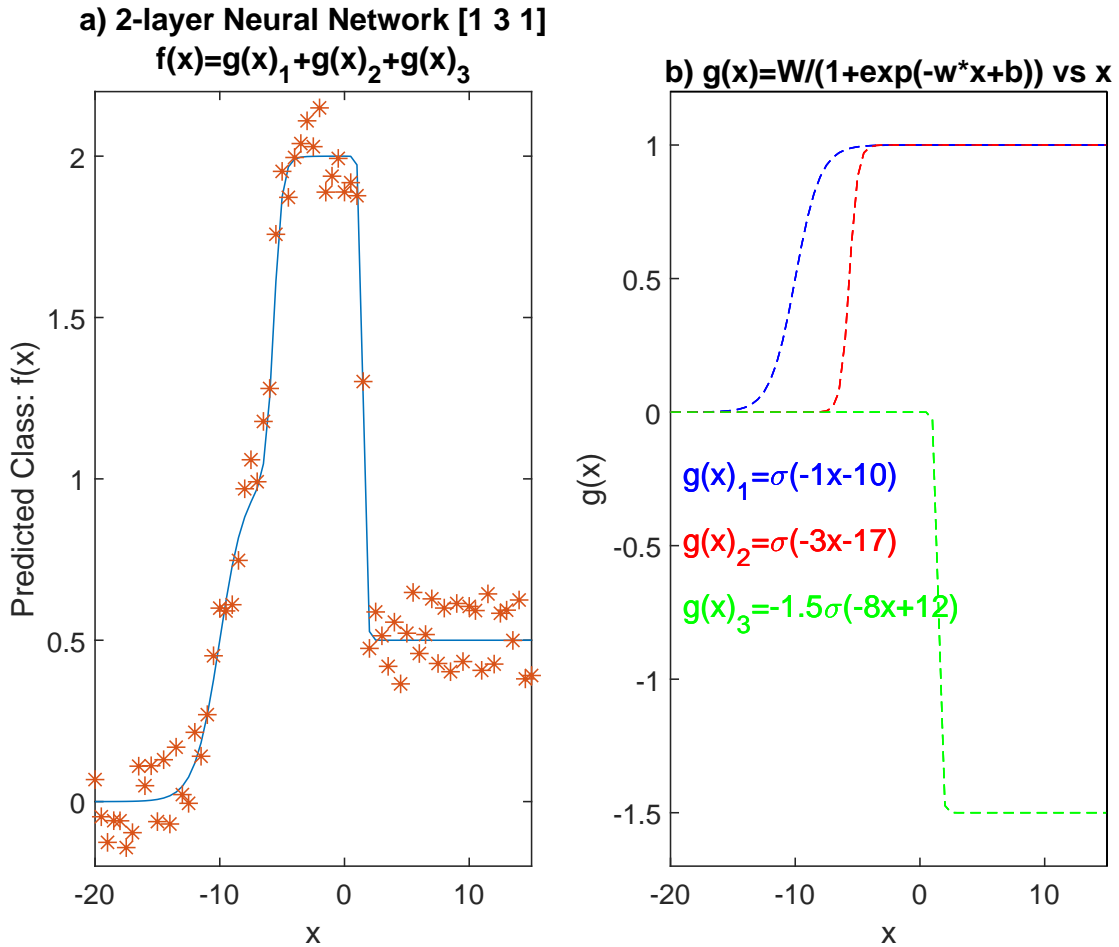


Figure 4.4: Plots of training pairs a)  $(x^{(i)}, y^{(i)})$  and the best fit  $f(x) = g(x)_1 + g(x)_2 + g(x)_3$  and b) the individual sigmoids  $g(x)_1, g(x)_2$  and  $g(x)_3$ . The curve  $g(x)_1 + g(x)_2$  can only monotonically increase, but the addition of the term  $g(x)_3$  with  $W_3 = -1.5$  in equation 4.5 allows it to decrease because  $W_3 < 0$ . This represents a two-layer neural network [1 3 1] with one node in the input layer, three nodes in the hidden layer, and one node in the output layer.

## Neuron

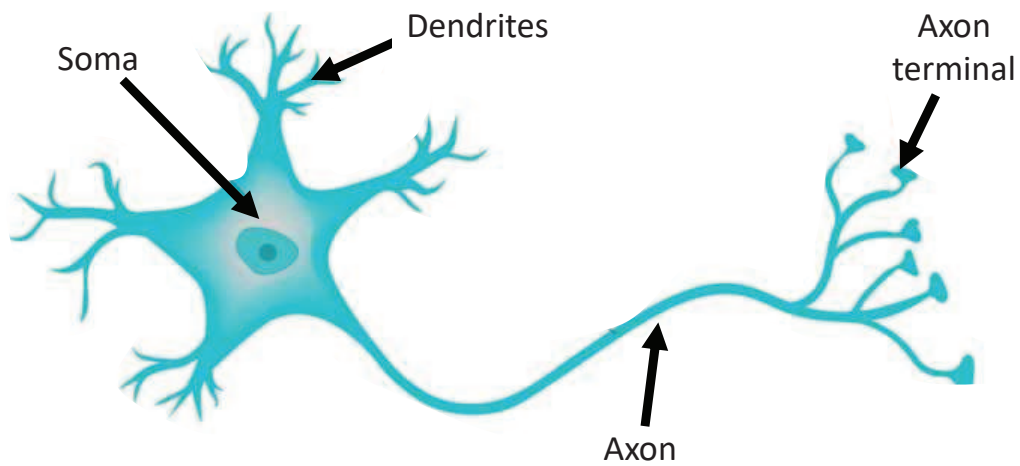


Figure 4.5: Diagram of a neuron with its axon, soma and dendrites. Credit to David Baillot/ UC San Diego and <https://medicalxpress.com/news/2018-07-neuron-axons-spindly-theyre-optimizing.html>. Recent work by Beniaguev et al. (2021) suggests that the dendritic branches of the cortical neuron can act as spatial pattern detectors. Thus, a single neuron can be simulated by a 5-layer deep neural network.

signal chemicals. The weight can be decreased by transmitting signal inhibitors (i.e. oppositely charged ions) along the dendrite.

- **Soma.** The soma in the neuron sums the weighted input signals from the connected dendrites. The positive and negative ions are mixed together in the solution inside the cell's body, which represents the matrix-vector operation in equation 4.8.
- **Axon.** The axon receives the weighted sum of signals from the soma, and once it exceeds a threshold potential, the axon will pass an "all-in" sign down its length. This signal is then passed as new input into neighboring neurons.

The above components served as a simple model of the biological neuron, but its practical use in computational science had to wait until the advent of practical computers in the 1950s. In the late 1950s, Rosenblatt (1958) presented a computational demonstration of McCulloch and Pitt's (1943) neuron model. Rosenblatt denoted it as the *perceptron* whose use "... is designed to illustrate some of the fundamental properties of intelligent systems in general... The analogy between the perceptron and biological systems should be readily apparent to the reader.". The multilayer perceptron model is largely represented by the architecture shown in Figure 4.6, also known as a multilayer neural network (<https://en.wikipedia.org/wiki/Perceptron>) when used for more general tasks.

The one-layer perceptron in Figure 4.3a can be an accurate *linear classifier* if the input data are linearly separable. Here, linearly separable points  $\mathbf{x}^{(k)}$  means that the points labeled as  $y = 0$  can be separated by a hyperplane (denoted as a line if  $\mathbf{x}^{(k)} = (x_1^{(k)}, x_2^{(k)})$ ) from those points labeled as  $y = 1$ . However, a one-layer perceptron will not be able to accurately label a training set if the points are not linearly separable (<https://en.wikipedia.org/wiki/Perceptron>). In this case, a multilayer neural network with two or more layers and a sufficient number of nodes/layer<sup>3</sup> is used to find the high-dimensional manifold that accurately separates the two classes of points. See sections 4.1.3-4.1.4.

#### 4.1.2 General Neural Network

A neural network (NN) is a non-linear mathematical model

$$f(\mathbf{x}^{(k)}, \mathbf{W}) \approx \mathbf{y}^{(k)}, \quad (4.6)$$

that maps the  $k^{th}$  input data  $\mathbf{x}^{(k)}$  to its expected (or true) output  $\mathbf{y}^{(k)}$ , where  $\mathbf{W}$  represent the model parameters to be determined. A simple example is the classification problem in Figure 4.1b where

$$\mathbf{W} = (b, m); \quad \mathbf{x} = (1, x), \quad (4.7)$$

and  $\mathbf{y} = (y)$ . Here, a 1 is incorporated into the vector  $\mathbf{x}$  so that the  $1 \times 2$  matrix  $\mathbf{W}^T \rightarrow \mathbf{w}^T = (b \ m)$  is multiplied by the  $2 \times 1$  vector  $\mathbf{x} = (1, x)$  to get  $\mathbf{w}^T \mathbf{x} = wx + b$  so that the predicted class label is  $\sigma(\mathbf{w}^T \mathbf{x})$ .

---

<sup>3</sup>The dimension  $M$  of the output of the first layer is equal to the number  $M$  of nodes in that layer. For an output with a higher dimension manifold then the number of nodes in the first layer must exceed the dimension of the input vector  $\mathbf{x}$  for a single-layer NN.

In general, the mathematical operations of  $f(\mathbf{x}^{(k)}, \mathbf{W})$  are a recursive sequence of matrix-vector multiplications and non-linear thresholding operations applied to the input data  $\mathbf{x}^{(k)}$  to predict  $\mathbf{y}$ . The input-output combination  $(\mathbf{x}^{(k)}, \mathbf{y}^{(k)})$  is known as a training pair and a large training set of pairs is used to train the network for the model parameters  $\mathbf{W}$ . Once properly trained, the neural network (NN) can be used to accurately predict  $f(\mathbf{x}^{(k)}, \mathbf{W}) \approx \mathbf{y}$  from a never-before-seen input  $\mathbf{x}$ . This prediction operation from  $\mathbf{x}$  is known as inference<sup>4</sup>.

If the matrix (or matrices)  $\mathbf{W}$  is (are) densely populated, then the NN is known as a *fully-connected neural network* (FCNN). On the other hand, if the matrix-vector operation  $\mathbf{W}$  represents a sparsely populated convolution matrix<sup>5</sup> then this known as a convolutional neural network (CNN). This chapter and Chapters 5-6 exclusively discuss FCNNs, and Chapters 9-12 present the theory and practice of CNNs.

The architecture of a three-layer<sup>6</sup> neural network is illustrated in Figure 4.6. Each node represents two operations: matrix-vector multiplication

$$z_i^{[n]} = \sum_{j=1}^J w_{ij}^{[n]} a_j^{[n-1]}, \text{ weighted sum of inputs } a_j^{[n-1]} \text{ at } i\text{th node of } n\text{th layer} \quad (4.8)$$

and the operation of a non-linear *squashing* function

$$a_i^{[n]} = g(z_i^{[n]}) \text{ activation function at } i\text{th node of } n\text{th layer}. \quad (4.9)$$

Here, the superscript denoting the example index is kept silent. The upper limit  $J$  in the summation is the number of nodes in the  $(n-1)^{th}$  layer and  $a_i^{[n]}$  is the  $i$ th activation component of the activation vector  $\mathbf{a}^{[n]}$  in the  $n^{th}$  layer. The bias term is contained within the matrix associated with  $w_{ij}$  and the vector  $\mathbf{x}$  is adjusted to incorporate this bias as demonstrated in equation 4.7. Each vertical column of nodes in which computations are performed is known as a layer, and the ones between the first and last columns of nodes are known as *hidden layers*. The input column of nodes is technically not a layer because its elements are specified, not computed, so it will be denoted as the zeroth-layer with the input  $\mathbf{a}^{(0)} = \mathbf{x}$ .

The neural network solution can now be succinctly defined as finding the model weights  $w_{ij}^{[n]}$  that minimize the, for example,  $L^2$  objective function:

$$\epsilon = 1/2 \sum_i \sum_k \left( \overbrace{g(z_i^{[N](k)})}^{\text{predicted}} - \overbrace{y_i^{(k)}}^{\text{true}} \right)^2 + \lambda \text{ regularization term}, \quad (4.10)$$

<sup>4</sup>Bishop (2006) defines the inference stage as learning the model from training data. However, inference is often defined as making a prediction from the trained model (<https://www.intel.com/content/www/us/en/artificial-intelligence/posts/deep-learning-training-and-inference.html>). We will use this later definition of inference.

<sup>5</sup>For a spatial convolution of two 1D signals, each row of the periodic convolution matrix is a shifted copy of the previous row. In other words, each row represents a convolution filter that is spatially invariant.

<sup>6</sup>There does not seem to be uniform agreement on how to count the number of layers, some authors, e.g. Duda et al. (2000), count the input nodes, aka units or neurons, as a layer and so refer to Figure 4.6 as a four-layer network. Our counting notation will be consistent with Bishop (2006) and not count the input layer.



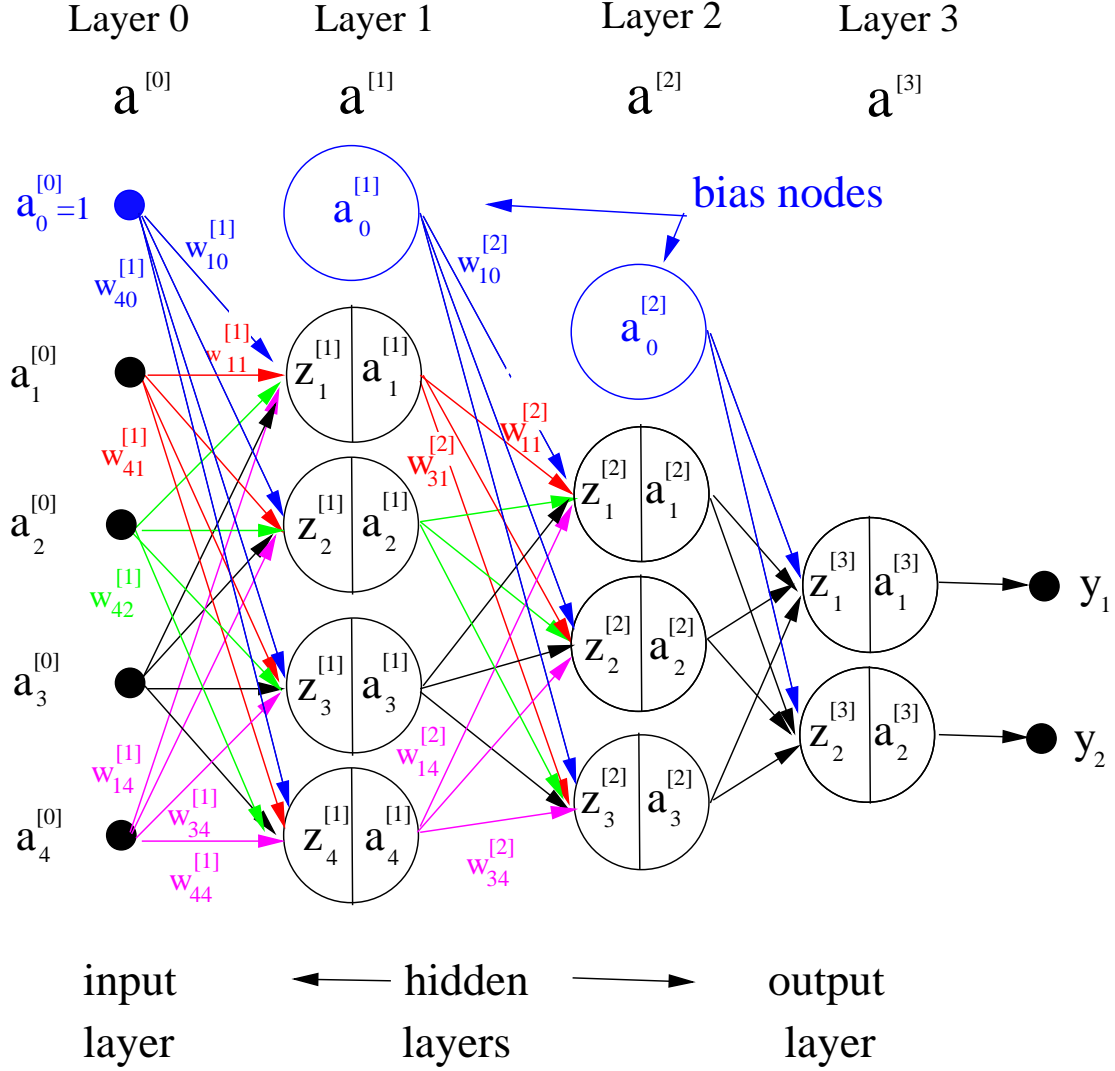


Figure 4.6: Notation for a three-layer, i.e.  $N = 3$ , neural network with four input nodes at the input layer. The activation output from the  $i^{th}$  node in the  $n^{th}$  layer is denoted as  $a_i^{[n]} = g(z_i^{[n]})$ , where  $z_i^{[n]} = \sum_j w_{ij}^{[n]} a_j^{[n-1]}$ ; note, the superscript  $[n-1]$  for the activation component is one less integer than the superscript  $[n]$  for the weight  $w_{ij}^{[n]}$ . The input data  $\mathbf{x}$  are represented by the activation vector  $\mathbf{a}^{[0]}$  for the  $0^{th}$  layer and the predicted data are denoted by the  $2 \times 1$  activation vector  $\mathbf{y} = \mathbf{a}^{[3]}$  at the last layer. The layers between the first (zereth layer) and last columns (third layer) of nodes are denoted as hidden layers. The bias vector for the 1st layer is  $(w_{10}^{[1]}, w_{20}^{[1]}, w_{30}^{[1]}, w_{40}^{[1]})$ .

where  $\lambda > 0$  is the regularization parameter and the summation in  $k$  is over the indices of the training pairs. The summation in  $i$  is over the number of nodes in the output layer and, for the  $k^{th}$  input example,  $a_i^{[N](k)} = g(z_i^{[N](k)})$  is the predicted output at the  $i^{th}$  node of the  $N^{th}$  layer, also referred to as the output layer. Here, the superscript  $(k)$  in  $\mathbf{z}^{[N](k)}$  denotes the data sample index, but it will often be made silent to avoid notation clutter.

The pseudo-code for the  $L^2$  iterative solution by steepest descent without regularization is given in Box 4.1.1.

**Code 4.1.1.** *MATLAB Pseudo-code for Minimizing Equation 4.10 by Steepest Descent*

Uniform distribution of random starting weights:  $w_{ij}^{[n]} \forall i, j, n$ ,

Starting residual  $(r_i^{(k)}) = g^{[N]}(z_i^{[N](k)}) - y_i^{(k)} \forall i, k$ ,

Starting objective function  $\epsilon$  & step length  $\alpha$

```

for it = 1 : niter           Loop over iterations
    for n = 1 : N           Loop over layers
        for i = 1 : I       Loop over weight index i
            for j = 1 : J    Loop over weight index j
                 $w_{ij}^{[n]} := w_{ij}^{[n]} - \alpha \frac{\partial \epsilon}{\partial w_{ij}^{[n]}}$ 
            end
        end
    end
end
Update  $\mathbf{r}^{(k)}$ ,  $\alpha$  and  $\epsilon$ 
end
```

(4.11)

where  $:=$  indicates that the variable  $w_{ij}^{[n]}$  on the left is reset to the value given on the righthand side. Here, the objective  $\epsilon$  is an implicit function of the iteration index  $it$  because its residual is updated at each iteration, and  $\alpha$  is the step length at iteration  $it$ . Chapter 6 derives recursive formulas for the efficient computation of the gradient in the Box 4.1.1 pseudo-code. In practice, the data are typically too large so the data are subdivided into mini-batches, and the gradient and model update for a mini-batch are computed in one iteration. Then the updated model parameters  $w_{ij}^{[n]}$  are used as the starting model for the next mini-batch. This procedure is known as stochastic gradient descent (SGD). See section 8.

### 4.1.3 Geometrical Interpretation of a Neural Network

The residual  $a_i^{[N]} - y_i$  in equation 4.10 says that the *true* component  $y_i$  in the last layer is predicted by the  $i^{th}$  component of the activation function  $a_i^{[N]} = \sigma(z_i^{[N]})$ , where  $g(z) = \sigma(z)$  is taken to be a sigmoid function with the scalar input  $z$ ; as promised, the example superscript  $(k)$  superscript is made silent. According to equation 4.8, the input

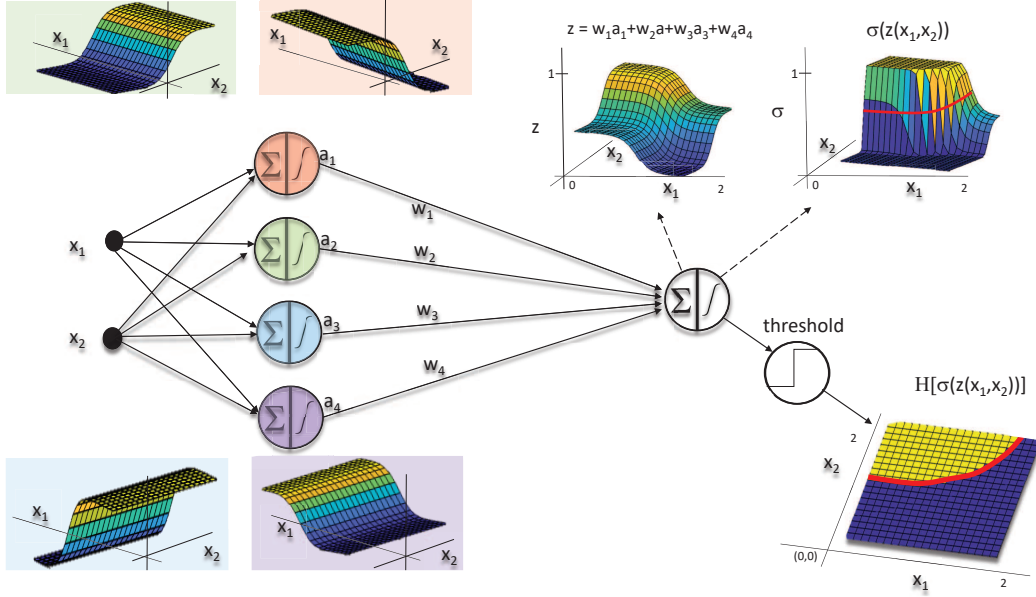


Figure 4.7: Two-layer network with a  $2 \times 1$  input vector  $\mathbf{x}$  and 4 nodes in the hidden layer. In this example the activation functions are sigmoids and the bias nodes are ignored. The hard-threshold symbol is the Heavyside ideogram within the lower-right circle.

$z_i^{[N]}$  into this activation function is the weighted sum of the activation components  $a_j^{[N-1]}$  from the  $N - 1$  layer, as illustrated in Figure 4.7 for  $N = 2$  in the two-layer example. If  $a_j^{[N-1]} \rightarrow a_j$  is a sigmoid function then the linear algebra interpretation of equation 4.7 is that the vector  $z = \sum_{j=1}^4 w_j a_j$  is a weighted sum of  $J = 4$  sigmoids, which act as basis functions in the estimation of the output vector (Duda et al., 2000). The geometrical orientation of each plotted sigmoid function in Figures 4.7-4.8 depends on the learned weights  $w_{ij}^{[2]} \rightarrow w_j$  in their arguments.

Each of the sigmoid functions in Figure 4.7 plot with a different orientation as described by either  $\sigma(x_1)$ ,  $\sigma(-x_1)$ ,  $\sigma(x_2)$ , or  $\sigma(-x_2)$ . Weighting the outputs of the sigmoids and summing them up gives the  $z = \sum_{i=1}^4 w_i a_i$  plot in the top middle, and applying the soft-threshold operation  $\sigma(z)$  gives the graph of the output prediction  $y^{pred.} = \sigma(z(x_1, x_2))$  at the top right. In this classification example, the output should ideally be either  $y = 1$  or  $y = 0$ , so a threshold is set along the red curve, which is the decision boundary where  $\sigma(z) = 0.5$ . The projection of the decision boundary onto the  $x_1 - x_2$  plane is plotted in the lower right plot of Figure 4.7. Here, the hard-threshold function is defined as  $H(\sigma(z)) = 1$  if  $\sigma(z) > 0.5$ , otherwise  $H(\sigma(z)) = 0$ . Any input sample in the yellow area is predicted to be class type 1, otherwise it is 0.

The shape of the red decision boundary can be even more complex by including more nodes in the hidden layer and/or more layers, where the sigmoid plots of many different orientations can be used to form decision boundaries of almost arbitrary shape. See Exercise 4.7.5.

If the output layer has more than two nodes then there will be many classes and many

different decision boundaries, one for each class. More generally, section 4.2 presents different types of activation functions that can serve as basis functions to reconstruct the output. For example, Sitzmann et al. (2019) use sine activation functions

$$\sigma(z_i) = \sin\left(\sum_j w_{ij}x_j + b_i\right), \quad (4.12)$$

for interpolation of input data by a neural network where, in this case,  $x_j$  represent the components of the coordinates at each point of the image. Interpolating data by a NN is a regression problem where training is performed on sparsely sampled data and the output is finely sampled data, as illustrated in Figure 14.20. For inpainting, the input data can be missing or be damaged as in Figure 14.2, and the output is the full data set (Pathak et al., 2016). For sine activation functions, the weights  $w_{ij}$  can be interpreted as components of wavenumber vectors and the bias term  $b_i$  is the component of a phase-shift vector. A benefit of this representation compared to non-smooth activation functions (see section 4.2) is that the sine basis functions are differentiable everywhere so that, once the NN is trained, all spatial derivatives of the data can be computed analytically. Derivatives exist for the sigmoid function, but according to the results in Sitzmann et al. (2019), *their derivatives are often not well behaved and also fail to represent fine details. The sine functions provide better resolved features compared to many other types of activation functions.*

#### 4.1.4 Generality of a Neural Network

Is there a generalization of neural networks that can represent any continuous mapping from  $\mathbf{x}$  to  $f(\mathbf{x})$  in equation 4.6? The answer is yes as Kolomogorov proved (Duda et al., 2000). Specifically, Kolomogorov proved that any continuous function  $f(\mathbf{x})$  can be related to the input  $\psi_{ij}(x_i)$  by

$$f(\mathbf{x}) = \sum_{j=1}^{2n+1} \tilde{\sigma}\left(\sum_{i=1}^d \psi_{ij}(x_i)\right), \quad (4.13)$$

where  $d$  is the dimension of the input vector  $\mathbf{x}$ ,  $n \geq 2$  for the dimension of the hypercube  $I^n$  ( $I = [0, 1]$ ) and  $\psi$  and  $\tilde{\sigma}$  are properly chosen functions. This formula resembles that of a neural network for two or more layers if  $\tilde{\sigma}$  is replaced by a sigmoid activation function.

Unfortunately,  $\tilde{\sigma}$  and  $\psi$  in the above equation are complicated functions that do not resemble the smooth activation function of a sigmoid. However, Duda et al. (2000) present intuitive arguments that suggest that the sum of 4 or more sigmoids in Figure 4.7 can reconstruct a wide variety of decision curves for a 2-layer NN. In fact, changing the coefficients  $w_i$  in the first layer gives rise to island-like decision curves of almost any shape, even the Gaussian output shown in Figure 4.9 which is computed by the MATLAB code in Exercise 4.7.5. This implies that a NN with two or more layers with a sufficient number of nodes per layer is able to reconstruct many different continuous mappings  $f(\mathbf{x}, \mathbf{W}) = \mathbf{y}$  between the input  $\mathbf{x}$  and target  $\mathbf{y}$ . In practice, we typically need many nodes per layer and more than two layers to get useful results from real-data examples.

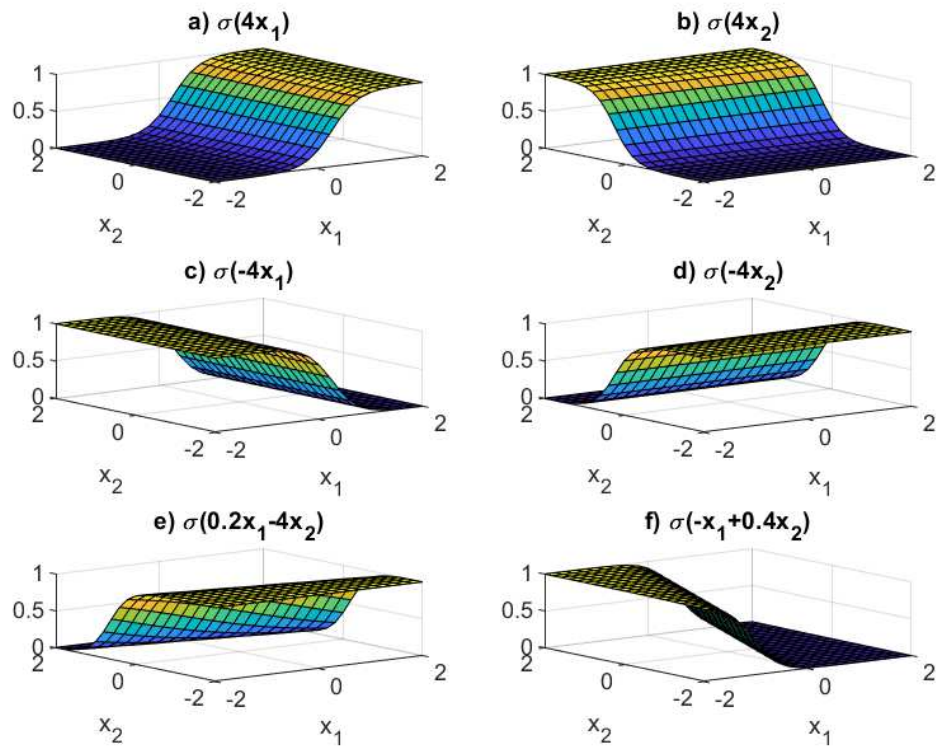


Figure 4.8: Six different sigmoids, each oriented along a different azimuthal direction in the  $x_1 - x_2$  plane. The weights of the NN are the numbers multiplying  $x_1$  and  $x_2$  in the argument of the sigmoid.

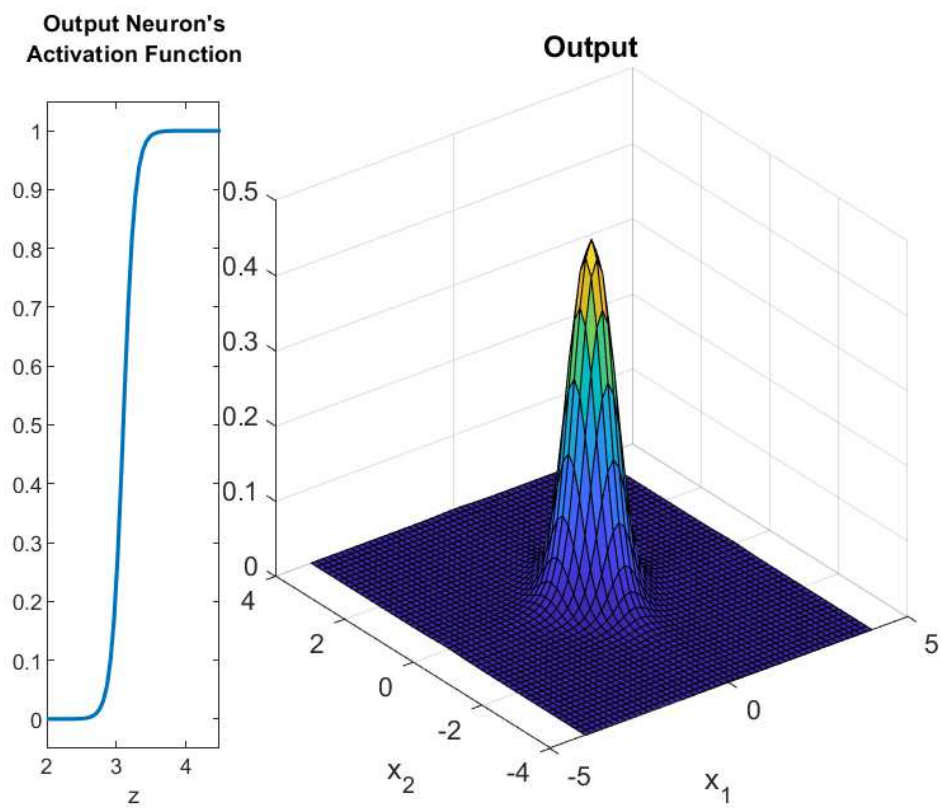


Figure 4.9: Gaussian curve constructed from the 2-layer NN in Figure 4.7.

### 4.1.5 Neural Network Workflow

The workflow for computing the coefficients  $\mathbf{W}$  of the neural network model is illustrated in Figure 4.10 and described in Appendix 4.9. If the target vectors  $\mathbf{y}$  are class labels for the input data  $\mathbf{x}$ , then the main steps for the workflow of a supervised neural network are the following. The labeled data are divided into two parts: 80% for training and 20% for validation and testing.

1. **Training.** Use the training data to estimate the optimal values of the network's coefficients in  $\mathbf{W}$  so the loss function is minimized. How many training examples are needed? The more the better as long as the variety of training examples represent any examples in the unseen and unlabeled data for supervised classification. Ng and Jordan (2011) cite a rule-of-thumb that the number of training examples should be at least the same number of unknown parameters in the NN model. A gradient descent method is the optimization algorithm.
2. **Validation and Testing.** The accuracy of the trained NN is refined by applying it to labeled validation data and tuning the hyperparameters. If acceptable accuracy is achieved then apply the NN to a remaining batch of unused labeled data for testing its accuracy.
3. **Inference.** Predict the output for completely new data which are unlabeled.

### 4.1.6 Recursive Feed-forward Formula

The input vector  $\mathbf{x} = \mathbf{a}^{[0]}$  in Figure 4.6 is inserted on the left, and the forward modeling operations are represented by the recursive feed-forward operations

$$\mathbf{y}^{pred.} = g(\mathbf{W}^{[N]}\mathbf{a}^{[N-1]}), \quad (4.14)$$

$$\begin{aligned} &= g(\mathbf{W}^{[N]}g(\mathbf{W}^{[N-1]}\mathbf{a}^{[N-2]})), \\ &\vdots \\ &= g(\mathbf{W}^{[N]}g(\mathbf{W}^{[N-1]}\dots g(\mathbf{W}^{[1]}\mathbf{a}^{[0]})\dots)), \end{aligned} \quad (4.15)$$

where, for convenience, we assume the same type of activation functions  $g(\cdot)$  for each layer and the bias terms are incorporated into the  $\mathbf{W}^{[n]}$  matrices. In the case where the activation functions  $g(\cdot)$  are different at each layer then a layer superscript can be appended to the notation  $g(\cdot) \rightarrow g(\cdot)^{[n]}$ . The activation function acts on each element of a  $J \times 1$  vector  $\mathbf{z}^{[n]} = \mathbf{W}^{[n]}\mathbf{a}^{[n-1]}$  to return the vector  $\mathbf{a}^{[n]} = g(\mathbf{W}^{[n]}\mathbf{a}^{[n-1]})$ .

The feed-forward operations in equation 4.15 can be computed in an efficient manner, as will be shown in Chapter 6 for a multi-layer multi-node NN. It starts by computing  $g(\mathbf{W}^{[1]}\mathbf{a}^{[0]})$  in the first layer, then reuses this result for the input to the calculations of the second layer. This procedure is repeated until the last layer is reached to efficiently compute the predicted output  $\mathbf{y}^{pred.}$ .

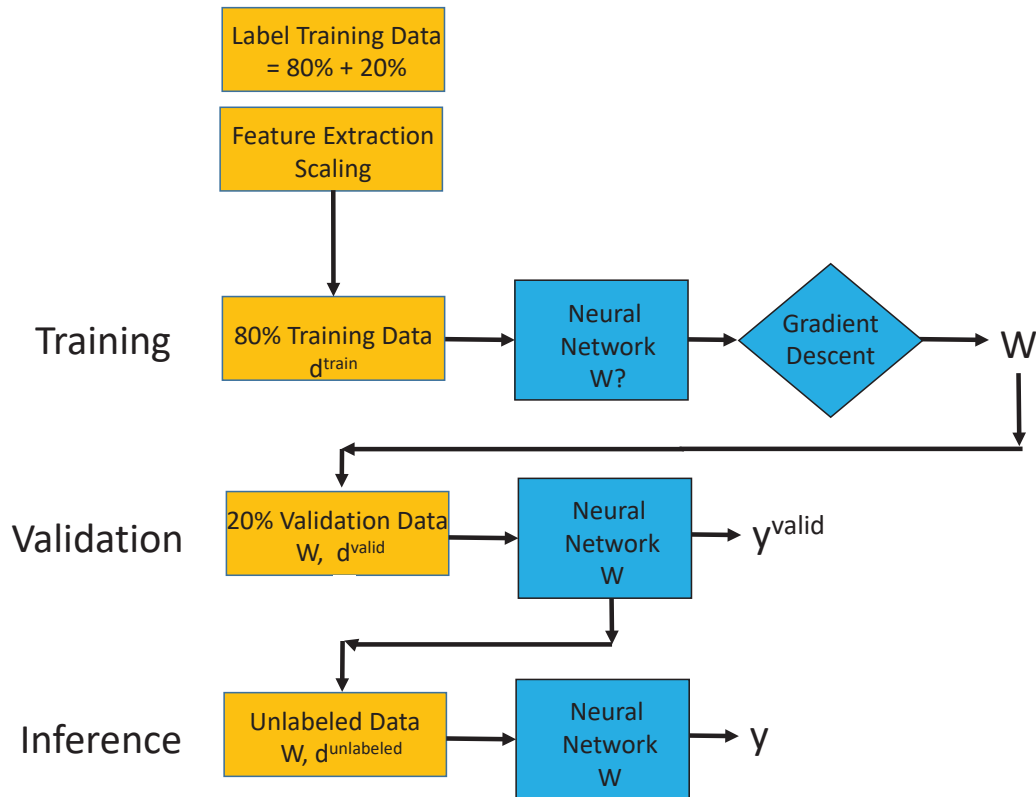


Figure 4.10: Workflow for supervised learning of a neural network for classification of the input data  $\mathbf{x}^{(k)}$ . The training data consist of the labeled training pairs  $(\mathbf{x}^{(k)}, y^{(k)})$ .



### 4.1.7 Recursive Back-propagation Formula

For a long chain of single-node layers where  $\mathbf{W}^{[n]} \rightarrow w^{[n]}$ , each with the unknown weight  $w^{[n]}$  for  $n = \{1, 2, \dots, N\}$ , equations 4.14-4.15 become

$$\begin{aligned}
 y^{pred.} &= g(\overbrace{w^{[N]} a^{[N-1]}}^{z^{[N]}}, \\
 &= g(w^{[N]} g(\overbrace{w^{[N-1]} g(w^{[N-2]} a^{[N-3]})}^{z^{[N-1]}})) , \\
 &= g(w^{[N]} g(w^{[N-1]} (g(w^{[N-2]} (\dots w^{[2]} g(\overbrace{w^{[1]} a^{[0]}}^{z^{[1]}}) \dots))))), \tag{4.16}
 \end{aligned}$$

where we have used the recursive definitions

$$z^{[n]} := w^{[n]} a^{[n-1]} = w^{[n]} g(z^{[n-1]}); \tag{4.17}$$

$$g^{[n]} := g(z^{[n]}) = g(w^{[n]} a^{[n-1]}); \tag{4.18}$$

and will use the identities

$$\frac{\partial g(z^{[n]})}{\partial z^{[n]}} = (1 - g^{[n]}) g^{[n]}; \tag{4.19}$$

$$\begin{aligned}
 \frac{\partial g(z^{[n]})}{\partial w^{[n]}} &= \frac{\partial g(z^{[n]})}{\partial z^{[n]}} \frac{\partial z^{[n]}}{\partial w^{[n]}}, \\
 &= (1 - g^{[n]}) g^{[n]} g^{[n-1]}. \tag{4.20}
 \end{aligned}$$

Here, the input data are defined as  $x = a^{[0]} = g^{[0]}$  at layer 0 and we assume  $g(z)$  is the sigmoid so that  $\frac{\partial g(z^{[n]})}{\partial z^{[n]}} = (1 - g^{[n]}) g^{[n]}$ .

For a chain of single-nodes, the gradient components of the composite function  $\epsilon = \frac{1}{2}(y^{pred.} - y)^2$  are given by

$$\frac{\partial \epsilon}{\partial w^{[n]}} = \overbrace{\frac{\partial \epsilon}{\partial y^{pred.}}}^{g(z^{[N]})-y} \overbrace{\frac{\partial y^{pred.}}{\partial w^{[n]}}}^{Frechet\ derivative} \quad n \in \{1, 2, \dots, N\}, \tag{4.21}$$

where the Fréchet derivative characterizes the sensitivity of the predicted output  $y^{pred.}$  to the change in the model parameter  $w^{[n]}$ . The chain rule<sup>7</sup> can be applied to the derivatives

---

<sup>7</sup>If  $f(t) = f(h(t))$  is a composite function, then the chain rule says that  $\frac{\partial f}{\partial t} = \frac{\partial f}{\partial h} \frac{\partial h}{\partial t}$ .

of equation 4.16 w/r to the layer weights  $w^{[N]}$  and  $w^{[N-1]}$  to get the Fréchet derivative

$$\begin{aligned} \frac{\partial y^{pred.}}{\partial w^{[N]}} &= \overbrace{\frac{\partial g(z^{[N]})}{\partial z^{[N]}}}^{n=N \text{ in eqn. 4.19}} \overbrace{\frac{\partial z^{[N]}}{\partial w^{[N]}}}^{g^{[N-1]}}, \\ &= (1 - g^{[N]})g^{[N]} g^{[N-1]}. \end{aligned} \quad (4.22)$$

$$\begin{aligned} \frac{\partial y^{pred.}}{\partial w^{[N-1]}} &= \overbrace{\frac{\partial g(w^{[N]}g^{[N-1]})}{\partial w^{[N]}g^{[N-1]}}}^{n=N \text{ in eqn. 4.19}} \overbrace{\frac{\partial w^{[N]}g^{[N-1]}}{\partial g(z^{[N-1]})}}^{w^{[N]}} \overbrace{\frac{\partial g(z^{[N-1]})}{\partial w^{[N-1]}g^{[N-2]}}}^{n=N-1 \text{ in eqn. 4.19}} \overbrace{\frac{\partial w^{[N-1]}g^{[N-2]}}{\partial w^{[N-1]}}}^{g^{[N-2]}}, \\ &= (1 - g^{[N]})g^{[N]} w^{[N]} (1 - g^{[N-1]})g^{[N-1]} g^{[N-2]}. \end{aligned} \quad (4.23)$$

We can see the development of a pattern so that the Fréchet derivative for the  $N - 2$  layer is

$$\frac{\partial y^{pred.}}{\partial w^{[N-2]}} = (1 - g^{[N]})g^{[N]}w^{[N]} \cdot (1 - g^{[N-1]})g^{[N-1]}w^{[N-1]} \cdot (1 - g^{[N-2]})g^{[N-2]}g^{[N-3]}. \quad (4.24)$$

An inductive argument shows that the derivative w/r to the weight at the  $N - k$  layer is

$$\begin{aligned} \frac{\partial y^{pred.}}{\partial w^{[N-k]}} &= (1 - g^{[N]})g^{[N]}w^{[N]} (1 - g^{[N-1]})g^{[N-1]}w^{[N-1]} \dots \\ &\quad (1 - g^{[N-k+1]})g^{[N-k+1]}w^{[N-k+1]} (1 - g^{[N-k]})g^{[N-k]}g^{[N-k-1]}. \end{aligned} \quad (4.25)$$

Therefore, the Fréchet derivatives  $\frac{\partial y^{pred.}}{\partial w^{[n]}}$  for  $n \in \{N, N - 1, N - 2, \dots, 1\}$  can be recursively computed by first computing  $\frac{\partial y^{pred.}}{\partial w^{[N]}}$  with equation 4.22 for the  $N^{th}$  layer, retrieving  $w^{[N]}$  from memory, and storing the value of  $(1 - g^{[N]})g^{[N]}w^{[N]}$ . Then reuse it for the computation of  $\frac{\partial y^{pred.}}{\partial w^{[N-1]}}$  in equation 4.23, retrieving  $w^{[N-1]}$  from memory, and storing  $(1 - g^{[N-1]})g^{[N-1]}w^{[N-1]}$ . This procedure can be recursively repeated until  $\frac{\partial y^{pred.}}{\partial w^{[1]}}$  is computed with equation 4.25 for  $k = N - 1$  and  $g^{[0]} = x$ . If each layer has more than one node, then the scalar multiplications are replaced by the matrix-vector multiplications described in Chapter 6.

In summary, recursive back-propagation starts at the output layer to compute the Fréchet derivative w/r  $w^{[N]}$ , and then progressively computes the Fréchet derivatives at shallower layers until the input layer is reached. This is opposite to the feed-forward operations, where we start at the input layer and progressively compute  $z^{[n]}$  at deeper layers until we reach the deepest layer where  $n = N$ . The computational cost of computing the Fréchet derivatives is  $O(NM)$  multiplications, where we assume  $O(M)$  operations at each link of the chain. If recursion is not used then the computational count is  $O(N^2M)$  operations to compute all the Fréchet derivatives.

The recursion formulas that progressively update the weights from the output data layer

to the shallowest input layer are similar to those for an iteration of least squares migration (LSM). For LSM (Schuster, 2017), the seismic data residual  $\Delta \mathbf{d}$  is back-propagated from the recording layer at the surface to deeper layers. This back-propagation operator is represented by the matrix  $\mathbf{L}^T$ , and the reflectivity weights updated at each interface are denoted by  $\mathbf{m} = \mathbf{L}^T \Delta \mathbf{d}$ . The operator  $\mathbf{L}^T$  is also known as the migration operator, which is the adjoint of the forward modeling operator  $\mathbf{L}$ .

## 4.2 Activation Functions

The sigmoid function in Figure 4.11a was one of the earliest activation functions used for neural networks. It is also used to describe a NN in terms of probability theory as discussed in Box 4.2.1.

**Key Idea 4.2.1. Key Idea Box: Sigmoid Function and Probability Theory**

The sigmoid function  $g(z)$  has the two main properties of a binary probability mass distribution:  $0 \leq g(z) \leq 1$  and it sums to 1 for the two outcomes  $y = 0$  and  $y = 1$  (see Exercises 4.7.10-4.7.11). For example, define  $p$  as the probability that, given the input predictor variable  $\mathbf{x}$ , it predicts the binary response variable of  $y = 1$ , referred to as *true*. Similarly the false response of  $y = 0$  has probability  $1 - p$ . The log-odds of this outcome is defined as

$$\text{odds} = \ln\left[\frac{p}{1-p}\right], \quad (4.26)$$

which is the logarithm of the ratio of the true and false probabilities. For example, assume that an unfair flipped coin is highly likely to land on a head. Therefore,  $p = 0.9$  and the denominator is  $1 - p = 1 - 0.9 = 0.1$  so that the odds  $p/(1 - p)$  of a head are 10 times that of a tail.

Setting  $\text{odds} = \mathbf{w}^T \mathbf{x}$  says that the outcome probability of the experiment is a function of the weighted input vector  $\mathbf{x}$ . For example,  $\mathbf{w}^T \mathbf{x}$  might contain information on how asymmetrical the coin might be that would affect the outcome of a coin flip. This function can be derived by setting  $\text{odd} = \mathbf{w}^T \mathbf{x}$  and exponentiating equation 4.26 to give

$$\frac{p}{1-p} = e^{\mathbf{w}^T \mathbf{x}}, \quad (4.27)$$

which can be solved for  $p$  to give

$$p = \frac{e^{\mathbf{w}^T \mathbf{x}}}{e^{\mathbf{w}^T \mathbf{x}} + 1} = \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}}}. \quad (4.28)$$

This is the same formula as the sigmoid function in equation 4.3. Therefore the sigmoid function can now be interpreted as the probability of the predictor variable  $\mathbf{x}$  being true for the target variable  $y = 1$ . If  $\mathbf{x}$  is a random variable then the conditional probability of it having the response  $y = 1$  is

$$p(y = 1 | \mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}}}, \quad (4.29)$$

where  $\mathbf{w}$  is the model parameter vector. If the target variable is  $y = 0$  then

$$\begin{aligned} p(y = 0 | \mathbf{x}, \mathbf{w}) &= 1 - p(y = 1 | \mathbf{x}, \mathbf{w}), \\ &= 1 - \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}}} = \frac{1}{1 + e^{+\mathbf{w}^T \mathbf{x}}}. \end{aligned} \quad (4.30)$$

However, the concatenated sequence of back-propagated sigmoid functions  $dg(z)/dz$  tended to produce outputs that peaked around  $z = 0$  and vanished for  $|z| > 0$  with deep (14 or more layers) neural networks (DNN). Consequently, updating the weights deep into the network was minimal and so prevented convergence to useful solutions (Bengio et al., 1994). This motivated the creation of more effective activation functions, especially the *rectified linear unit*  $ReLU(z)$  (Maas et al., 2013; Goodfellow et al., 2016; Brownlee, 2019):

$$ReLU(z) = \max(0, z) = \begin{bmatrix} 0 & \text{if } z < 0 \\ z & \text{if } z \geq 0. \end{bmatrix}, \quad (4.31)$$

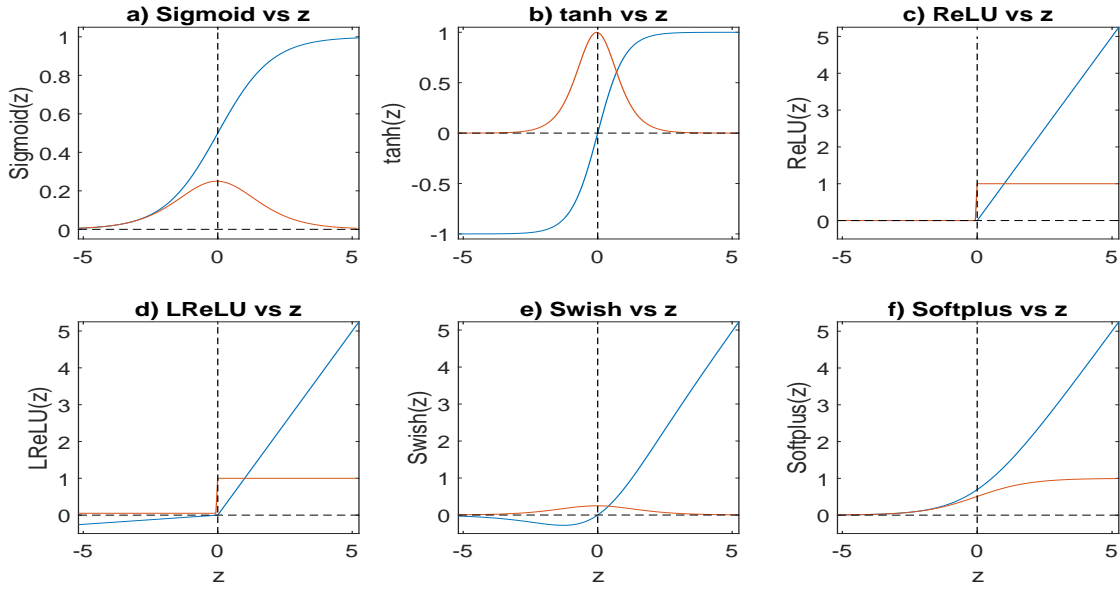


Figure 4.11: Activation functions (blue curves) and their derivatives (orange curves). The derivative of the ReLU function (orange line) in c) is a constant for  $z > 0$ , and zero otherwise. In comparison, the leaky ReLU in d) can output non-zero values for negative input values, and thereby avoids the dying ReLU problem (Doshi, 2019) and keeps some neurons activated.

plotted as the blue solid curve in Figure 4.11c. The derivative of the ReLU (orange curve in Figure 4.11c) is  $\frac{\partial \text{ReLU}(z)}{\partial z} = 0$  for  $z < 0$  and  $\frac{\partial \text{ReLU}(z)}{\partial z} = 1$  if  $z > 0$ , so the gradient is a constant for  $z > 0$ . This can significantly remedy the *vanishing gradient problem* with deep sigmoid-based networks and lead to substantially faster convergence (Zeiler et al., 2013). A mathematical rationale for networks with ReLU activation functions is that they promote solutions with sparse models under the  $L^1$  norm (see Chapter 24).

Unfortunately, neural networks with ReLU can also lead to the *exploding gradient problem* in, for example, recurrent neural networks in Chapter 12. The ad hoc partial remedy is to simply clip the gradient if it exceeds a specified threshold (Goodfellow et al., 2016).

Other activation functions and their derivatives are plotted in Figure 4.11. For example, the leaky ReLU (LReLU) in Figure 4.11d is an improved version of the ReLU function that is defined as

$$\text{LReLU}(z) = \max(\alpha z, z) = \begin{cases} \alpha z & \text{if } z < 0 \\ z & \text{if } z \geq 0. \end{cases}, \quad (4.32)$$

where  $0 < \alpha \leq 1$ . The leaky ReLU provides a small non-zero gradient for negative values of  $z$ , which can activate most of the neurons for learning even if  $z$  has negative polarity. In contrast, the standard ReLU has a zero gradient for  $z < 0$  so some neurons are unnecessarily deactivated. However, the leaky ReLU in some cases does not perform significantly better than ReLU (Maas et al., 2013).

A close cousin to leaky ReLU is the softplus activation function in Figure 4.11f (Nwankpa

et al., 2018)

$$g(z) = \ln(1 + e^z), \quad (4.33)$$

whose derivative  $dg(z)/dz$  is the sigmoid function. The merit of this function is that its derivative is continuous, but it still can lead to the exploding gradient problem because its output becomes large if the input is large. The multivariate version of softplus is

$$g(z_1, z_2, \dots, z_N) = \ln(1 + e^{z_1} + e^{z_2} + \dots + e^{z_N}), \quad (4.34)$$

where its derivative becomes the softmax function in equation 4.55.

The hyperbolic tangent or *tanh* function Figure 4.11b (Nwankpa et al., 2018)

$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}, \quad (4.35)$$

is similar in shape to the sigmoid function, except it is perfectly anti-symmetrical such that  $g(z) = -g(-z)$ . Some prefer it over the sigmoid because it sometimes gives better convergence (Karlick et al., 2011) for natural language processing (Dauphin et al., 2017) and speech recognition (Maas et al., 2013), and the gradients are not restricted to move in only one direction. However, it still suffers from the vanishing gradient problem of the sigmoid function for deep networks (Nwanka et al., 2018).

A smoothed version of the sigmoid function is the Swish function in Figure 4.11e discovered by Google researchers (Ramachandran et al., 2017):

$$g(z) = \frac{z}{1 + e^{-z}}, \quad (4.36)$$

which is as computationally efficient as ReLU and shows better performance than ReLU on deeper models (<https://medium.com/@neuralnets/swish-activation-function-by-google-53e1ea86f820>). Note, its derivative exists everywhere, even at  $z = 0$ . Google reports that its experiments "train deeper Swish networks than ReLU networks when using BatchNorm (Ioffe and Szegedy, 2015) despite having gradient squishing property. Up to 40 iterations with the input as the MNIST data set, both Swish and ReLU networks were both comparable in convergence rate. However, Swish outperforms ReLU by a large margin in the range between 40 and 50 layers when optimization becomes difficult. In very deep networks, Swish achieves higher test accuracy than ReLU."

### 4.3 Examples of Neural Networks

We will now present several examples of neural network architectures for solving simple classification problems. The equations for the objective functions and gradients of single and double hidden layer NNs will now be defined.

### 4.3.1 Single-Layer Neural Network

The solution to the fitting problem illustrated in Figure 4.1b finds the optimal  $2 \times 1$  model vector  $\mathbf{w} = (w_0, w_1) = (b, w)$  that minimizes a specified objective function  $\epsilon$ . For a single-node NN, this problem can be generalized to a  $M \times 1$  input vector  $\mathbf{x}$  so that  $\mathbf{w}$  is also the same dimension  $M \times 1$  and incorporates the bias term  $b$ . In this case,  $N = 1$ ,  $g(z_i^{[N](k)}) \rightarrow g(\mathbf{w}^T \mathbf{x}^{(k)})$  and  $y_i^{(k)} \rightarrow y^{(k)}$  so the  $L^2$  misfit function in equation 4.10 becomes

$$\epsilon = \frac{1}{2} \sum_{k=1}^K (g(\mathbf{w}^T \mathbf{x}^{(k)}) - y^{(k)})^2; \quad (4.37)$$

and

$$\frac{\partial g(\mathbf{w}^T \mathbf{x}^{(k)})}{\partial w_i} = \frac{\partial g(\mathbf{w}^T \mathbf{x}^{(k)})}{\partial \mathbf{w}^T \mathbf{x}^{(k)}} \frac{\partial \mathbf{w}^T \mathbf{x}^{(k)}}{\partial w_i} = (1 - g(\mathbf{w}^T \mathbf{x}^{(k)})) g(\mathbf{w}^T \mathbf{x}^{(k)}) x_i^{(k)}, \quad (4.38)$$

where the last equation follows from equation 4.20 for  $\frac{\partial z^{[n]}}{\partial w_i}|_{n=1} = \frac{\partial \mathbf{w}^T \mathbf{x}^{(k)}}{\partial w_i} = x_i^{(k)}$ . Therefore, the steepest descent formula for the single-node network is

$$\begin{aligned} w_i^{(l+1)} &= w_i^{(l)} - \alpha \sum_{k=1}^K (g(\mathbf{w}^T \mathbf{x}^{(k)}) - y^{(k)}) \overbrace{\frac{\partial g(\mathbf{w}^T \mathbf{x}^{(k)})}{\partial w_i}}^{\text{eq. 4.38}} \\ &= w_i^{(l)} - \alpha \sum_{k=1}^K \overbrace{g(\mathbf{w}^T \mathbf{x}^{(k)}) (1 - g(\mathbf{w}^T \mathbf{x}^{(k)})) x_i^{(k)}}^{\frac{\partial r^{(k)}}{\partial w_i}} \overbrace{(g(\mathbf{w}^T \mathbf{x}^{(k)}) - y^{(k)})}^{r^{(k)} = k^{th} \text{ residual}}, \end{aligned} \quad (4.39)$$

where  $\mathbf{w}$  in the arguments of  $g$  is the weight vector  $\mathbf{w}^{(l)}$  computed at the  $l^{th}$  iteration. As a test case, Example 4.3.1 uses equation 4.39 to solve the binary classification problem with a 1-layer neural network.

A regularization term  $\lambda(w_1, w_2, \dots)^T(w_1, w_2, \dots)$  might be added to the objective function in order to reshape an undesirable objective function with a long valley in Figure 2.6b into the desirably round one in Figure 2.6a. Here, the components  $(w_1, w_2, \dots)$  control the shape of the contours, so regularization adjusts their values accordingly. *In contrast, the bias component  $w_0$  should not be included in regularization because it only shifts the contours associated with the objective function* (see Exercises 4.7.7 and 4.7.14).

**Example 4.3.1.** *Binary Classification with a 1-Layer Neural Network*

A MATLAB code is used to compute the 1-layer NN solution to the following binary classification problem. There are 100 training pairs of labeled data  $(\mathbf{x}, y)$  where  $\mathbf{x} = (1, x_1, x_2, x_3, x_4, x_5)$  is a  $6 \times 1$  vector with  $x_i \in \{0, 1\}$  for  $i = \{1, 2, 3, 4, 5\}$ . The first term of  $\mathbf{x}$  is  $x_0 = 1$  because it accounts for the bias term  $b$ , where  $\mathbf{w} = (b, w_1, w_2, w_3, w_4, w_5)$  for the single-node NN model in Figure 4.3a. The target output is the binary classification value of  $y = 1$  if  $x_i = 0$  for  $i = \{1, 2, 3, 4, 5\}$ , otherwise  $y = 0$ . An example of 100  $5 \times 1$  data vectors  $(x_1, x_2, x_3, x_4, x_5)$  is displayed in Figure 4.12.

One hundred training examples are used to train the network to give the results shown in Figure 4.13. The predicted values  $y^{pred.} = g(\mathbf{w}^T \mathbf{x})$  in Figure 4.13b are never equal to 1 for the first 400 iterations. One might consider this to be a failure, except with many more iterations the solution converges to the correct predictions. However, the correct classifications can be determined by noticing that any  $y^{pred.}$  above the threshold value of 0.34 might be considered as the class denoted as  $y = 1$ . With this threshold constraint there is an almost perfect accuracy in the predicted class shown in Figure 4.13c. Here, the actual class is a red star and the predicted class (after the threshold constraint) is a green star. If the class predictions are correct then the red stars should be completely hidden by green stars, which is hardly the case.

Why is it possible that a 1-layer NN can correctly classify the input data? Assume that the input data are  $3 \times 1$  vectors  $(x_1, x_2, x_3)$  where  $x_i \in \{0, 1\}$ . In this case, the vertices at the purple and red dots in Figure 4.14 represent all 8 possible  $3 \times 1$  input vectors. The  $y = 1$  point  $\mathbf{w} = (0, 0, 0)$  is at the origin, and it can clearly be separated from the other vertices by the slanted red triangle. Thus, these points are linearly separable and they can be separated by a single-node network with a sigmoid activation. See Exercise 4.7.4 for the extension of this argument to  $5 \times 1$  input vectors.

**4.3.2 Two-layer Neural Network**

In general, the one-node neural network cannot emulate complex relationships between the input data vectors and the target vectors (Bishop, 2006). However, multinode and multilayer neural networks have been found to be universal approximators (Hornik et al., 1989; Hornik, 1991; Ripley, 1996).

The unknown model parameters for the two-layer network in Figure 4.3 are  $\mathbf{w}$  and  $v$  in the first and second layers, respectively. The Fréchet derivatives for the two-layer network were derived in equations 4.22-4.24, where the input data is the scalar  $x$  so there is only one model parameter  $w$  in the first layer. In contrast, the Figure 4.3 model assumes that the input is the  $4 \times 1$  vector  $\mathbf{x} = (x_1, x_2, x_3, x_4)$  and so  $\mathbf{w} = (w_1, w_2, w_3, w_4)$ . Using equation 4.17



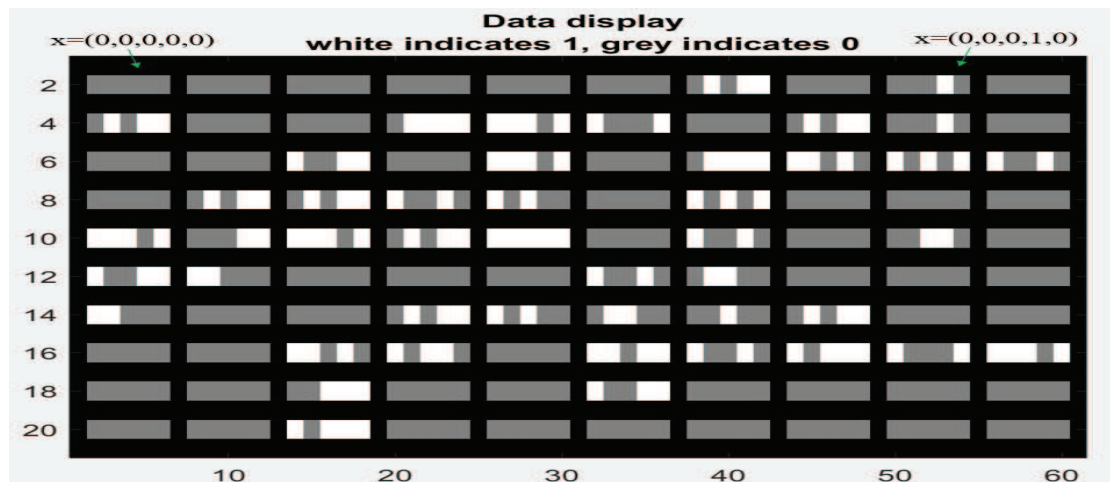


Figure 4.12: Images of 100  $5 \times 1$  data vectors  $(x_1, x_2, x_3, x_4, x_5)$  where gray denotes the number 0 and white denotes the number 1. Each data vector is labeled as  $y = 1$  if  $(x_1, x_2, x_3, x_4, x_5) = (0, 0, 0, 0, 0)$ , otherwise it is labeled  $y = 0$ .

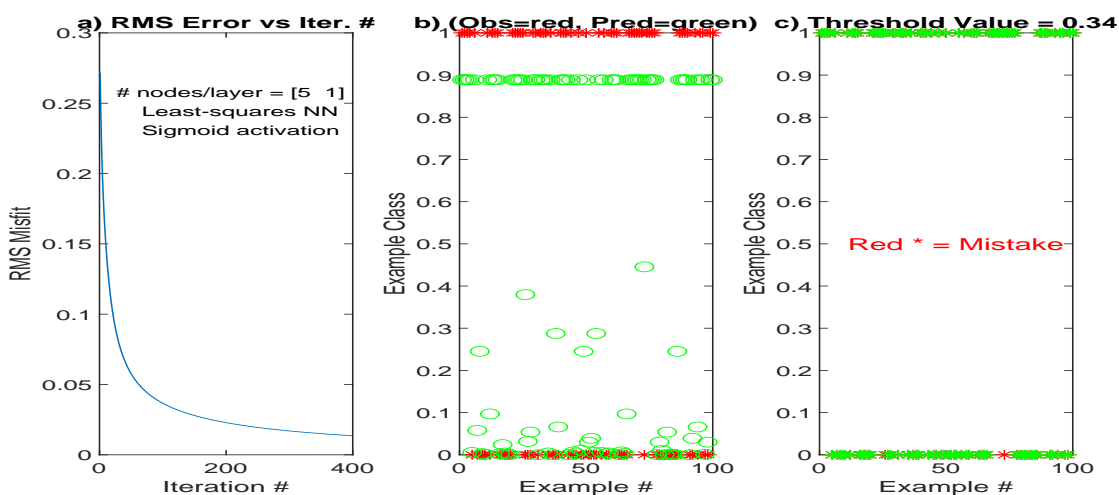


Figure 4.13: a) RMS error versus iteration number, b) predicted (green) versus actual (red) classes after 400 iterations, and c) predicted binary classes after a hard threshold of 0.29. The numbers in  $[5 \ 1]$  indicate the number of nodes/layer, and the NN architecture is characterized by a sigmoid activation function and a  $L^2$  misfit function.

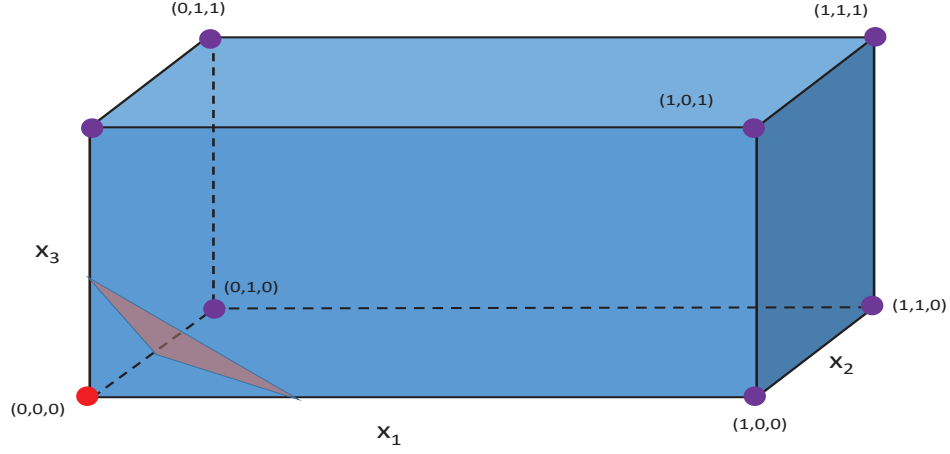


Figure 4.14: Each vertex represents one of the eight possible examples with the  $3 \times 1$  vector  $\mathbf{x} = (x_1, x_2, x_3)$ , where  $x_i \in \{0, 1\}$ . The red plane represents one of the decision planes that separates  $(0, 0, 0)$  at the red point from the other vertices.

and  $N = 2$  we get

$$w^{[2]} = v; \quad z^{[2]} = vg(\mathbf{w}^T \mathbf{x}); \quad \frac{\partial z^{[2]}}{\partial w^{[2]}} \rightarrow \frac{\partial vg(\mathbf{w}^T \mathbf{x})}{\partial v} = g(\mathbf{w}^T \mathbf{x}). \quad (4.40)$$

In this case the gradient for the model parameter  $v$  can be obtained from equations 4.16-4.21 to get

$$\frac{\partial \epsilon}{\partial v} = \sum_{k=1}^K r^{(k)} \overbrace{g(\theta^{(k)})(1 - g(\theta^{(k)}))g(z^{(k)})}^{\frac{\partial r^{(k)}}{\partial v}}; \quad (4.41)$$

$$(4.42)$$

$$\text{where } \theta^{(k)} = g(\mathbf{w}^T \mathbf{x}^{(k)})v \text{ and } r^{(k)} = g(\theta^{(k)}) - y^{(k)}. \quad (4.43)$$

Similarly, the components  $\partial \epsilon / \partial w_i$   $i \in \{1, 2, 3, 4\}$  in the first layer can be obtained from equation 4.23 to give

$$\frac{\partial \epsilon}{\partial w_i} = \sum_{k=1}^K r^{(k)} g(\theta^{(k)})(1 - g(\theta^{(k)}))g(z^{(k)})(1 - g(z^{(k)}))v x_i^{(k)}. \quad (4.44)$$

Equations 4.41-4.44 are now used to solve the binary classification problem posed in Example 4.3.1. The details for these computations are described in Example 4.3.2.

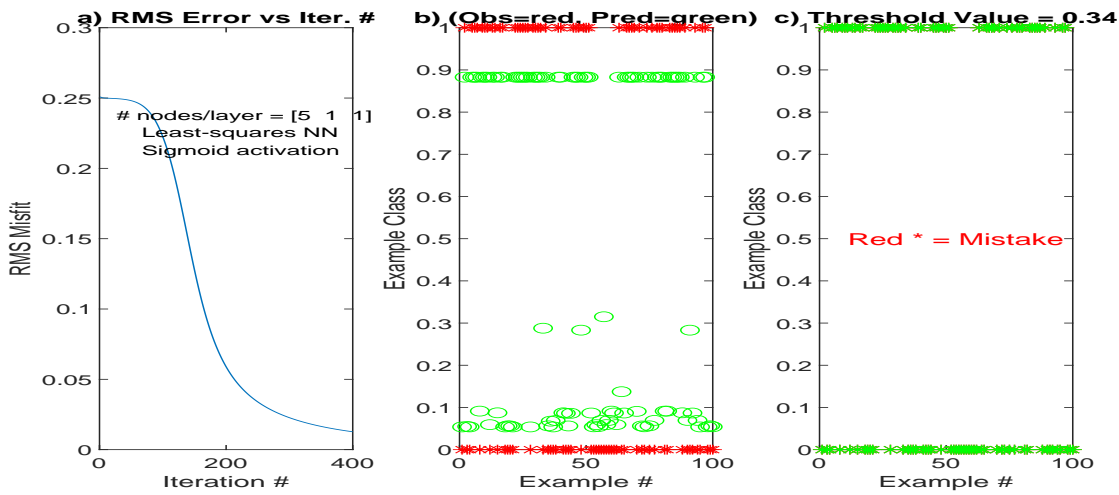


Figure 4.15: Same as Figure 4.13, except we now use the neural network with the  $[5 \ 1 \ 1]$  architecture characterized by the two single-node layers in Figure 4.3b. The MATLAB code in `/LAB1/Chapter.Book.FCN/Chapter.Gradient.Neural/lab.html` is used to compute these results.

#### Example 4.3.2. Binary Classification with a 2-Layer Neural Network

The two-layer network in Figure 4.3b is used to solve the binary classification problem posed in Example 4.3.1. Figure 4.15 shows the results which reveal a greater density of green circles below the class value of 0.2 (which approximates  $y = 0$ ) than seen in the single-layer results in Figure 4.13. However, the convergence rate of the single-node NN is somewhat faster than that of the two-node network. This is consistent with our empirical tests where an increase in the number of nodes often requires more iterations for convergence.

### 4.3.3 Neural Network MATLAB Code

The results in Figures 4.13 and 4.15 are computed by the pseudo-code in Code 4.3.1.

**Code 4.3.1.** *MATLAB Pseudo-code for a Neural Network*

```

MATLAB Pseudo-code for Neural Network

% x(N,M) - input- M input feature vectors with size Nx1
% t(1,M) - input- M input target vectors with size 1x1
% t= 1 or 0
%layer_size- input- # of layers
%obj_option- input- 1=12 & 2=x-entropy
%act_option- input- 1=sigmoid & 2=ReLU
% lambda - reg. param
% ww - input- random starting Nx1 weight vector for each node
M=100; % # of equations constraint
N=5; % # of unknowns (w1, w2, w3, w4, w5) for 1st node !modified by Zongcai
x=zeros(N,M);t=zeros(1,M);
layer_size=[N,1,1];
obj=2;act=2;
[x,t,alpha,lim,ww,layer_size,obj,act]=DataIn1(M,N,x,t,obj,act,layer_size); % Create training data
nit=400;res=zeros(nit,1);
layer_num=numel(layer_size); %layer number include the input and output

%%
%x(:,1:50)=zeros(5,50);t(1:50)=0;
thresh=.34;
%%
ww_old=ww;
for k=1:nit % Looping over iterations
    alpha=1; % step size (orig, case: 1)
    [grad,res(k)]=gradientnn(M,x,t,ww_old,layer_size,obj,act);
    for ilayer=1:layer_num-1
        ww{ilayer}=ww_old{ilayer}-alpha*grad{ilayer};
    end
    %
    [~,res1]=gradientnn(M,x,t,ww,layer_size,obj,act);
    %-----
    % Bisection line search for step length
    while (res1>res(k)) && (alpha>lim)
        alpha=alpha*0.5; % halve value of current step length
        for ilayer=1:layer_num-1
            ww{ilayer}=ww_old{ilayer}-alpha*grad{ilayer}; % updated gradient with 1/2 step length
        end
        [~,res1]=gradientnn(M,x,t,ww,layer_size,obj,act);
    end
    ww_old = ww; %%% update successful
    %-----
end
Display1(M,res,nit,ww,layer_size,x,t,obj,act,thresh)
print -depsc Fig2.balanced.TwoNode.Xentropy.ReLU.eps

```

**Bisection Line Search.** The bisection line-search method (see code between the dashed lines in Code 4.3.1) compares the value of the objective function  $\epsilon(\mathbf{w}^{(k+1)})$  at  $\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \alpha \nabla \epsilon(\mathbf{w}^{(k)})$  to  $\epsilon(\mathbf{w}^{(k)})$  at the  $k^{th}$  iterate. If  $\epsilon(\mathbf{w}^{(k+1)})$  is larger than  $\epsilon(\mathbf{w}^{(k)})$  then it halves the value of  $\alpha$  and compares again until  $\epsilon(\mathbf{w}^{(k+1)})$  at the new iterate is less than that at the previous iterate. Our empirical tests suggest that a quick-and-dirty bisection line search leads to much faster convergence compared to a fixed-step length.

#### 4.3.4 NN with a Cross-Entropy Objective Function

Instead of the least squares objective function, the neural network community often uses the cross-entropy objective function for solving the binary classification problem. Empirical evidence suggests it has better convergence properties compared to the least squares approach. In fact, it can be shown that minimizing the binary cross-entropy for the logistic regression is a convex problem (Loiseau, 2020). This means that the objective function has just one minimum, which is the global one. See Exercise 4.7.24.

The starting point for the cross-entropy method ([https:// rdipietro.github.io/ friendly-intro-to-cross-entropy-loss/](https://rdipietro.github.io/friendly-intro-to-cross-entropy-loss/)) is relate the sigmoid function to the binary probability distri-

bution (see Box 4.2.1). Here, the sigmoid function  $0 \leq g(z) \leq 1$  is the likelihood function for the binary random variable  $y$ :

$$p(y|\mathbf{x}, \mathbf{w}) = \left( \overbrace{g(\mathbf{w}^T \mathbf{x})}^{\text{probability } y=1} \right)^y \left( \overbrace{1 - g(\mathbf{w}^T \mathbf{x})}^{\text{probability } y=0} \right)^{1-y}, \quad (4.45)$$

which says that, given  $\mathbf{w}$  and the random variable  $\mathbf{x}$ , the probability for the binary random variable  $y = 1$  is  $g(\mathbf{w}^T \mathbf{x})$ , and for  $y = 0$  it is  $1 - g(\mathbf{w}^T \mathbf{x})$ . Finding the optimal  $\mathbf{w}$  that maximizes  $p(y|\mathbf{x}, \mathbf{w})$  is known as the maximum likelihood solution (Bishop, 2006).

Maximizing equation 4.45 is equivalent to finding  $\mathbf{w}$  that minimizes the cross-entropy objective function:

$$\epsilon = -y \ln g(\mathbf{w}^T \mathbf{x}) - (1 - y) \ln(1 - g(\mathbf{w}^T \mathbf{x})), \quad (4.46)$$

which is derived by taking the natural logarithm of equation 4.45 and multiplying the result by  $-1$ . Since the logarithm is a monotonic function of its argument then the optimal  $\mathbf{w} \rightarrow \mathbf{w}^*$  that maximizes the likelihood function in equation 4.45 also minimizes  $\epsilon$  in equation 4.46.

If we have  $K$  training pairs of independent random variables  $(\mathbf{x}^{(k)}, y^{(k)})$  then the likelihood of  $K$  training pairs predicting the labels  $(y^{(1)}, y^{(2)}, \dots, y^{(K)})$  is the product  $\prod_{k=1}^K p(y^{(k)}|\mathbf{x}^{(k)}, \mathbf{w})$  of the individual likelihood functions. Taking the negative logarithm of this product and dividing by  $K$  gives a weighted summation of cross-entropy error functions:

$$\epsilon = -\frac{1}{K} \sum_{k=1}^K [y^{(k)} \ln g(\mathbf{w}^T \mathbf{x}^{(k)}) + (1 - y^{(k)}) \ln(1 - g(\mathbf{w}^T \mathbf{x}^{(k)}))], \quad (4.47)$$

where  $p(y^{(k)}|\mathbf{x}^{(k)}, \mathbf{w}) \rightarrow g(\mathbf{w}^T \mathbf{x}^{(k)})$ .

What is advantage of the cross-entropy objective function compared to the  $L^2$  misfit function? The short answer is that empirical tests often show that a NN with cross-entropy objective functions often has faster convergence for solutions of classification problems. A plausibility argument that is consistent with these empirical results is shown in Figure 4.16 by plotting the cross-entropy and  $L^2$  objective functions for a) one data point with  $y = 1$  and b) two data points with  $y = 1$  and  $y = 0$ . Here, the blue cross-entropy curves rise much more steeply<sup>8</sup> than those for the  $L^2$  objective function, so the gradients of the cross-entropy functions are larger. Therefore, the iterative solutions to the cross entropy objective function can be expected to have faster convergence than those for the  $L^2$  objective function in this example.

The gradient for the cross entropy function is similar to that in equations 4.39 for the

---

<sup>8</sup>The steeper rise in  $\ln \sigma(z)$  is expected because  $0 \leq \sigma(z) \leq 1$  so that  $-\ln \sigma(z)$  inflates a small range of numbers between  $\sigma(z) = 10^{-22} \approx 0$  and  $\sigma(z) = 1$  to be between 50 and 0. In comparison, squaring  $0 \leq \sigma(z) \leq 1$  keeps the same range between 0 and 1.

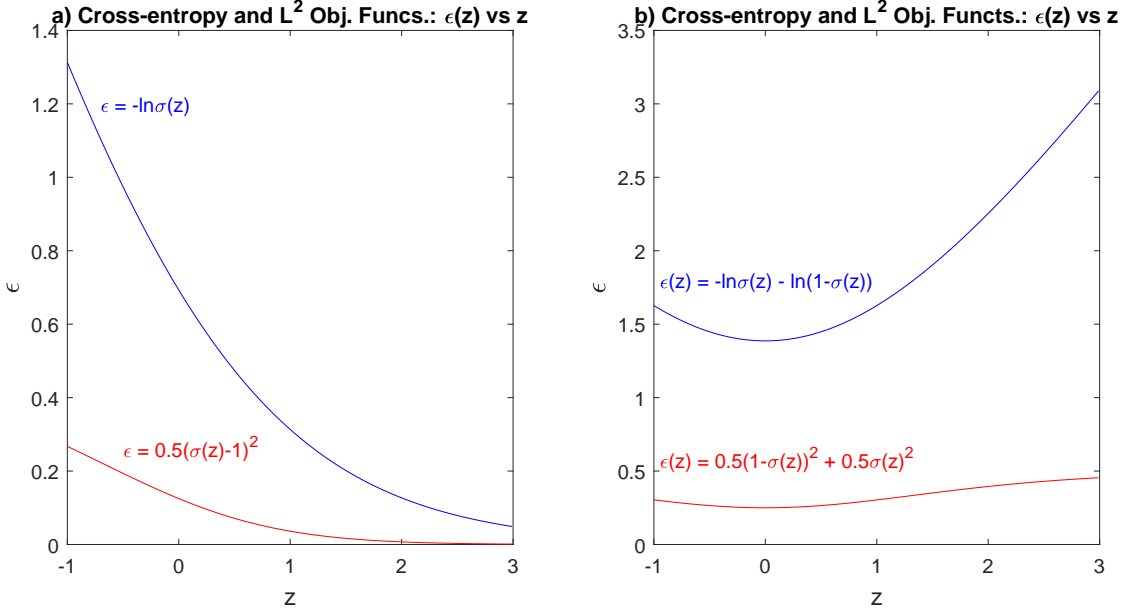


Figure 4.16: Objective functions plotted for  $L^2$  misfit function cross-entropy functions for a) one data point with the target value  $y = 1$  and b) two data points with  $y = 1$  and  $y = 0$ . The slope for the cross-entropy function is much steeper than the maximum slope in a).

least squares objective function, except

$$\begin{aligned}
 \frac{\partial \epsilon}{\partial w_i} &= -\frac{1}{K} \sum_{k=1}^K \left( \frac{y^{(k)}}{g(\mathbf{w}^T \mathbf{x}^{(k)})} - \frac{1 - y^{(k)}}{1 - g(\mathbf{w}^T \mathbf{x}^{(k)})} \right) \overbrace{g(\mathbf{w}^T \mathbf{x}^{(k)})(1 - g(\mathbf{w}^T \mathbf{x}^{(k)}))}^{\frac{\partial g(\mathbf{w}^T \mathbf{x})}{\partial w_i} = \frac{\partial g(\mathbf{w}^T \mathbf{x})}{\partial \mathbf{w}^T \mathbf{x}} \frac{\partial \mathbf{w}^T \mathbf{x}}{\partial w_i}} x_i^{(k)}, \\
 &= \frac{1}{K} \sum_{k=1}^K \left( \overbrace{g(\mathbf{w}^T \mathbf{x}^{(k)})}^{pred.} - \overbrace{y^{(k)}}^{true} \right) x_i^{(k)}.
 \end{aligned} \tag{4.48}$$

Here, the cross entropy gradient does not have the multiplicative factor  $g(z^{(k)})(1 - g(z^{(k)}))$  seen in the  $L^2$  gradient of equation 4.39 (see Exercise 4.7.9). This means that  $g(z^{(k)})(1 - g(z^{(k)})) < 1$  so the cross-entropy gradient is larger and therefore is characterized by steeper objective functions in Figure 4.16 and a faster learning rate.

If the NN has many layers such that  $N \gg 1$  then the composite gradient of  $\epsilon$  for many layers will tend to zero because it is the concatenation  $O(N)$  layer gradients, where each one of them can be near zero. This leads to the vanishing gradient problem where convergence stalls (Bengio et al., 1994). This problem can be partly remedied by using other types of activation functions (see section 4.2) and will be addressed in Chapter 9.

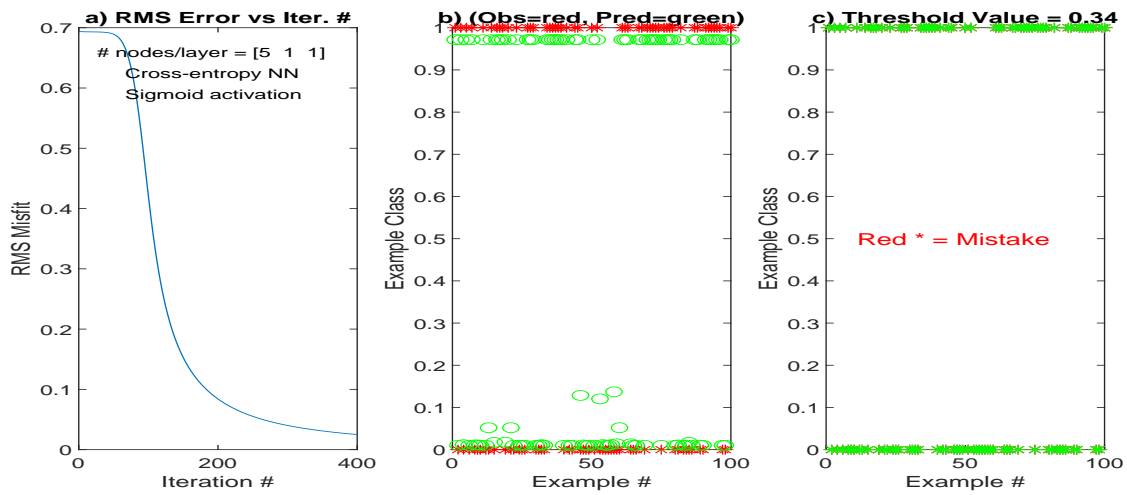


Figure 4.17: Same as Figure 4.15 except the cross-entropy objective function is used.

**Example 4.3.3.** *Binary Classification with Sigmoid and Cross-Entropy Objective Function*

The binary classification problem posed in Example 4.3.1 will now be solved using a cross-entropy objective function with a two-layer network. Results are shown in Figure 4.17 and reveal much faster convergence and a more accurate prediction of the classes than demonstrated by the two-layer NN results in Figure 4.15 with an  $L^2$  objective function. This is consistent with the plausibility argument that the cross-entropy objective function has a steeper slope (see Figure 4.16) so it is characterized by faster convergence than the  $L^2$  objective function.

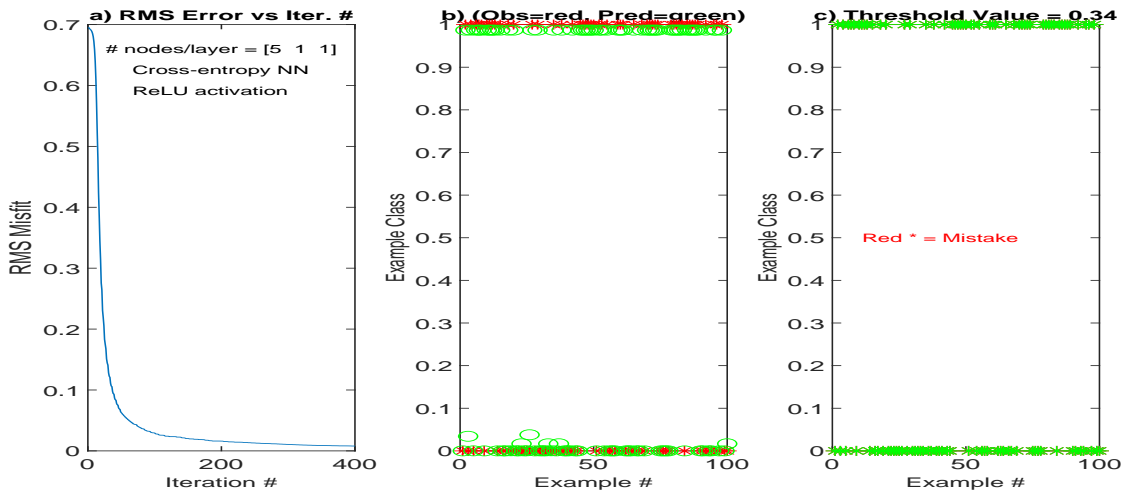


Figure 4.18: Same as Figure 4.17 except the cross-entropy objective function and ReLU activation function in the middle layer are used.

**Example 4.3.4.** *Binary Classification with ReLU and Cross-Entropy Objective Function*

A cross-entropy objective function and a ReLU activation function (in the middle layer) with a two-layer network is used to solve the binary classification problem posed in Example 4.3.1. The last layer in our new network still uses a sigmoid activation function because the output there needs to be between 0 and 1. Here, we use a steepest descent method with a bisection line-search procedure as outlined between the dashed lines in Code 4.3.1. The results are shown in Figure 4.18 and reveal the fastest convergence and the most accurate predictions compared to the results in Figures 4.15 and 4.17. In this case, the ReLU significantly enhanced the convergence rate compared to the exclusive use of the sigmoid function.

To test the speedup sensitivity of ReLU w/r to the number of nodes and layers, the binary classification problem is solved by a NN with the [5 5 5 2 1] architecture. We now use a conjugate gradient method with a more sophisticated polynomial line-search method (see Chapter 3 and <https://www.mathworks.com/matlabcentral/mlc-downloads/downloads/submissions/56645/versions/3/previews/fmincg.m/index.html>) coupled to a Polack-Ribiere formula. The results are shown in Figures 4.19 and 4.20. Here, the ReLU activation functions always provided the results with the fastest convergence rate. This conclusion is consistent with the results (not shown) when the [561] architecture is used to solve the same problem.



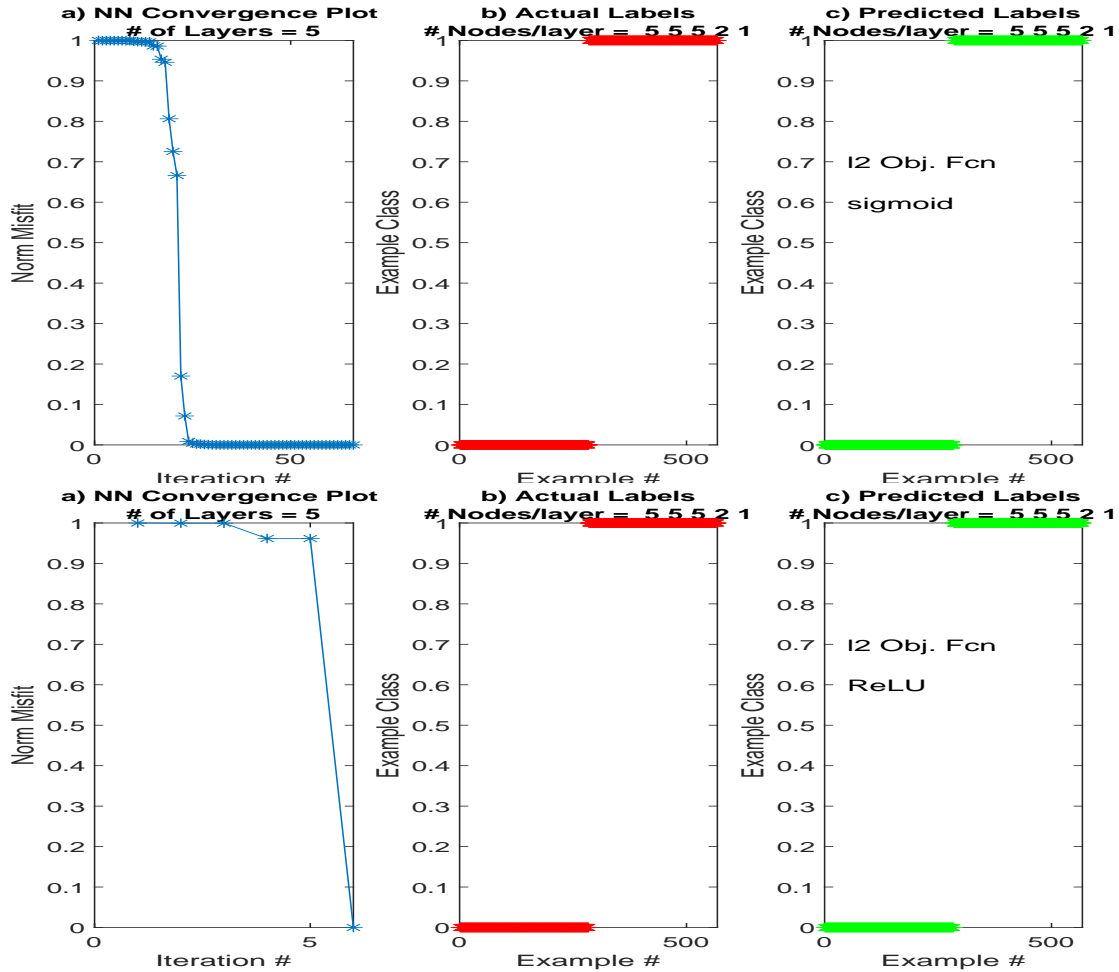


Figure 4.19: Binary classification results in c) using the  $[5\ 5\ 5\ 2\ 1]$  and  $L^2$  objective function is used with the (top row) sigmoid and (bottom row) ReLU activation functions. Here a conjugate gradient code with a Polack-Ribiere step length is used to compute the iterative solutions.

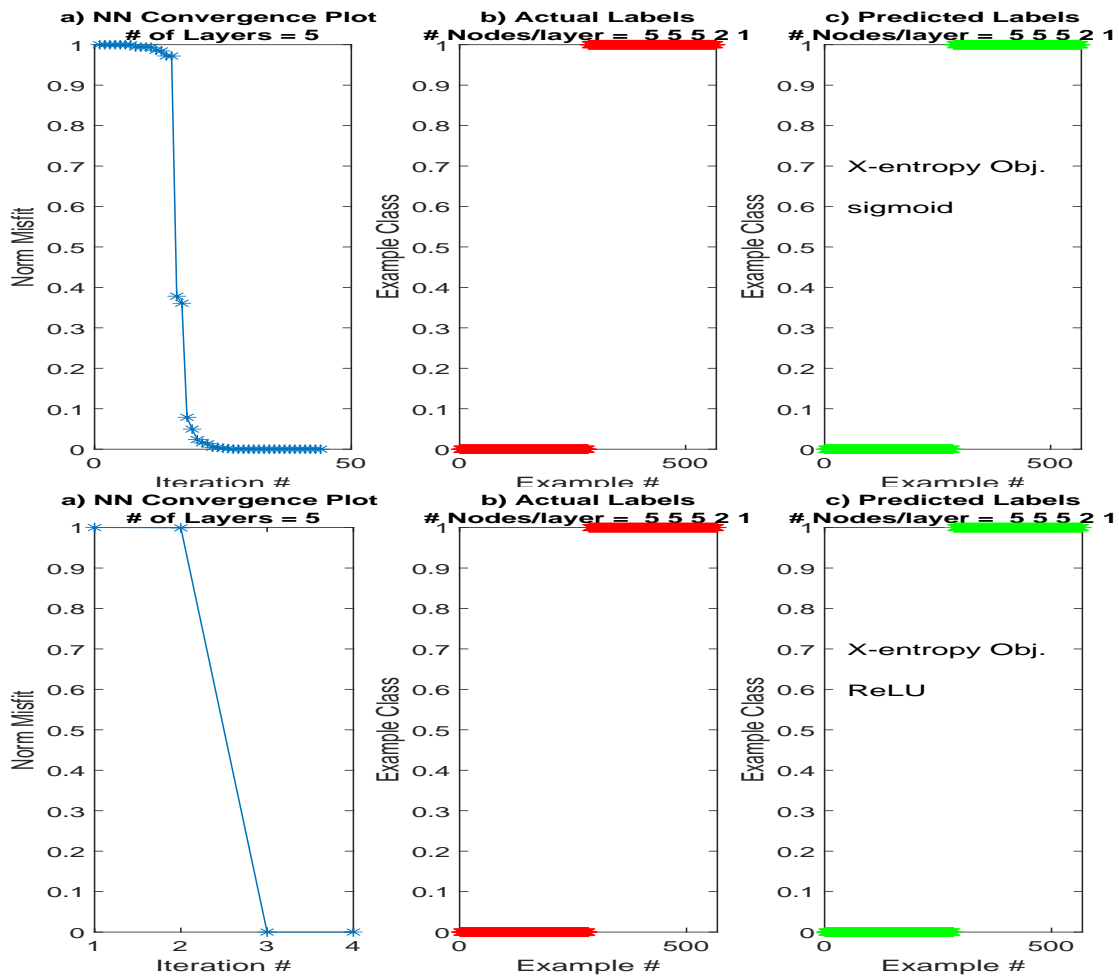


Figure 4.20: Same as Figure 4.19 except a cross-entropy objective function is used instead of the  $L^2$  objective function.

## 4.4 Types of Classification

There are at least three types of *classification* problems: 1). binary classification where the number of class types is  $C = 2$ , 2). multiclass where  $C > 2$  and only one of the output components of the  $C \times 1$  vector  $\mathbf{y}$  is assigned  $y_i = 1$ , while the rest are assigned to be 0, and 3). multilabel classification where  $C > 2$  but each input can be assigned more than one class type. In this case, many components of  $\mathbf{y}$  can be assigned the value  $y_i = 1$ , while the rest are zero.

### 4.4.1 Multilabel Classification

The multilabel classification scheme can assign more than one label to an input image (Bishop, 2006). If there are  $C > 2$  different classes of input objects, then we can use a network with  $C$  output nodes in the form of the  $C \times 1$  target vector  $\mathbf{y}$ . Each output node in  $\mathbf{y}$  has a separate and independent binary class label  $y_c \in \{0, 1\}$  for  $c = \{1, 2, \dots, C\}$ . For example, assume that the label  $y_1 = 1$  in the first output node says that the input image has eyes, otherwise  $y_1 = 0$  says it has no eyes. If the second output node is labeled  $y_2 = 1$ , then the input image has a tongue; if not, then  $y_2 = 0$ . Therefore, the input image of a human will have a target vector  $\mathbf{y} = (y_1, y_2, y_3, \dots, y_C)$  with  $y_1 = y_2 = 1$ .

Since the class types  $(y_1, y_2, \dots, y_C)$ , are independent of one another then the conditional probability  $p(y_1, y_2, \dots, y_C | \mathbf{x}, \mathbf{w})$  is a concatenation of the conditional probabilities  $p(y_c | \mathbf{x}, \mathbf{w})$  in equation 4.29 for each output node:

$$p(y_1, y_2, \dots, y_C | \mathbf{x}, \mathbf{w}) = \prod_{c=1}^C p(y_c | \mathbf{x}, \mathbf{w}). \quad (4.49)$$

If there are  $K$  independent training pairs then the conditional probability for predicting all of the labels  $\mathbf{y}^{(k)} = (y_1^{(k)}, y_2^{(k)}, \dots, y_C^{(k)})$  is a concatenation of the conditional probabilities in equation 4.49:

$$p(\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \dots, \mathbf{y}^{(K)} | \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(K)}, \mathbf{w}) = \prod_{k=1}^K \prod_{c=1}^C p(y_c^{(k)} | \mathbf{x}^{(k)}, \mathbf{w}). \quad (4.50)$$

Substituting the expression for the binary conditional probability function in equation 4.45 into equation 4.50 and taking its negative logarithm gives the cross-entropy objective function for a multilabel classifier:

$$\epsilon = -\frac{1}{K} \sum_{c=1}^C \sum_{k=1}^K [y_c^{(k)} \ln g_c(\mathbf{w}^T \mathbf{x}^{(k)}) + (1 - y_c^{(k)}) \ln(1 - g_c(\mathbf{w}^T \mathbf{x}^{(k)}) )], \quad (4.51)$$

where  $g_c$  is the sigmoid function. Here, the multiplicative weight of  $1/K$  is used to normalize the objective function so it provides an average value.

### 4.4.2 Multiclass Classification

The standard multiclass, aka multinomial, classification is for  $C > 2$  where the input image belongs to only one class, and no other. That is, the  $C \times 1$  target vector  $\mathbf{y}$  only has one element labeled as 1 and all the other elements are labeled as 0. This is known as

a  $1 - of - C$  coding multinomial scheme (Bishop, 2006), sometimes denoted as multiclass classification.

For example, assume the output class for node 1 is for the input image being a cat, node 2 is for dogs, and node 3 is for an animal being neither a dog or cat. Clearly any input image of a single animal can only be labeled as 1 for just one of the three output nodes. We cannot assign an independent binary conditional probability to each output node because, for example, the probability output of a sigmoid at nodes 1 and 2 might both take on the value 1. This would violate the *only one-class* requirement of a multiclass classification. The discrete probabilities must be coupled to one another so only one wins at the end with the highest probability, and they all sum to 1.

To fix this problem we need to generalize the sigmoid function so that it can serve as the conditional probability for more than two classes  $C > 2$  of outputs. The hint as to how to do this is to multiply the top and bottom of equation 4.29 by  $e^{\mathbf{w}_1^T \mathbf{x}}$  to get

$$p_1 = \frac{e^{\mathbf{w}_1^T \mathbf{x}}}{e^{\mathbf{w}_1^T \mathbf{x}} + e^{\mathbf{w}_2^T \mathbf{x}}}, \quad (4.52)$$

where  $\mathbf{w}_2 = -\mathbf{w} + \mathbf{w}_1$  and  $p_1$  is the discrete probability function of  $\mathbf{x}$  being classified as  $y_1 = 1$  at output node #1 for a given  $\mathbf{w}_1$  and  $\mathbf{w}_2$ .

Equation 4.52 might represent the output of a two-node network because there are two model vectors  $\mathbf{w}_1$  and  $\mathbf{w}_2$  at the last layer. In the context of the NN in Figure 4.6, the first row of the corresponding NN matrix  $\mathbf{W}$  at the output layer would be  $\mathbf{w}_1$ , and the second row would be  $\mathbf{w}_2$ . This suggests that the probability  $p_2$  of the target value is  $y_2 = 1$  at the second output node is

$$p_2 = \frac{e^{\mathbf{w}_2^T \mathbf{x}}}{e^{\mathbf{w}_1^T \mathbf{x}} + e^{\mathbf{w}_2^T \mathbf{x}}}. \quad (4.53)$$

We can combine these two probability functions to get the joint conditional probability function

$$p(y_1, y_2 | \mathbf{x}, \mathbf{w}_1, \mathbf{w}_2) = p_1^{y_1} p_2^{y_2}, \quad (4.54)$$

where the only possible values for  $\mathbf{y} = (y_1, y_2)$  are either  $(1, 0)$  or  $(0, 1)$ .

If the input image to the last layer of the NN is denoted as the  $M \times 1$  vector  $\mathbf{z}$ , and there are  $C$  output nodes where only one of the nodes is classed as  $y_c = 1$ , and the others have class type 0, then the probability of the  $c^{th}$  node taking the class  $y_c = 1$  is the softmax function:

$$p_c = p(y_c | \mathbf{x}, \mathbf{w}_1, \mathbf{w}_2) = \frac{e^{a_c}}{\sum_{c'=1}^C e^{a_{c'}}}, \quad (4.55)$$

where  $a_c = \mathbf{w}_c^T \mathbf{z}$  denotes the  $c^{th}$  output component of the matrix-vector multiplication at the last layer of the NN. Here,  $\mathbf{W}$  is the  $C \times M$  matrix of the last layer with the  $c^{th}$  row denoted by  $\mathbf{w}_c$ . Clearly,  $\sum_{c=1}^C p_c = 1$  and  $0 \leq p_c \leq 1$ , which satisfy the conditions for the softmax function being a discrete conditional probability function.

More generally, the joint conditional probability function  $p(y_1, y_2, \dots, y_C | \mathbf{x}, \mathbf{W})$  for  $C$

output nodes (or classes) is obtained in a way similar to that in equation 4.55, i.e. take the products of  $p_c^{y_k}$

$$p(\overbrace{y_1, y_2, \dots, y_C}^y | \mathbf{x}, \mathbf{W}) = \prod_{c=1}^C p_c^{y_c}. \quad (4.56)$$

Here, the term raised to the  $1 - y$  power in equation 4.45 is not needed because this is no longer the expression of a binary classification at a single node. In the binary classifiers with a one-node output discussed in section 4.3, the joint conditional probability was assessed at the same output node so it was the product of the two possible predicted probabilities  $p^y(1 - p)^{1-y}$  at this one node. In contrast, the burden for multiclassification with  $C = 2$  is spread between two nodes, with node #1 indicating one class  $y_1 = 1$  and node #2 indicating the other class  $y_2 = 1$ . Thus the joint conditional probability function is  $p_1^{y_1} p_2^{y_2}$  in equation 4.54, which can be shown to reduce to the same equation for a single-node binary classifier.

If there are  $K$  independent training pairs then the conditional probability for predicting all of the labels  $\mathbf{y}^{(k)} = (y_1^{(k)}, y_2^{(k)}, \dots, y_C^{(k)})$  is a concatenation of the conditional probabilities in equation 4.56. Similar to that in equation 4.50, the cross-entropy loss function for the multinomial classifier is obtained by taking the negative logarithm of the concatenation of conditional probabilities in equation 4.56 to get

$$\epsilon = - \sum_{c=1}^C \sum_{k=1}^K y_c^{(k)} \ln p(\mathbf{y}_c^{(k)} | \mathbf{x}^{(k)}, \mathbf{W}), \quad (4.57)$$

where we have included the index  $k$  associated with the training pair  $(\mathbf{z}^{(k)}, \mathbf{y}^{(k)})$  and taken the product of the joint probabilities  $p(\mathbf{y}_c^{(k)} | \mathbf{x}^{(k)}, \mathbf{W})$  of each training pair because they are independent of one another.

As an example, consider the case of one training example  $K = 1$  and three classes  $C = 3$  so that equation 4.57 becomes

$$\epsilon = -y_1 \ln p((1, 0, 0) | \mathbf{x}, \mathbf{W}) - y_2 \ln p((0, 1, 0) | \mathbf{x}, \mathbf{W}) - y_3 \ln p((0, 0, 1) | \mathbf{x}, \mathbf{W}), \quad (4.58)$$

where the superscript notation  $(k)$  is silent. If the actual target vector  $\mathbf{y} = (y_1, y_2, y_3) = (0, 0, 1)$  then equation 4.58 becomes

$$\epsilon = - \ln p((0, 0, 1) | \mathbf{x}, \mathbf{W}) = - \ln \left\{ \frac{\overbrace{e^{\mathbf{w}_3^T \mathbf{x}}}^{p_3}}{e^{\mathbf{w}_1^T \mathbf{x}} + e^{\mathbf{w}_2^T \mathbf{x}} + e^{\mathbf{w}_3^T \mathbf{x}}} \right\}. \quad (4.59)$$

However, the element values in the model matrix  $\mathbf{W}$  might be a poor choice so the soft-max term  $p_3$  gives a small probability value near zero, e.g.  $p_3 = e^{-4} = 0.01$  so  $\epsilon = -\ln e^{-4} = 4$ . This is a relatively large number and an unacceptably large objective function. A better choice would be to choose a  $\mathbf{W}$  that gives a probability close to 1, e.g.  $p_3 = e^{-.001} = 0.99$  so that  $\epsilon = -\ln e^{-.001} = .001$ , which is almost at the minimum of the cross-entropy objective function.

In summary, we have different loss functions for different types of classification problems.

1. **Binary.** If the output is labeled as just one of two classes, then there is only a single output node. The binary cross-entropy loss function in equation 4.51 is used with  $C = 1$ .
2. **Multilabel.** If the class labels are independent of one another, then equation 4.51 is used for the multilabel loss function.
3. **Multiclass.** If the labeling is 1-of- $C$  coding, i.e. there is only one element in the unit vector  $\mathbf{y}$  with the value 1, then equation 4.57 is used for the cross-entropy loss function.
4. **Alternative.** An alternative to a binary classification scheme, with just one output node and the loss function in equation 4.51 with  $C = 1$ , is to use two output nodes and use the softmax output activation function in equation 4.55 at each node. In this case, equation 4.57 is used as the loss function.

**Example 4.4.1.** *Multiclass Classification with a Neural Network*

We will now employ a NN for multiclass classification of the input data used in Example 4.3.4.1. Instead of classifying the output to be binary  $\mathbf{y} = (y_1)$  for  $y_1 \in \{0, 1\}$ , a multiclass NN will label the input as one of the, e.g., 6 different classes of the input vector  $\mathbf{x} = (x_1, x_2, x_3, x_4, x_5)$ , where the multiclass target vector is the unit vector  $\mathbf{y} = (y_0, y_1, y_2, y_3, y_4, y_5)$  where  $y_i \in \{0, 1\}$ . Here,  $y_i = 1$  indicates that there are  $i$  1's in the input vector  $(x_1, x_2, x_3, x_4, x_5)$ . For example, if  $y_0 = 1$  this indicates that there are no 1's in the input vector  $\mathbf{x}$ . Similarly, if  $y_1 = 1$  then this indicates that there is only a single 1 in  $\mathbf{x}$ . If  $y_2 = 1$  then this indicates that there are two 1's in  $\mathbf{x}$ , and so on. Figure 4.21a-4.21b depict an example of 100 input vectors and their classes, respectively.

We will use a NN architecture that consists of two layers, a hidden layer with one node and the output layer has 6 nodes. This architecture is denoted as [5 1 6] because the input layer consists of 5 nodes. The last layer in our multiclass network uses the softmax activation function to compute the  $6 \times 1$  output vector  $\mathbf{y}^{pred.}$ . The output node with the highest value of the softmax function is used to classify the corresponding input  $\mathbf{x}$ .

The classification results for a NN designed with the  $L^2$ -objective and sigmoid-activation functions are shown in Figures 4.22-4.23. Accurate classifications are achieved within a reasonable number of iterations. However, the fastest convergence rates are those that used the ReLU activation function in the NN.

## 4.5 Best Practices

Practical tips for successfully executing neural networks, both fully connected and convolutional, include the following.

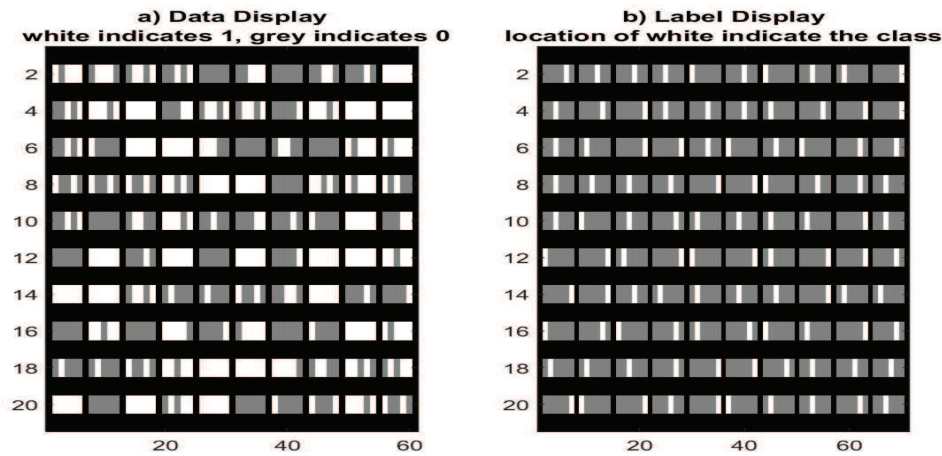


Figure 4.21: Distributions of a) input vectors  $(x_1, x_2, x_3, x_4, x_5)$  and b) their class  $\mathbf{y} = (y_0, y_1, y_2, y_3, y_4, y_5)$  vectors.

1. **Data Scaling.** Demean the  $N \times 1$  input data  $\mathbf{x}$  and scale by the standard deviation so the variance of  $\mathbf{x}$  is 1. Scaling can also be used to scale input values so that  $0 \leq x_i \leq 1$ . For example, the amplitudes in a seismic trace can be normalized to be between  $-1/2$  and  $+1/2$ , and then  $1/2$  is added so that the input amplitudes are between 0 and 1. Alternatively, the tanh function can be used as the thresholding function in the NN.
2. **Weight Initialization.** The NN weights  $\mathbf{W}^{(0)}$  at the first iteration can be initialized by a random number generator to be between -1 and 1. There is no rigorous advice here, but an early attempt known as Xavier initialization (Glorot and Bengio, 2010) suggested that *"Weights are set at random values sampled uniformly from a range proportional to the size of the number of nodes in the previous layer (specifically  $\pm 1/\sqrt{n}$  where  $n$  is the number of nodes in the prior layer)"*. However, He et al. (2015) suggested this was not appropriate for NNs with ReLU, so they proposed the He initialization: *He initialization: specify the weights as  $\pm \sqrt{2/n}$  where  $n$  is the number of nodes in the prior layer known as the fan-in. In practice, both Gaussian and uniform versions of the scheme can be used.* (Brownlee, 2019). See Exercise 4.7.23.
3. **Bias Initialization.** The initial bias terms can be set to a small number such as 0.1 for ReLU (Brownlee, 2019). This makes it very likely that the rectified linear units will be initially active for most inputs in the training set and allow for effective updating of the model parameters (Goodfellow et al., 2016).
4. **Activation Functions.** Use ReLUs or some version of them for the activation units in the hidden layers. Empirical evidence suggests that NNs with ReLU or Swish activation functions have faster convergence than those using sigmoid-like activation functions. To quote Goodfellow et al. (2016): *"In modern neural networks, the default recommendation is to use the rectified linear unit or ReLU."* However, the recurrent neural networks (RNNs) still use the tanh or sigmoid activation functions.

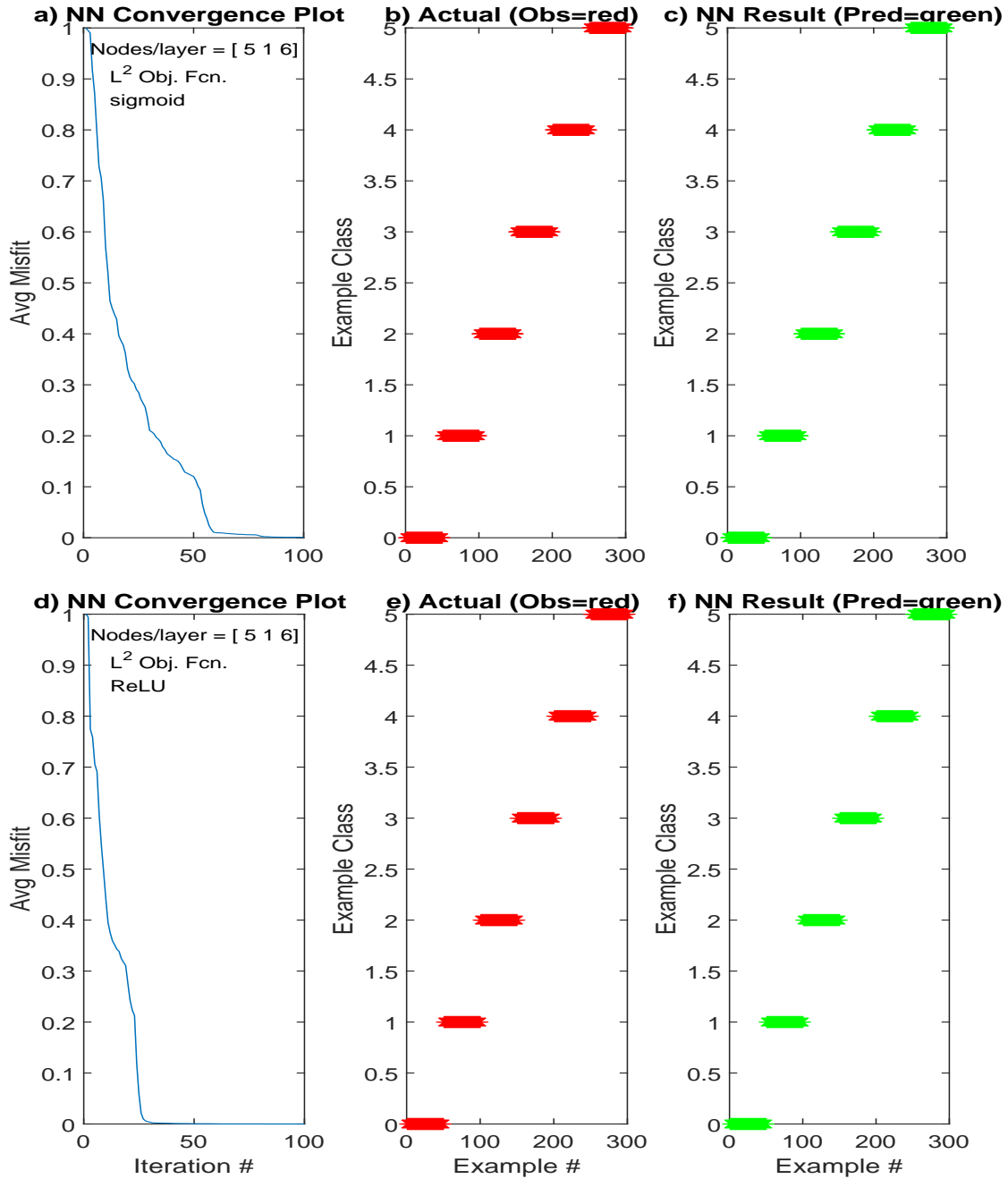


Figure 4.22: Plots of a)  $L^2$  objective function versus iteration number, b) actual class labels of 100 data vectors ( $x_1, x_2, x_3, x_4, x_5$ ) and c)  $\mathbf{y} = (y_1, y_2, y_3, y_4, y_5, y_6)$  class-label predictions computed by the NN with the node/layer structure denoted by [5 1 6]. Here, the input layer is counted as a layer. The second row d)-f) of images is the same as the first row except a ReLU activation function is used instead of a sigmoid in the hidden layer.



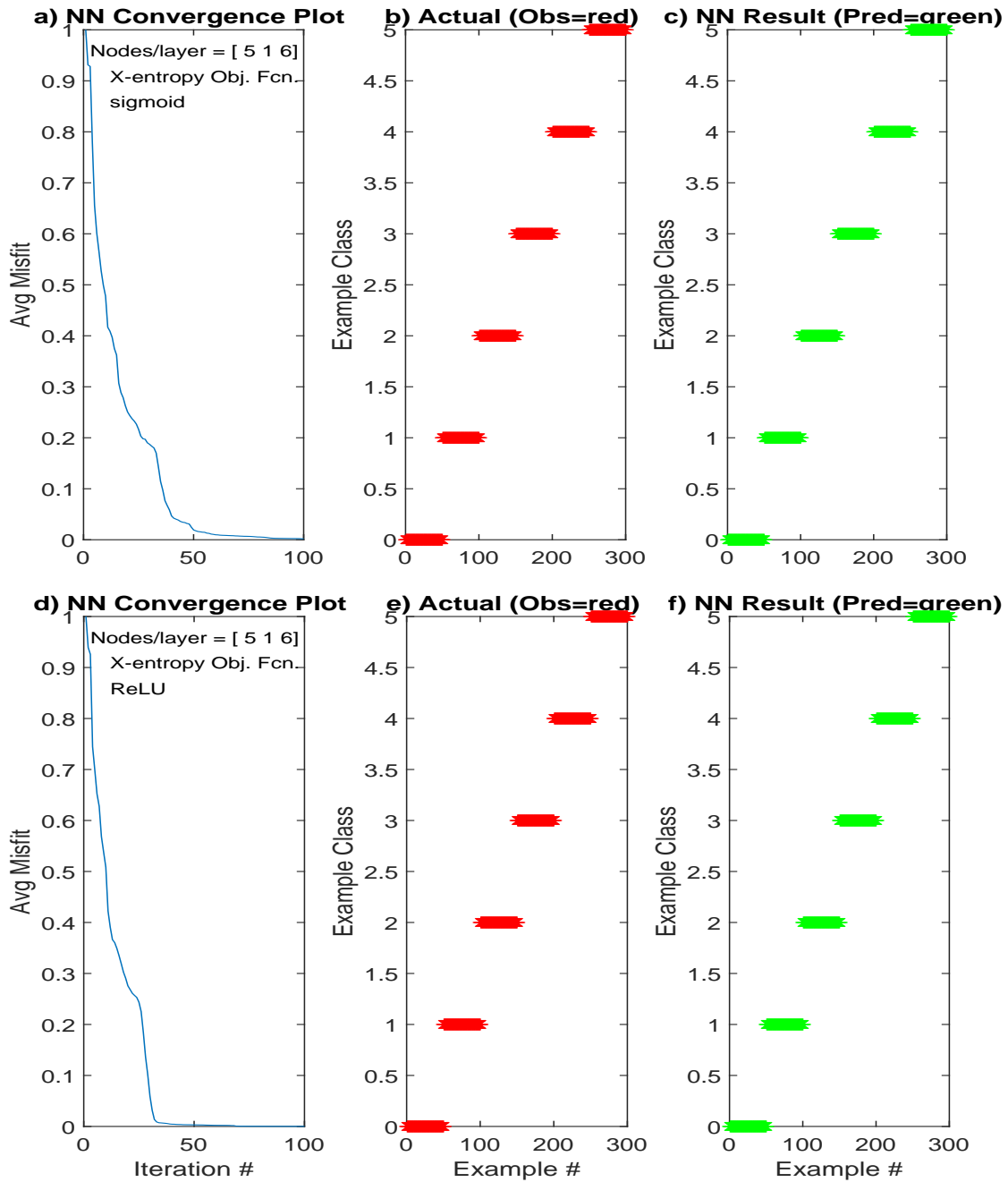


Figure 4.23: Same as Figure 4.22 except the  $L^2$  objective function is replaced by the cross-entropy objective function.

5. **Convolutional Neural Networks.** If the NN is deep with more than five layers, then convolutional NNs (CNNs) is often preferred compared to the fully-connected NN (FCNN). See Chapter 9.
6. **Unknown Label.** For multilabel classification, always include an "unknown" label as an output node that represents the class not recognizable. This mitigates the problem of misidentification of a, for example, strange-looking dog being mislabeled as a cat. If a strange-looking dog picture, not seen in the training data, is in the out-of-training data then the FCNN will likely choose it as "unknown" if it has the option of an unknown label. See Singer (2019).
7. **Regularization.** To avoid overfitting, regularization is introduced into the solution. It typically steers the solution to be less complex than the solution for the overfitting case, and can be implemented in many ways. Examples include incorporating an explicit penalty term in the objective function (see equation 4.10), reducing the size of the weights in each layer by such a penalty term, early stopping of the iterations so the objective function does not go below a threshold value, reducing the size of the network by dropping out units in each layer (see Chapters 9 and 14), or adding noise to the input (see Chapter 14). See Exercises 4.7.7-4.7.8 and Exercise 4.7.14.
8. **Mini-batch Processing.** It is often too computationally demanding to compute the composite gradient if the entire *batch* of data cannot be fitted into the computer's physical memory. Instead, the data are divided into  $B$  non-overlapping mini-batches (Duchi et al., 2011; Kingma and Ba, 2014; Ruder, 2017), and the solution is sequentially updated for sequential moves along the gradients  $\nabla \epsilon^{\{ib\}}$  for  $ib \in \{1, 2, \dots, B\}$ , where the  $ib^{th}$  objective function is  $\epsilon^{\{ib\}}$ . This procedure is known as stochastic gradient descent<sup>9</sup> (SGD), where the batch size should easily fit within the physical memory. Its pseudo-code is presented in Box 4.5.1, where  $\epsilon^{\{ib\}}$  is the objective function for the  $ib^{th}$  mini-batch. Figure 4.24 illustrates the updates (4 black arrows at the peak) for the mini-batch consisting of 4 examples. Here, the gradient is computed at the peak for four examples, and summed to give the long black arrow. This is repeated for the next four examples at the end point of the long black arrow.

---

<sup>9</sup>The term stochastic gradient descent is sometimes restricted to gradient algorithms where the mini-batch consists of just one training pair; if the input is more than one training pair then it is referred to as mini-batch SGD. We will ignore these distinctions and refer to SGD as the iterative code in Box 4.5.1 for any size mini-batch as long as it is less than the size of the original training data.

**Code 4.5.1.** *Pseudo-code for Mini-batch Stochastic Gradient Descent*

```

for it = 1 : nit                                Loop over epochs
    for ib = 1 : B                                Loop over B mini-batches
         $w^{\{ib+1\}} = w^{\{ib\}} - \alpha \nabla \epsilon^{\{ib\}}$ 
    end
     $w^{\{1\}} = w^{\{B+1\}}$                                 Reshuffle data  $\rightarrow$  B mini-batches
end

```

The new solution  $\mathbf{w}^{\{1\}}$  is computed by moving along the gradient direction  $\nabla \epsilon^{\{1\}}$  for a single iteration until it gets to  $\mathbf{w}^{\{2\}}$ , the point that minimizes  $\epsilon^{\{1\}}$ . This new point  $\mathbf{w}^{\{2\}}$  serves as the starting point for the second mini-batch of data with misfit  $\epsilon^{\{2\}}$ , where a new gradient  $\nabla \epsilon^{\{2\}}$  is computed. We move along  $\nabla \epsilon^{\{2\}}$  until there is no further reduction in  $\epsilon^{\{2\}}$ . This process is repeated until the iterative solution  $\mathbf{w}^{\{B\}}$  is found for the last batch of data. This is defined as one *epoch* of iterations. The data are randomly reshuffled into a new group of  $B$  non-overlapping batches, and the process is repeated until the solution at epoch 2 is reached. This iterative procedure can be repeated for more epochs until the objective function value is below a specified threshold value.

The iterative solution by the mini-batch stochastic gradient descent method is guaranteed to converge to the global minimum for convex objective functions (Ruder, 2017), and to a local minimum for non-convex functions. The mini-batch can be just one data example, but typically 50 to 256 are often preferred (Ruder, 2017). The averaging of gradients in mini-batch SGD efficiently reduces the noise in the gradients of individual training pairs to give more stable convergence than standard steepest descent so that *mini-batch SGD is typically the algorithm of choice when training a neural network (Ruder, 2017)*. Ruder (2017) claims that the SGD gradient can jump in dramatically different directions from one example to another one, which can lead to *new and potentially better local minima*. The mini-batch SGD is very similar to the Monte Carlo approach to FWI, where mini-batches of shot gathers are inverted from a randomly selected batch of data (Krebs et al., 2009; Boonyasiriwat and Schuster, 2010; Yang and Wu, 2017; van Herwaarden et al., 2020).

An example of the convergence rate with a CNN is shown in the top plots of Figure 4.25 for different sized mini-batches. In this example, smaller mini-batches show faster convergence than the larger batch sizes. Empirical experiments by Keskar et al. (2017) suggest that large batch sizes are often characterized by objective functions with *sharp minima*, and so tend to get stuck in local minima. This is illustrated with the sharp minimum (with steep sides) shown in the bottom plot of Figure 4.25. In this example, training with SD will not generalize well to out-of-the-training data. In contrast, the smaller batch sizes are often characterized by flatter landscapes in the objective function, and so have less of a tendency to getting stuck in local minima. If

## Mini-Batch Stochastic Gradient Descent



Figure 4.24: Update arrows for mini-batch SGD using single-example per batch (red) and gradients for four-examples per-batch (dashed black arrows). The red single-example arrows move to the next update point after each gradient calculation. The four-example arrows compute four mini-batch gradients (4 black dashed arrows) evaluated at the same point, sum them together to get the update direction, and move along this update to the next model point (long solid black arrow).

they do get stuck, then they can escape with the next mini-batch iteration.

Dinh et al. (2017) and Kawaguchi et al. (2017) disputed the idea that small-batch SGD produces *flat* minimizers that generalize well, while large batches produce *sharp* minima with poor generalization. They argue that that generalization is not directly related to the curvature of loss surfaces, citing some authors that achieve good performance with large batch sizes. What seems to be true here is that it is difficult to make general conclusions from empirical results on a limited number of sets. This highlights a weakness of neural networks which lacks a deep theoretical understanding of its fundamental properties.

9. **Step Length.** There are many step-length strategies (see Chapter 3), but the adaptive ones such as momentum, Adam and Adagrad are quite popular. Nevertheless, getting trapped in a local minimum and recognizing this is still a challenge. As the solution gets near the minimum, the step length can be decreased (or *annealed*) to avoid overshooting the desired solution. The term *step length*, like many other terms in optimization theory, was defined long before the machine learning community *re-discovered* this concept, so we often use the term *step length* rather than the machine learning (ML) terminology of *learning rate*.
10. **Gradient Verification.** If the NN code has a bug in it then the results will likely be unreliable. To avoid this problem, the gradient calculation of the code  $\frac{\partial \epsilon(\mathbf{w})}{\partial w_i}$  can be compared to that of the finite-difference estimation of the gradient:

$$\left[ \frac{\partial \epsilon(\mathbf{w})}{\partial w_i} \right]^{FD} = \frac{\epsilon(\mathbf{w} + \delta w_i \hat{\mathbf{e}}_i) - \epsilon(\mathbf{w})}{\delta w_i}, \quad (4.60)$$

where  $\delta w_i$  is the small perturbation of the  $i^{th}$  model component along the unit model vector  $\hat{\mathbf{e}}_i$ . The results for validating the NN gradient by a finite-difference computation are given in Figure 4.26, where the classification problem is the same as that described in Figure 4.12. Appendix 4.10 presents a fragment of the MATLAB code that computes this graph.

Alternatively, the gradient computed from your own self-written code can be checked for accuracy using the automatic differentiation codes that are widely available and discussed in Chapter 26.

## 4.6 Summary

The equations for a neural network  $f(\mathbf{x}, \mathbf{W}) = \mathbf{y}$  are presented, and their solution by an iterative gradient method is derived. The NN maps input data to output data by a concatenated sequence of two operations, matrix-vector multiplication and non-linear thresholding by an activation function. The greater the number of layers and units/layer, the more complexity a NN can model in the mapping between  $\mathbf{x}$  and  $\mathbf{y}$ . Each matrix in the neural network is densely populated, which increases the costs of storage and computation time compared to that of a layer for the sparsely populated convolutional neural network.

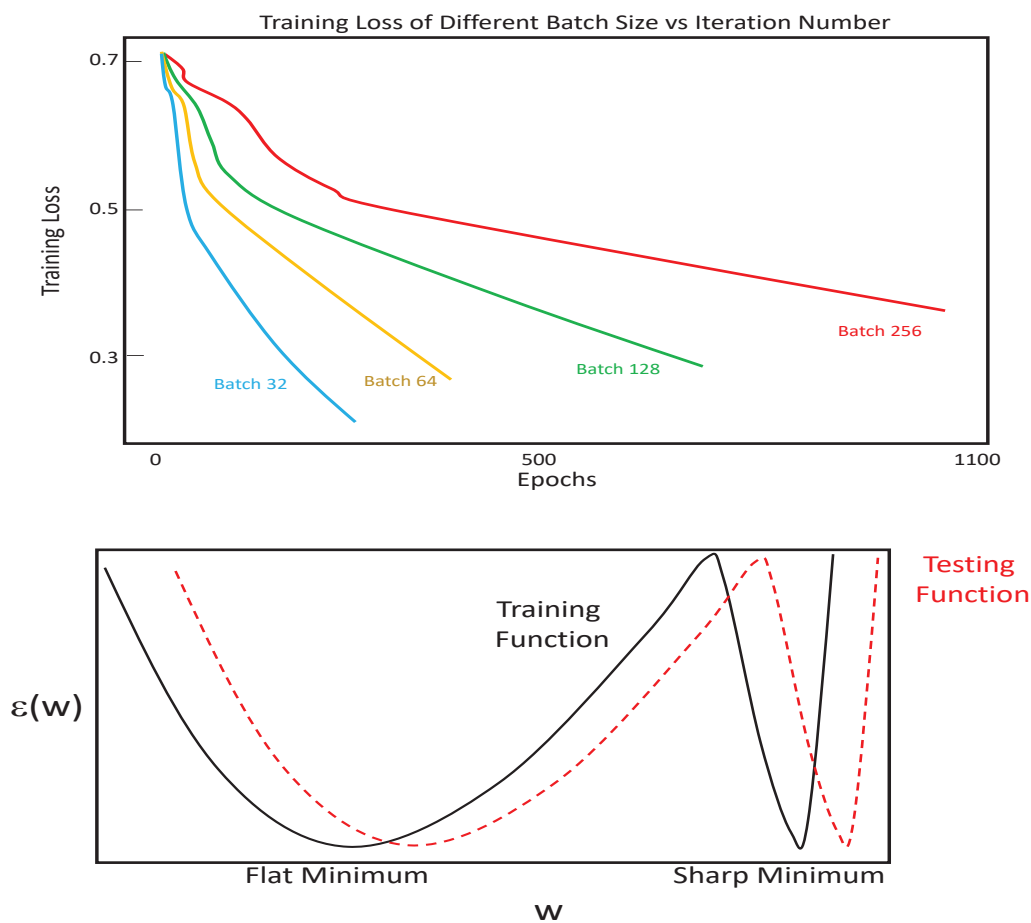


Figure 4.25: Top: Objective function loss versus epoch number for different sizes of mini-batches in a classification example from Chang (2020). Bottom: Flat and sharp objective functions. The geometry of the flat landscape allows SGD to escape from a local minimum at the next mini-batch iteration. Dashed lines represent the objective function of the test data. Typically, validation data is for tuning the NN optimization parameters such as the number of layers, the step length etc., and testing data is for determining the final performance of the NN algorithm. Illustration adapted from Keskar et al. (2017).

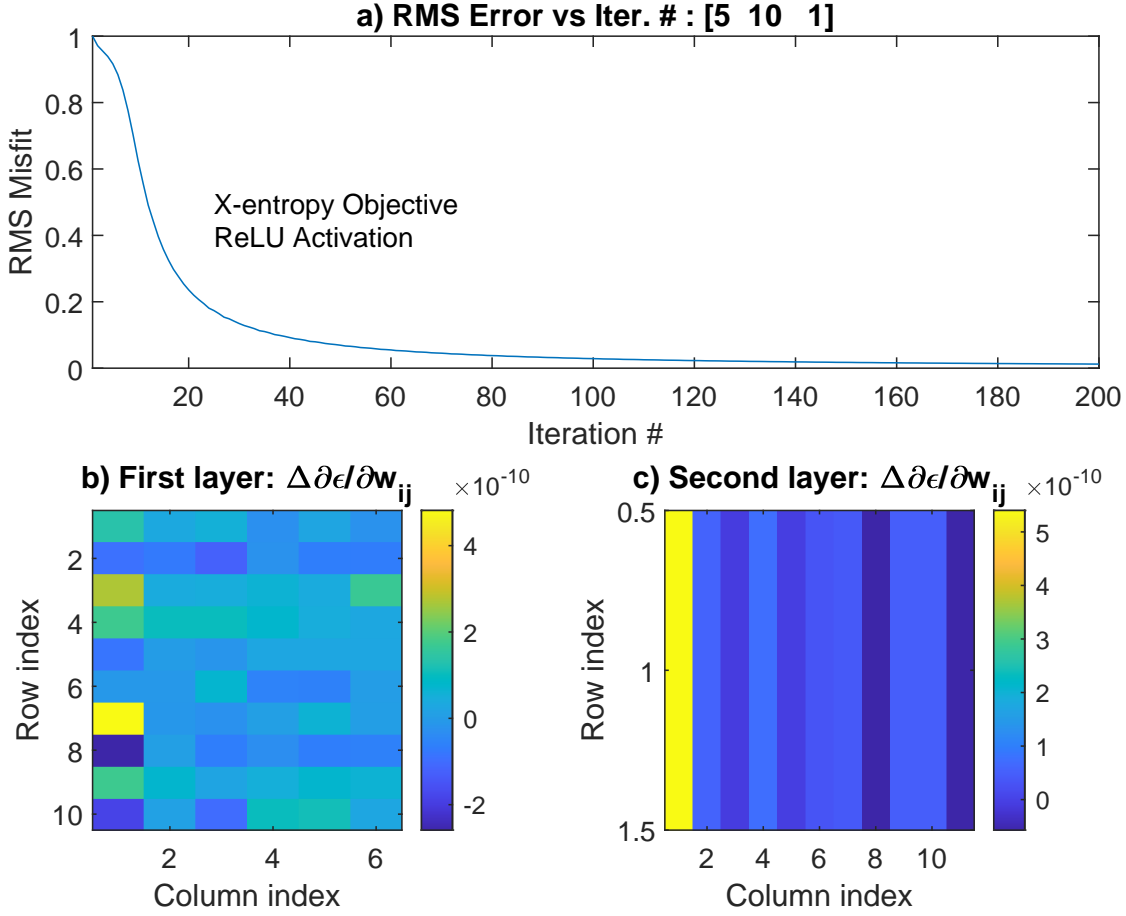


Figure 4.26: Plot of a) RMS misfit versus iteration number. Graphs b)-c) present the normalized differences  $\Delta \frac{\partial \epsilon}{\partial w_{ij}} = (\frac{\partial \epsilon}{\partial w_{ij}} - [\frac{\partial \epsilon}{\partial w_{ij}}]^{FD}) / \gamma$  between the NN gradient  $\frac{\partial \epsilon}{\partial w_{ij}}$  and the finite-difference gradient  $[\frac{\partial \epsilon}{\partial w_{ij}}]^{FD}$ . Here, the normalization factor  $\gamma = |\frac{\partial \epsilon}{\partial w_{ij}}|$  and the architecture of the 2-layer NN is denoted by [5 10 1] with 5 nodes in the input layer, 10 in the hidden layer, and one in the output layer. The classification problem is the same as that described in Figure 4.12. The column and row indices are those for the b)  $10 \times 6 \mathbf{W}^{[1]}$  and c)  $1 \times 10 \mathbf{W}^{[2]}$  matrices in the NN, where the bias variables are included in the model variables. .

Two types of problems can be solved by a NN. The first one is a regression problem where the output elements of  $\mathbf{y}$  can be a continuous range of numbers on the real line. The second one is the classification problem where the output values are limited to a small discrete set of numbers. Each number can be assigned a different class type. The classification problems typically favor a cross-entropy type objective function while regression problems also use a  $L^2$  misfit function. MATLAB codes are provided and can be used to assess the solution sensitivity to different parameters such as the damping term, iteration number, step length, and errors in the data. The more complex equations for finding the gradient of a general neural network are presented in Chapter 6.

## 4.7 Exercises

1. Show that  $\frac{\partial g(z)}{\partial z} = g(z)(1 - g(z))$ , where  $g(z)$  is the sigmoid function.
2. A linear operator  $g(z)$  has the property that  $g(\alpha z) = \alpha g(z)$ . Show that  $g(\alpha z) = \alpha g(z)$  is not true if  $g(z)$  is the sigmoid function.
3. Assume that the activation function is linear such that  $g(\mathbf{x}) = \mathbf{W}\mathbf{x}$ , where  $\mathbf{x}$  is an  $N \times 1$  vector and  $\mathbf{W}$  is an  $N \times N$  matrix. Show that the multilayer system can be represented by  $g(g(g(\mathbf{x}))) = \mathbf{W}'\mathbf{x}$  where  $\mathbf{W}' = \mathbf{W}^{[3]}\mathbf{W}^{[2]}\mathbf{W}^{[1]}$  and  $\mathbf{W}^{[i]}$  is the square matrix associated with the  $i^{th}$  layer. Is a multilayer linear system more capable at modeling a non-linear model than a single-layer linear system? Can the system with a linear activation function accurately model a non-linear system of equations?
4. Assume that  $\mathbf{x}$  is the  $5 \times 1$  input vector such that  $x_i \in \{0, 1\} \forall i \in \{1, 2, 3, 4, 5\}$ . Use the same reasoning used for Figure 4.14 to argue that a single-node NN can distinguish  $\mathbf{x} = (0, 0, 0, 0, 0)$  from all the other possible inputs. Hint: Each vertex in Figure 4.14 has two other hidden dimensions, whose points belong to the hidden planar surface in  $x_4 - x_5$ . This means that there are three extra vertices for each vertex in Figure 4.14. For example, the vertex  $(0, 0, 1)$  has the hidden planar surface with vertices at  $(0, 0, 1, 1, 0)$ ,  $(0, 0, 1, 1, 1)$ ,  $(0, 0, 1, 0, 1)$ .
5. Create two separated Gaussian functions from a two-layer NN with four sigmoid activation functions. The MATLAB code for creating one Gaussian function is in Box 4.7.1.



**Code 4.7.1.** *MATLAB code for Generating the Figure 4.9 Gaussian Function Using a 2-layer Neural Network (Yunsong Huang).*

```
x_ax = linspace(-4,4,51);
[X,Y] = meshgrid(x_ax, x_ax); % x_ax == y_ax
spc = 0.7;
gain1 = 1.75;
thresh2= 3.1;
gain2 = 12;
[output, hiddenSgm, outActvFnt, outActvZ] = assemble4hidden_nodes_fnc(...
    X,Y, spc, gain1, thresh2, gain2);

figure
for i = 1:4
    subplot(2,2,i)
    surf(X,Y,hiddenSgm{i})
    xlabel('x_1')
    ylabel('x_2')
end
% print -dpng FourHiddenNeurons.png

figure
subplot(1,4,1)
plot(outActvZ, outActvFnt, 'linewidth', 1.5)
axis([outActvZ(1), outActvZ(end), -0.05, 1.05])
xlabel('z')
title({'Output neuron''s', 'activation function', ' '})

subplot(1,4,[2,3,4])
surf(X,Y,output)
xlabel('x_1')
ylabel('x_2')
title('The output')
print -depsc INN.Yunsong.eps
```

and

**Code 4.7.2.** *MATLAB code for Generating the Figure 4.9 Gaussian Function Using a 2-layer Neural Network (cont.)*

```
function [output, hiddenSgm, outActvFnt, outActvZ] = assemble4hidden_nodes_fnc(...
    X,Y, spc, gain1, thresh2, gain2)
% Taking inputs in 2-D,
% 4 hidden neuron each has a sigmoid (with gain=gain1, threshold=0), spaced at
% 2*spc apart, a pair facing each other in x, another pair facing each
% other in y. Those 4 sigmoids add up to a max center value at (0,0).
% At the Output neuron taking the 4 hidden neurons as inputs, the threshold
% and gain parameters are thresh2, gain2

% outActvFnt, outActvZ 1D plot of the out neuron's activation function
%
% X,Y generated by meshgrid
% spc hidden neuron's facing sigmoids are spaced at 2*spc apart

sz = size(X);
hiddenSgm = cell(1,4);
for i = 1:4, hiddenSgm{i} = zeros(sz); end % init memory
% hiddenSgm's 1 & 2 are facing each other in x
% 3 & 4 are facing each other in y
% 1 & 3 are increasing functions
% 2 & 4 are decreasing functions
hiddenSgm{1} = 1 ./ (1+exp(-gain1.*(X+spc)));
hiddenSgm{2} = 1 ./ (1+exp( gain1.*(X-spc)));
hiddenSgm{3} = 1 ./ (1+exp(-gain1.*(Y+spc)));
hiddenSgm{4} = 1 ./ (1+exp( gain1.*(Y-spc)));

in2 = zeros(sz); % input to the output neuron
for i = 1:4, in2 = in2 + hiddenSgm{i}; end
%--
output = 1 ./ (1+exp( -gain2.*(in2-thresh2)));
%--
outActvZ = linspace(2, 4.5, 50);
outActvFnt = 1 ./ (1+exp( -gain2.*(outActvZ-thresh2)));

return
```

6. The bias<sup>10</sup> is the average of the squared difference between the prediction  $\hat{g}(x)$  and the noisy data  $y = g(x) + \epsilon$  in the equation below:

$$\langle (y - \hat{g}(x))^2 \rangle = [Bias(\hat{g}(x))]^2 + Var(\hat{g}(x)) + \sigma^2, \quad (4.61)$$

$$Bias(\hat{g}(x)) = \langle \hat{g}(x) \rangle - g(x) \quad \text{and} \quad Var(\hat{g}(x)) = \langle (\hat{g}(x) - \langle \hat{g}(x) \rangle)^2 \rangle. \quad (4.62)$$

Here  $\langle \rangle$  represents averaging over the ensemble of the random variables  $\epsilon$ , where  $\epsilon$  is white noise. Note,  $y$  and  $\hat{g}(x)$  are random variables because they are functions of the random variable  $\epsilon$ . We seek an optimal estimate  $\hat{g}(x)$  that minimizes the sum of squared errors in equation 4.61. The function  $g(x)$  is a deterministic function so  $\langle g \rangle = g$  and  $Var(g(x)) = 0$ , and we will use the abbreviations  $g(x) \rightarrow g$  and  $\hat{g}(x) \rightarrow \hat{g}$ .

<sup>10</sup>The use of the bias term here is different than that which refers to the affine translation term in the  $\mathbf{W}\mathbf{x}$  operation of a NN.

Show that

$$\begin{aligned} \langle (y - \hat{g})^2 \rangle &= \langle (y - g + g - \hat{g})^2 \rangle, \\ &= \sigma^2 + \langle (g - \hat{g})^2 \rangle + 2 \langle (y - g)(g - \hat{g}) \rangle. \end{aligned} \quad (4.63)$$

Show that

$$\begin{aligned} \langle (g - \hat{g})^2 \rangle &= \langle (\hat{g} - \langle \hat{g} \rangle + \langle \hat{g} \rangle - g)^2 \rangle, \\ &= \text{Var}(\hat{g}) + \text{Bias}^2(\hat{g}) + 2 \langle (\hat{g} - \langle \hat{g} \rangle)(\langle \hat{g} \rangle - g) \rangle. \end{aligned} \quad (4.64)$$

Show that both cross-product terms are equal to zero:

$$\begin{aligned} \langle (y - g)(g - \hat{g}) \rangle &= \langle yg - g^2 - y\hat{g} + g\hat{g} \rangle, \\ &= g^2 - g^2 - \langle y\hat{g} \rangle + g \langle \hat{g} \rangle, \\ &= - \langle (g + \epsilon)\hat{g} \rangle + g \langle \hat{g} \rangle = 0. \end{aligned} \quad (4.65)$$

Assume that  $\hat{g}$  and  $\epsilon$  are uncorrelated so  $\langle \epsilon\hat{g} \rangle = 0$ .

Show that

$$\begin{aligned} \langle (\hat{g} - \langle \hat{g} \rangle)(\langle \hat{g} \rangle - g) \rangle &= \langle \hat{g} \langle \hat{g} \rangle - \langle \hat{g} \rangle \langle \hat{g} \rangle - \hat{g}g + \langle \hat{g} \rangle g \rangle \\ &= 0. \end{aligned} \quad (4.66)$$

Krogh and Vedelsby (1994) show how the bias-variance formula can be used with cross-validation to estimate NN models that generalize well (Hyndman and Athanasopoulos, 2014).

7. Assume a 1-node perceptron with one layer such that  $\epsilon^{misfit} = \frac{1}{2}(g(\mathbf{w}^T \mathbf{x}) - \mathbf{y})^2$ . Show that the regularized steepest descent code for updating the bias term  $w_0 = b$  and the weights  $(w_1, w_2, \dots, w_I)$  is

$$\begin{aligned} &\text{for } k = 1 : niter && \text{Loop over iterations} \\ &\quad \text{for } i = 1 : I && \text{Loop over weight index i} \\ &\quad \quad w_i := w_i - \alpha^{[k]} \frac{\partial \epsilon^{misfit}}{\partial w_i} - \lambda w_i, \\ &\quad \text{end} \\ &\quad b = b - \frac{\partial \epsilon^{misfit}}{\partial b}, \\ &\quad \text{end} \end{aligned} \quad (4.67)$$

The bias term  $b = w_0$  does not undergo regularization because its role, in the case of the linear relationship  $y = mx + b$ , is to shift the decision line, not change its shape.

8. Does the MATLAB code regularize the bias terms in *NNode1.m* in */LAB1/Chapter.Book.FCN/Chapter.Gradient.Neural/lab.html*? Change the code so you can compare the results where the bias terms are regularized versus not regularized. Is there a noticeable difference in convergence rates? Why?
9. Derive equation 4.48.
10. Show that  $\sum_{i=1}^2 p_i = 1$ , where the binary probability mass distribution function  $p_i$  described by the sigmoid function is  $p_1 = \sigma(z)$  for  $y = 1$  and  $p_2 = (1 - \sigma(z))$  for  $y = 0$ .
11. The previous exercise showed that  $p(y|z) = (1/(1 + e^{-z}))^y (1/(1 + e^{-z}))^{1-y}$  is a PMF (conditioned on  $-\infty \leq z \leq \infty$ ) in the discrete RV  $y \in \{0, 1\}$  because  $1 \geq p(y|z) \geq 0$  and  $\sum_{y=0,1} p(y|z) = 1$ . If the continuous RV  $z$  is drawn from a Gaussian distribution, can we say that  $p(y|z)$  is a probability function in the RV  $z$ ? Why?
12. Training with inconsistent data. Many data sets have labeling errors, so this results in a system of NN equations that are inconsistent. For example, the picture of a dog might be mislabeled as a cat. Introduce 10% error into the labeled training data in *NNode1.m* in */LAB1/Chapter.Book.FCN/Chapter.Gradient.Neural/lab.html*, i.e. the labels are wrong so the labeling equations are inconsistent. Rerun the code and determine the accuracy of predicted labels. Repeat this for different percent errors in labeled data. Plot percent input error versus percent of prediction error. Is there a linear relationship between input errors versus label error? Explain.
13. Same as previous question except use the multiclass neural network code in */LAB1/Chapter.Book.FCN/Chapter.MultiGNN/NNode\_multiclass.m*.
14. Consider the linear affine<sup>11</sup> problem where the residual is  $\mathbf{r} = \mathbf{X}\mathbf{w} + \mathbf{b} - \mathbf{y}$ . The  $L^2$  misfit function is

$$\epsilon = 1/2 \mathbf{w}^T \mathbf{X}^T \mathbf{X} \mathbf{w} + \mathbf{w}^T \mathbf{X}^T \mathbf{b} - \mathbf{w}^T \mathbf{X}^T \mathbf{y} - \mathbf{b}^T \mathbf{y} + \mathbf{b}^T \mathbf{b} + \mathbf{y}^T \mathbf{y}. \quad (4.68)$$

Show the following.

- Property 1: the second derivatives of  $\epsilon$  with respect to the parameters in  $\mathbf{b}$  do not depend on the input data in  $\mathbf{X}$ .
- Property 2: the second derivative of  $\epsilon$  with respect to  $(w_1, w_2, \dots)$  does depend on the input data.
- Recall that regularization tends to reduce overfitting so that the predicted output will not be sensitive to small changes in the input as seen in Figure 4.28b. Use Properties 1 and 2 to argue why  $(w_1, w_2, \dots)$  needs to be regularized to adjust the shape of the objective function while  $w_0 = b$  does not need to be regularized.

---

<sup>11</sup>A linear function  $f(x)$  preserves the origin of the system of equations such that  $f(0) = 0$ , while an affine function is a composite of a linear function and a translation. For example,  $z = \alpha x_1 + \beta x_2$  is a linear operation for constants  $\alpha$  and  $\beta$ , but  $z = \alpha x_1 + \beta x_2 + 4$  is an affine operation because it has shifted the origin by the factor 4.

15. The `Datatin.m` and `Dipsplay.m` codes are given in Boxes 4.7.3-INN.ZCcode2 and are in the MATLAB code `NNode1.m` in `/LAB1/Chapter.Book.FCN/Chapter.Gradient.Neural/lab.html`. Use these codes in conjunction with the one presented in the text to test the performance of a single neural network code that classifies  $5 \times 1$  vectors  $\mathbf{x}$  whose element values are either 1 or 0. The goal is to train a neural network that can recognize an input vector  $\mathbf{x}$  where there is only a single 1 in one of its elements.

- Compare the performance of the least squares algorithm against that of the cross-entropy algorithm. Which has a faster convergence rate.
- The number of data examples that are classified as 1 should be about the same as the ones that are classified as 0, otherwise the system of equations will be ill-conditioned and convergence will slow down. Numerically demonstrate this fact.

**Code 4.7.3.** *MATLAB Data  $\mathbf{x} = (x_1, x_2, x_3, x_4, x_5)$  Generation Subroutine where  $x_i$  is either 1 or 0. (Zongcai Feng)*

```
function [x,t,alpha,lim,ww,layer_size,obj,act]=Datain1(M,N,x,t,obj,act,layer_size)

for i=1:M
    x(:,i)=round(rand(N,1));
end
for i=1:M
    if sum(x(:,i))==0;t(i)=1;end
end

% Now we are setting about 1/2 of x(:, :) to have no 1 in the Nx1 vector
% so about 1/2 data are classed at t=1
t0=find(t==0); t1=find(t==1);
x(:,t0(1:round(M/2)-numel(t1)))=0.0;
t(t0(1:round(M/2)-numel(t1)))=1;
rank = randperm(M);
x=x(:,rank);
t=t(rank);

alpha=1.0; % step size
lim=.00001; % stopping criterion
%layer_size=[N,10,1]; % first and last elements are the input and output layer size
%layer_size=[N,10,1];
%obj=1; %objective function, 1 for L2 norm, 2 for likelihood
%act=1; %active function, 1 for sigmoid norm, 2 for ReLU

layer_num=numel(layer_size); %layer number include the input and output
ww=cell(layer_num-1,1);
for ilayer=1:layer_num-1
    % initialize the weights of the neural network with random number
    ww{ilayer}=rand(layer_size(ilayer+1),layer_size(ilayer)+1)*0.1; % +1 for bias
end

end
```

**Code 4.7.4.** *MATLAB Display of  $\mathbf{x} = (x_1, x_2, x_3, x_4, x_5)$  Subroutine (Zongcai Feng).*

```
function Display1(M,res,nit,ww,layer_size,x,t,obj,act,thresh)
tp=t*0; tpp=tp;
%thresh=.34;
layer_num=numel(layer_size); %layer number include the input and output
aa=cell(layer_num-1,1);
zz=cell(layer_num-1,1);
aa{1}=x;
for iter =1:layer_num-1
    aa{iter} = [ones(1, M); aa{iter}];
    zz{iter}=ww{iter}*aa{iter};
    if iter ==layer_num-1
        [aa{iter+1},~] = activation(zz{iter},1);
    else
        [aa{iter+1},~] = activation(zz{iter},act);
    end
end
tp=aa{layer_num};
for m=1:M
    if tp(m)>=thresh;tpp(m)=1;end
    if tp(m)<thresh;tpp(m)=0;end
end
subplot(131);plot(res(1:nit));
title('a) RMS Error vs Iter. #');

if obj==2;text(nit/6,max(res)*.9,'Cross-entropy NN','fontsize',8);
else;
text(nit/6,max(res)*.9,'Least-squares NN','fontsize',8);end

text(nit/15,max(res)*.95,...
    ['# nodes/layer = ',num2str(layer_size),'],','fontsize',8)

if act==2;text(nit/6,max(res)*.85,...
    'ReLU activation','fontsize',8);else;
text(nit/6,max(res)*.85,...
    'Sigmoid activation','fontsize',8);end

ylabel('RMS Misfit');xlabel('Iteration #')
subplot(132);plot([1:M],t,'r',[1:M],tp,'go')
ylabel('Example Class');xlabel('Example #')
title('b) (Obs=red, Pred=green)')
subplot(133);plot([1:M],t,'r',[1:M],tpp,'g*')
ylabel('Example Class');xlabel('Example #')
title(['c) Threshold Value = ',num2str(thresh)])
text(M/9,.5,'Red * = Mistake','color','red')
end

function [ g,dg ] = activation( z,type )
%activation function
%input:
%z: input for ativation function
%type: objctive function type: 1 for sigmoid, 2 for ReLU
%output:
% g is the value of the activation function
% dg is the gradient of g with respectve to z

if type==1 % for sigmoid
    g = 1.0 ./ (1.0 + exp(-z));
    dg = g .* (1 - g); %Compute the gradient
elseif type==2 % for ReLU
    g=z*0.0; dg=z*0.0;
    g(z>0)=z(z>0);
    g(z<=0)=z(z<=0)*0.0;
    dg(z>0)=1.0;
    dg(z<=0)=0.0;
end
end
```

16. Prove that the derivative of the univariate softplus function in equation 4.33 is a sigmoid.
17. Prove that the derivative of the multivariate softplus function in equation 4.34 is the softmax function for one output node.
18. Create a large training set of data similar to that shown in Figure 4.21, except include

some mislabeled data. Tune the parameters (such as step length, starting model etc) of a multilayer NN to give reasonably good convergence and a small RMS error. Now take a holdout data set that was not used for training and use the trained coefficients in  $\mathbf{W}$  to classify each example in the holdout data set. Plot RMS error versus iteration number for both the training and test data. Is the error rate similar to that of the training data? Did you overfit the data? Was your training set sufficiently diverse to account for the characteristics of the holdout data? if you overfitted, what are the remedies?

19. From the previous problem, generate an RMS error versus network complexity graph by using deeper networks (or more nodes/layer) so you can compute a graph like that in Figure 4.27. Can you identify a critical point?
20. Compare the results of using a two-node MATLAB algorithm with the single-node algorithm. Which one is more tolerant of complexity and noise in the training examples.
21. In Exercise 4.7.12, how do you determine if you are stuck in a local minima? Hint: try different starting models. Show results.
22. Show that the derivative of the cross-entropy loss function

$$\epsilon = - \sum_{k=1}^K y_k \ln p_k, \quad (4.69)$$

is

$$\frac{\partial \epsilon}{\partial o_i} = p_i - y_i, \quad (4.70)$$

where the softmax function is

$$p_j = \frac{e^{o_j}}{\sum_{k=1}^K e^{o_k}}. \quad (4.71)$$

where  $K$  is the number of output nodes in the last layer.

23. For the data set from Exercise 4.7.18, test the convergence rate's sensitivity to different types of weight initialization for  $w_{ij}$ : a) random normalized weights of  $w_{ij}$  between -1 and 1, b) constant weights with value  $w_{ij} = 1$  for all weights, c)  $w_{ij} = 0$  for all weights.
24. An objective function with just one minimum is a convex function (Loiseau, 2020). The condition for this is that the Hessian matrix  $\mathbf{H}$  is positive definite  $\mathbf{H} \mathbf{a} \mathbf{a}^T \geq 0$  for any vector  $\mathbf{a}$ . The first step is to show that the following objective function is convex:

$$\epsilon = -\frac{1}{m} [\mathbf{y}^T \ln \sigma(\mathbf{X}\mathbf{w}) + (\mathbf{1} - \mathbf{y})^T \ln(-\mathbf{X}\mathbf{w})], \quad (4.72)$$

where  $m$  is the number of training examples and

$$\nabla_{\mathbf{w}}\epsilon = -\mathbf{X}^T\mathbf{D}\mathbf{X}. \quad (4.73)$$

Each row of  $\mathbf{X}$  is one of the training examples and each element of  $\mathbf{y}$  is a label for a training example. Then show that the Hessian matrix is

$$\mathbf{H} = \mathbf{X}^T\mathbf{D}\mathbf{X}, \quad (4.74)$$

where

$$\mathbf{D} = \frac{1}{m} \text{diag}[\sigma(\mathbf{X}\mathbf{w})(1 - \sigma(\mathbf{X}\mathbf{w}))]. \quad (4.75)$$

The second step is to show that  $\mathbf{w}^T\mathbf{H}\mathbf{w} \geq 0$ .

## 4.8 Computational Labs

The binary classification lab is in `/LAB1/Chapter.Book.FCN/Chapter.Gradient.Neural/lab.html`. After downloading and unzipping `Chapter.Gradient.Neural.zip`, execute `lab.html`. You will run the MATLAB code `NNode1.m` to get the results similar to those seen in Figures 4.13, 4.15, 4.17, and 4.18.

The gradient validation of the NN binary classification code is in `/LAB1/Chapter.Book.FCN/Chapter.Gradient.Neural/lab.html`. The MATLAB code `NNode_test.m` generates Figure 4.26.

The multiclass classification lab is in `/LAB1/Chapter.Book.FCN/Chapter.MultiGNN/lab.html`. After downloading and unzipping `./FNN_multiclass_code.zip`, execute `lab.html`. Now run the MATLAB code `NNode_multiclass.m` to get the results similar to those seen in Example 4.4.2.

## 4.9 Appendix: Neural Network Workflow

The workflow for finding the optimal model parameters  $w_{ij}^{[n]}$  for  $n \in \{1, 2, \dots, N\}$ ,  $i \in \{1, 2, \dots, I\}$ ,  $j \in \{1, 2, \dots, J\}$  that solve the *supervised* neural network problem consists of three steps.

1. **Training.** A large training set of labeled data pairs  $(\mathbf{x}^{(k)}, \mathbf{y}^{(k)})$  are obtained, and around 80% are randomly selected for training while the rest are selected as the set of *validation data*. Computing the optimal values of  $w_{ij}$  that minimize equation 4.10 is also known as training. The optimal model parameters  $w_{ij}^{[n]}$  are found by an iterative optimization method, e.g. equation 4.11. The training set should be comprehensive enough to include characteristics seen in all relevant data outside the training set. *If our trained NN model can accurately predict out-of-the-training data then we say that the NN is well generalized.*

How much data is needed for training? According to [https:// en.wikipedia.org/ wiki / Curse\\_of\\_dimensionality](https://en.wikipedia.org/wiki/Curse_of_dimensionality): *There should be many more data examples than the to-*



tal number of model dimensions. A typical rule-of-thumb is that there should be at least 5 training examples for each dimension in the representation (Koutroumbas and Theodoridis, 2008). Ng and Jordan (2013) suggest the folk wisdom, i.e., the number of input examples should be linearly related to the number of unknown model parameters. This has some theoretical basis for many models, where the Vapnik–Chervonenkis (VC) dimension is linear or at most some low-order polynomial in the number of model parameters. In other words, the sample complexity for the NN is linear in the VC dimension (Vapnik, 1995).

With a fixed number of training samples, the predictive power of a classifier or regressor first increases as the number of dimensions or features used is increased but then decreases (Trunk, 1979), which is known as Hughes phenomenon (Hughes, 1968) or peaking phenomena (Koutroumbas and Theodoridis, 2008). For example, if 5 points distributed along the real line were considered sufficient to represent the data density, then in  $N$  dimensions we need  $5^N$  points with sufficient density. If the input vector  $\mathbf{x}$  consist of the brightness intensity values in the  $N = 10^6$  pixels of a digital picture, then according to the rule-of-thumb there should be  $5^{10^6}$  training examples! This is too pessimistic of a rule-of-thumb because the data points mostly exist on a much lower-dimensional manifold. Nevertheless, it is not surprising that the most *labor-intensive task in supervised learning* is often labeling of the large set of training data. The amount of data needed for training is often a trial-and-error exercise.

For logistic regression, a widely used rule of thumb, the "one in ten rule", states ([https://en.wikipedia.org/wiki/Logistic\\_regression](https://en.wikipedia.org/wiki/Logistic_regression)) that *logistic regression models give stable values for the explanatory variables*<sup>12</sup> if based on a minimum of about 10 events per explanatory variable (EPV); where event denotes the cases belonging to the less frequent category in the dependent variable. If the network has  $10^6$  explanatory variables, this suggests that  $O(10^7)$  training examples are needed.

2. **Validation.** The accuracy of the computed model parameters in  $\mathbf{W}$  are tested<sup>13</sup> on the validation data  $f(\mathbf{x}, \mathbf{W}) \approx \mathbf{y}$ , where  $(\mathbf{x}, \mathbf{y})$  belong to the validation set, which is a random selection of about 20% of the labeled data (see Figure 2.8). A satisfactory model is attained if the prediction accuracy for the validation data is similar, i.e. within a few percent, to that for the training data. Otherwise, a more comprehensive data set and/or a better model architecture is needed.

The value of the damping term  $\lambda > 0$  in equation 4.10 determines which objective is more important, reducing the data misfit or the regularizing penalty term. Figure 4.27 illustrates how increasing the complexity of the neural network to the right of the critical point (red asterisk) leads to overfitting where the accuracy of the test-data predictions starts to decrease. As an example, the overfitted curve in Figure 4.28b fits almost every wiggle in the training data because the model complexity is too high.

<sup>12</sup>The explanatory variables are the coefficients  $w_{ij}$  that predict the given data.

<sup>13</sup>Here,  $\mathbf{W}$  corresponds to the coefficients in  $\mathbf{W}^{[n]}$  for all of the layers.

As discussed in section 2.1.6, the tendency to overfit data can be reduced by using a larger value of the regularization parameter  $\lambda > 0$  or by reducing the model complexity with fewer layers and/or nodes in the network. However, if the model is too simple then it underfits the data as illustrated by the simple linear model in Figure 4.28a. Choosing the best model that neither overfits or underfits is sometimes known as the bias-variance tradeoff, where increasing the bias (see Exercise 4.7.6) will decrease the variance<sup>14</sup> or vice versa (Bishop, 2006). As shown in Figure 4.28c, the well-balanced model fits the actual pattern in the data and largely ignores the noise. Such models tend to generalize well to holdout data if its main features are contained in the training data.

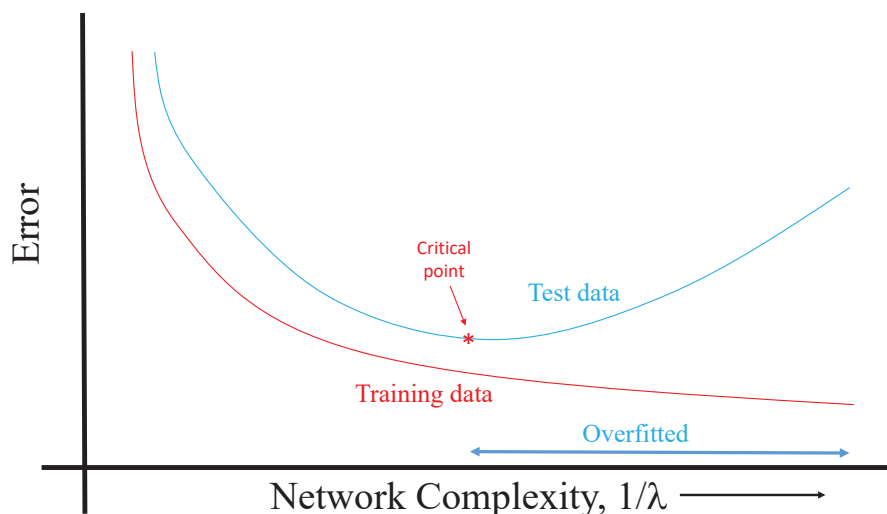


Figure 4.27: Prediction accuracy of the training and test data versus complexity and  $1/\lambda$  of the NN. If the network is too complex, as we move to the right of the red critical point, then it begins to memorize (Arpit et al., 2017; Radhakrishnan et al., 2019) both noise and signal in the input images. The consequence is that the NN can have difficulty in accurately predicting the class type of new input data. Increasing the value of the regularization parameter  $\lambda$  and/or decreasing the number of nodes or layers tends to mitigate memorization problems at the possible loss of accuracy.

One type of a robust training procedure is cross-validation, which resamples the labeled data and trains on different subsets of it. Cross-validation randomly divides your training data into, say, 10 subsets  $\mathbf{D}^{(s)}$  for  $s \in \{1, 2, \dots, 10\}$ , holding out each subset

<sup>14</sup>Variance is the variability of the model's prediction of output target values, which informs us about the spread of our predictions. Models with high variance tend to overfit, i.e. memorize (Arpit et al., 2017; Radhakrishnan et al., 2019), training data and cannot easily be generalized to accurately fit *holdout* or *test* data. Small changes in the holdout data lead to large changes in the predictions for an overfitted model.

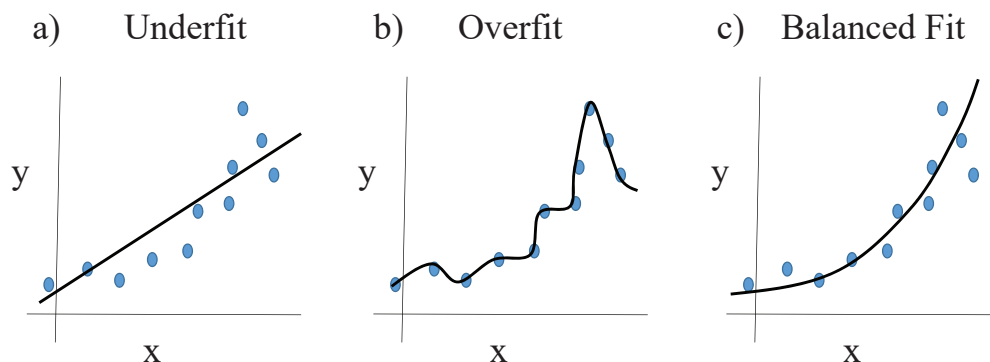


Figure 4.28: Plots of a) underfit ted, b) overfitted and c) well-fitted  $(x, y)$  points.

while training on the other 9 subsets, testing each learned classifier  $\mathbf{W}^{(s)}$  on the examples it did not see. The RMS errors from all ten trained classifiers  $\mathbf{W}^{(s)}$  are averaged to get an accurate prediction of NN performance ([https://en.wikipedia.org/wiki/Cross-validation\\_\(statistics\)](https://en.wikipedia.org/wiki/Cross-validation_(statistics))). In some cases, the  $\mathbf{W}^{(s)}$  with the best performance is used for the holdout data (<https://www.datarobot.com/wiki/cross-validation/>). As an example of best performance for classification, assume that ten models can be used to make predictions, and the final class type for an input is by majority vote. For regression the final output is by averaging the output values at an output node (Krogh and Vedelsby, 1994).

3. **Inference.** Once properly trained to get the optimal coefficients  $\mathbf{W}^{[n]}$  for  $n \in \{1, 2, \dots, N\}$ , the model is used to predict the output  $\mathbf{y}^{pred}$  from unlabeled data  $\mathbf{x}$ . Accurate predictions are achieved if the NN has been well generalized so that  $\mathbf{W}^{[n]}$  has captured the hidden patterns from a large set of training examples. However, even if the network is over-parameterized, Arpit et al. (2019) state the following: “*Yet deep neural networks (DNNs) often achieve excellent generalization performance with massively over-parameterized models. This phenomenon is not well-understood.*”.

## 4.10 Appendix: MATLAB Code for Validation of the NN Gradient

**Code 4.10.1.** *MATLAB Code Fragment for Validating a 2-layer NN Gradient (Zongcai Feng).*

```

for obj=1:2
    for act=1:2

        % disp(['obj = ',num2str(obj),', act = ',num2str(act),]);

        [grad,~]=gradientnn(M,x,t,ww,layer_size,obj,act);

        grad_numel=grad;
        ww_numel=ww;

        e = 1e-4;
        for ilayer=1:layer_num-1
            grad_numel{ilayer}=grad{ilayer}*0.0;
            for irow=1:layer_size(ilayer+1)
                for icolumn=1:layer_size(ilayer)+1
                    ww_numel=ww;
                    ww_numel{ilayer}(irow,icolumn)=ww{ilayer}(irow,icolumn)-e;
                    [~,res1]=gradientnn(M,x,t,ww_numel,layer_size,obj,act);
                    ww_numel{ilayer}(irow,icolumn)=ww{ilayer}(irow,icolumn)+e;
                    [~,res2]=gradientnn(M,x,t,ww_numel,layer_size,obj,act);
                    grad_numel{ilayer}(irow,icolumn) = (res2 - res1) / (2*e); %numerical gradient

                    disp(['Analytic grad = ',num2str(grad{ilayer}(irow,icolumn)),...
                        ', Nume grad = ',num2str(grad_numel{ilayer}(irow,icolumn)), ...
                        ', Diff = ',num2str(grad{ilayer}(irow,icolumn)-grad_numel{ilayer}(irow,icolumn))]);
                end
            end
        end

        end

        diffgrad1=(grad{1}(:,:)-grad_numel{1}(:,:))/max(abs(grad{1}(:)));
        diffgrad2=(grad{2}(:,:)-grad_numel{2}(:,:))/max(abs(grad{2}(:)));
        figure(2);
        subplot(121);imagesc( diffgrad1);colorbar;
        subplot(122);imagesc( diffgrad2);colorbar;
        figure(1)

```

## Chapter 5

# Processing Geophysical Data with Fully-Connected Neural Networks

We now present several examples where a fully-connected neural network (FCNN) or a support vector machine method (see Chapter 7) is used to process geophysical data. The first two examples will be for filtering artifacts from migration images and the third example will be for designing windows in the frequency-wavenumber (FK) domain so that the fundamental-mode spectrum is separated from that of the higher-mode arrivals. Readers not familiar with seismic imaging are referred to Appendix 27.

### 5.1 FCNN Filtering of Migration Images in $\mathbf{z} = (x, z)$

Seismic migration (see Appendix 27.2) is an imaging method that estimates the subsurface reflectivity from seismic data (Yilmaz, 2001); the final result is denoted as the migration image  $m(\mathbf{z})$  where  $\mathbf{z} = (x, z)$ . For the example in Figure 5.1a, seismic data recorded on the top horizontal surface are migrated to give the large amplitude values of  $m(\mathbf{z})$ , which denote the locations of the fan-like reflecting interfaces. Some of the large amplitudes are artifacts from the migration method and do not indicate an actual reflector. The NN and support vector machine (SVM) methods will now be used to denoise the migration images computed from synthetic seismic data. A tutorial for seismic migration is in Appendix 27.2 and more details are in Yilmaz (2001). The SVM method, described in Chapter 7, is closely related to that of a NN in that it finds the best decision surface that separates one class of objects from another.

The goal is to use a fully-connected neural network to distinguish noise from the reflection signals in the Figure 5.1a migration image. As discussed in Appendix 27.2, the elliptical features are typically due to aliasing artifacts in the migration image and can be identified as noise by a geophysical interpreter. If the geophysicist can assign local labels to a representative set of noisy migration images, then these data can be used to train a NN to distinguish noise from signal. The trained NN can then be used to suppress coherent artifacts in test migration images.

The workflow for NN identification of migration noise consists of the following steps, where the migration amplitude  $m(\mathbf{z}_i)$  at the  $i^{th}$  pixel in Figure 5.1a is denoted as  $m_i$ .

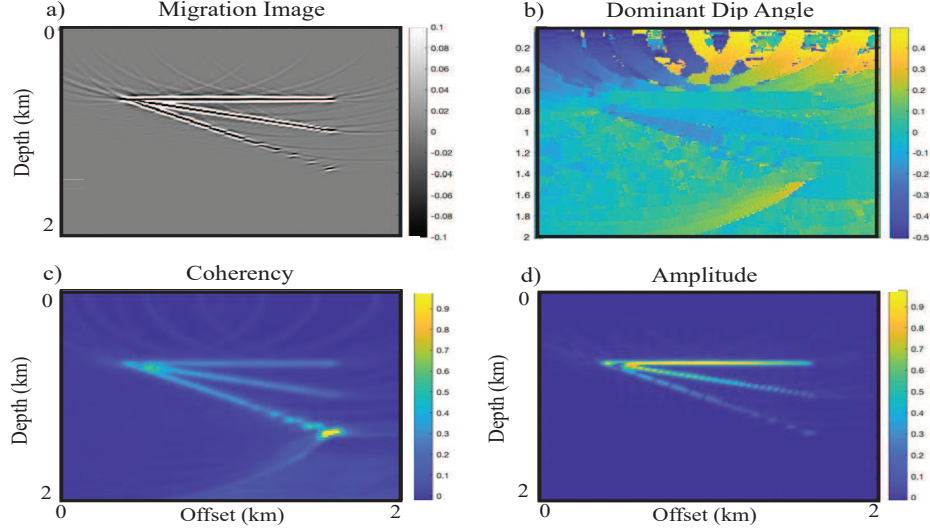


Figure 5.1: Images of the a) raw migration section  $m_i$ , b) dominant dip angle  $\theta_i$ , c) migration coherency  $\phi_i$ , and d) envelope amplitudes  $A_i$  of the migration image for all pixels  $i \in \{1, 2, \dots, N\}$ ; the  $A_i$  is related to the amplitude of the trace's Hilbert transform (see Appendix 5.8). Here, the migration image is a  $201 \times 201$  digital image so  $N = 201 \cdot 201 = 40,401$ . Colorbar values are normalized.

- **skeletonized Features in  $m(\mathbf{z})$ .** To train the neural network, the first step is to define a small  $5 \times 5$  window centered at the  $i^{th}$  pixel. Three skeletonized characteristics are then computed for the image in this small window: dominant dip angle, coherency and maximum amplitude as described in Appendix 5.8. The human interpreter classifies selected pixels in the training migration image as either noise or signal. The computer computes the three skeletal characteristics  $(x_1^{(i)}, x_2^{(i)}, x_3^{(i)})$  at each of the selected pixels in  $m(\mathbf{z})$  and assigns the classification value  $y^{(i)}$ . Therefore, the  $i^{th}$  input vector<sup>1</sup> is the  $4 \times 1$  vector  $(1, x_1^{(i)}, x_2^{(i)}, x_3^{(i)})$  with the label  $y^{(i)}$ :  $y^{(i)} = 1$  for signal, otherwise  $y^{(i)} = 0$ . For the migration examples, only 70 image pixels (0.17 % of original image) are selected for training from the  $201 \times 201$  migration image  $m(\mathbf{z})$  in Figure 5.1a.

What are the optimal skeletal characteristics of  $m(\mathbf{z})$  that can distinguish noise from signal? The answer is to use intuition and experience to identify the most significant characteristics that distinguish noise from signal. For example, we recognize that migration artifacts should have weak coherency  $\phi_i$ , large dip angles  $\theta_i$ , and weak amplitudes  $A_i$  at the  $i^{th}$  pixel, while the reflection signal should mostly be characterized by strong coherency, modest dip angles, and strong amplitudes. As an example, Figure 5.1 plots the  $\phi_i$  values at all pixels where weak coherency is indicated by the deep-blue color. Therefore we use  $(\theta_i, \phi_i, A^{(i)})$  as the skeletonized characteristics of

<sup>1</sup>The  $i^{th}$  superscript in  $\mathbf{x}^{(i)}$  denotes the  $i^{th}$  example, and in this case also denotes the  $i^{(th)}$  pixel in the image.

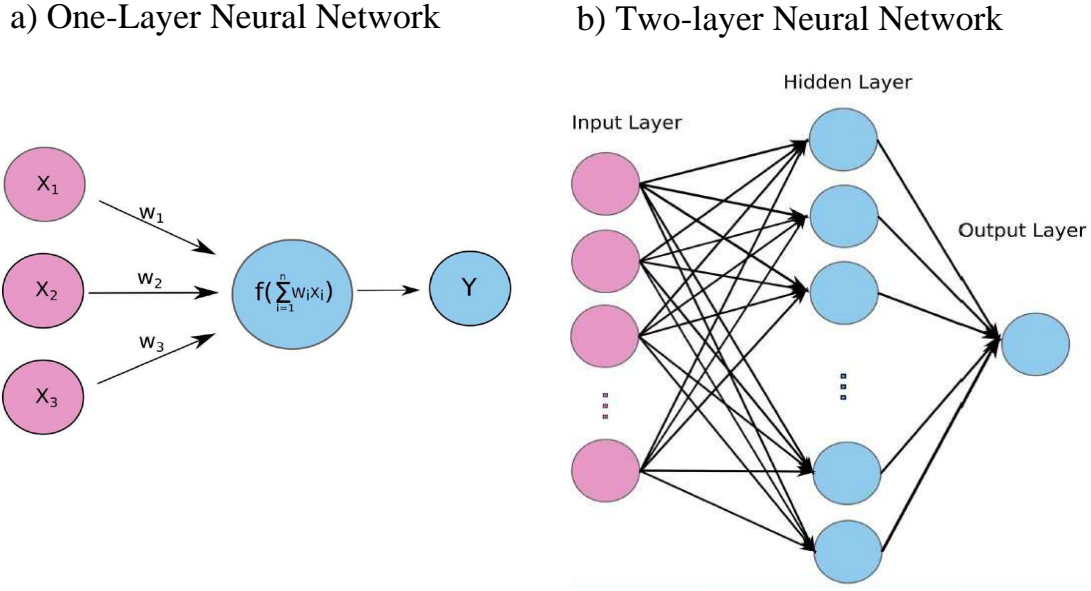


Figure 5.2: Neural network architectures for a) one- and b) two-layer neural networks. The one-layer one-node network diagram is for the logistic network and the two-layer multi-node architecture is for the neural network used in the examples.

the migration image to distinguish noise from signal. Other characteristics might be more effective and should be used if needed for better accuracy.

- NN Training.** The input training pair is now defined as  $(\mathbf{x}^{(i)}, y^{(i)}) = (1, \theta_i, \phi_i, A^{(i)}, y^{(i)})$  for 70 points in the migration image in Figure 5.1a. The value 1 is assigned to the first element of  $\mathbf{x}^{(i)}$  because it is associated with the bias term in the NN. The  $i^{th}$  example, which is also the specified  $i^{th}$  pixel, in the training set is manually labeled as either signal  $y^{(i)} = 1$  or noise  $y^{(i)} = 0$ . There will be two neural-network models, one of them will consist of a single hidden layer with one node (see Figure 5.2a), and the other will be two layers (see Figure 5.2b), with the hidden layer consisting of 15 nodes. All of the layers use sigmoid-activation functions.
- Decision Surface.** The NN procedure is then used to find the weights of the NN (including the bias terms for each node) that cleanly label the noise and signal pixels. This generates a decision surface in the feature spaces shown in Figure 5.3a-5.3b. Here, the yellow-colored planes in a) and b) define the decision surfaces in the 3D feature space. On one side of this surface the points are labeled as  $y = 1$ , while the other side has  $y = 0$ . This decision boundary was found by using the trained neural network to predict the class type (either 0 or 1) for all points in the discretized 3D volume. Then the 2D surface can be found by inspection.
- Fully Connected Neural Network Inference.** Once the FCNN model parameters are found then the other parts of the migration image (i.e., out-of-the-training-set data) are automatically labeled by inference. If the amplitude of the automatically

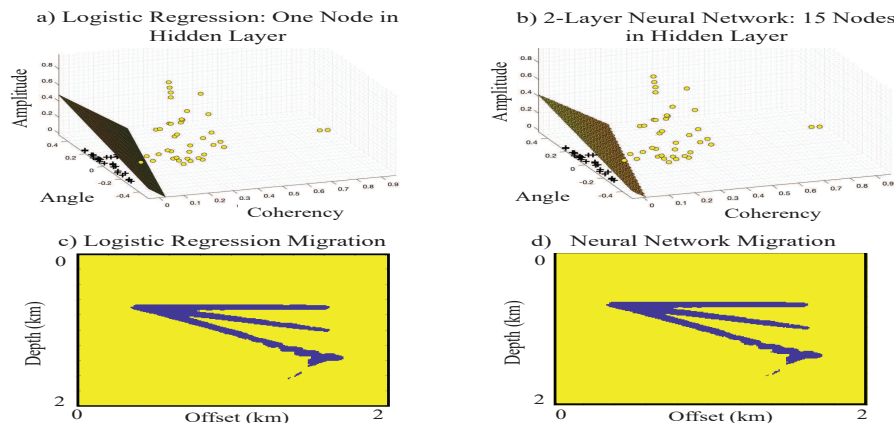


Figure 5.3: Decision boundaries computed by a) single-layer logistic regression and a b) two-layer NN with 15 nodes in hidden layer. The corresponding decision areas for the migration images are directly below the feature plots where yellow denotes noise and blue is signal. The yellow circles in a) and b) denote signal and the black '+' symbols denote noise.

labeled image is noise then that amplitude can later be filtered by some method such as muting. The automatically labeled features are shown in Figure 5.3c-5.3d, where yellow indicates noise and dark blue indicates signal. Here, the amplitudes classified as noise are suppressed by muting in the post-stack image and the resulting migration images are shown in c) and d).

Instead of a linear model  $\mathbf{w}^T \mathbf{x}$  inside the argument of the activation function, we can employ a non-linear model such as the example shown in Figure 2.7. That is, if the skeletal data is  $\mathbf{x} = (1, x_1, x_2, x_3)$  at each pixel<sup>2</sup> then our adjusted input data can be the  $9 \times 1$  feature vector

$$\mathbf{x} = (1, x_1, x_2, x_3, x_1^2, x_2^2, x_3^2, x_1x_2, x_1x_3, x_2x_3), \quad (5.1)$$

compared to the  $4 \times 1$  input data vector used for Figure 5.3. The results are shown in Figure 5.4a, where the decision boundary has more curvature than those shown in Figure 5.3. However, there is not much of a difference in the filtered migration images. As a comparison, the support vector machine results (see Chapter 7) are shown in Figure 5.4b and 5.4d. Here, the decision boundary is along a non-linear surface, as expected. The problem with this simple example is that the strongest amplitudes usually distinguished noise from signal. This is typically not the case with realistic seismic data and their associated migration images.

<sup>2</sup>The pixel superscript ( $i$ ) is silent in this example.



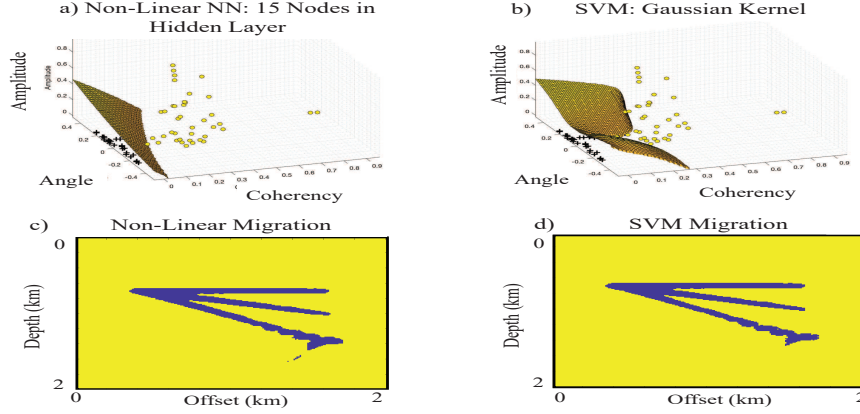


Figure 5.4: Similar to the previous figure except the input at each pixel is the  $10 \times 1$  feature vector in equation 5.1. For comparison, the support vector machine (SVM) results are shown in b) and d). In both cases the decision boundary is not along a plane.

## 5.2 Filtering of Migration Images in the Angle Domain

One of the problems with the above NN strategy is conflicting dips, where signals intersect the coherent noise in the migration image  $m(\mathbf{z})$ . Undesirably, muting the noise at the noise+signal intersection can also kill useful signal. A better approach is to transform the traces into a domain where the signals separate from the coherent noise. In this new domain, NN training and filtering can be done to eliminate the noise; here, filtering is nothing more than muting the noise separated from the signal. Then the filtered data are inverse transformed to get the filtered migration image in the  $x - z$  domain. Instead of explicitly using NN, we will use the support vector machine to distinguish noise from signal in the angle domain. The reason for doing this is that the previous section showed that the NN with several layers had the same capability as that of the SVM.

### 5.2.1 Angle Domain Transform

The transform that separates different dipping events in the migration image is a local slant stack applied to each migration trace in the extended domain image  $m(H, x, z)$  to get  $\tilde{m}(\theta, x, z)$ , which is denoted the dip-angle angle-domain common image gather (ADCIG) (see lower left *Dip Gather* image in Figure 5.5 and Chen et al. (2020)) for a specified value of the  $x$  coordinate. Here,  $H$  is the subsurface offset variable. The ADCIG is displayed for  $-30^\circ \leq \theta \leq 30^\circ$  and  $0 \leq z \leq z_{max-depth}$  at the fixed offset coordinate  $x$ . Three feature panels can be extracted from the ADCIG for SVM training: coherency, amplitude, and local dip-angle feature panels. Similar to training in the previous examples, a small set of selected pixels and a few features at each selected pixel in the ADCIG domain are defined,

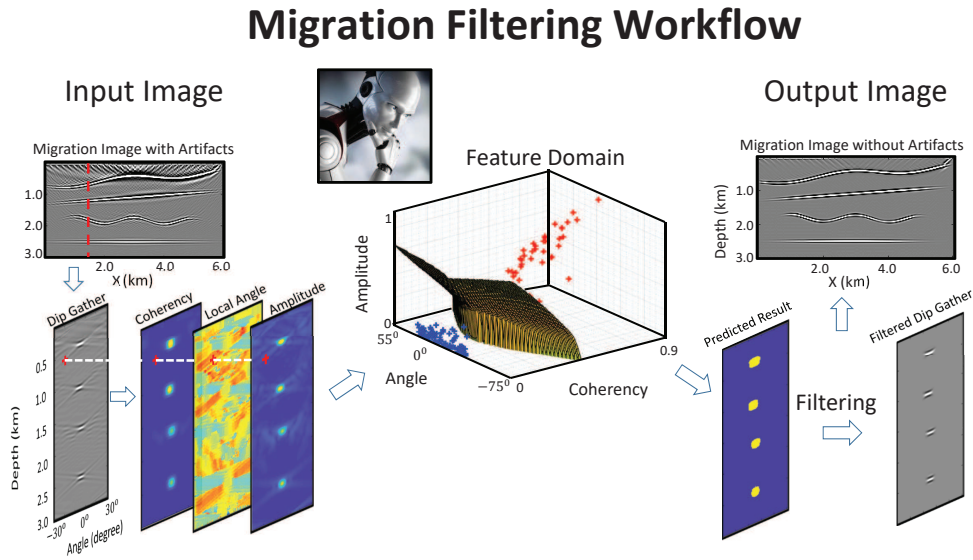


Figure 5.5: Workflow for SVM (or NN) applied to ADCIG's computed from the red migration trace (and a few neighboring traces) in the upper-left panel. Three feature images are extracted from each ADCIG, the noise and signals are labeled, and the SVM is trained. The trained SVM is then used to label pixels in the new ADCIGs, and then the pixels labeled as noise are muted (Chen et al., 2020).

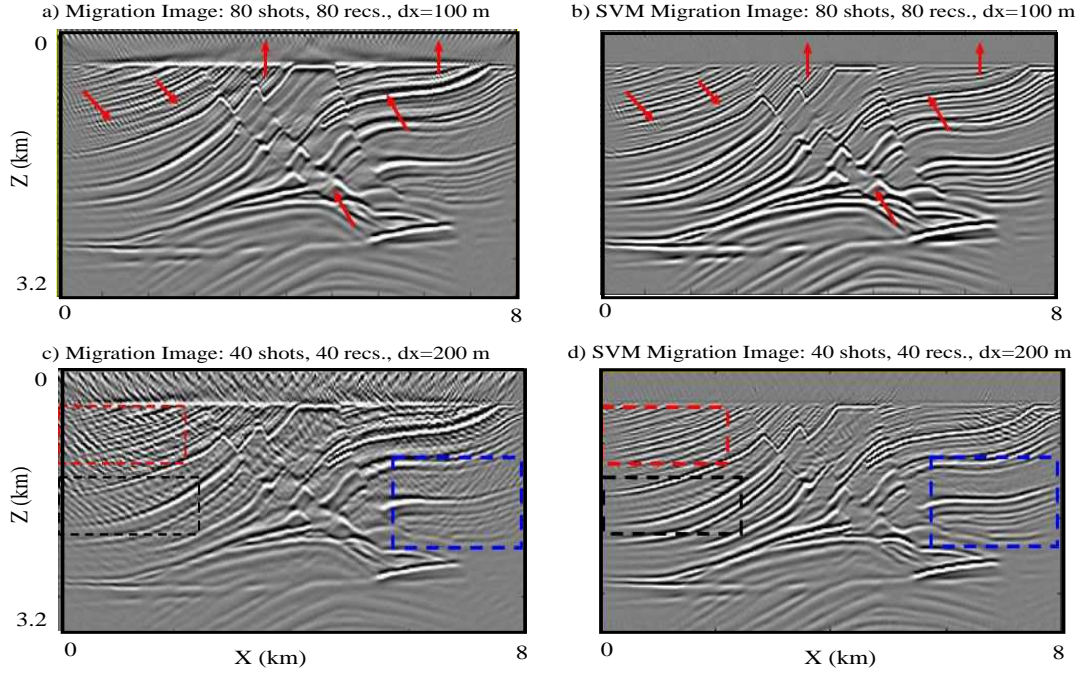


Figure 5.6: Migration images a), c) before and b), d) after SVM filtering of the ADCIGs. The top and bottom rows of migration images were computed from aliased data traces with source and receiver intervals of 100 m and 200 m, respectively (Chen et al., 2020).

and the selected pixel is labeled as either noise or signal. This allows for muting filters to be applied to the data in the ADCIG domain, which are then inverse transformed back into the  $x - z$  domain of the migration image (Chen et al., 2020). An example is shown in Figure 5.6 where aliasing artifacts in the a) and c) migration images are largely eliminated after filtering in the b) and d) images.

### 5.2.2 Filtering Angle Domain Migration Images from Marine Data

Filtering of migration images is next applied to marine data from the North Sea (Chen et al., 2020). Figure 5.7 shows a vertical slice of the 3D migration image (top) before and (bottom) after SVM filtering of the migration image, which used a Gaussian kernel. We will only show the results for the SVM and not the NN with two layers (50 nodes in the 1st layer and 25 in the second) because the synthetic tests showed them to be almost identical in performance.

In the training phase, noise is labeled in the angle-domain gathers of Figure 5.5, and the network is then trained over a small set of  $4 \times 1$  feature vectors. The top image in Figure 5.7 has many aliasing artifacts that show up as jittery interfaces at the shallow depths  $0 < z < 1.5$  km. The filtered image is the bottom image and shows that they are largely eliminated by SVM filtering.

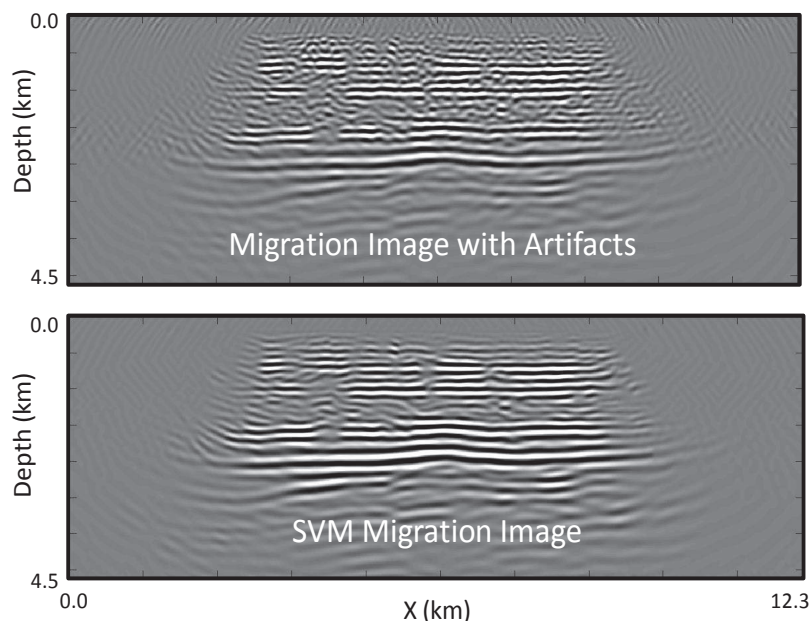


Figure 5.7: Migration images from the North Sea (top) before and (bottom) after SVM filtering of ADCIGs with a Gaussian kernel.

### 5.3 NN Filtering of Surface Waves

The Rayleigh waves in the common shot gather (see Appendix 27.3) displayed in Figure 5.8b can be inverted to get an estimate of the S-velocity model  $v(x, z)_s$  in Figure 5.8a. Such models can be very useful for assessing the strength of near-surface soil and rocks (Yilmaz, 2001). The Rayleigh waves in Figure 5.8b have the largest amplitudes and propagate with an apparent horizontal velocity that is a weighted average of the shear velocity to a depth of about  $1/3$  to  $1/2$  of a wavelength. Wavelength decreases with increasing frequency so the higher (lower) frequencies are affected by the S velocities in the shallower (deeper) portions of the medium.

To invert for  $v(x, z)_s$ , the common shot gather  $d(x, t)$  on the leftside of Figure 5.9 is Fourier transformed along both the  $x$  and  $t$  axes to get the magnitude spectrum  $\tilde{D}(k, \omega)$  shown on the rightside of Figure 5.9. The standard strategy is to either pick the fundamental dispersion curve<sup>3</sup>  $\kappa(\omega)$  in the  $\tilde{D}(k, \omega)$  spectrum and invert it for  $v(x, z)_s$ , or transform this curve to phase velocity  $C(\omega)$  and invert this for the shear-velocity model. The problem is that the higher-order modes can interfere with accurate picking of the fundamental dispersion curve, especially if an auto-picker is used. Another problem is that manual, or semi-manual, picking of dispersion curves for a large number of shot gathers requires an enormous amount of manual labor.

To avoid such problems, the strategy will be to define a window in the  $k - \omega$  domain

---

<sup>3</sup>The fundamental dispersion mode can be conveniently picked because it has the largest magnitude spectrum in the  $k - \omega$  domain.

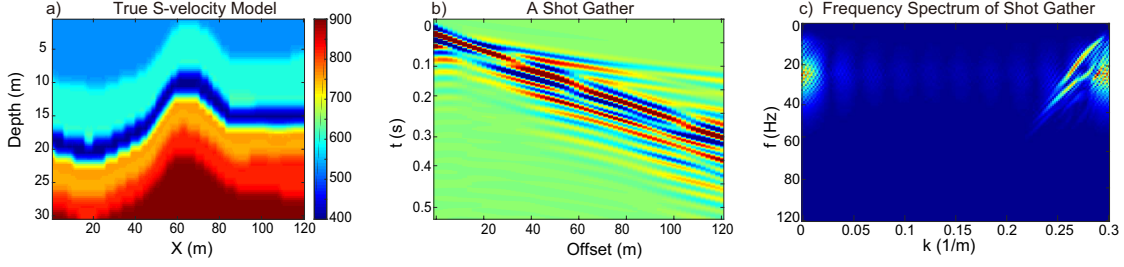


Figure 5.8: a) True S-velocity layer model, b) a shot gather and c) the frequency spectrum of the raw shot gather. The Rayleigh waves in the CSG are the strongest amplitude arrivals denoted as the dark red-blue events which intersect the time axis at around 0.3 s and the offset of 120 m. From Li et al. (2020).

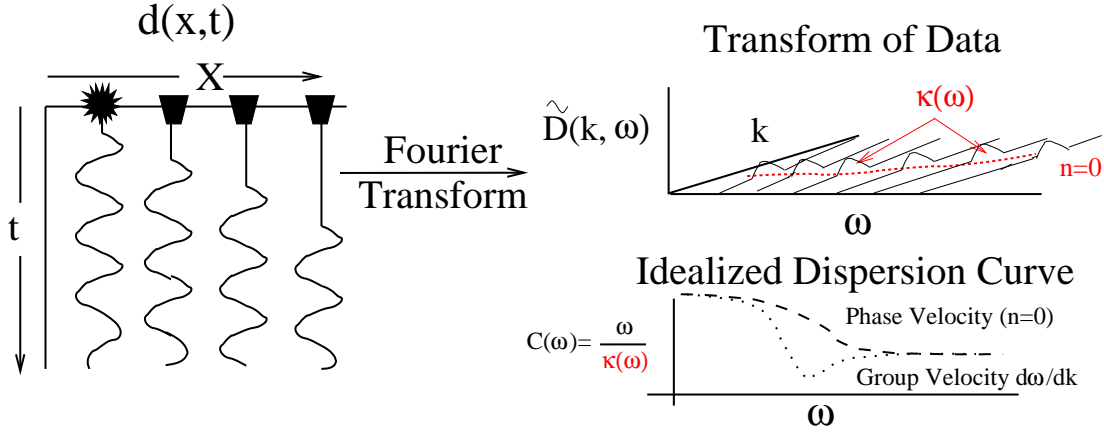


Figure 5.9: Common shot gather  $d(x, t)$  on the left and the fundamental ( $n=0$ ) dispersion curves for Love waves in the (top right)  $k - \omega$  and (bottom right)  $C(\omega) - \omega$  domains. Here the phase velocity is  $C(\omega) = \omega / \kappa(\omega)$  and  $\kappa(\omega)$  is the picked dispersion curve, i.e. the skeletonized data (Li et al., 2017).

that separates the fundamental from the higher-order mode energy. All energy outside this window is muted so that the autopicker can more easily identify the fundamental dispersion curve. To automatically create this window, we will train a neural network to define it in the  $k - \omega$  domain, and then apply it to out-of-the-training set data to isolate the fundamental-mode energy in  $\tilde{D}(k, \omega)$ . A standard auto-picker can then be used to pick  $\kappa(\omega)$ , which can then be inverted for  $v(x, z)_s$ .

Figure 5.11 illustrates the workflow for NN identification of the window that only contains fundamental-mode energy in  $\tilde{D}(k, \omega)$ . The workflow consists of the following steps.

- **skeletonized Features of  $\tilde{D}(k, \omega)$ .** Three features that will be used to identify the window for the fundamental-mode energy in  $\tilde{D}(k, \omega)$  are coherency, local-dip angle, and frequency range as illustrated in Figure 5.10 and described in Appendix 5.8. The brown window in the bottom right of Figure 5.11 partly coincides with the steeply dipping orange zone seen in Figure 5.10b. Local dip angle and coherency have been

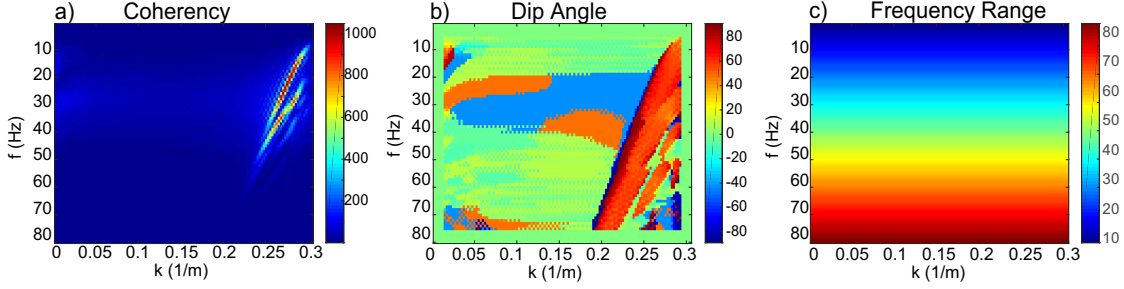


Figure 5.10: Three feature images associated with the magnitude spectra of a shot gather: a) amplitude coherency, b) dip angle by slant stacking, and c) the frequency range.

defined in the previous sections.

Frequency range is the third skeletonized feature selected for the fundamental dispersion window. The boundary of the orange slab in Figure 5.10b has shallow dip angles at low frequencies 5-12 Hz and steep dip angles at higher frequencies 30-80 Hz. This suggests that the frequency and wavenumber of the fundamental mode are correlated with one another, and roughly follow the pattern of the orange slab. To ensure that this correlation between dip and frequency is noticed by the NN, we also include the skeletal data denoted as the *frequency range* map in Figure 5.10c. It associates each pixel in the skeletal input image with the actual frequency in the magnitude spectrum. For example, comparing Figure 5.10c with Figure 5.10b allows the human observer to guess that the orange slab has steep dips at higher frequencies and shallow dips at lower frequencies. If the human can detect this correlation, so will the NN.

- **Training.** Training is similar to that for the migration filtering in section 5.1. The difference is that the three features are obtained from the magnitude spectrum of a shot gather, and the three feature maps are shown in the lower left of Figure 5.11. If the pixel in the  $k - \omega$  domain belongs to the Rayleigh-wave window, then that pixel is labeled  $y = 1$ . As an example, the window for the fundamental mode is shown on the lower right side of Figure 5.11 as a brown finger-like body. The model is trained until there is a clear separation boundary in the feature space that separates regions within (blue symbols in middle plot) and outside (orange symbols in middle plot) the fundamental mode window.
- **Decision Surface.** The decision boundary that separates the  $y = 1$  and  $y = 0$  points can be determined by finding the labels for every point in the middle plot and finding the decision boundary by inspection.
- **Inference.** Once the NN model is trained it is then applied to unlabeled magnitude spectra to determine the window associated with the fundamental modes. The fundamental curve can then be auto-picked more easily because there are no distracting higher-order modes. The auto-picked curves for the fundamental mode are then inverted using the procedure described in Li et al. (2017, 2020) to get  $v(x, z)_s$ .



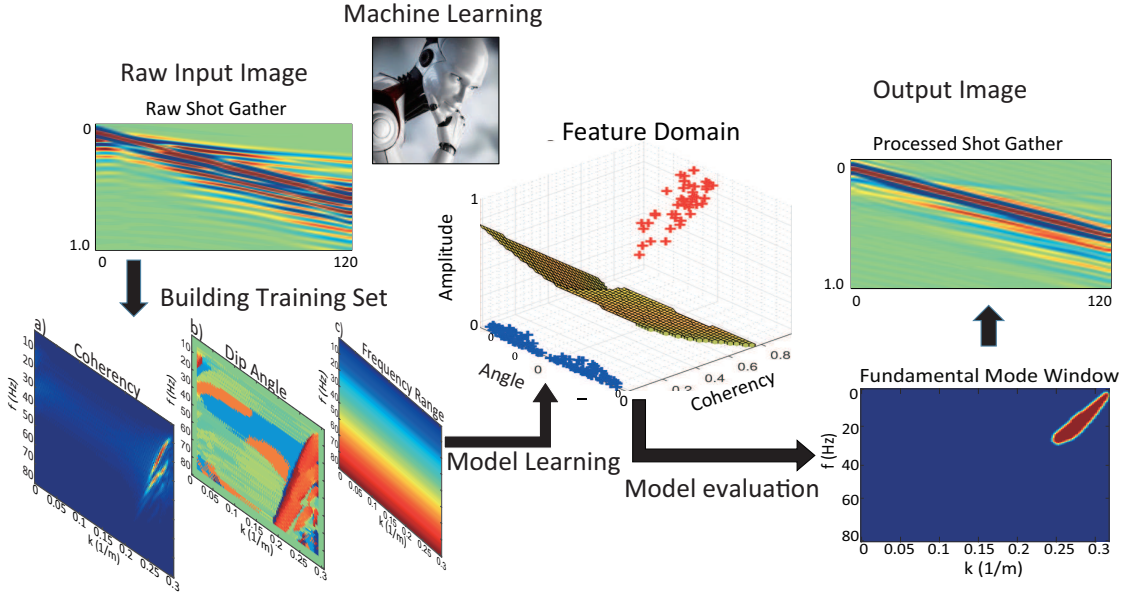


Figure 5.11: The classical machine learning workflow consists of four steps: data preprocessing, feature extraction, model learning and model evaluation.

As an example, the shot gather in Figure 5.8b is computed for a shot at the surface of the Figure 5.8a model. The P-velocity model is the same as the S-velocity model except that it is scaled by the multiplicative value of, e.g., 2. We will test four different machine learning models for binary classification: a single-hidden layer NN with one node in the hidden layer (also denoted as logistic regression), a single-layer NN with 10 nodes in the layer, a SVM with a Gaussian kernel, and a two-layer neural network with 50 nodes in the first layer and 25 nodes in the layer just before the output.

The frequency spectrum of the raw shot gather in Figure 5.8b is shown in Figure 5.12a. After training, the windowed spectra computed by logistic regression, single-layer NN, and SVM are shown in Figure 5.12b, 5.12c, and 5.12d, respectively. The window for the two-layer NN is almost identical to that for the SVM. Inverse transforming these windowed spectra gives the shot gathers in Figure 5.13. The 2-layer NN gives a shot gather almost identical to the SVM shot gather.

Which filtered shot gather in Figure 5.13 is correct? To partially answer this question we assume a three-layer approximation to the Figure 5.9a model, and analytically compute its fundamental-dispersion curve. The analytic dispersion curve is plotted as a dashed white line in Figure 5.14. It is seen that the fundamental-dispersion curve most closely follows the spectrum associated with the SVM window. The results for the SVM window are almost identical to those in Figure 5.15 computed by the 2-layer NN window.

## 5.4 MATLAB Codes

The migration images in Figure 5.1 are created by MATLAB codes, with the main code fragments given below. Before they are executed the user selects and labels  $y = 0$

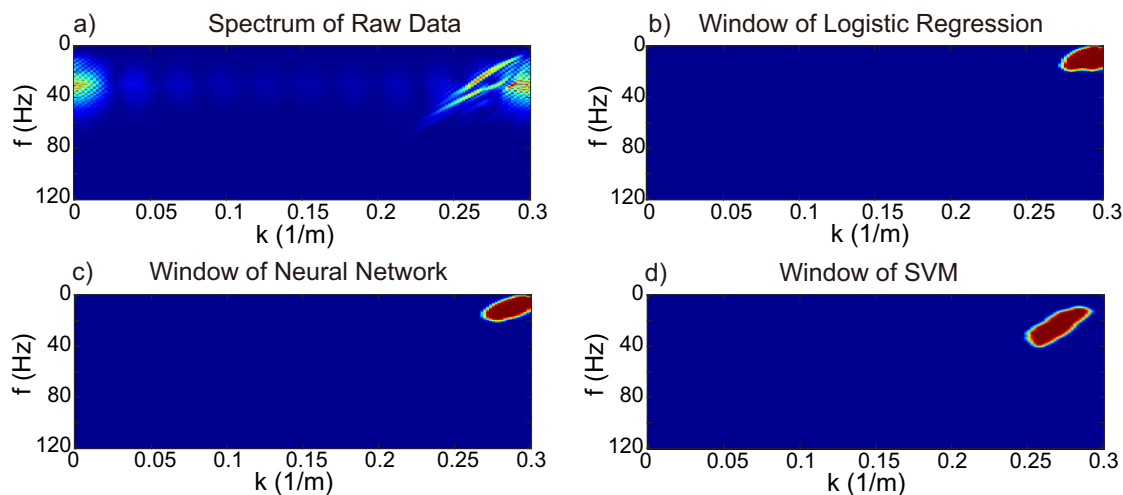


Figure 5.12: a) The frequency spectrum of the raw shot gather. Also, b), c) and d) show the output mute-window delineated by the logistic regression, neural network and SVM windows, respectively (Li et al., 2020).

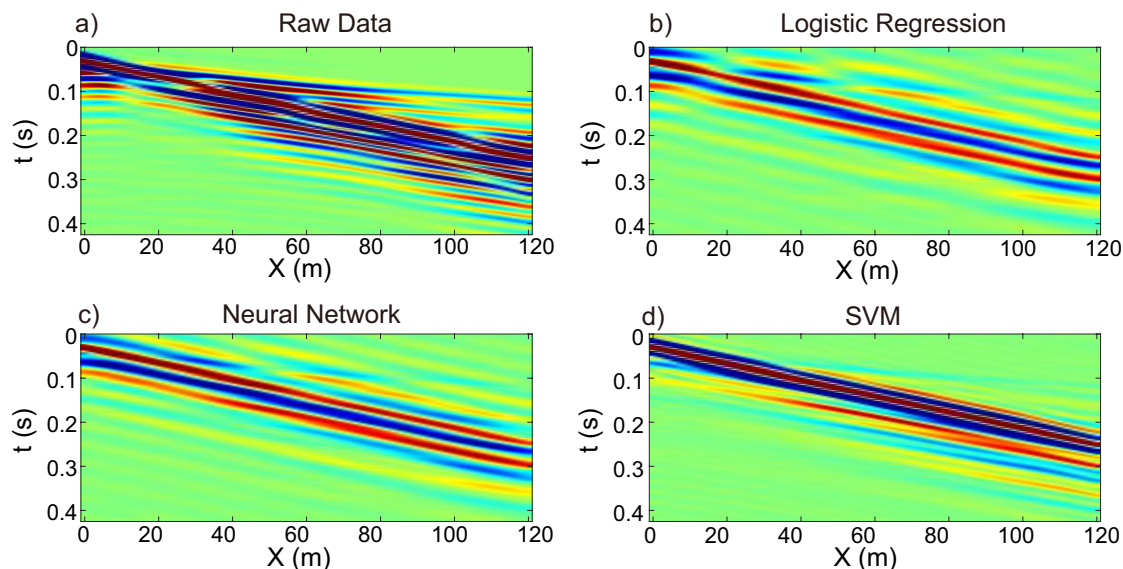


Figure 5.13: a) Raw shot gather. The b), c) and d) panels show the shot gathers filtered by the logistic regression, neural network and SVM windows, respectively. (Li et al., 2020).

for about 50 points in the migration image that are migration artifacts. About the same number of points are selected as reflectors with the label  $y = 1$ . These labeled points are then used to train the algorithms. This lab is in the Computational labs directory *LAB1/Chapter.Book.GNN/NNMigration/lab.html*. The main fragment of the logistic regression code is given in Appendix 5.9.



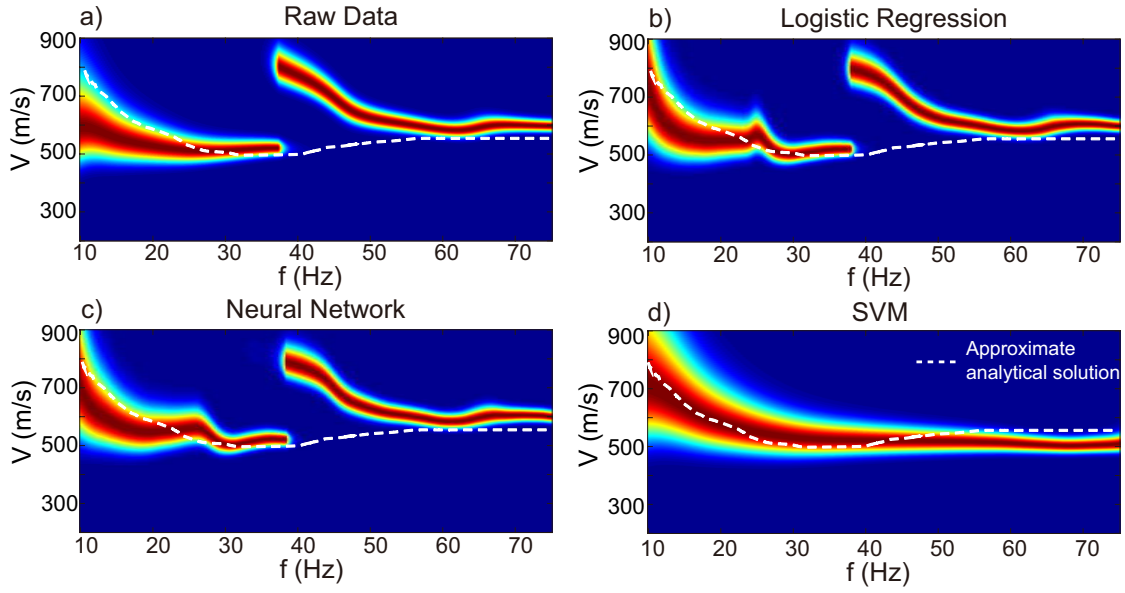


Figure 5.14: The dispersion curves of a) the raw shot gather and those filtered by the b) logistic regression, c) neural network, and d) kernel SVM windows.

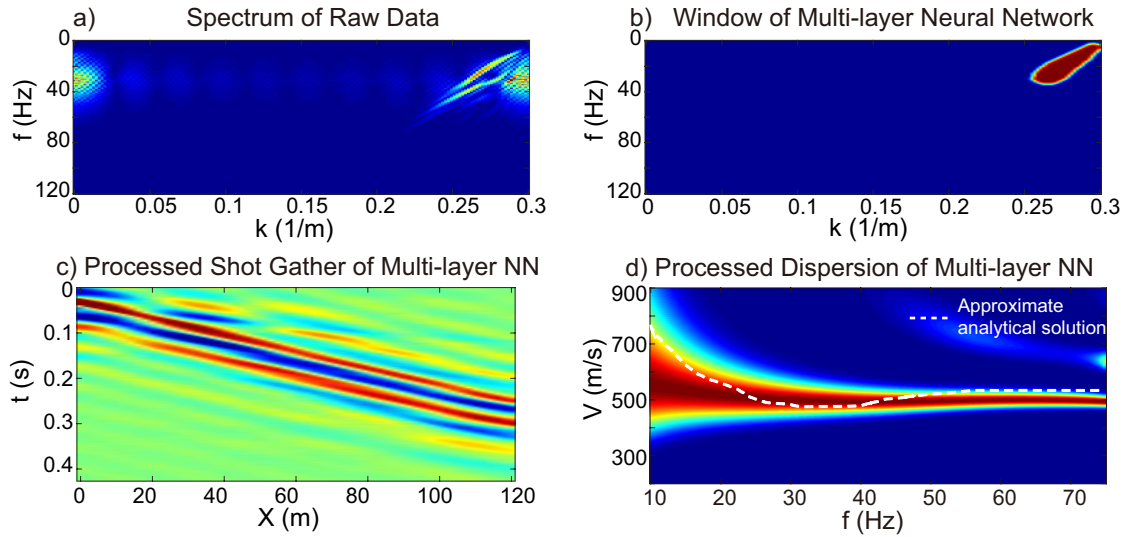


Figure 5.15: a) The frequency spectrum of the raw shot gather, b) the fundamental-dispersion curve window delineated by the two-layer neural network, c) the shot gather filtered by the multilayer NN and d) the dispersion curve of the multilayer NN processed data.

## 5.5 Summary

Geophysical examples are presented for processing seismic data by neural networks. In each of the cases, the raw data were skeletonized as features that were thought to distinguish

signal from noise. Sometimes this separation is best achieved by transforming to a domain where noise and signal are untangled from one another. The challenge is to use experience, trial-and-error, and physics-based intuition to identify the skeletal features in the data that strongly distinguish noise from data. MATLAB codes are provided in order to reproduce some of the examples, and they can be adjusted to tackled other geophysical problems.

## 5.6 Exercises

1. What other skeletonized features could be used to distinguish migration noise from signal? Adjust the code in `LAB1/Chapter.Book.GNN/NNMigration/lab.html` and try your new ideas.
2. Use the code in `LAB1/Chapter.Book.GNN/NNSurfaceWave/lab.html` to separate left and right going waves in a shot gather. Compare these results to those using conventional quadrant muting in the FK domain.
3. Subsample your traces in the shot gather at more than  $1/2$  of a horizontal wavelength, which means that the data are aliased. Train a ML filter to eliminate the leakage of the aliased energy in the FK domain, assuming it does not overlap the signal.
4. Create some traces in time. Add in large amplitude noise at short time intervals, also known as burst noise. Train a NN to eliminate this burst noise.

## 5.7 Computational Labs

1. Go to the lab in `LAB1/Chapter.Book.GNN/NNMigration/lab.html` and run the lab. This MATLAB code computes the decision boundaries that separate signal from noise in migration images, and then mutes the noise in the migration image in the  $(x, z)$  domain. The performance of the SVM is compared to that of the neural network and logistic regression algorithms.
2. Go to the lab in `LAB1/Chapter.Book.GNN/NNSurfaceWave/lab.html` and run the lab. This MATLAB code computes the window in the  $k - \omega$  domain that isolates the fundamental mode Rayleigh wave from the higher-order modes.

## 5.8 Appendix: Dip Angles, Coherency and Amplitudes of a Migration Image

The dip angle, coherency and amplitude at each pixel of a migration image are skeletonized characteristics that can be used to sometimes distinguish aliasing noise from signal. We now describe the computation of each feature.

To compute these three skeletonized characteristics we first define a sliding  $5 \times 5$  window centered at the  $i^{th}$  pixel in the migration image. The  $5 \times 5$  window is used because it is about one wavelength in size for our migration example, and it is slid from one pixel to the next to compute each pixel-centered feature. We apply a local slant stack (Yilmaz, 2001) to

the windowed data centered at the  $i^{th}$  pixel to get the dip angle of the dominant reflection energy. The local slant stack is the summation of amplitudes of migration traces that intersect a line slanted at a specified angle. Only the neighboring six or so traces (i.e., 1-2 wavelengths) centered around the  $i^{th}$  pixel are used for this summation. The dip angle of the line is iteratively incremented by about 5-15 degrees and the dip angle with the largest summed amplitude is assigned to be  $\theta_i$ . To reduce noise, the summation can be over trace amplitudes that intersect a "fat line", where the thickness of the "fat" line is about one-half of a wavelength.

The  $5 \times 5$  window is shifted over by one pixel and the slant-stack procedure is repeated to give  $\theta_{i+1}$ . Repeating this procedure for all the training set pixels in Figure 5.1a gives the dip angle image shown in Figure 5.1b. In practice, only 70 pixels are in the actual training set.

The coherency  $\phi_i$  at the  $i^{th}$  pixel of Figure 5.1a is computed by forming a common image gather (CIG) or a common angle gather (CAG) from the prestack migration images (Yilmaz, 2001). If the migration velocity is accurate then summing along the offset axis (or angle axis if the CAG is used) should give a strong coherency value  $\phi_i$  at the  $i^{th}$  pixel. Repeating this procedure for all the pixels in Figure 5.1a gives the coherency image shown in Figure 5.1c.

The amplitude image in Figure 5.1d is obtained by computing the Hilbert transform of a trace in the migration image (Yilmaz, 2001). Then the envelope amplitude  $A_i$  is computed in the window centered at the  $i^{th}$  pixel.

## 5.9 Appendix: MATLAB Code for Logistic Regression

The main fragment of the logistic regression code is given below.

**Code 5.9.1. MATLAB Logistic Regression Code (Yuqing Chen)**

```

[m,n]=size(X);      % Get the size of training example
X2=[ones(m,1) X]; % Add intercept term to the training set

disp('Calculating decision boundary')
maxiter=500;        % Iteration numbers
theta=rand(n+1,1)-0.5; % Initialize the model parameters
step=10;            % Set the step length

for iter=1:maxiter
    [cost, grad] = costFunction2(theta, X2, Y); % Compute data misfit (cost) and gradient (grad)
    if(mod(iter,20)==0) % Show the data misfit at every 20 iterations
        fprintf('iter= %d res = %f\n',iter,cost)
    end
    theta2=theta-step.*grad; % Update the model parameters with a
                             % trial step length
    [cost1, grad_temp] = costFunction2(theta2, X2, Y); % Compute the new misfit (cost1)
                                                         % and gradient (grad_temp)
                                                         % fprintf('res1 = %f\n',cost1)

    if(cost1>cost) % If the new misfit (cost1) is larger than the previous misfit (cost),
                  % then we reduce the step length and try again
        step=step*0.8; % Reduce the step length
        theta2=theta-step.*grad; % Update the model parameters with reduced trail step length
        [cost1, grad_temp] = costFunction2(theta2, X2, Y); % Compute the new misfit
        fprintf('step = %f, res1 = %f\n',step,cost1) % Print the step length and data misfit
    end % When the new misfit is smaller than the
        %misfit at the previous iteration (cost1 < cost), we update the model parameter

    theta=theta2;
end
%
%
disp('Display decision boundary')
[X_surf1,Y_surf1]=meshgrid(min(cohe_vec2):0.01:max(cohe_vec2),min(angl_vec2):0.01:max(cohe_vec2));
Z_surf1=-(theta(1)+theta(2).*X_surf1+theta(3).*Y_surf1)./theta(4); % compute the hyperplane
% predicted by the logistic
% regression
hold on
surf(X_surf1,Y_surf1,Z_surf1) % Display the hyperplane

```

The main fragment of the neural network code is given below.

**Code 5.9.2.** *MATLAB Neural Network Code (Yuqing Chen)*

```
lamda=0.0; step=10; maxiter=500; % lambda: regularization term (higher lambda can avoid
                                % over-fitting but will lost accuracy)
for iter=1:maxiter
    [cost,grad] = nnCostFunction(nn_params, input_layer_size, hidden_layer_size, ...
                                num_labels, X2, Y, lamda); % nnCostFunction is the kernel used to
                                                            % compute the data misfit and gradient
    if(mod(iter,50)==0||iter==1)
        fprintf('iter= %d, res = %f\n',iter,cost)
    end
    nn_params2=nn_params-step*grad;
    [cost1,grad_temp] = nnCostFunction(nn_params2, input_layer_size, hidden_layer_size, ...
                                    num_labels, X2, Y, lamda);
    while(cost1>cost)
        step=step*0.8;
        nn_params2=nn_params-step*grad;
        [cost1,grad_temp] = nnCostFunction(nn_params2, input_layer_size, hidden_layer_size, ...
                                    num_labels, X2, Y, lamda);
        fprintf('step = %f, res1 = %f\n',step,cost1)
    end
    nn_params=nn_params2;
end
```



## Chapter 6

# Multilayer Fully-Connected Neural Networks

The recursive equations for both the feed-forward and backward-propagation operations are derived for a multilayered fully-connected neural network. For each iteration, the feed-forward and gradient computations cost about  $O(N^2M)$  algebraic operations for  $N$  layers, each one associated with an  $M \times M$  weight matrix. The workflow and pseudo-MATLAB codes are presented for implementing the multilayer neural network algorithm.

### 6.1 Introduction

A fully-connected neural network consists of a sequence of computational nodes arranged in distinct layers (see Figure 5.11). Each layer is a vertical sequence of computational nodes, where we assume  $N$  layers in the neural network and each layer can have a different number of nodes. The input nodes are not counted as a network layer because there is no computational processing at these nodes. The FCNN matrix for each layer is dense and fully populated with mostly non-zero numbers.

There are two operations required for computing the gradient of the objective function: feed-forward and back-propagation. The next two sections derive the formulas for these two operations so they can be computed in an efficient recursive fashion.

### 6.2 Feed-forward Operation

The feed-forward operation of the neural network inputs the features  $a_i^{[0]}$  from the starting  $0^{th}$  layer and produces the predicted target value  $a_i^{[N]}$  at the last  $N^{th}$  layer:

$$a_i^{[N]} = g\left(\sum_{j=1}^{\mu_{N-1}} w_{ij}^{[N]} g\left(\sum_{k=1}^{\mu_{N-2}} w_{jk}^{[N-1]} g\left(\dots g\left(\sum_{n=1}^{\mu_1} w_{mn}^{[1]} a_n^{[0]}\right)\dots\right)\right)\right), \quad (6.1)$$

where  $\mu_i$  is the number of nodes in the  $i^{th}$  layer and we have assumed the same type of activation function  $g(\cdot)$  in each layer. If different activation functions are used then we can append a superscript such that  $g^{[k]}(\cdot)$  denotes the activation function in the  $k^{th}$  layer. Here, the variable  $z_i^{[k]}$  and output parameters  $a_i^{[k]} = g(z_i^{[k]})$  of the  $k^{th}$  layer can be assembled into the  $\mu_k \times 1$  vectors  $\mathbf{z}^{[k]}$  and  $\mathbf{a}^{[k]} = g(\mathbf{z}^{[k]})$ , respectively. The activation function  $g(\cdot)$  is applied to each element of  $\mathbf{z}^{[n]}$  so that  $g(\mathbf{z}^{[n]})$  is a vector of the same dimension as  $\mathbf{z}^{[n]}$ , with the components

$$z_i^{[n]} = \sum_{j=1}^{\mu_{n-1}} w_{ij}^{[n]} a_j^{[n-1]}, \quad (6.2)$$

for  $i \in \{1, 2, \dots, \mu_n\}$ .

The components of the  $\mu_n \times 1$  vector  $\mathbf{a}^{[n]}$  can be recursively computed from the components of  $\mathbf{a}^{[n-1]}$  in the lower-order  $(n-1)^{th}$  layer. This recursion can lead to a significant gain in efficient computations as seen in the pseudo-MATLAB code below.

```
load  $\mathbf{a}^{[0]}$ ;
for  $n = 1 : N$ 
    load  $\mathbf{W}^{[n]}$ 
     $\mathbf{z}^{[n]} = \mathbf{W}^{[n]} \mathbf{a}^{[n-1]}$ 
     $\mathbf{a}^{[n]} = g(\mathbf{z}^{[n]})$ 
end
```

where the matrix coefficients  $w_{kl}^{[n]}$  for the  $n^{th}$  layer are the elements of the  $\mu_n \times \mu_{n-1}$  matrix  $\mathbf{W}^{[n]}$  and the  $\mu_n \times 1$  vector  $\mathbf{z}^{[n]}$  contains the components  $z_i^{[n]}$  in equation 6.2. For convenience the bias terms  $b_i^{[n]}$  for the  $n^{th}$  layer are implicitly represented in the first column of the matrix  $\mathbf{W}^{[n]}$ . Assuming that all of the layers have the same number  $M$  of nodes, then each matrix-vector product/layer costs  $O(M^2)$  operations. Thus, the computational count for recursively computing the predicted target vectors  $\mathbf{a}^{[n]}$  for  $n \in \{1, 2, \dots, N\}$  is  $O(M^2 N)$ . Without using this recursive procedure, the computational count is  $O(M^2 N^2)$  to compute  $\mathbf{a}^{[n]}$  for all of the layers.

### 6.3 Back-propagation Operation

The gradient term in equation 4.11, i.e.

$$w_{ij}^{[n]} := w_{ij}^{[n]} - \alpha \frac{\partial \epsilon}{\partial w_{ij}^{[n]}}, \quad (6.3)$$

can be recursively computed *backward* by updating  $w_{ij}^{[N-1]}$  from the higher-order terms in the  $N^{th}$  layer. Unlike forward propagation where recursion computes the higher-order activation term from the previous-order one, back-propagation recursively computes lower-order terms from the higher-order ones. Similar to forward propagation, the computational count for



recursively computing gradients at all of the layers is  $O(M^2N)$ .

### 6.3.1 Formula for $\partial\epsilon/\partial w_{jk}^{[N]}$

The gradient of the misfit function in equation 4.11 for the weights in the  $N^{th}$  layer is

$$\frac{\partial\epsilon}{\partial w_{jk}^{[N]}} = \sum_{i=1}^{\mu_N} \overbrace{(a_i^{[N]} - y_i)}^{\text{residual}} \frac{\partial a_i^{[N]}}{\partial w_{jk}^{[N]}}, \quad (6.4)$$

where we set  $\lambda = 0$  for pedagogical simplicity. From the chain rule, the definition  $a_i^{[N]} = g(z_i^{[N]})$  and equation 6.2 we have

$$\frac{\partial a_i^{[N]}}{\partial w_{jk}^{[N]}} = \frac{\partial a_i^{[N]}}{\partial z_i^{[N]}} \frac{\partial z_i^{[N]}}{\partial w_{jk}^{[N]}} = g(z_j^{[N]})' a_k^{[N-1]} \delta_{ij}, \quad (6.5)$$

and

$$\frac{\partial a_m^{[N]}}{\partial a_i^{[N-1]}} = \frac{\partial g(z_m^{[N]})}{\partial z_m^{[N]}} \frac{\partial z_m^{[N]}}{\partial a_i^{[N-1]}} = g(z_m^{[N]})' w_{mi}^{[N]}, \quad (6.6)$$

where  $g(z_j^{[N]})' = \frac{\partial g(z_j^{[N]})}{\partial z_j^{[N]}}$  and  $\delta_{ij}$  represents the Kronecker delta function. Plugging equation 6.5 into equation 6.4 gives the gradient wrt  $w_{jk}^{[N]}$  in the  $N^{th}$  layer:

$$\frac{\partial\epsilon}{\partial w_{jk}^{[N]}} = \overbrace{(a_j^{[N]} - y_j)g(z_j^{[N]})'}^{\Delta_j^{[N]}} a_k^{[N-1]}, \quad (6.7)$$

where  $\Delta_j^{[N]}$  is the  $j^{th}$  component of the weighted residual that vector multiplies the "forward" field value  $a_k^{[N-1]}$  at the  $k^{th}$  node in the  $(N-1)^{th}$  layer. If  $g(z)$  is a sigmoid function then  $g(z)' = g(z)(1 - g(z))$ . Other types of activation functions will provide different formulas for the derivative.

Can we recursively use the term  $\Delta_j^{[N]}$  in the formula for the calculation of the next lower-order gradient? The answer is yes as the next section demonstrates.

### 6.3.2 Formula for $\partial\epsilon/\partial w_{ij}^{[N-1]}$

To derive the expression for  $\partial\epsilon/\partial w_{ij}^{[N-1]}$ , recall the multidimensional chain rule where  $f(a_1(t), a_2(t), a_3(t), \dots, a_M(t))$  is a regular function of  $M$  smooth functions  $a_i(t)$ . In this case the derivative  $\partial f(t)/\partial t$  can be expressed as

$$\frac{\partial f(a_1^{[N]}, a_2^{[N]}, \dots, a_M^{[N]})}{\partial t} = \sum_{m=1}^M \frac{\partial f(a_1^{[N]}, a_2^{[N]}, \dots, a_M^{[N]})}{\partial a_m^{[N]}} \frac{\partial a_m^{[N]}}{\partial t}, \quad (6.8)$$

where we append the superscript  $[N]$  to the  $a_i$  variables. Setting  $t \rightarrow a_i^{[N-1]}$ , this equation says that, for forward propagation of the feature values, a perturbation in  $a_i^{[N-1]}$  will affect the values of the outputs  $a_i^{[N]}$  in the next highest-order layer. Since this perturbation  $t$  will also affect the misfit function  $\epsilon(a_1^{[N]}, a_2^{[N]}, \dots, a_{\mu_n}^{[N]})$ , then according to the chain rule in equation 6.8 the gradient w/r to the weights in the  $N - 1$  layer becomes

$$\begin{aligned}
\frac{\partial \epsilon}{\partial w_{jk}^{[N-1]}} &= \sum_{m=1}^{\mu_N} \overbrace{\frac{\partial \epsilon}{\partial a_m^{[N]}}}^{(a_m^{[N]} - y_m)} \sum_{i=1}^{\mu_{N-1}} \overbrace{\frac{\partial a_m^{[N]}}{\partial a_i^{[N-1]}}}^{eq. 6.6: g(z_m^{[N]})' w_{mi}^{[N]}} \overbrace{\frac{\partial a_i^{[N-1]}}{\partial w_{jk}^{[N-1]}}}^{eq. 6.5: g(z_j^{[N-1]})' a_k^{[N-2]} \delta_{ij}}, \\
&= \sum_{m=1}^{\mu_N} \overbrace{(a_m^{[N]} - y_m)}^{\mathbf{r}^{[N]}} g(z_m^{[N]})' g(z_j^{[N-1]})' a_k^{[N-2]} \sum_{i=1}^{\mu_{N-1}} w_{mi}^{[N]} \delta_{ij}, \\
&= \overbrace{g(z_j^{[N-1]})'}^{\mathbf{dg}^{[N-1]}} \overbrace{a_k^{[N-2]}}^{\mathbf{a}^{[N-2]}} \left( \overbrace{\sum_{m=1}^{\mu_N} w_{mj}^{[N]} g(z_m^{[N]})'}^{\mathbf{W}^{[N]T} \mathbf{dg}^{[N]}} \overbrace{(a_m^{[N]} - y_m)}^{\mathbf{r}^{[N]}} \right), \tag{6.9}
\end{aligned}$$

where the summation in the  $m$  index is over the  $\mu_N$  nodes in the  $N^{th}$  layer and  $j$  is the index of the output vector associated with this matrix-vector multiplication. Here, the role of the  $\{\mathbf{W}^{[N]T} \mathbf{dg}^{[N]} \cdot \mathbf{r}^{[N]}\}_j$  term is to back-propagate the weighted residual vector from the  $N^{th}$  layer to the  $j^{th}$  node in the  $(N - 1)^{th}$  layer. Note that  $\mathbf{dg} \cdot \mathbf{a}$  indicates an element-by-element MATLAB multiplication of the two vectors and the summation associated with the matrix-vector multiplication is over the first subscript  $m$  of  $w_{mj}^{[N]}$ . Thus, we use the transpose symbol  $T$  in the superscript of  $\mathbf{W}^{[N]}$ .

### 6.3.3 Formula for $\partial \epsilon / \partial w_{ij}^{[N-2]}$

The previous section derived the gradient formula for the  $N - 1$  hidden layer next to the output layer  $N$ . We now derive the gradient  $\partial \epsilon / \partial w_{ij}^{[N-2]}$  for the hidden layer surrounded only by hidden layers. In this case the objective function will be a function of the activation parameters in two neighboring links of the chain of layers. This means we need a two-link chain rule for  $f = f(g(h(t)))$ , where  $f(g)$  is a function of  $g$  and  $g(h)$  is a function of  $h(t)$ . Therefore, the single-link chain rule in equation 6.8 can be extended to the two-link chain rule where  $a_i(t) \rightarrow a_i(b_1(t), b_2(t) \dots b_M(t))$ :

$$\frac{\partial f(a_1, a_2, \dots, a_M)}{\partial t} = \sum_{m=1}^M \sum_{n=1}^M \frac{\partial f(a_1, a_2, \dots, a_M)}{\partial a_m} \frac{\partial a_m}{\partial b_n} \frac{\partial b_n}{\partial t}. \tag{6.10}$$

Setting  $a_i \rightarrow a_i^{[N]}$ ,  $b_i \rightarrow a_i^{[N-1]}$ ,  $f \rightarrow \epsilon$  and  $t \rightarrow a_i^{[N-2]}$  gives the formula for  $\frac{\partial \epsilon}{\partial w_{jk}^{[N-2]}}$ :

$$\begin{aligned}
\frac{\partial \epsilon}{\partial w_{jk}^{[N-2]}} &= \sum_{i=1}^{\mu_{N-2}} \frac{\partial \epsilon}{\partial a_i^{[N-2]}} \frac{\partial a_i^{[N-2]}}{\partial w_{jk}^{[N-2]}}, \\
&= \sum_{m=1}^{\mu_N} \overbrace{\frac{\partial \epsilon}{\partial a_m^{[N]}}}^{(a_m^{[N]} - y_m)} \sum_{n=1}^{\mu_{N-1}} \overbrace{\frac{\partial a_m^{[N]}}{\partial a_n^{[N-1]}}}^{g(z_m^{[N]})' w_{mn}^{[N]}} \sum_{i=1}^{\mu_{N-2}} \overbrace{\frac{\partial a_n^{[N-1]}}{\partial a_i^{[N-2]}}}^{g(z_n^{[N-1]})' w_{ni}^{[N-1]}} \overbrace{\frac{\partial a_i^{[N-2]}}{\partial w_{jk}^{[N-2]}}}^{g(z_j^{[N-2]})' a_k^{[N-3]} \delta_{ij}}, \\
&= \overbrace{g(z_j^{[N-2]})' a_k^{[N-3]}}^{\mathbf{dg}^{[N-2]} \mathbf{a}^{[N-3]}} \left( \overbrace{\sum_{n=1}^{\mu_{N-1}} w_{nj}^{[N-1]} g(z_n^{[N-1]})'}^{\mathbf{W}^{[N-1]T}} \overbrace{g(z_n^{[N-1]})'}^{\mathbf{dg}^{[N-1]}} \right) \left( \overbrace{\sum_{m=1}^{\mu_N} w_{mn}^{[N]} g(z_m^{[N]})'}^{\mathbf{W}^{[N]T}} \overbrace{g(z_m^{[N]})'}^{\mathbf{dg}^{[N]}} \overbrace{(a_m^{[N]} - y_m)}^{\mathbf{r}^{[N]}} \right), \tag{6.11}
\end{aligned}$$

where the boldface variables are also seen in equation 6.9 and  $g(z_m^{[N]})'(a_m^{[N]} - y_m)$  is the residual component in equation 6.7.

Each layer of the neural network has the same type of structure, so the formula for the gradient in the  $P^{th}$  layer, where  $1 < P < N - 2$ , is given by

$$\frac{\partial \epsilon}{\partial w_{jk}^{[P]}} = g(z_j^{[P]})' a_k^{[P-1]} \left( \mathbf{V}^{[P+1]} \mathbf{V}^{[P+2]} \dots \mathbf{V}^{[N-1]} \mathbf{V}^{[N]} \mathbf{r}^{[N]} \right)_j, \tag{6.12}$$

where

$$(\mathbf{V}^{[P]})_{ij} = w_{ij}^{[P]} g(z_i^{[P]})', \quad i \in [1, 2 \dots \mu_P], \quad j \in [1, 2 \dots \mu_{P-1}], \tag{6.13}$$

The numerical implementation of equation 6.12 is ripe for recursion. Unlike the recursive algorithm of forward propagation that starts at layer 1, backward recursion starts from the  $N^{th}$  layer to get  $\mathbf{V}^{[N]} \mathbf{r}^{[N]}$ , and then recursively computes the next lower-order terms until reaching the layer of interest. This is known as backward propagation of the residual.

The pseudo-code for recursively computing the  $M \times M$  gradient matrix  $\mathbf{d}\epsilon$  at the  $(J-1)^{th}$  layer is given below. Here we assume the bias values are absorbed into the matrix  $\mathbf{W}^{[N]}$  and  $N > J > 1$ , where  $J$  is a positive integer. Assume  $\mathbf{W}^{[i]}$ ,  $\mathbf{a}^{[i]}$ ,  $\mathbf{dg}^{[i]}$  are the  $M \times M$  weight,  $M \times 1$  activation and  $M \times 1$  gradient  $\frac{\partial g}{\partial z}$  matrices, respectively, that have been pre-computed for all  $N$  layers. For notational simplicity we assume that the number of nodes in each layer

is  $M$ .

```

Load ( $\mathbf{a}^{[0]}$ ,  $\mathbf{a}^{[1]} \dots \mathbf{a}^{[N]}$ ),  $\mathbf{y}$ , ( $\mathbf{dg}^{[0]}$ ,  $\mathbf{dg}^{[1]} \dots \mathbf{g}^{[N]}$ ), ( $\mathbf{W}^{[0]}$ ,  $\mathbf{W}^{[1]} \dots \mathbf{W}^{[N]}$ )
     $\mathbf{in} = \mathbf{a}^{[N]} - \mathbf{y}$ 
for  $i = N : -1 : J$ 
     $\mathbf{in} = \mathbf{dg}^{[i]} * \mathbf{in}$ 
     $\mathbf{in} = \mathbf{W}'^{[i]} * \mathbf{in}$ 
end
     $\mathbf{in} = \mathbf{in} * \mathbf{dg}^{[i-1]}$ 
     $d\epsilon(j, k) = in(j) * a(k)^{[i-2]}$ 

```

Each gradient calculation for a layer's weights will cost about  $O(M^2)$  algebraic operations per iteration because of the matrix-vector product. If the number of layers is  $N$  then the total cost will be about  $O(M^2N)$  algebraic operations for computing the weight gradients for all the layers.

## 6.4 MATLAB Code

The MATLAB code for the fully-connected neural network can be derived from the above pseudo-code. There are two principal portions of the code, forward and backward propagation in *gradientnn.m* and the main code *NNode.m*. Below is the major fragment from *NNode.m*.

**Code 6.4.1. MATLAB *NNode.m* Code**

```

M=100; % # of equations constraint
N=5; % # of unknowns (w1, w2, w3, w4, w5) for 1st node
x=zeros(N,M);t=zeros(1,M);
[x,t,alpha,lim,ww1,layer_size,obj,act]=Datain(M,N,x,t); % Create training data
act=1;obj=1;nit=400;res=zeros(nit,1);ww_old=ww1;
layer_num=numel(layer_size); % layer number include the input and output

for k=1:nit % Looping over iterations
    alpha=1; % step size (orig, case: 1)
    [grad,res(k)]=gradientnn(M,x,t,ww_old,layer_size,obj,act);
    for ilayer=1:layer_num-1
        ww{ilayer}=ww_old{ilayer}-alpha*grad{ilayer};
    end
    % Bisection line search for step length
    [~,res1]=gradientnn(M,x,t,ww,layer_size,obj,act);
    while (res1>res(k)) && (alpha>lim)
        alpha=alpha*0.5;
        for ilayer=1:layer_num-1
            ww{ilayer}=ww_old{ilayer}-alpha*grad{ilayer};
        end
        [~,res1]=gradientnn(M,x,t,ww,layer_size,obj,act);
    end
    ww_old = ww; %%% update successful
end
Display(M,res,nit,ww,layer_size,x,t,obj,act)

```

A fragment of the pseudo-code for forward propagation modeling in *gradientnn.m* is shown below.

**Code 6.4.2. MATLAB *Forward-propagation* Code**

```

%% Part 1: Feedforward the neural network and return the cost in the variable res.
% Do forward propagation
for iter =1:layer_num-1
    aa{iter} = [ones(1, M); aa{iter}]; % z[n]=W[n]a[n-1], 1 is for bias
    zz{iter}=ww{iter}*aa{iter}; % a[n]=g(z[n])
    if iter ==layer_num-1 %output layer use sigmoid
        [aa{iter+1},~] = activation(zz{iter},1);
    else
        [aa{iter+1},~] = activation(zz{iter},act);
    end
    penalize =penalize+ sum(sum(ww{iter}.^ 2)); % add regularization
end
t_pred = aa{layer_num};
[res,in]= misfit( t_pred,t,1/M,obj); %obj=1 for L2, 2 for likelihood

```

and the back-propagation MATLAB code is below

**Code 6.4.3. MATLAB Back-propagation Code**

```

%% Part 2: Implement the backpropagation algorithm to compute the gradients.
% Implement backpropagation
for iter=layer_num-1:-1:1
    if iter ==layer_num-1 %output layer has to use sigmoid
        [~,dg]=activation(zz{iter},1);
    else
        [~,dg]=activation(zz{iter},act); %in=dg[i].*in
    end
    in=dg.*in; % in is the backward field
    grad{iter}=in*aa{iter}'; % de(j,k)=in*(a[i-2])T
    in=ww{iter}'*in; % in=W'[i]*in
    in = in(2:end, :);
end
end

```

The activation function *activation.m* is given by

**Code 6.4.4. MATLAB Activation Function Code**

```

function [ g,dg ] = activation( z,type )
%z - input- for activation function
%type: objective function type: 1 for sigmoid, 2 for ReLU
% g - output- value of the activation function
% dg- output- gradient of g with respective to z

if type==1 % for sigmoid
    g = 1.0 ./ (1.0 + exp(-z));
    dg = g .* (1 - g); %Compute the gradient of the sigmoid function
elseif type==2 % for ReLU
    g=z*0.0; dg=z*0.0;
    g(z>0)=z(z>0);
    g(z<=0)=z(z<=0)*0.0;
    dg(z>0)=1.0;
    dg(z<=0)=0.0;
else
    display('You entered the wrong type for the activation function');
    stop
end
end

```

A vectorized version of the back-propagation and forward propagations algorithms are in the pseudo-code of Appendix 6.8.

## 6.5 Summary

The recursive equations are derived for the forward-propagation and back-propagation operations of a neural network. These operations can be used to iteratively update the weights  $w_{ij}^{[n]}$  in each layer that minimize the objective function. The recursive nature of these equations provide for an efficient steepest descent procedure that requires  $O(M^2N)$  computations per iteration, where  $M$  is the number of nodes per layer and  $N$  is the number of layers. Here, we assume that the number of nodes  $M$  is the same for each layer.

The next chapter presents the automatic differentiation algorithm, which computes the exact values of the derivatives of a complicated objective function. These values are com-

puted by a basic computer code that is applicable to any neural network.

## 6.6 Exercises

1. Derive the gradient formula with the cross-entropy loss function instead of the  $L_2$  loss function in equation 6.11.
2. Derive the  $L_2$  gradient formula with the ReLU activation function instead of the sigmoid activation function in equation 6.11.
3. Show that equation 6.9 reduces to equation 4.44 for the two-layer neural network.
4. Write the pseudo-MATLAB code for the steepest descent method, where the bias vector and its gradient are explicitly computed.
5. Write the pseudo-MATLAB code for the steepest descent method where regularization is used.
6. Write the pseudo-MATLAB code where the input is a batch matrix that represents the training set.
7. Write the pseudo-MATLAB code for the steepest descent method where the number of nodes per layer is different from one another.

## 6.7 Computational Labs

1. Go to the Fully-Connected Neural Network lab in */LAB1/Chapter.Book.FCN/Chapter.Gradient.Neural/lab.html*. After downloading and unzipping *Chapter.Gradient.Neural.zip*, run the MATLAB code *NNode1.m* to get the results seen in Figures 4.13, 4.15, 4.17, and 4.18. *LAB1/Chapter.Book.FCN/Chapter.GenNN/lab.html* and run the MATLAB code. It uses a fully-connected neural network to classify input vectors that have 1's and 0's for its elements.

## 6.8 Appendix: Vectorized Steepest Descent Formula

Assume a network architecture where there are  $M$  nodes per layer. The vectorized stochastic steepest descent formula for computing  $w_{ij}^{[n]}$  for each layer is given by the following

MATLAB pseudo-code.

```

Load  $\mathbf{a}^{[0]}$ ,  $\mathbf{y}$ ; Initialize  $\alpha$ , matrices  $\mathbf{W}^{[i]}$  for  $i = [1, 2, \dots N]$ 
    for  $m = 1 : N$ 
         $\mathbf{a}^{[m]} = \text{forw}(\mathbf{a}^{[0]}, \mathbf{W}^{[1]} \dots \mathbf{W}^{[N]})$  Compute predicted data  $\mathbf{a}^{[N]}$ 
         $\mathbf{dg}^{[m]} = \text{deriv}(\mathbf{a}^{[m]})$  Compute predicted derivative  $\mathbf{dg}^{[m]}$ 
    end
    for  $n = 1 : \text{niter}$  Iterate steepest descent
         $\mathbf{in} = \mathbf{a}^{[N]} - \mathbf{y}$  Residual
        for  $i = N : -1 : 1$  Compute  $\mathbf{W}^{[i]}$  at each layer
             $\mathbf{in} = \mathbf{dg}^{[i]} .* \mathbf{in}$ 
             $\mathbf{d\epsilon} = (\mathbf{in} .* \mathbf{dg}^{[i-1]}) * \mathbf{a}^{[i-1]T}$ 
             $\mathbf{in} = \mathbf{W}'^{[i]} * \mathbf{in}$ 
             $\mathbf{W}^{[i]} := \mathbf{W}^{[i]} - \alpha \mathbf{d\epsilon}$ 
        end
        for  $m = 1 : N$  Compute new prediction  $\mathbf{a}^{[N]}$ 
             $\mathbf{a}^{[m]} = \text{forw}(\mathbf{a}^{[0]}, \mathbf{W}^{[1]}, \dots \mathbf{W}^{[N]})$ 
             $\mathbf{dg}^{[m]} = \text{deriv}(\mathbf{a}^{[m]})$ 
        end
    end
end

```

The above code is vectorized because we have almost entirely eliminated the explicit use of *for loops*, which can speed up computations by more than order-of-magnitude. If the input data are a batch of data then this can be accommodated by an outer loop over the different training examples. However, this is inefficient so a vectorized version can be obtained by replacing the input  $M \times 1$  data vector  $\mathbf{a}^{[0]} \rightarrow \mathbf{A}^{[0]}$ , where the  $M \times P$  batch-data matrix has  $P$  columns, each one a different training example of input values. Similarly, the  $M \times 1$  target vector  $\mathbf{y}$  can be transformed into a  $M \times P$  target matrix. This assumes that there are  $M$  nodes per layer.



## Chapter 7

# Support Vector Machines

A support vector machine (SVM) is a supervised learning method that classifies *binary* data into two classes (Cortes and Vapnik, 1995). It has the same purpose as logistic regression or NN in classifying data, but SVM is sometimes preferred because it is simpler in that there is no need for the design of a complicated network architecture. Similar to NN or CNN, there are three stages for building a SVM model: training on a set of classified data, validating the model on a smaller set of validation data, and finally applying the SVM to test data. Unlike a NN, the linear SVM is always guaranteed to converge for linearly separable data because it is free of local minima<sup>1</sup>. Consequently, it is a very popular classifier in the biomedical community. If the data are not linearly separable, then they can be transformed to a higher-dimensional space to find the separation boundary between the binary classes. SVM can also be generalized to classifying data that has more than two classes (Bishop, 2006) and its popularity has been growing over the last two decades (see Figure 7.1 and Cervantes et al. (2020)). The SVM is also extended to regression problems with the support vector regression (SVR) algorithm.

### 7.1 Introduction

There are many useful geophysical applications of SVM, including well-log classification (Al-Anazi and Gates, 2010a,b; Sebtosheikh, M. and A. Salehi, 2015; Sebtosheikh et al., 2015; Deng et al., 2017), migration filtering (Chen et al., 2020a), frequency filtering of surface waves (Li et al., 2019), AVO inversion (Kuzma, 1999; Kuzma and Rector, 2004), and seismic trace interpolation and inversion (Kuzma et al., 2006). Earthquake seismologists and engineers also use SVMs for classification of seismic events (Appoloni, 2009; Körtström et al., 2016; Tang et al., 2019) and for picking the traveltimes of events (Jiang and Ning, 2020). Traffic engineers also use SVR for predicting the flow of traffic at different times of the week (Wu et al., 2004; Akter et al., 2015), where the SVRs are trained on historical traffic data. Wu et al. (2004) show that the SVR predictor significantly outperforms the other baseline predictors for traffic data analysis.

---

<sup>1</sup>Wu et al. (2004) reinforce the claim that SVMs are free of local minima: "This is largely due to the structural risk minimization (SRM) principle in SVM, which has greater generalization ability and is superior to the empirical risk minimization (ERM) principle as adopted in neural networks. In SVM, the results guarantee global minima, whereas ERM can only locate local minima. "

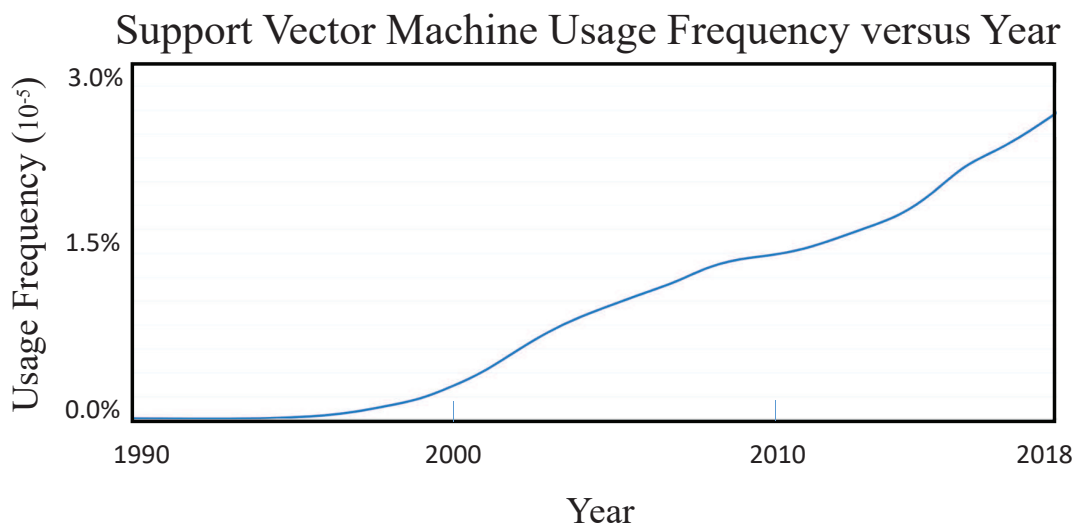


Figure 7.1: Usage of the word *support vector machine* plotted against calendar year.

Both rock physics and simultaneous seismic inversion can be used to identify the reservoir zones by applying SVM to seismic data (Torres and Reveron, 2013). Such data can be geologically interpreted by SVM methods as well. For example, Zhao et al. (2014) introduced the efficient proximal support vector machines (Fung and Mangasarian, 2001) into lithofacies classification in the Barnett Shale. They demonstrated the validity of the proximal support vector machines classifier for binary classification of shales and limestones. Seismic interpreters can use multi-attribute SVMs for reliable fault identification in seismic sections (Di et al., 2017) and use multiclass SVMs (Crammer and Singer, 2001) to interpret geology from seismic sections (Liu et al., 2020a,c). The next section describes the theory of SVM.

## 7.2 SVM Theory

In the binary classification problem we are given a set of training data consisting of  $N$  feature vectors

$$(\mathbf{x}^{(i)}, y^{(i)}), \quad (7.1)$$

where  $\mathbf{x}^{(i)}$  is the  $i^{th}$  feature vector with dimension  $D \times 1$  and  $y^{(i)} = \pm 1$  is the binary label value for classification. For a set of two-dimensional feature<sup>2</sup> vectors  $\mathbf{x}^{(i)}$ , the goal of SVM is to find the *best* line that separates the two classes of data, denoted as +’s and -’s, in Figure 7.2. Here, the *best* line is the dashed black one with the fattest margin width<sup>3</sup>  $2d$

<sup>2</sup>Sometimes feature vectors  $\phi(\mathbf{x})$  are defined as the input data  $\mathbf{x}$  transformed into a higher-dimensional vectors  $\mathbf{z} = \phi(\mathbf{x})$ .

<sup>3</sup>We will use Bishop’s definition of *margin*: the shortest distance between the *decision* boundary and any of the sample points. Unfortunately, *margin* has a different definition in the dictionary so the use of *margin*

and is equidistant between the solid black lines that are parallel to one another. This dashed line, also known as the *decision line* or *decision boundary*, is mathematically characterized by the normal vector  $\mathbf{w}$  and intercept value  $b$ , where any vector  $\mathbf{x}$  on the decision line satisfies  $w_1x_1 + w_2x_2 + b = 0$ . The black solid lines intersect the circled margin points  $\mathbf{x}^{(i)}$  known as *support vectors* for the specified index values  $i$ . These support vectors define the points with the closest perpendicular distance to the decision boundary.

A less than optimal *decision line* is represented by the red dashed line in Figure 7.2 that also separates the two classes of data. Here, the margin thickness is skinnier than the one associated with the black lines. The skinnier the margin thickness, the easier it is to misclassify data subject to errors in the training pair  $(\mathbf{x}^{(i)}, y^{(i)})$ .

Once the  $\mathbf{w}$  is found after sufficient training, the SVM classifies new points  $\tilde{\mathbf{x}}^{(i)}$  that are out of the training set. Figure 7.2 depicts points in a two-dimensional space, but points in a  $D$ -dimensional space can be separated by a  $(D - 1)$ -dimensional hyperplane. In this case the point  $\mathbf{x}$  on the hyperplane satisfies  $\mathbf{w}^T \mathbf{x} + b = w_1x_1 + w_2x_2 + \dots + w_Dx_D + b = 0$ . If the points are not linearly separable then a nonlinear transformation of coordinates can be found where the data are linearly separable. In this new domain, the linear SVM can be used to find the correct decision boundary (see section 7.4).

The advantage of SVM over classification by logistic regression is that the optimal SVM line has the thickest margin, which is not necessarily the case with logistic regression when the points are linearly separable. In addition, the linear SVM solution never gets stuck in a local minimum, and requires fewer points for training because it only needs to be trained with the support vectors. The SVM procedure is very popular as a classification method, as evidenced by its citations in Figure 7.1 and some of the examples shown in this book. It is still one of the most effective classification methods.

### 7.2.1 SVM Applications

Many fields of science and engineering use SVM for classification, which include the following.

1. SVMs are helpful in text and hypertext categorization.
2. Classification of images can also be performed using SVMs. According to Wikipedia ([https://en.wikipedia.org/wiki/Support\\_vector\\_machine](https://en.wikipedia.org/wiki/Support_vector_machine)) "experimental results show that SVMs achieve significantly higher search accuracy than traditional query refinement schemes after just three to four rounds of relevance feedback. This is also true of image segmentation systems, including those using a modified version SVM that uses the privileged approach as suggested by Vapnik (DeCoste, 2002).".
3. Hand-written characters can be recognized using SVM (DeCoste, 2002).
4. According to Wikipedia ([https://en.wikipedia.org/wiki/Support\\_vector\\_machine](https://en.wikipedia.org/wiki/Support_vector_machine)) "The SVM algorithm has been widely applied in the biological and other sciences. They have been used to classify proteins with up to 90% of the compounds classified

---

as a distance in the SVM literature is in conflict with its definition in English language.

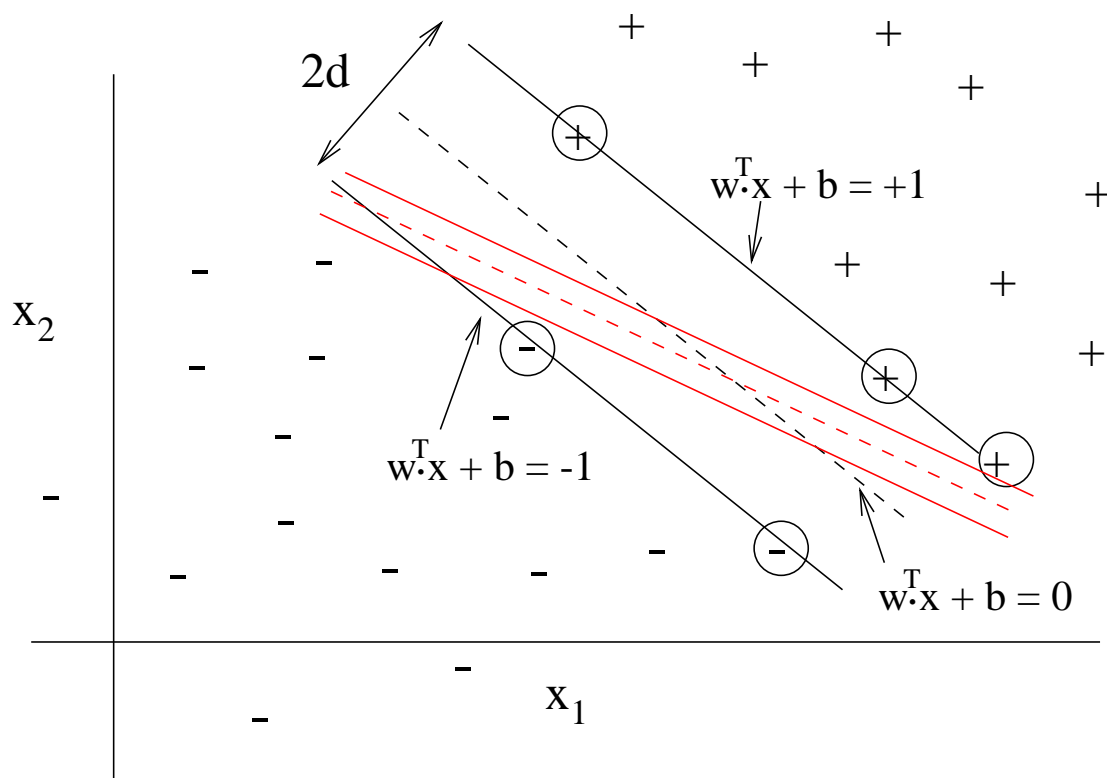


Figure 7.2: The points  $\mathbf{x}^{(i)}$  denoted by the class symbols  $\pm$  are separable by a decision line. The decision line with the thickest margin (dashed black line) is the optimal one estimated by SVM and the circled margin points  $\mathbf{x}^{(i)}$  are known as *support vectors*. In this example we assume the special case  $d = |\mathbf{w}^T \mathbf{x} + b| = 1$  for  $\mathbf{x}$  on the solid black lines.

correctly. Permutation tests based on SVM weights have been suggested as a mechanism for interpretation of SVM models (Gaonkar and Davatzikos, 2013). Support vector machine models can be used to identify features in the biological sciences.”.

5. According to a review article on medical hyperspectral imaging, Lu and Fei (2014) and Fei (2020) write the following: ”SVM is a kernel-based machine learning technique which has been widely used in the classification of hyperspectral images. Due to its strong theoretical foundation, good generalization capability, low sensitivity to the curse of dimensionality, and ability to find global classification solutions, SVMs is usually preferred by many researchers over other classification paradigms.”.
6. SVM is also useful for detection of code maliciously placed in computers. Kauser et al. (2011) write the following: ”Support vector machine (SVM) is used for classification in (computer operating systems) intrusion detection systems (IDS) due to its good generalization ability and nonlinear classification using different kernel functions and performs well as compared to other classifiers.”.
7. SVM is widely used in remote sensing methods for classification and detection of objects in aerial and satellite images (Mountrakis et al., 2011). They state the following: ”First, SVMs are not as well-known as other classifiers (e.g., decision trees, variants of neural networks) in the general remote sensing community, yet they can match if not exceed the performance of established methods. Second, their performance gains seem well-suited for some remote sensing applications, where a limited amount of reference data is often provided. Third, even though the method is not widely popular, in recent years there has been a significant increase in SVM works on remote sensing problems suggesting this review is current and appropriate.”.

### 7.3 Linear SVM

The linear SVM finds the hyperplane is a binary classification scheme that finds the hyperplane, or linear decision surface, that separates one class of feature vectors from the other. It can be adjusted for multiclass classification of points (Bishop, 2006) and it can be adapted to find non-linear decision surfaces.

Define the  $N$  feature vectors  $(\mathbf{x}^{(n)}, y^{(n)})$  for  $n \in [1, 2 \dots N]$ , with the binary classification of  $y^{(n)} = \pm 1$ . We assume that the set of points  $(\mathbf{x}^{(n)}, y^{(n)})$  are linearly separable so that there exist lines that separate the positive and negative classes from one another in Figure 7.2. SVM seeks the decision boundary, i.e. line, with the fattest margin thickness. This margin thickness can be defined by first defining the perpendicular distance  $\tilde{d}$  between  $\mathbf{x}$  on the

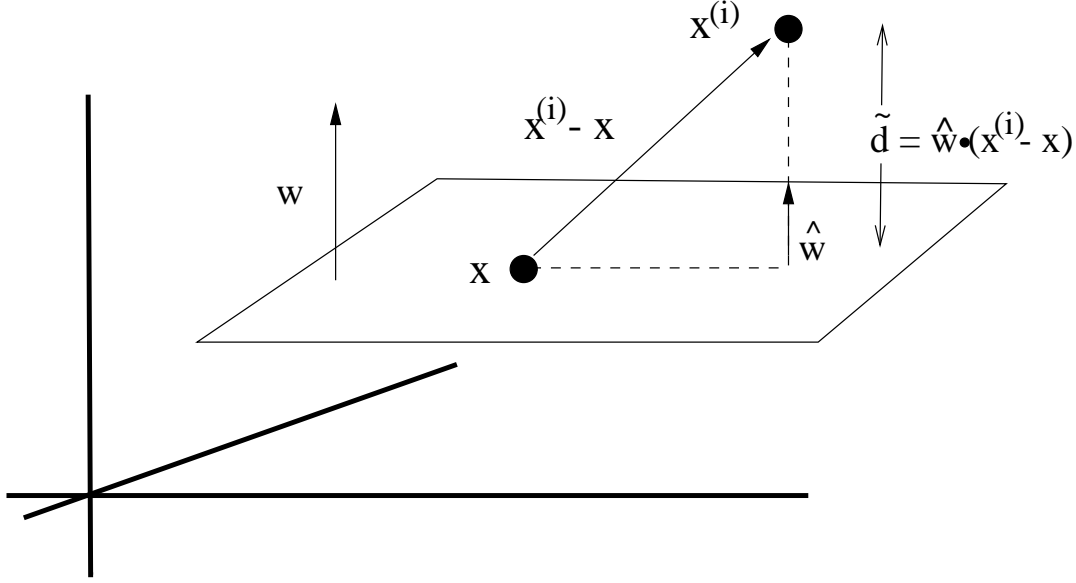


Figure 7.3: The perpendicular distance between  $\mathbf{x}$  on the plane defined by  $\hat{\mathbf{w}}^T \mathbf{x} + \tilde{b} = 0$  and  $\mathbf{x}^{(i)}$  is given by  $\tilde{d} = |\hat{\mathbf{w}}^T (\mathbf{x}^{(i)} - \mathbf{x})|$ . Here,  $\hat{\mathbf{w}}$  is the unit vector perpendicular to the decision plane and  $\tilde{b} = -\hat{\mathbf{w}}^T \mathbf{x}$ .

decision plane in Figure 7.3 and any other point  $\mathbf{x}^{(n)}$  as

$$\begin{aligned}
 \tilde{d} &= \frac{1}{\|\mathbf{w}\|} |\mathbf{w}^T (\mathbf{x}^{(n)} - \mathbf{x})|, \\
 &= \frac{1}{\|\mathbf{w}\|} |\mathbf{w}^T \mathbf{x}^{(n)} + b - \underbrace{(\mathbf{w}^T \mathbf{x} + b)}_{= 0 \text{ for } \mathbf{x} \text{ on decision boundary}}|, \\
 &= \frac{1}{\|\mathbf{w}\|} |\mathbf{w}^T \mathbf{x}^{(n)} + b|,
 \end{aligned} \tag{7.2}$$

where  $\hat{\mathbf{w}} = \mathbf{w}/\|\mathbf{w}\|$  is the unit vector perpendicular to the plane in Figure 7.3 and  $b = \tilde{b}\|\mathbf{w}\|$ . Note, if  $\mathbf{x}^{(n)}$  is on the decision line then  $\mathbf{w}^T \mathbf{x}^{(n)} + b = 0$ . If  $\mathbf{x}^{(n)}$  is on the same side of the decision plane that  $\mathbf{w}$  is pointing towards then  $\mathbf{w}^T \mathbf{x}^{(n)} + b > 0$ . If  $\mathbf{x}^{(n)}$  is on the other side then  $\mathbf{w}^T \mathbf{x}^{(n)} + b < 0$ .

The annoying absolute value operator in equation 7.2 can be eliminated by replacing it with the product of the target value  $y^{(n)}$  and  $\mathbf{w}^T \mathbf{x}^{(n)} + b$ :

$$\tilde{d} = \frac{1}{\|\mathbf{w}\|} y^{(n)} (\mathbf{w}^T \mathbf{x}^{(n)} + b), \tag{7.3}$$

which is always positive if  $\mathbf{x}^{(n)}$  is correctly classified as the value of  $y^{(n)}$ . For example, if  $\mathbf{w}$  in Figure 7.2 points to the right then the product  $y^{(n)} (\mathbf{w}^T \mathbf{x}^{(n)} + b) > 0$  is positive for  $\mathbf{x}^{(n)}$  to the right of the decision line. This is because  $y^{(n)} = +1$  and  $(\mathbf{w}^T \mathbf{x}^{(n)} + b) > 0$  are also positive. For points  $\mathbf{x}^{(n)}$  to the left of the decision line  $y^{(n)} = -1$ , then the value

$y^{(n)}(\mathbf{w}^T \mathbf{x}^{(n)} + b) > 0$  is positive because both  $(\mathbf{w}^T \mathbf{x}^{(n)} + b) < 0$  and  $y^{(n)} = -1$  are negative.

For a given  $\mathbf{w}$  and  $b$  that cleanly separates the data, the margin thickness is given by the perpendicular distance between the decision boundary and the closest point in the data set that satisfies the following conditions:

$$\begin{aligned} & \frac{1}{\|\mathbf{w}\|} \min_{\mathbf{x}^{(n)}} [y^{(n)}(\mathbf{w}^T \mathbf{x}^{(n)} + b)] \\ \text{subject to } & y^{(n)}(\mathbf{w}^T \mathbf{x}^{(n)} + b) > 0 \quad \forall \mathbf{x}^{(n)}, \end{aligned} \quad (7.4)$$

where the inequality constraint can be described as the *clean-separation* condition.

As illustrated by the dashed black and red lines in Figure 7.2, there are many different sets of  $(\mathbf{w}, b)$  that satisfy the above conditions. However, there is only one set of  $(\mathbf{w}, b)$  that satisfies the following conditions:

$$(\mathbf{w}^*, b^*) = \arg \max_{\mathbf{w}, b} \left\{ \frac{1}{\|\mathbf{w}\|} \min_{\mathbf{x}^{(n)}} [y^{(n)}(\mathbf{w}^T \mathbf{x}^{(n)} + b)] \right\}, \quad (7.5)$$

$$\text{subject to } y^{(n)}(\mathbf{w}^T \mathbf{x}^{(n)} + b) > 0 \quad \forall \mathbf{x}^{(n)}. \quad (7.6)$$

where the  $1/\|\mathbf{w}\|$  term is outside the  $\min_{\mathbf{x}^{(n)}}$  operation because it does not depend on the points  $\mathbf{x}^{(n)}$ . The optimal set  $(\mathbf{w}^*, b^*)$  and the support vectors on the margin boundary describe the SVM slab with the thickest width.

Assume that the smallest value on the lefthand side of equation 7.6 is  $\alpha > 0$  when  $\mathbf{x}^{(n)}$  is a support vector. In this case we can replace the inequality  $> 0$  in equation 7.5 by  $\geq \alpha$ . Dividing by  $\alpha$  transforms equation 7.6 into

$$y^{(n)}(\mathbf{w}^T \mathbf{x}^{(n)} + b) \geq 1 \quad \forall \mathbf{x}^{(n)}, \quad (7.7)$$

where

$$y^{(n)}(\mathbf{w}^T \mathbf{x}^{(n)} + b) = 1, \quad \forall \mathbf{x}_{supp}^{(n)}. \quad (7.8)$$

Here,  $\mathbf{x}_{supp}^{(n)}$  defined as a support vector on the margin boundary.

The vectors  $\mathbf{x}^{(n)}$  in equation 7.5 that minimize the numerator  $\min_{\mathbf{x}^{(n)}} [y^{(n)}(\mathbf{w}^T \mathbf{x}^{(n)} + b)]$  are the support vectors  $\mathbf{x}_{supp}^{(n)}$  on the margin boundaries. Therefore, equation 7.8 says that the numerator in equation 7.5 can be replaced by the value 1 so that the optimization problem can be redefined as finding  $(\mathbf{w}, b)$  such that

$$\begin{aligned} & (\mathbf{w}^*, b^*) = \arg \max_{\mathbf{w}, b} \left\{ \frac{1}{\|\mathbf{w}\|} \right\}, \\ \text{subject to } & y^{(n)}(\mathbf{w}^T \mathbf{x}^{(n)} + b) \geq 1 \quad \forall \mathbf{x}^{(n)}. \end{aligned} \quad (7.9)$$

It is algorithmically convenient to transform this maximization problem into a minimization

problem by replacing  $1/\|\mathbf{w}\|$  with  $\frac{1}{2}\|\mathbf{w}\|^2$ :

$$\begin{aligned} (\mathbf{w}^*, b^*) &= \arg \min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2, \\ \text{subject to } y^{(n)}(\mathbf{w}^T \mathbf{x}^{(n)} + b) &\geq 1 \quad \forall \mathbf{x}^{(n)}. \end{aligned} \quad (7.10)$$

The inequality demands that  $(\mathbf{w}, b)$  cleanly separates the training set  $(\mathbf{x}^{(n)}, b)$  and the minimization requirement is satisfied by the decision boundary with the thickest margin  $\tilde{d} \propto 1/\|\mathbf{w}\|$  (see equation 7.3). A numerical solution by, e.g. quadratic programming (Nocedal and Wright, 1999; Press et al., 2007), will give the optimal values of  $(\mathbf{w}, b)$  that have the fattest margin in Figure 7.2. We can treat quadratic programming as a black box, where the algorithmic details are given in Nocedal and Wright (1999). More efficient solution methods such as sub-gradient descent and coordinate descent algorithms can be applied to the dual problem ([https://en.wikipedia.org/wiki/Support\\_vector\\_machine](https://en.wikipedia.org/wiki/Support_vector_machine)).

## 7.4 Nonlinear SVM

A set of training data might not be linearly separable, such as the data points shown in Figure 7.4a. In this case, a line (or hyperplane) cannot separate the different classes of data but higher-order polynomials might be able to describe a wiggly separation boundary. To determine this nonlinear separation boundary we use a nonlinear transformation  $z_i = \phi(\mathbf{x})_i$  of the coordinates in  $\mathbf{x}$  to add extra dimensions to the solution space, just as we did for the motorcycle model in equation 2.35.

As an example, the black and green classes of data in Figure 7.4a can be separated by the red circle. This decision boundary can be described by the transformation from 2D feature coordinates  $(x_1, x_2)$  to the 3D space of  $(z_1, z_2, z_3)$  defined by

$$(x_1, x_2) \rightarrow (z_1, z_2, z_3) = (\underbrace{\phi(\mathbf{x})_1}_{x_1}, \underbrace{\phi(\mathbf{x})_2}_{x_2}, \underbrace{\phi(\mathbf{x})_3}_{\sqrt{x_1^2 + x_2^2}}). \quad (7.11)$$

A hyperplane in the transformed space is described by the  $3 \times 1$  normal vector  $\tilde{\mathbf{w}} = (\tilde{w}_1, \tilde{w}_2, \tilde{w}_3)^T$  and the bias term  $\tilde{b}$ . Figure 7.4b depicts the data plotted in this transformed coordinate system, which are now separated by the horizontal plane defined by  $z_3 = 1$ . Once the coordinates  $(z_1^b, z_2^b, z_3^b)$  of the boundary are determined from  $(\tilde{\mathbf{w}}, \tilde{b})$ , then the associated decision boundary in the original space  $(x_1, x_2)$  can be found by the inverse mapping from  $\mathbf{z}$  to  $\mathbf{x}$ .

Therefore, a nonlinear SVM defines a nonlinear transformation such that the new feature coordinates are  $z_i = \phi(\mathbf{x})_i$  and we seek the solution  $(\tilde{\mathbf{w}}, \tilde{b})$  to the following optimization problem with inequality constraints.



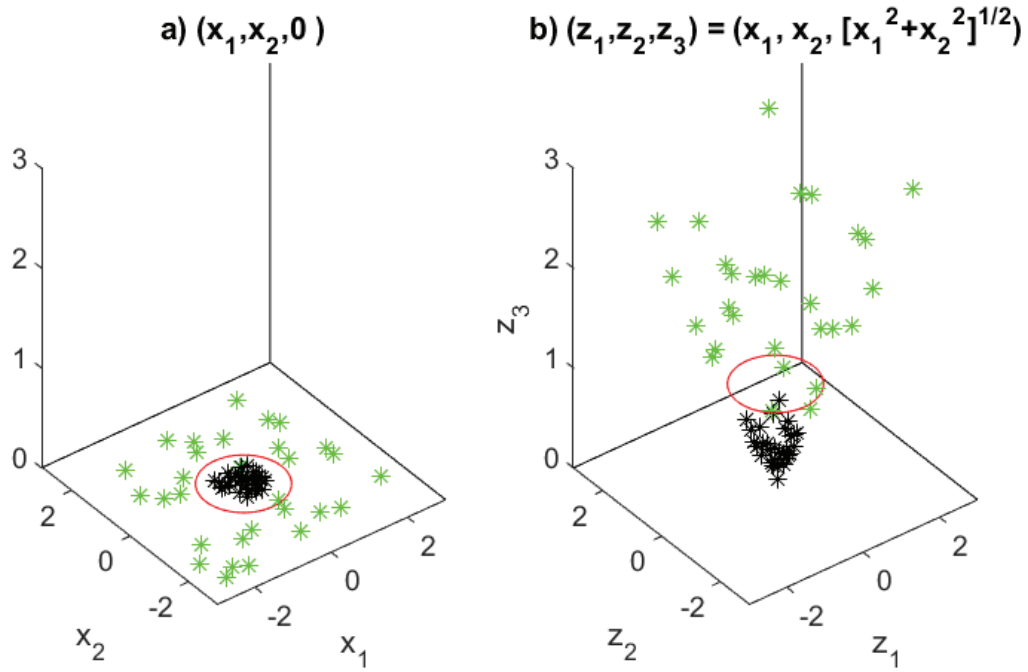


Figure 7.4: Data with a binary classification are plotted in a) the original 2D and b) transformed 3D spaces using equation 7.11. The green points above the plane  $z_3 = 1$  in b) are cleanly separated from the black points below it.

**Key Idea 7.4.1. Primal Optimization Problem with Inequality Constraints**

The primal optimization problem in the transformed coordinate system is defined as finding  $(\tilde{\mathbf{w}}^*, \tilde{b}^*)$  such that

$$\begin{aligned} (\tilde{\mathbf{w}}^*, \tilde{b}^*) &= \arg \min_{\tilde{\mathbf{w}}, \tilde{b}} \frac{1}{2} \|\tilde{\mathbf{w}}\|^2, \\ \text{subject to } y^{(n)}(\tilde{\mathbf{w}}^T \mathbf{z}^{(n)} + \tilde{b}) &\geq 1. \end{aligned} \quad (7.12)$$

where the linear inequality constraint is with respect to the  $z$ -coordinates and the tilde indicates the parameters associated with the hyperplane in the  $z$ -space. The optimal parameters  $(\tilde{\mathbf{w}}^*, \tilde{b}^*)$  define the hyperplane with the fattest margin that cleanly separates the transformed training data in  $z$ -space. After  $(\tilde{\mathbf{w}}^*, \tilde{b}^*)$  is computed, new unlabeled points  $\mathbf{x}$  can be transformed into the  $\mathbf{z}$  domain and the sign of  $\tilde{\mathbf{w}}^T \mathbf{z} + \tilde{b}$  will determine its class type. The computational cost of finding the optimal solution to equation 7.12 by a direct matrix inversion is dominated by  $O(D^3)$ , which is the same as inverting the normal matrix in a  $D \times D$  system of normal equations  $\mathbf{X}^T \mathbf{X} \mathbf{w} = \mathbf{X}^T \mathbf{y}$ . Here,  $\mathbf{w}$  is the  $D \times 1$  vector,  $\mathbf{y}$  is the  $N \times 1$  target vector and  $\mathbf{X} \mathbf{w} = \mathbf{y}$  is the  $N \times D$  system of equality constraint equations.

As in the original problem, a quadratic programming method can be used to find the solution.

The penalty in going to a higher dimension is that it can increase the computational cost by a significant amount. For example, if the feature vector is a  $D \times 1 = 10^5 \times 1$  vector and the nonlinear transform increases the dimension by a factor of four, then the cost of the quadratic programming solution increases from  $O(D^3)$  to  $O(4^3 D^3)$  algebraic operations. For a very large  $D$ , this extra cost can be reduced by solving the dual Lagrangian problem as discussed in the next section.

## 7.5 Primal and Dual Solutions

The constrained optimization problem with inequality constraints in equation 7.10 or equation 7.12 is defined as the primal problem. Its numerical solution can be as high as  $O(D^3)$  algebraic operations, where  $D$  is the dimension of the  $D \times 1$  vector  $\mathbf{w}$ . This can be expensive for  $D \gg 1$  so the alternative is to solve the dual problem.

**Key Idea 7.5.1. Dual Optimization Problem with Inequality Constraints**

The dual problem is defined as finding  $\mathbf{w}$  and  $b$  that minimize and  $\alpha$  that maximizes the Lagrangian  $L(\mathbf{w}, b, \alpha)$ :

$$(\mathbf{w}, b, \alpha) = \max_{\alpha} \min_{\mathbf{w}, b} L(\mathbf{w}, b, \alpha) = \max_{\alpha} \min_{\mathbf{w}, b} \left[ \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{n=1}^N \alpha_n \overbrace{(y^{(n)} (\mathbf{w}^T \mathbf{x}^{(n)} + b) - 1)}^{d_n} \right], \quad (7.13)$$

$$\nabla L(\mathbf{w}, b, \alpha) = \mathbf{0}, \quad \text{subject to } \alpha_n \geq 0 \text{ for } n = 1, 2 \dots N, \quad (7.14)$$

where  $L$  is the Lagrangian function discussed in Appendix 7.16. Here,  $N$  is the number of inequality constraints,  $\alpha_n$  is the Lagrange multiplier for the  $n^{\text{th}}$  inequality equation and  $\alpha = (\alpha_1, \alpha_2 \dots \alpha_N)^T$  is the  $N \times 1$  Lagrange multiplier vector. We will show that this approach only requires finding  $N$  unknowns. For SVM the number of inequality constraints is equal to the number of feature vectors in a mini-batch of data. This can result in a significant reduction in computational costs if  $N$  is much less than the number of elements in the  $D \times 1$  vector  $\mathbf{w}$ . However, the most significant benefit in the dual space is that the kernel method can be devised with the reduced dual optimization algorithm described in Box 7.5.2.

The solution to the dual problem can be obtained by a quadratic programming method (Nocedal and Wright, 1999) or algorithms such as the sub-gradient or coordinate descent methods ([https://en.wikipedia.org/wiki/Support\\_vector\\_machine](https://en.wikipedia.org/wiki/Support_vector_machine)). In this case, the non-support vectors  $\mathbf{x}^{(i)}$   $i \in I_{\text{non-support}}$  will be associated with  $\alpha_i = 0 \forall i \in I_{\text{non-support}}$  to minimize the cost function. The only non-zero parameters  $\alpha_i \neq 0 \forall i \in I_{\text{support}}$  will be those associated with the support vectors  $\mathbf{x}^{(i)}$   $\forall i \in I_{\text{support}}$ . This means that, unlike a NN, training of a SVM model does not require a large training set if it contains the necessary support vectors.

The starting point for computing the solution to equation 7.14 with a reduced Lagrangian is to find  $\mathbf{w}$  and  $b$  in terms of  $\alpha_n$ . The resulting Lagrangian is denoted as the reduced Lagrangian. Once this is done we can pass the problem to a quadratic programming code to solve for the  $N$  values of  $\alpha_i$ .

We have replaced the primal problem with inequality constraints in equation 7.10 or equation 7.12 with a similar one in equation 7.14, so what have we gained? Nothing is gained until we eliminate the  $\mathbf{w}$  and  $b$  in terms of the variables  $\alpha_n$ . This can be done by setting the gradients of the Lagrangian  $L$  w.r.t. to  $w_i$  and  $b$  to zero:

$$\frac{\partial L}{\partial w_i} = w_i - \sum_{n=1}^N \alpha_n y^{(n)} x_i^{(n)} = 0 \rightarrow \quad \mathbf{w} = \sum_{n=1}^N \alpha_n y^{(n)} \mathbf{x}^{(n)}, \quad (7.15)$$

and

$$\frac{\partial L}{\partial b} = \sum_{n=1}^N \alpha_n y^{(n)} = 0. \quad (7.16)$$

Inserting equation 7.16 into equation 7.14 gives the transformed Lagrangian:

$$\begin{aligned} L(\mathbf{w}, b, \boldsymbol{\alpha}) &= \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{n=1}^N \alpha_n (y^{(n)} (\mathbf{w}^T \mathbf{x}^{(n)} + b) - 1), \\ &= \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{n=1}^N \alpha_n y^{(n)} \mathbf{w}^T \mathbf{x}^{(n)} + \sum_{n=1}^N \alpha_n - \overbrace{b \sum_{n=1}^N \alpha_n y^{(n)}}^{\boldsymbol{\alpha}^T \mathbf{y} = 0 \text{ by eq. 7.16}}, \\ &= \sum_{n=1}^N \alpha_n + \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{n=1}^N \alpha_n y^{(n)} \mathbf{w}^T \mathbf{x}^{(n)}, \end{aligned} \quad (7.17)$$

where  $b$  is now eliminated.

#### Key Idea 7.5.2. Reduced Dual Optimization Problem with Inequality Constraints

The  $\mathbf{w}$  in equation 7.17 can be eliminated by plugging  $\mathbf{w} = \sum_{n=1}^N \alpha_n y^{(n)} \mathbf{x}^{(n)}$  from equation 7.15 into equation 7.17 to get the *reduced* Lagrangian:

$$\begin{aligned} \mathcal{L}(\boldsymbol{\alpha}) &= \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N y^{(n)} y^{(m)} \alpha_n \alpha_m \mathbf{x}^{(n)T} \mathbf{x}^{(m)}, \\ \text{subject to } &\alpha_n \geq 0; \quad \boldsymbol{\alpha}^T \mathbf{y} = 0, \end{aligned} \quad (7.18)$$

where the only unknowns are the  $N$  variables  $\alpha_n$  that maximize  $\mathcal{L}(\boldsymbol{\alpha})$  subject to the inequality constraints. Here,  $N$  is also the number of feature vectors in the training set which can be small if the original training data set is broken up into mini-batches. In this case, solving the reduced dual optimization problem is much more efficient if the dimension of the  $D \times 1$  feature vector is huge compared to the number  $N$  of feature vectors in a mini-batch. An important SVM example that demands the more efficient dual optimization approach is in medical diagnosis where each input  $D \times 1$  vector can be a tomographic scan with millions of pixel per scan (Gaonkar and Davatzikos, 2013).

Multiplying the reduced Lagrangian by  $-1$  transforms it to a minimization problem w.r.t.  $\alpha_i$ :

$$\min_{\alpha} \frac{1}{2} \alpha^T \overbrace{\begin{bmatrix} y^{(1)}y^{(1)}\mathbf{x}^{(1)T}\mathbf{x}^{(1)} & y^{(1)}y^{(2)}\mathbf{x}^{(1)T}\mathbf{x}^{(2)} & \dots & y^{(1)}y^{(N)}\mathbf{x}^{(1)T}\mathbf{x}^{(N)} \\ y^{(2)}y^{(1)}\mathbf{x}^{(2)T}\mathbf{x}^{(1)} & y^{(2)}y^{(2)}\mathbf{x}^{(2)T}\mathbf{x}^{(2)} & \dots & y^{(2)}y^{(N)}\mathbf{x}^{(2)T}\mathbf{x}^{(N)} \\ \dots & \dots & \dots & \dots \\ y^{(N)}y^{(1)}\mathbf{x}^{(N)T}\mathbf{x}^{(1)} & y^{(N)}y^{(2)}\mathbf{x}^{(N)T}\mathbf{x}^{(2)} & \dots & y^{(N)}y^{(N)}\mathbf{x}^{(N)T}\mathbf{x}^{(N)} \end{bmatrix}}^{\text{quadratic coefficients}} \alpha + (-\mathbf{1}^T)\alpha \quad (7.19)$$

$$\text{subject to} \quad \mathbf{y}^T \alpha = 0 \text{ and } \alpha_n \geq 0 \ \forall n, \quad (7.20)$$

where  $\mathbf{1}$  is the  $N \times 1$  vector  $\mathbf{1} = (1, 1, \dots, 1)$ . If  $y_i = 1 \ \forall i$  then the  $N \times N$  matrix of dot products is known as the Gram matrix (Bishop, 2006). For  $y_i$  taking on the values of  $\pm 1$  then we will refer to it as a scaled-Gram matrix. The optimal solution can be found, for example, by a quadratic programming method with a computational cost as high as  $O(N^3)$  for inverting this matrix.

If the nonlinear SVM method is used then we are seeking the  $(\tilde{\mathbf{w}}, \tilde{b})$  that gives rise to the same type of scaled-Gram matrix seen in equation 7.19, except  $\mathbf{x}^{(n)T}\mathbf{x}^{(m)}$  is replaced by  $\mathbf{z}^{(m)T}\mathbf{z}^{(n)}$ :

$$\min_{\alpha} \frac{1}{2} \alpha^T \overbrace{\begin{bmatrix} y^{(1)}y^{(1)}\mathbf{z}^{(1)T}\mathbf{z}^{(1)} & y^{(1)}y^{(2)}\mathbf{z}^{(1)T}\mathbf{z}^{(2)} & \dots & y^{(1)}y^{(N)}\mathbf{z}^{(1)T}\mathbf{z}^{(N)} \\ y^{(1)}y^{(2)}\mathbf{z}^{(2)T}\mathbf{z}^{(1)} & y^{(2)}y^{(2)}\mathbf{z}^{(2)T}\mathbf{z}^{(2)} & \dots & y^{(2)}y^{(N)}\mathbf{z}^{(2)T}\mathbf{z}^{(N)} \\ \dots & \dots & \dots & \dots \\ y^{(N)}y^{(1)}\mathbf{z}^{(N)T}\mathbf{z}^{(1)} & y^{(N)}y^{(2)}\mathbf{z}^{(N)T}\mathbf{z}^{(2)} & \dots & y^{(N)}y^{(N)}\mathbf{z}^{(N)T}\mathbf{z}^{(N)} \end{bmatrix}}^{\text{quadratic coefficients}} \alpha + (-\mathbf{1}^T)\alpha \quad (7.21)$$

$$\text{subject to} \quad \mathbf{y}^T \alpha = 0 \text{ and } \alpha_i \geq 0. \quad (7.22)$$

Here, we have to construct  $\mathbf{z}^{(n)} \ \forall n$  and then use  $\mathbf{z}^{(n)}$  to compute the dot products. However, computation of these dot products is impractical if the dimension of  $\mathbf{z}^{(n)}$  is large or has an infinite dimension. The remedy is to specify the dot products as a kernel  $\mathbf{z}(\mathbf{x})^T \mathbf{z}(\mathbf{x}') \rightarrow k(\mathbf{x}, \mathbf{x}')$ , as will be discussed in the next section. In this way we avoid the explicit computation of the dot product in the  $\mathbf{z}$  domain, and only compute the value of the kernel  $k(\mathbf{x}, \mathbf{x}')$  in the  $\mathbf{x}$  domain, e.g.  $k(\mathbf{x}, \mathbf{x}') = e^{-\|\mathbf{x} - \mathbf{x}'\|^2 / \sigma^2}$ . This is the kernel trick or kernel substitution (Bishop, 1996), but the kernel must honor the symmetry and semi-positive definite properties described in the next section.

## 7.6 Kernel Methods

A problem with the nonlinear SVM method is that the choice of a very high-order transformation  $\mathbf{z} = \phi(\mathbf{x})$  will increase the number of unknowns per feature vector and therefore increase the computational cost of, for example, the dot products. However, we must be cautious about overfitting of the training data if a highly oscillatory polynomial is selected. This problem can be mitigated by soft-margin regularization in section 7.9.

To overcome the dot-product expense in the  $\mathbf{z}$ -domain we can reformulate the nonlinear SVM problem in the dual space by specifying an infinite-dimensional kernel function  $k(\mathbf{x}, \mathbf{x}') = \mathbf{z}(\mathbf{x})^T \mathbf{z}(\mathbf{x}')$ .

In this case, equation 7.21 becomes the scaled-Gram matrix:

$$\min_{\alpha} \frac{1}{2} \alpha^T \begin{bmatrix} y^{(1)}y^{(1)}k(\mathbf{x}^{(1)}, \mathbf{x}^{(1)}) & y^{(1)}y^{(2)}k(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) & \dots & y^{(1)}y^{(N)}k(\mathbf{x}^{(1)}, \mathbf{x}^{(N)}) \\ y^{(2)}y^{(1)}k(\mathbf{x}^{(2)}, \mathbf{x}^{(1)}) & y^{(2)}y^{(2)}k(\mathbf{x}^{(2)}, \mathbf{x}^{(2)}) & \dots & y^{(2)}y^{(N)}k(\mathbf{x}^{(2)}, \mathbf{x}^{(N)}) \\ \dots & \dots & \dots & \dots \\ y^{(N)}y^{(1)}k(\mathbf{x}^{(N)}, \mathbf{x}^{(1)}) & y^{(N)}y^{(2)}k(\mathbf{x}^{(N)}, \mathbf{x}^{(2)}) & \dots & y^{(N)}y^{(N)}k(\mathbf{x}^{(N)}, \mathbf{x}^{(N)}) \end{bmatrix} \alpha + (-\mathbf{1}^T)\alpha. \quad (7.23)$$

The stationary conditions in equations 7.15-7.16 become

$$\frac{\partial L}{\partial \tilde{w}_i} = \tilde{w}_i - \sum_{n=1}^N \alpha_n y^{(n)} z_i^{(n)} = 0 \rightarrow \quad \tilde{\mathbf{w}} = \sum_{n=1}^N \alpha_n y^{(n)} \mathbf{z}^{(n)}, \quad (7.24)$$

and

$$\frac{\partial L}{\partial \tilde{b}} = \sum_{n=1}^N \alpha_n y^{(n)} = 0. \quad (7.25)$$

Here  $\tilde{b}$  and  $\tilde{\mathbf{w}}$  define, respectively, the bias and  $D_z \times 1$  hyperplane normal vector in the  $z$ -space.

For a high-dimensional  $\mathbf{z}$ , there is a huge computational savings in explicitly computing the kernel  $k(\mathbf{x}, \mathbf{x}')$  in equation 7.23 rather than computing  $\mathbf{z}(\mathbf{x})^T \mathbf{z}(\mathbf{x}')$  in equation 7.21. If  $\mathbf{z}$  is an infinite-dimensional vector  $\mathbf{z}$  then its dot product  $\mathbf{z}(\mathbf{x})^T \mathbf{z}(\mathbf{x}')$  cannot be practically computed. In contrast, if  $k(\mathbf{x}, \mathbf{x}')$  is specified as a, e.g. Gaussian kernel

$$k(\mathbf{x}, \mathbf{x}') = e^{-\frac{1}{\sigma^2} \|\mathbf{x} - \mathbf{x}'\|^2}, \quad (7.26)$$

then the kernel  $k(\mathbf{x}, \mathbf{x}')$  can easily be computed for any specified pair of vectors  $\mathbf{x}$  and  $\mathbf{x}'$ . Here,  $\frac{1}{\sigma^2}$  is a scalar that determines the width of the Gaussian kernel, and the Gaussian kernel is also known as the radial basis function (RBF). If  $\|\mathbf{x} - \mathbf{x}'\|^2$  is replaced with  $\|\mathbf{x} - \mathbf{x}'\|$  then this known as an exponential or Laplacian kernel.

As an example, take the scalar case where  $\sigma = 1$  and

$$\begin{aligned} k(x', x) &= e^{-(x' - x)^2}, \\ &= e^{-x^2} e^{-x'^2} e^{2xx'}, \\ &= e^{-x^2} e^{-x'^2} \sum_{k=0}^{\infty} \frac{2^k x^k x'^k}{k!}, \\ &\quad \text{Taylor series} \\ &= e^{-x^2} e^{-x'^2} \overbrace{(1 + 2xx' + 2x^2x'^2 + \dots)}^{\mathbf{z}(x) \cdot \mathbf{z}(x')}, \\ &= \overbrace{(e^{-x^2} (1, \sqrt{2}x, \sqrt{2}x^2, \dots))}^{\mathbf{z}(x)} \overbrace{(e^{-x'^2} (1, \sqrt{2}x', \sqrt{2}x'^2, \dots))}^{\mathbf{z}(x')}. \end{aligned} \quad (7.27)$$

Here, the infinite-dimensional vector  $\mathbf{z}(x)$  and its inner product with  $\mathbf{z}(x')$  cannot be practically computed, but the computation of the kernel  $k(x, x') = e^{-\|x - x'\|^2}$  is trivial.

The kernel  $k(\mathbf{x}, \mathbf{x}')$  must satisfy two properties (Press et al., 2007) in order to be used as a substitute for  $\mathbf{z}(\mathbf{x}')^T \mathbf{z}(\mathbf{x})$  in equation 7.21:

- $k(\mathbf{x}, \mathbf{x}')$  must be symmetric such that  $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$ .
- The Gram matrix must be positive semi-definite. Symmetry for the Gaussian kernel is proven

by inspection and the positive semi-definite property can be empirically tested by using an example matrix.

Some kernels (Bishop, 2006) that satisfy the above properties include the radial basis function or Gaussian kernel, the polynomial kernel  $(\mathbf{x}^T \mathbf{x} + 1)^p$  where the polynomial order  $p$  is specified by the user, the linear kernel where  $p = 1$  and the two-layer neural net SVM with the kernel  $\tanh(\beta_0 \mathbf{x}^T \mathbf{x} + \beta_1)$ . Here,  $\beta_0$  and  $\beta_1$  are specified by the user and only some values are effective ([https://en.wikipedia.org/wiki/Support\\_vector\\_machine](https://en.wikipedia.org/wiki/Support_vector_machine)).

**Key Idea 7.6.1. SVM Kernel Algorithm**

In summary, the nonlinear SVM kernel algorithm with an infinite-dimensional kernel is the following.

1. Specify the functional form of the kernel, e.g.  $k(\mathbf{x}^{(n)}, \mathbf{x}^{(m)}) = e^{-\gamma \|\mathbf{x}^{(n)} - \mathbf{x}^{(m)}\|^2}$ , and compute the  $N \times N$  Gram matrix in equation 7.23. Here,  $\gamma$  is a constant specified by the user where large values of  $\gamma$  indicate that the main influence of this basis function is only around points  $\mathbf{x}^{(n)}$  and  $\mathbf{x}^{(m)}$  that are close neighbors to one another. Use quadratic programming (QP), the Lagrangian barrier (Nocedal and Wright, 1999), sub-gradient or coordinate descent methods to solve for  $\alpha_n \forall n$ .
2. Define the feature vector  $\mathbf{x}$  so that its predicted classification value  $y$  is given by  $\text{sign}(\tilde{\mathbf{w}}^T \mathbf{z}(\mathbf{x}) + \tilde{b})$ , which is also known as the decision function. Substituting  $\tilde{\mathbf{w}}$  from equation 7.24 gives the predicted classification value:

$$\begin{aligned} y &= \text{sign}\left(\sum_{n=1}^N \alpha_n y^{(n)} \mathbf{z}(\mathbf{x})^T \mathbf{z}(\mathbf{x}^{(n)}) + \tilde{b}\right), \\ &= \text{sign}\left(\sum_{n \in B_{\text{support}}} \alpha_n y^{(n)} k(\mathbf{x}, \mathbf{x}^{(n)}) + \tilde{b}\right), \end{aligned} \quad (7.28)$$

where  $B_{\text{support}}$  is the set of indices associated with the support vectors where  $\alpha_i \neq 0$ . Note, the final class decision for  $\mathbf{x}$  is computed by summing the weighted classes of its neighboring points, where the weights are  $\alpha_n k(\mathbf{x}, \mathbf{x}^{(n)})$ . If  $k(\mathbf{x}, \mathbf{x}^{(n)})$  is a Gaussian kernel where  $\gamma$  is large then only the classes of its very nearest neighbors determine the class of  $\mathbf{x}$ . This is analogous to nearest neighbor interpolation.

The value of  $\tilde{b}$  in equation 7.28 is obtained by noting that the support vector  $\mathbf{z}(\mathbf{x}_i)$  for  $i \in B_{\text{support}}$  is associated with the non-zero Lagrange multiplier  $\alpha_n \neq 0$  that satisfies the constraint equation

$$\begin{aligned} y^{(i)} &= \tilde{\mathbf{w}}^T \mathbf{z}(\mathbf{x}^{(i)}) + \tilde{b}, \\ &= \sum_{n \in B_{\text{support}}} \alpha_n y^{(n)} \mathbf{z}(\mathbf{x}^{(n)})^T \mathbf{z}(\mathbf{x}^{(i)}) + \tilde{b} \quad \text{for } i \in B_{\text{support}}, \end{aligned} \quad (7.29)$$

where equation 7.24 is used for  $\tilde{\mathbf{w}}$ . Substituting  $k(\mathbf{x}^{(n)}, \mathbf{x}^{(i)}) = \mathbf{z}(\mathbf{x}^{(n)})^T \mathbf{z}(\mathbf{x}^{(i)})$  gives

$$\sum_{n \in B_{\text{support}}} \alpha_n y^{(n)} k(\mathbf{x}^{(i)}, \mathbf{x}^{(n)}) + \tilde{b} = y^{(i)}. \quad (7.30)$$

This equation is now solved for in terms of  $\tilde{b}$ . To get a more robust estimate of  $\tilde{b}$ , compute the average of the different support vectors to get  $\bar{b}$ :

$$\bar{b} = \frac{1}{N_{\text{support}}} \sum_{i \in B_{\text{support}}} [y^{(i)} - \sum_{n \in B_{\text{support}}} \alpha_n y^{(n)} k(\mathbf{x}^{(i)}, \mathbf{x}^{(n)})], \quad (7.31)$$

where  $N_{\text{support}}$  is the number of support vectors in the dual space where  $\alpha_i \neq 0$ .

3. All of the feature vectors in the validation and test sets can now be classified with equation 7.28. An example of a wiggly decision boundary is given in Figure 7.5.



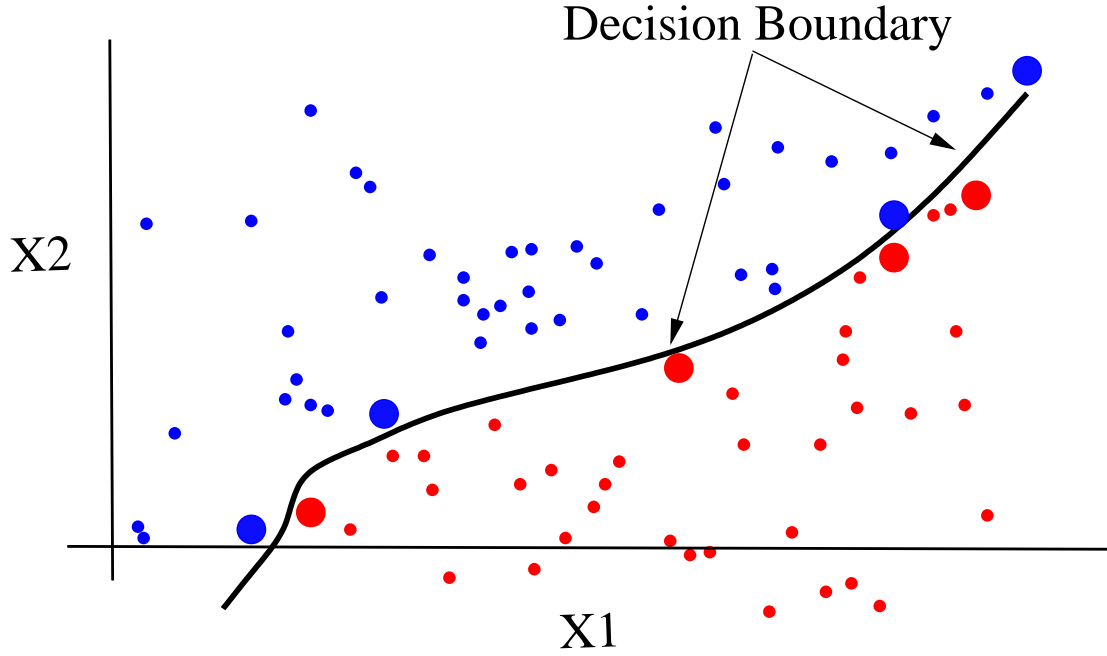


Figure 7.5: Feature points in  $(x_1, x_2)$  space with the black-decision boundary computed by the Gaussian-kernel method in the dual space. The support points are the large filled circles and the input examples are classified as either blue or red points.

## 7.7 Numerical Examples

Chapter 5 presented several examples of filtering migration images and dispersion spectra of surface waves using the SVM method. The SVM is trained to distinguish noise from signal in the classified images, and once identified, a suitable filtering method can be used to suppress the noisy artifacts in the migration image.

To train the SVM for migration filtering, the first step is to define a small  $5 \times 5$  window centered at the  $i^{th}$  pixel. Three skeletonized characteristics are then computed for the image in this small window: dominant dip angle, coherency and maximum amplitude as described in section 5.8. The user classifies each point in the training migration image as either noise or signal. The computer computes the three skeletal characteristics  $(x_1^{(i)}, x_2^{(i)}, x_3^{(i)})$  at each point of  $m(\mathbf{x})$  and assigns the classification value  $y^{(i)}$ . Therefore, the input feature vectors are the  $4 \times 1$  vectors  $(x_1^{(i)}, x_2^{(i)}, x_3^{(i)}, y^{(i)})$ . For the migration examples, fewer than 100 image points are selected for training from the  $201 \times 201$  migration image  $m(\mathbf{x})$ , which means that less than 0.2 % of the original migration image is used for training.

Figure 7.6 depicts the migration images after filtering by either a NN or an SVM method. The training was conducted using the input data of the three features maximum dip angle, coherence, and amplitude for each pixel. The signals in the images are colored yellow in Figure 7.6 as classified by a) two-node logistic regression, b) a two-layer neural network, c) SVM with a Gaussian kernel without regularization, and d) SVM with a Gaussian kernel with regularization. The SVM image

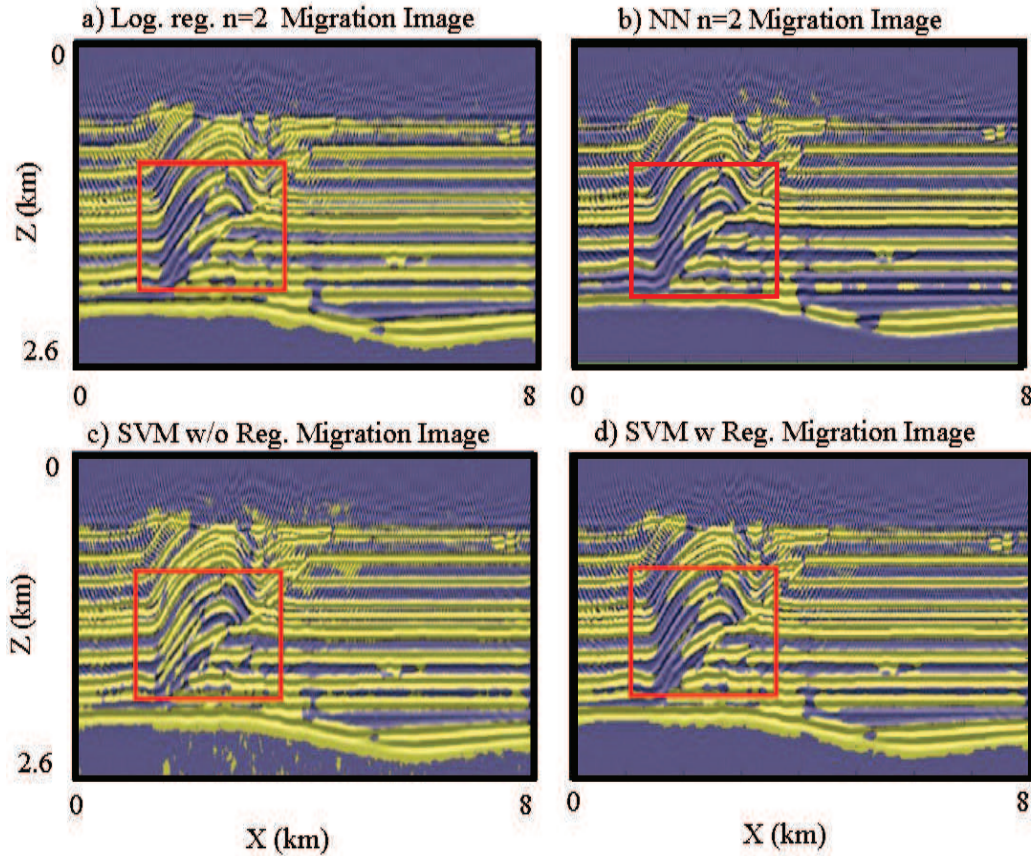


Figure 7.6: Migration images classified as signal (yellow) by a) logistic regression of order two, b) a neural network with 2 layers and 15 nodes in the hidden layer, c) SVM with a Gaussian kernel w/o regularization and d) SVM with a Gaussian kernel with regularization. Images courtesy of Yuqing Chen.

w/o regularization in Figure 7.6c is judged to be the best result because it picks out most of the steeply dipping interfaces in the red box. This suggests that the regularization strategy should be improved to not exclude steep dips.

### 7.7.1 MATLAB Codes

The migration images in Figure 7.6 were created by MATLAB codes<sup>4</sup>. Before they are executed the user trained the models by selecting 50 pixels in the raw migration image that are migration artifacts and then selects 50 pixels that are reflectors. These classified pixels are then used to train the algorithms. This lab is in the Computational labs directory<sup>5</sup>.

<sup>4</sup>[LAB1/Chapter.Book.SVM/Chapter.SVM1/lab.html](#)

<sup>5</sup>See [LAB1/Chapter.Book.SVM/Chapter.SVM1/lab.html](#).

## 7.8 Multiclass SVM

Sometimes the data need to be divided into more than two types of data, in which case a multiclass SVM method can be used. In this case the *one-versus-all* SVM approach (Vapnik, 1998; Crammer and Singer, 2001) can be used. In this method if there are  $K$  classes of data, then one of the data types, denoted as the  $k = 1$  class, is assigned one binary value by the SVM method and the rest of the data are assigned the other one. This procedure is repeated for the  $k = 2$  class, and then again one at a time until all the classes are separated. However, Bishop (2006) points out that this approach can sometimes lead to inconsistent classifications.

Another problem is that the data are imbalanced, with fewer data in the specified class than the rest of the data, and promote an imbalanced gradient that mainly reduces the residuals for the rest of the class compared to that for the specified class. This can lead to slow convergence if an iterative solver is used and a large percentage error in the predicted classifications. Balancing the gradient, also known as preconditioning in full waveform inversion, can be a partial remedy where the weight of the minority class has +1 while the other data have a weight of  $1/(K - 1)$ . If the number of points in each class differs significantly from one another then each point in a class is divided by the number of points in that class. Rebalancing the data in this way is similar to the rebalancing of drone photos in Chapter 11.

Sometimes data points are assigned by the *one-versus-all* SVM to more than one class, which is an error in classification. This contradiction is sometimes resolved by assigning the misclassified point to the class with the largest value of the predicted target function  $y$ .

A number of other heuristic approaches to multiclass SVM are described in Bishop (2007), but he states that the *one-versus-all* approach is the one that is most widely used today. Research into better multiclass SVM methods is still an ongoing effort, including the multi-kernel SVM which is used to classify multiclass lithofacies from seismic data (Liu et al., 2020c). A good overview of this issue and its partial remedies is in Cervantes et al. (2020).

## 7.9 Hinge-Loss and Soft-Margin SVM

The linear SVM assumes that the binary data points are linearly separable with the consequence that the QP solution to the dual problem will give a *hard-margin* decision boundary. This is sometimes referred to as *hard-margin* SVM<sup>6</sup>. However, data are often polluted with errors in misclassification and/or feature coordinates. To account for these errors a regularization function  $\gamma(z^{(n)})$  can be added to the SVM objective function

$$\epsilon = \underbrace{\lambda \|\mathbf{w}\|^2}_{\text{penalty}} + \underbrace{\frac{1}{N} \sum_{n=1}^N \gamma(z^{(n)})}_{\text{unregularized objective}}, \quad (7.32)$$

where  $\lambda > 0$  is a damping term and

$$\gamma(z^{(n)}) = \max(0, 1 - \overbrace{y^{(n)}(\mathbf{w}^T \mathbf{x}^{(n)} + b)}^{z^{(n)}}). \quad (7.33)$$

---

<sup>6</sup>When the dimensionality of  $\mathbf{w}$  is in the millions while the sample sizes are in the hundreds, one can always find 'hyperplanes' (the high-dimensional analogue to lines) that can separate any possible labeling of points (Gaonkar and Davatzikos, 2013). This allows us to use the hard margin SVM formulation of Vapnik (1995) instead of the soft-margin SVM described in this section.

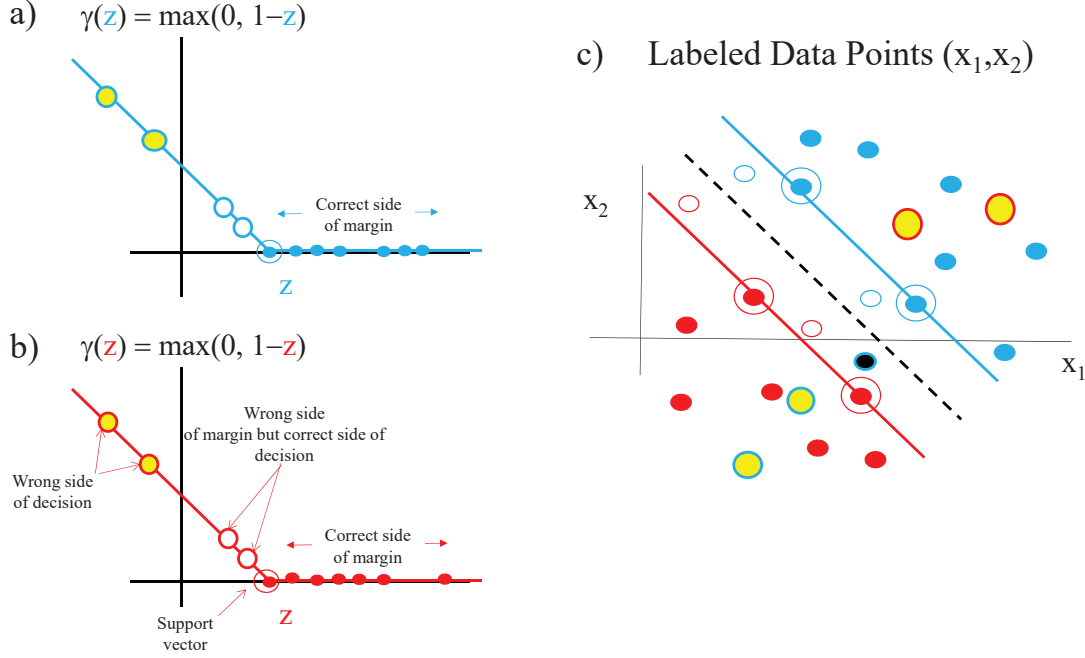


Figure 7.7: Plots of  $\gamma(z)$  versus  $z$  for a) blue and b) red points for the input data  $(x, z)$  in c). The filled yellow circles in a) and b) are data points incorrectly labeled and the open circles are correctly labeled but within the margins. This is useful because the margin thickness can be widened to tolerate these deviant points by minimizing the soft-margin objective function in equation 7.36.

Now  $\lambda \|\mathbf{w}\|^2$  is considered as the regularization term, aka the penalty function, while  $\sum_{n=1}^N \gamma(z^{(n)})$  is the unregularized objective function. If  $\lambda$  is nearly zero then the optimal model  $(\mathbf{w}^*, b^*)$  will devote most of its computations to minimizing  $\frac{1}{N} \sum_{n=1}^N \gamma(z^{(n)})$  and, therefore, create a skinny margin thickness where no points are misclassified.

The term  $\gamma(z^{(n)})$  is denoted as the hinge-loss function (Bishop, 2006) because the loss plots as the hinged lines in Figure 7.7a-7.7b. The loss becomes larger as the point  $\mathbf{x}^{(n)}$  plots farther away on the wrong side of the margin line. As examples, the open blue or open red circles, where  $0 < z < 1$ , belong to points on the wrong side of the margin boundary but are still on the correct side of the decision line. The hinge losses for these points are relatively small between 0 and 1, so a modest-sized value of  $\lambda > 0$  will allow for a thick margin that contains the open-circle points between the decision and margin boundaries. This margin thickness will certainly be thicker than the one for  $\lambda \approx 0$  where all the points have to be on the correct side of the margin boundaries. This is equivalent to the original hard-margin SVM margin described in the previous sections.

The hard-margin boundaries can be considerably softened if  $\lambda \gg 0$ . In this case, the hinge-loss function will be insignificant compared to  $\lambda \|\mathbf{w}\|^2$  even if there are many misclassified point. The consequence is that the optimal model  $(\mathbf{w}^*, b^*)$  will produce a very small value for  $\|\mathbf{w}\|$  and therefore allow for a very wide margin thicknesses<sup>7</sup> at the expense of many misclassified points. These misclassified points are the yellow-filled circles for  $z < 0$  in Figure 7.7a-7.7b and in Figure 7.7c.

In summary, the parameter  $\lambda$  in equation 7.32, determines the tradeoff between maximizing the margin width and ensuring that as many points  $\mathbf{x}^{(n)}$  are correctly classified. For a small enough

<sup>7</sup>Recall, the margin thickness is inversely proportional to  $\|\mathbf{w}\|$ .

value of  $\lambda \approx 0$ , the decision boundary is the same as that for the hard-margin SVM. In this case the margin thickness can be quite skinny if there are slight errors in the feature coordinates. However, the margin thickness can get larger in the presence of such errors if  $\lambda \gg 0$ . A concern is that strong outliers might unduly influence the final solution, so these points should be eliminated if possible.

### Key Idea 7.9.1. Soft-Margin Regularization: Primal Problem

Another form of regularization widens the margin thickness even if the classification boundary correctly classifies all of the training data (Bishop, 2006). Widening the boundary admits that some of the support vectors might be in error, such as point  $\mathbf{x}_5$  in Figure 7.8a, and so the margin thickness might actually be wider and more accurate. This is illustrated in Figure 7.8b, where the point  $\mathbf{x}_5$  has been effectively ignored (by moving it) and the new margin thickness and classification boundary have changed.

We can't arbitrarily ignore a point, but we can move a point according to the expected error tolerance denoted by  $\xi_5$ . Moving  $\mathbf{x}^{(5)}$  to  $\mathbf{x}^{(5)} + \Delta\mathbf{x}^{(5)}$  where it is located farther from the decision boundary yields the adjusted primal problem as

$$\begin{aligned} \min_{\mathbf{w}, b} \epsilon &= \frac{1}{2} \|\mathbf{w}\|^2, \quad \text{subject to } y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b) \geq 1 \quad \forall i \neq 5 \quad \text{and} \\ y^{(5)}(\mathbf{w}^T \{\mathbf{x}^{(5)} + \Delta\mathbf{x}^{(5)}\} + b) &\geq 1 \rightarrow y^{(5)}(\mathbf{w}^T \mathbf{x}^{(5)} + b) \geq 1 - \xi_5, \end{aligned} \quad (7.34)$$

where  $\xi_5 = y^{(5)} \mathbf{w}^T \Delta\mathbf{x}^{(5)} > 0$ . Instead of manually deciding the points to be moved we will let the learning algorithm determine which ones get moved and which ones are left alone by defining the general soft-margin problem as

$$\min_{\mathbf{w}, b} \epsilon = \frac{1}{2} \|\mathbf{w}\|^2, \quad \text{subject to } y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b) \geq 1 - \overbrace{\xi_i}^{\text{slack variable}}; \quad i \in \{1, 2, \dots, N\}, \quad (7.35)$$

where  $\xi_i = y^{(i)} \mathbf{w}^T \xi'^{(i)}$  is the slack variable obtained by the inversion method<sup>a</sup>. It can also be interpreted as the effective distance the point  $\mathbf{x}^{(i)}$  is perpendicularly moved away from the decision boundary. However, this is not a good strategy to let the computer decide the values of  $\xi_i$  because it will expand the decision boundary to be as wide as possible. This means that there is no limit to how far the points will be moved. We need a penalty constraint to keep the move distance  $\xi_i$  to within a tolerable error so that large moves will be penalized. This leads to the final formula for the soft-margin regularization:

$$\min_{\mathbf{w}, b} \epsilon = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_i \xi_i, \quad \text{subject to } y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b) \geq 1 - \xi_i; \quad i \in \{1, 2, \dots, N\}, \quad (7.36)$$

where  $C \geq 0$  is a user-specified regularization parameter. If  $C \gg 0$  then the algorithm will tend to make the tolerable errors  $\xi_i$  as small as possible, so it will compute a thin margin thickness as if no errors can be tolerated. This would be equivalent to SVM without regularization. If  $C$  is nearly zero then the tolerable errors are large so that the computed decision boundary will have a large margin thickness, otherwise known as a soft margin.

The constant  $C$  is user-defined and controls the tradeoff between the maximization of the margin and the minimization of classification errors on the training set.

<sup>a</sup>It is possible that there is no solution to the problem where  $|y^{(i)} - (\mathbf{w}^T \mathbf{x}^{(i)} + b)| < \epsilon \quad \forall i$ . To remedy this problem, we introduce a slack variable  $\xi_i$  for each point. These slack variables allow regression errors to exist up to values  $|\xi_i|$ .

**Key Idea 7.9.2. Soft-Margin Regularization: Dual Problem**

The dual formulation of the primal problem in equation 7.36 is similar to that of equation 7.14, except we include the slack variable term. This leads to the Lagrangian:

$$L(\mathbf{w}, \boldsymbol{\xi}, \alpha, b) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_i \xi_i + \sum_i \alpha_i [1 - \xi_i - y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b)]; \quad i \in \{1, 2, \dots, N\} \quad (7.37)$$

The first step is to minimize  $L(\mathbf{w}, \boldsymbol{\xi}, \alpha, b)$  w/r to  $\mathbf{w}$ ,  $b$  and  $\boldsymbol{\xi} = (\xi_1, \xi_2, \dots, \xi_N)$ :

$$\min_{\mathbf{w}, b, \boldsymbol{\xi}} L(\mathbf{w}, \boldsymbol{\xi}, \alpha, b), \quad \text{subject to } \xi_i \geq 0 \quad \forall i. \quad (7.38)$$

Similar to the procedure in Box 7.5 for defining the SVM dual problem, we can use the stationary equations of constraint to eliminate all but the  $\alpha_i$  variables in equation 7.38 to get the reduced Lagrangian:

$$\mathcal{L}(\boldsymbol{\alpha}) = \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N y^{(n)} y^{(m)} \alpha_n \alpha_m \mathbf{x}^{(n)T} \mathbf{x}^{(m)}, \quad (7.39)$$

$$\text{subject to } 0 \leq \alpha_n \leq C; \quad \boldsymbol{\alpha}^T \mathbf{y} = 0. \quad (7.40)$$

The inequality constraint in equation 7.40 is bounded by  $C$  in order to ensure that  $\mathcal{L}(\boldsymbol{\alpha})$  is bounded (Ricci and Perfiti, 2007). For example, if  $C - \alpha_i$  is negative then  $\xi_i$  in equation 7.38 can go to  $-\infty$  to minimize  $\mathcal{L}(\boldsymbol{\alpha})$ .

The Python code in Box 7.10.1 implements the SVM with and without regularization, so that the regularized decision boundary in Figure 7.10b is thicker and has a different slope than the unregularized decision boundary in Figure 7.10a.

The soft-margin and hinge-loss SVM formulations are often treated separately (Bishop, 2006). However, Michael Jordan at Berkeley<sup>8</sup> shows that the soft-margin SVM is equivalent to hinge-loss SVM.

**Micheal Jordan's Claim:** The soft-margin SVM is a convex program for which the objective function is the hinge loss.

**Proof:** The soft-margin SVM is:

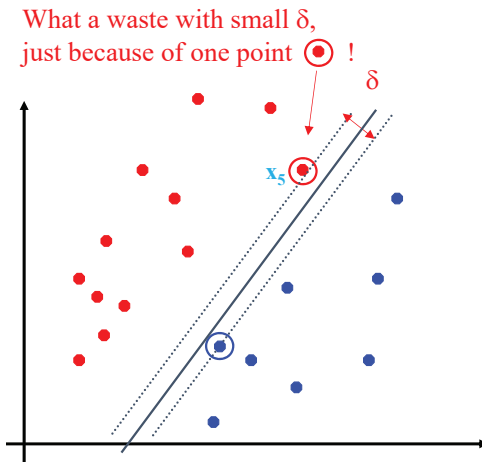
$$\arg \min_{\mathbf{w}, b, \boldsymbol{\xi}} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_i \xi_i, \quad \text{subject to } \xi_i \geq 1 - y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b); \quad \xi_i \geq 0. \quad (7.41)$$

When the data are misclassified s.t.  $y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b) \leq 0$ , the optimal  $\xi_i$  that minimizes the misfit function is  $\xi_i = 0$ . When the data are correctly classified, as indicated by  $y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b) > 0$ , the best value of  $\xi_i$  that minimizes the misfit function is  $y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b)$ . So, equation 7.41 can be expressed as the formula for hinge-loss regularization:

$$\arg \min_{\mathbf{w}, b, \boldsymbol{\xi}} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_i \max(0, y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b)). \quad (7.42)$$

<sup>8</sup><https://people.eecs.berkeley.edu/~jordan/courses/281B-spring04/lectures/lec6.pdf>

a) SVM with Thin Margin

b) Thick Margin after Moving  $\mathbf{x}_5 \rightarrow \mathbf{x}_5 + \Delta \mathbf{x}_5$ 

What is mathematical formula for moving  $\mathbf{x}_5$ ?  
 $y_5 (\mathbf{w}^T \mathbf{x}_5 + b) + y_5 (\mathbf{w}^T \Delta \mathbf{x}_5 + b) = y_5 (\mathbf{w}^T \mathbf{x}_5 + b) + \xi_5 \geq 1$

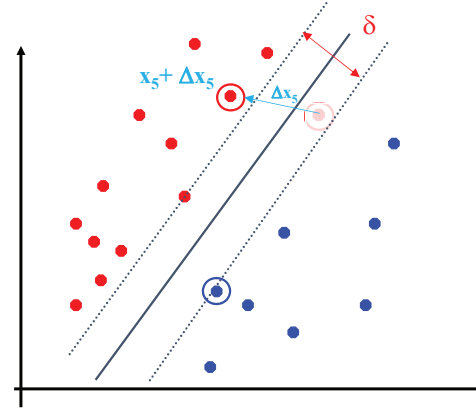


Figure 7.8: Black decision line a) before and b) after ignoring point  $\mathbf{x}_5$  in computing the optimal decision line.

## 7.10 SVM Lab with CoLab

The SVM lab<sup>9</sup> implements a SVM classifier on a small set of  $2 \times 1$  training points for different kernels. It allows the user to also test the soft-margin regularization on this data set. The Python code in Boxes 7.7.10.1-7.7.10.2 are authored by Gaël Varoquaux with documentation by Jaques Grobler.

Another lab<sup>10</sup> tests the effects of different values of  $\gamma$  and  $C$  for soft-margin SVM. Here, the RBF kernel in equation 7.26 is used where  $\gamma = 1/\sigma^2$ .

<sup>9</sup><https://colab.research.google.com/drive/1RyYlRjLqECz8ACglY1Tlz9Mx-gekCaD0?usp=sharing>

<sup>10</sup><https://colab.research.google.com/drive/1-SzbxFhuRgPyYdxaT1hvXLv1LAVtFwWx?usp=sharing>

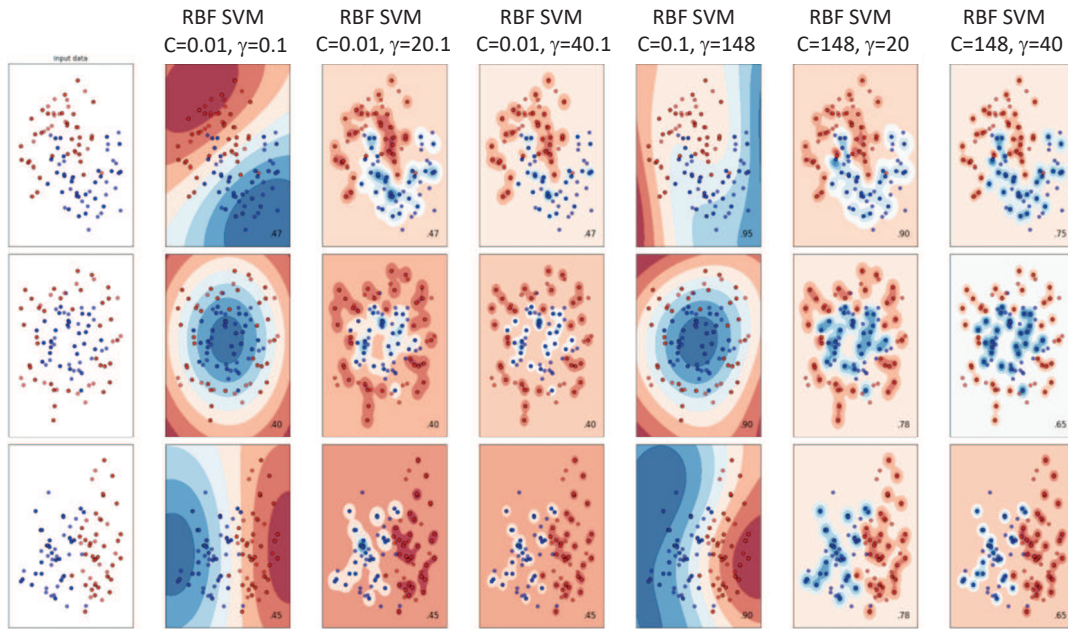


Figure 7.9: Binary labeled points on the left column and the columns to the right are the soft-margin SVM results for various values of  $\gamma = 1/\sigma^2$  in equation 7.26 and  $C$  in equation 7.36.

**Code 7.10.1.** *Python Code for SVM written by Gae"l Varoquaux, with documentation by Jaques Grobler: BSD Lic. 3.*

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
print(__doc__)

import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm

# we create 40 separable points
np.random.seed(0)
X = np.r_[np.random.randn(20, 2) - [2, 2], np.random.randn(20, 2) + [2, 2]]
Y = [0] * 20 + [1] * 20

# figure number
fignum = 1

# fit the model
for name, penalty in (('unreg', 1), ('reg', 0.05)):

    clf = svm.SVC(kernel='linear', C=penalty)
    clf.fit(X, Y)

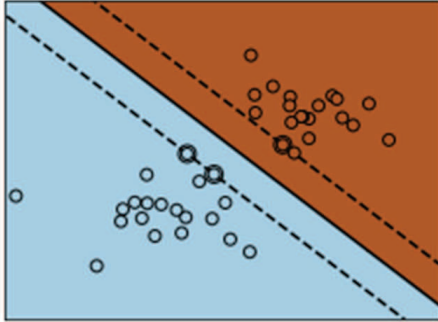
    # get the separating hyperplane
    w = clf.coef_[0]
    a = -w[0] / w[1]
    xx = np.linspace(-5, 5)
    yy = a * xx - (clf.intercept_[0]) / w[1]

    # plot the parallels to the separating hyperplane that pass through the
    # support vectors (margin away from hyperplane in direction
    # perpendicular to hyperplane). This is sqrt(1+a^2) away vertically in
    # 2-d.
    margin = 1 / np.sqrt(np.sum(clf.coef_ ** 2))
    yy_down = yy - np.sqrt(1 + a ** 2) * margin
    yy_up = yy + np.sqrt(1 + a ** 2) * margin

    # plot the line, the points, and the nearest vectors to the plane
    plt.figure(fignum, figsize=(4, 3))
    plt.clf()
    plt.plot(xx, yy, 'k-')
    plt.plot(xx, yy_down, 'k--')
    plt.plot(xx, yy_up, 'k--')
```



a) Data &amp; Non-regularized SVM



b) Data &amp; Regularized SVM

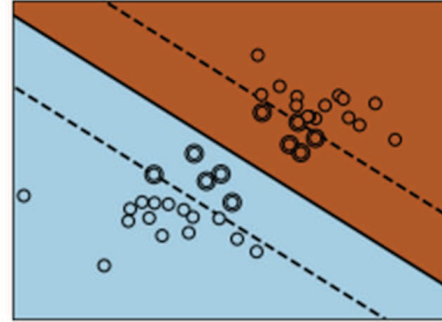


Figure 7.10: Solid black line is the SVM decision boundary a) before and b) after soft-margin regularization.

#### Code 7.10.2. Python Code for SVM (continued)

```
plt.scatter(clf.support_vectors_[0], clf.support_vectors_[1], s=80,
            facecolors='none', zorder=10, edgecolors='k')
plt.scatter(X[:, 0], X[:, 1], c=Y, zorder=10, cmap=plt.cm.Paired,
            edgecolors='k')

plt.axis('tight')
x_min = -4.8
x_max = 4.2
y_min = -6
y_max = 6

XX, YY = np.mgrid[x_min:x_max:200j, y_min:y_max:200j]
Z = clf.predict(np.c_[XX.ravel(), YY.ravel()])

# Put the result into a color plot
Z = Z.reshape(XX.shape)
plt.figure(figsize=(4, 3))
plt.pcolormesh(XX, YY, Z, cmap=plt.cm.Paired)

plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)

plt.xticks(())
plt.yticks(())
fignum = fignum + 1

plt.show()
```

The results in Figure 7.10 shows the data with the solid black line as the decision boundary a) before and b) after soft-margin regularization. Notice how the slope changed after regularization.

Another SciKit-Learning code compares the performance, displayed in Figure 7.11, of several clustering algorithms which includes a support vector machine with Gaussian kernel. The code<sup>11</sup> is described in Box 7.10.3. Note, the linear SVM decision boundary is straight and leads to larger errors, but it avoids overfitting .

<sup>11</sup> <https://colab.research.google.com/drive/1WB5GCIbrJnIHUP7ED9vNWyHXCqIDm68l#scrollTo=QvpQJqA1IsE0>

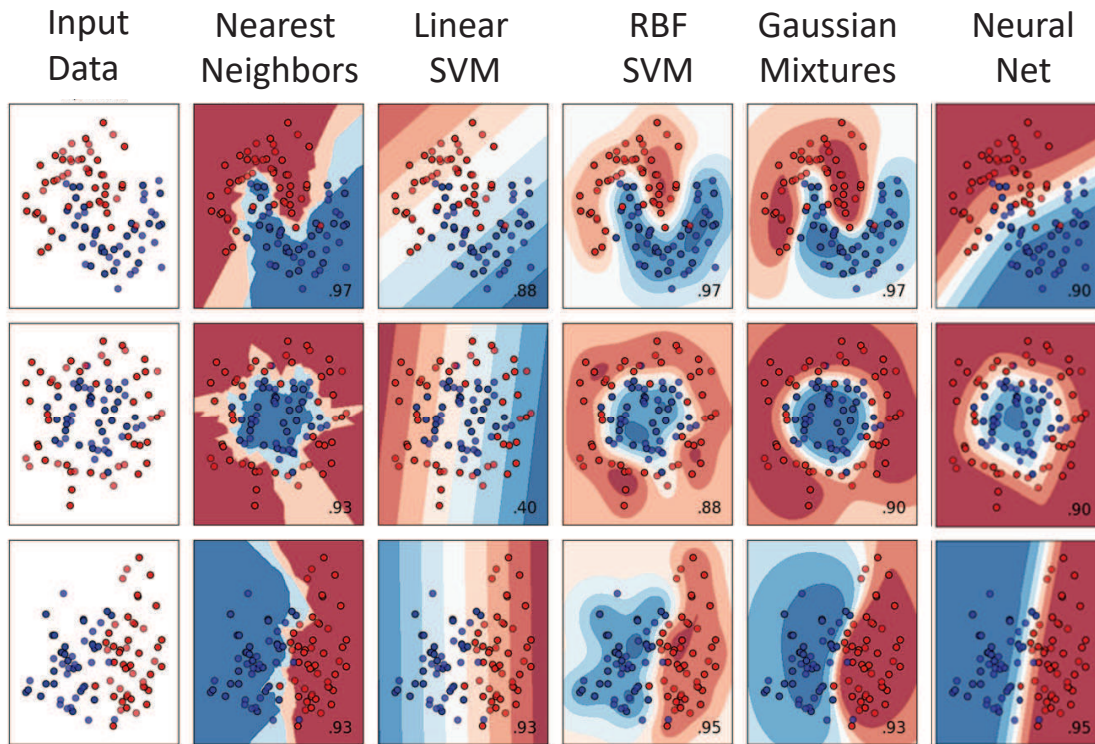


Figure 7.11: Ground truth point distributions on the left column and on the right are shown the results after classification by different clustering methods. The total testing error for classification is at the bottom of each classified image, and the training data are not shown. The term *RBF* denotes the radial basis function which is similar to the Gaussian basis function discussed in the previous section. The Python code in Box 7.10.2 reproduces these figures.

**Code 7.10.3.** *Python Code for Clustering<sup>a</sup> written by Gael Varoquaux, with documentation by Jaques Grobler: BSD Lic. 3. See Figure 7.11.*

```
# Code source: Gael Varoquaux, Andreas Muller & modified by Jaques Grobler

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_moons, make_circles, make_classification
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

h = .02 # step size in the mesh

names = ["Nearest Neighbors", "Linear SVM", "RBF SVM", "Gaussian Process",
         "Decision Tree", "Random Forest", "Neural Net", "AdaBoost",
         "Naive Bayes", "QDA"]

classifiers = [
    KNeighborsClassifier(3),
    SVC(kernel="linear", C=0.025),
    SVC(gamma=2, C=1),
    GaussianProcessClassifier(1.0 * RBF(1.0)),
    DecisionTreeClassifier(max_depth=5),
    RandomForestClassifier(max_depth=5, n_estimators=10, max_features=1),
    MLPClassifier(alpha=1, max_iter=1000),
    AdaBoostClassifier(),
    GaussianNB(),
    QuadraticDiscriminantAnalysis()]

X, y = make_classification(n_features=2, n_redundant=0, n_informative=2,
                          random_state=1, n_clusters_per_class=1)
rng = np.random.RandomState(2)
X += 2 * rng.uniform(size=X.shape)
linearly_separable = (X, y)

datasets = [make_moons(noise=0.3, random_state=0),
            make_circles(noise=0.2, factor=0.5, random_state=1),
            linearly_separable
            ]

figure = plt.figure(figsize=(27, 9))
i = 1
# iterate over datasets
for ds_cnt, ds in enumerate(datasets):
    # preprocess dataset, split into training and test part
    X, y = ds
    X = StandardScaler().fit_transform(X)
    X_train, X_test, y_train, y_test = \
        train_test_split(X, y, test_size=.4, random_state=42)

    x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
    y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                        np.arange(y_min, y_max, h))

    # just plot the dataset first
    cm = plt.cm.RdBu
    cm_bright = ListedColormap(['#FF0000', '#0000FF'])
    ax = plt.subplot(len(datasets), len(classifiers) + 1, i)
    if ds_cnt == 0:
        ax.set_title("Input data")
    # Plot the training points
    ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright,
              edgecolors='k')
    # Plot the testing points
    ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright, alpha=0.6,
              edgecolors='k')
    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(yy.min(), yy.max())
    ax.set_xticks(())
    ax.set_yticks(())
    i += 1

    # iterate over classifiers
    for name, clf in zip(names, classifiers):
        ax = plt.subplot(len(datasets), len(classifiers) + 1, i)
        clf.fit(X_train, y_train)
        score = clf.score(X_test, y_test)
```

<sup>a</sup><https://colab.research.google.com/drive/1WB5GCIbrJnIHUP7ED9vNWyHXCqIDm68l#scrollTo=QvpQJqAl1sE0>

**Code 7.10.4.** *Python Code for Clustering (continued)*

```

# Plot the decision boundary. For that, we will assign a color to each
# point in the mesh [x_min, x_max]x[y_min, y_max].
if hasattr(clf, "decision_function"):
    Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
else:
    Z = clf.predict_proba(np.c_[xx.ravel(), yy.ravel()])[:, 1]

# Put the result into a color plot
Z = Z.reshape(xx.shape)
ax.contourf(xx, yy, Z, cmap=cm, alpha=.8)

# Plot the training points
ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright,
          edgecolors='k')
# Plot the testing points
ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright,
          edgecolors='k', alpha=0.6)

ax.set_xlim(xx.min(), xx.max())
ax.set_ylim(yy.min(), yy.max())
ax.set_xticks(())
ax.set_yticks(())
if ds_cnt == 0:
    ax.set_title(name)
    ax.text(xx.max() - .3, yy.min() + .3, ('%.2f' % score).lstrip('0'),
           size=15, horizontalalignment='right')
    i += 1

plt.tight_layout()
plt.show()

```

## 7.11 SVM Regression

SVM can also be used for solving the regression problem (Drucker et al., 1996; Wu et al., 2004), and this method is known as support vector regression (SVR). It only requires a subset of the training data because the cost function does not care about points that are not support vectors. Training the SVR seeks the optimal vector  $\mathbf{w}$  that minimizes the following quadratic objective function  $\xi$  subject to the inequality constraints:

$$\begin{aligned}
 \min_{\mathbf{w}, b} \xi &= \frac{1}{2} \|\mathbf{w}\|^2, \\
 \text{subject to } &|y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)} - b| \leq \epsilon, \quad i \in \{1, 2, \dots, N\},
 \end{aligned} \tag{7.43}$$

where  $\epsilon$  is the specified threshold value of tolerable errors and  $N$  is the total number of relevant training examples. Note, the specified value of  $y^{(i)}$  is a scalar value that can be any number on the real line such that  $y^{(i)} \approx \mathbf{w}^T \mathbf{x}^{(i)} + b$ . In this case,  $\mathbf{w}^T \mathbf{x}^{(i)} + b$  is the predicted value of  $y^{(i)}$  and all predictions have to be within the specified tolerance of  $\epsilon$ . Slack variables and soft margin regularization are used to allow for approximate solutions in the presence of large errors. The description of SVM regression MATLAB codes<sup>12</sup> is at the MATLAB website<sup>13</sup>.

## 7.12 Practical Issues for Implementing SVM

There are three problems that must be addressed prior to the choice of a clustering method: scaling of feature variables, data augmentation, and computational cost.

<sup>12</sup><https://www.mathworks.com/help/stats/fitrsvm.html>

<sup>13</sup><https://www.mathworks.com/help/stats/support-vector-machine-regression.html>

### 7.12.1 Scaling

Each of the feature values might have different units (e.g. the migration filtering problem in section 7.7), so the input data need to be normalized. The SVM tries to maximize the distance between the separating plane and the support vectors. If one component, feature value, of the input vector has different units and much larger values, then it will dominate the other features when calculating the distance. If you rescale all features between, e.g. 0 and 1, they will all have the same influence on the distance metric.

This scaling can take the form of dividing each feature  $x_i$ , e.g. at the  $i^{th}$  pixel, by its standard deviation:

$$\hat{x}_i = \frac{x_i - \bar{x}}{\sigma}, \quad (7.44)$$

where the data are also demeaned by subtracting the mean value  $\bar{x}$  from the feature value  $x_i$ . An alternative is the mean normalization formula:

$$\hat{x}_i = \frac{x_i - \bar{x}}{\max(x_i) - \min(x_i)}, \quad (7.45)$$

where  $\max$  and  $\min$  take the maximum and minimum values of the feature values for all examples. This will provide an input data set with feature values varying between 1 and  $-1$  with a zero mean.

### 7.12.2 Data Augmentation

Data augmentation is used to increase the size of the training data, generalize the SVM predictive properties, balance the different types of data, and in some cases improve the accuracy of hybrid NN and SVM methods (Shima, 2018). Classification accuracy can be boosted by using dropout training and data augmentation for both linear and nonlinear SVMs (Chen, N. et al., 2018). As is well known, dropout tends to reduce the tendency for overfitting. In the case of Chen et al. (2018), input data were corrupted to give a new corrupted data set. This allowed the regularized SVM model to learn robust weights in the presence of noisy data.

Balancing the different types of data is important in order to increase the accuracy of the SVM prediction (Wu and Chang, 2003; Chawla et al., 2002; Ben-Hur and Weston, 2010). One way to do this is to oversample the minority class of samples that are undersampled (Chawla et al., 2002; Han et al., 2005) and over-weight the undersampled equations. The correction strategy used by Chawla et al (2002) randomly generated new synthetic samples along the border line between the minority samples and their random nearest neighbors. This algorithm first estimates the over-sampling rate, and then it randomly selects the minority samples and calculates their k-nearest neighbors. The final step is to generate new samples along the line between the minority sample and a random sample of the k neighbors. Han et al. (2004) only oversampled the minority class of samples near the borderline samples.

Deng et al. (2017) found that, after correcting for imbalanced data problems, the SVM classifier with a RBF could outperform a neural network in identifying the lithology of crystalline rocks taken from a deep borehole. Figure 7.12a depicts the human, NN and SVM interpretations of the well log. Figure 7.12b shows that the SVM interpretation is often more accurate than the NN interpretation. They used the *one-versus-all* strategy for the multiclassifier SVM discussed in section 7.8. Other possible remedies are reviewed in Cervantes et al. (2020).

### 7.12.3 Computational Cost

The computational cost of the kernel SVM method depends on the number  $N$  of training examples and the dimension of the feature space. As an example, the solution to the primal regularized

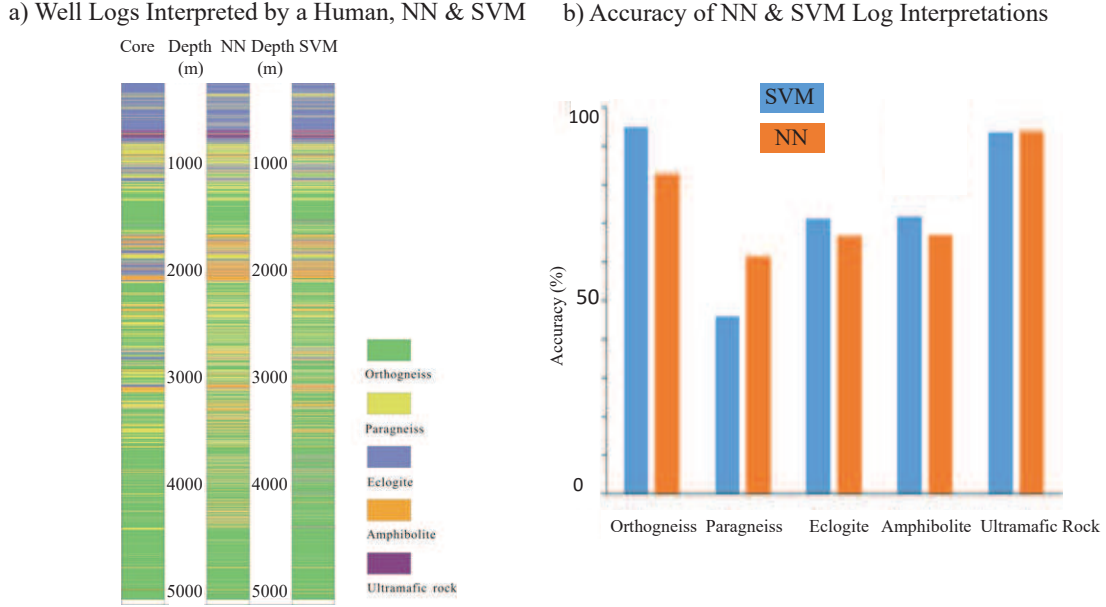


Figure 7.12: a) Well logs interpreted by a (left) human, (middle) NN, and (right) SVM. b) Accuracy of SVM and NN interpretations. Figure adapted from Deng et al. (2017).

least squares problem<sup>14</sup> requires the computation of the inverse to  $D \times D$  matrix  $(\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})$  and the computation of the matrix-matrix multiplication  $\mathbf{X}^T \mathbf{X}$ . This gives a computational complexity of  $O(ND^2 + D^3)$  (Chapelle, 2007), assuming a direct method for inversion of  $(\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})$ .

To solve the related dual problem, the computational complexity is  $O(DN^2 + N^3)$ , where the computational complexity is  $O(N^3)$  for the direct matrix inverse of  $(\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I})$  and  $O(DN^2)$  for the matrix multiplication  $\mathbf{X}\mathbf{X}^T$ ; here, the dimension of the matrix  $\mathbf{X}$  is  $N \times D$ . The matrix  $(\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I})$  corresponds to the scaled Gram matrix for SVM. If an efficient gradient optimization method is used then the computational cost can be as low as  $O(mN^2)$  for  $m \ll N$  iterations. It is argued that one should solve either the primal or dual problem depending on whether  $N$  is larger or smaller than  $D$ . In both cases, computations can also be speeded up with fast and efficient SVM algorithms for parallel computers (Perkins et al., 2015).

A solution strategy that avoids numerical inverses is the Sequential Minimal Optimization (SMO) strategy of Platt (1998). The SMO method breaks the QP solution into a sequence of much smaller QP solutions. These smaller QP solutions are analytically computed and avoid having to invert large matrices. The memory requirements grows as  $O(N)$  as the number of training examples grow. SMO's computation time is dominated by the SVM evaluation, hence SMO is fastest for linear SVMs and sparse data sets.

It is often the case that the dual problem is solved and the primal one is ignored. However, Chapelle (2007) shows that solving primal optimization can be superior to dual optimization when only approximate solutions are required.

<sup>14</sup>The primal solution to a regularized least squares (RLS) problem can be posed as finding  $\mathbf{w}^*$  that minimizes  $\lambda \mathbf{w}^T \mathbf{w} + \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$ , where the dimensions of  $\mathbf{w}$  and  $\mathbf{X}$  are  $D \times 1$  and  $N \times D$ , respectively. The RLS solution is  $\mathbf{w}^* = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$ , where  $(\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})$  is a  $D \times D$  matrix. The computational cost for minimizing the quadratic objective function with equality constraints is the same as that with inequality constraints.

If the number of training data and the dimension of the feature vectors are both large, then the computational cost of SVM can be prohibitive. An ad hoc strategy is to reduce the number of points for training by restricting most of the training set to be those with a high probability of being support vectors (Chau et al., 2010, 2013; Liu et al., 2017b). Otherwise, a NN or CNN approach can be more efficient and about as accurate.

### 7.12.4 Procedures for Users

Hsu et al. (2003) and Ben Hur (2010) proposed a list of procedures that users can follow when implementing SVM.

1. Transform data to the format of an SVM package.
2. In addition, data should be scaled and balanced w/r to the binary class types. The equations for the unbalanced class should be overweighted. The main importance of scaling is to prevent attributes with a wider range of numerical values from dominating those with a smaller range. For example, the input component  $x_j^{(i)}$  should be scaled such that  $-1 \geq x_j^{(i)} \geq 1$  or  $0 \geq x_j^{(i)} \geq 1 \forall i, j$ .

For example, assume that the first attribute of training data is scaled from  $[-10; +10]$  to  $[-1; +1]$ . If the first attribute of testing data lies in the range  $[-11; +8]$ , then we must scale the testing data to  $[-1 : 1; +0 : 8]$ .

3. Systematically try a few kernels listed in this chapter. The Gaussian kernel is usually favored (Hsu et al., 2003). The linear kernel provides a useful baseline, and in many bioinformatics applications provides the best results (Hsu et al., 2003). The flexibility of the Gaussian and polynomial kernels can lead to overfitting in high-dimensional datasets with a small number of examples. An SVM with a linear kernel is easier to tune since the only parameter that affects performance is the soft-margin constant. Once a result using a linear kernel is available, it can serve as a baseline to improve upon using a nonlinear kernel. Experience (Hsu et al., 2003) suggests that the Gaussian kernel usually outperforms the polynomial kernel in both accuracy and convergence time if the data are normalized correctly and an optimal value of the width parameter is chosen.
4. Use cross-validation described in section 4.9 to test the best values of the hyperparameters. For example,  $C$  and  $\gamma = 1/\sigma^2$  are the hyperparameters for the Gaussian kernels. Hsu et al (2003) successfully used an exponentially growing sequence of  $C$  values such as  $C = (2^{-5}, 2^{-3}, \dots, 2^3)$  to identify the optimal parameter. A coarse grid of  $C$  values is initially searched, then a fine grid search is used to find the best value of  $C$ .
5. Ben-Hur and Weston (2010) show examples for weighting unbalanced SVM data using an SVM library<sup>15</sup> of codes.

## 7.13 Summary

The linear SVM is a binary classification method that finds the optimal  $(\mathbf{w}, b)$  which gives the fattest margin distance in a  $D - 1$  hyperplane for  $N$  training examples. This is known as the solution to the primal problem where  $\mathbf{w}$  and  $b$  minimize  $\|\mathbf{w}\|^2$  subject to linear inequality constraints. Its solution can be computed by a quadratic programming method. If the dimension of the  $D \times 1$  feature vector  $\mathbf{x}^{(i)}$  is huge compared to the number  $N$  of feature vectors, then the computational cost is dominated by  $O(D^3)$  operations. In this case, solving the reduced dual problem is more efficient because its cost is dominated by  $O(N^3)$ .

<sup>15</sup><https://www.csie.ntu.edu.tw/~cjlin/libsvm/>

If the data are not linearly separable by a hyperplane, then a higher-order transform can be found that linearly separates the transformed points in the higher-dimensional space. This nonlinear transformation  $\mathbf{z}^{(i)} = \phi(\mathbf{x}^{(i)})$  maps the original feature vectors  $\mathbf{x}^{(i)}$  to the new  $D_z \times 1$  feature vector  $\mathbf{z}$ , where  $D_z > D$ . To reduce the computational costs, the primal SVM problem is transformed into the dual Lagrangian problem, where the number of unknowns  $N$  is the same as the number of training examples.

Kernel methods are used to avoid the problems of highly oscillatory basis functions with a nonlinear transform. The first step is to replace the dot product  $\mathbf{z}(\mathbf{x})^T \mathbf{z}(\mathbf{x}')$  with a suitable kernel  $k(\mathbf{x}, \mathbf{x}')$ , where there is no explicit specification of  $\mathbf{z}$ . This kernel must be symmetric and its Gram matrix must be positive semi-definite. An example is the Gaussian kernel  $k(\mathbf{x}, \mathbf{x}') = e^{-\|\mathbf{x} - \mathbf{x}'\|^2 / \sigma^2}$  and indicates the similarity of  $\mathbf{x}$  and  $\mathbf{x}'$  just like the dot product  $\mathbf{z}(\mathbf{x}')^T \mathbf{z}(\mathbf{x})$ ; if  $\mathbf{x} \approx \mathbf{x}'$  then  $k(\mathbf{x}, \mathbf{x}') = 1$  and  $\hat{\mathbf{z}}(\mathbf{x}')^T \hat{\mathbf{z}}(\mathbf{x}) = 1$ . The solution to the reduced Lagrangian is then obtained by a QP method with a computational complexity dominated by  $O(N^3)$  if a direct matrix inversion method is used.

There are benefits and liabilities of SVM compared to neural networks.

1. Mountrakis et al. (2011) state that "SVMs are particularly appealing in the remote sensing field due to their ability to successfully handle small training data sets, often producing higher classification accuracy than the traditional methods.". In contrast, the NN model typically requires a much larger training set to generalize to new data. The SVM is more efficient because it only requires training data with the support vectors to accurately determine the margin boundaries. In medical diagnosis, accurate classification of disease is the most important outcome, and so medical imaging diagnosis has a long successful use of SVM classification. However, autonomous vehicles (Kocić et al., 2019) often use NNs for classification even though SVMs can also be used.
2. Jeeva (2018) claims that "...published article trend for the neural nets vs support vector machines starting from 2000. There is a significant uptake in the article volume for the neural nets, and they have surpassed SVM significantly in active research in the last seven years."
3. The nonlinear SVM can find a higher-dimensional space where the data are linearly separable. In this case the SVM can properly classify the binary data. This is not always guaranteed for a NN because it might get stuck in a local minima (de Luca, 2020).
4. The SVM has only a few hyperparameters to choose, such as the type of kernel and the value of the regularization term. In contrast, the NN has many hyperparameters such as the number of filters, the number of layers, the filter sizes, the type of activation functions, and the architecture of the NN.
5. The SVM can have many fewer unknowns to solve for than the NN, so it can be computationally less expensive than a NN. SVMs are generally very fast to train (de Luca, 2020).
6. Both a NN and a SVM can classify with about the same accuracy. However, de Luca (2020) claims that "If given as much training and computational power as possible, however, NNs tend to outperform SVMs.". If the training set is huge then SVM can become too costly using the kernel method.
7. Unlike a NN, the number of unknowns increases with the number of input examples for an SVM. However, for well-separated classes the number of training examples is not large for an SVM (deLuca, 2020). If needed, the size of the input data can be reduced by only including data points that have a high probability of being a support vector.

Chapelle (2007) shows that solving primal optimization is superior to dual optimization when only approximate solutions are required.

8. The performance and accuracy of NNs depends on the initial weights and the number of iterations. In contrast, SVMs most often converge to the global minimum and don't require random initialization weight.



## 7.14 Exercises

1. Form the Lagrangian for the dual problem of soft-margin SVM in equation 7.32. Write down the equations for the stationarity conditions of  $L(\boldsymbol{\alpha}, \mathbf{w}, b)$ . The derivative of the *max* operator cannot be strictly defined. Replace it with a suitable approximation that can be differentiated.
2. Reformulate the objective to be  $\epsilon = 1/2\|\mathbf{w}\|^2 + C \sum_{n=1}^N \xi_n$  with the constraint

$$\mathbf{w}^T \mathbf{x}^{(n)} + b - 1 \geq \xi_n, \quad (7.46)$$

where  $\xi_n \geq 0$ . Here,  $C$  is a specified tradeoff parameter. What are the Karush-Kuhn-Tucker conditions? See Appendix 7.17 for the definition of the Karush-Kuhn-Tucker (KKT) conditions.

3. Derive the reduced Lagrangian for the above problem and state the constraints. See page 333 in Bishop (2007).
4. Go to the CoLab site<sup>16</sup> and execute the Python code for classifying a small set of  $2 \times 1$  vectors.
5. Download the SVM regression codes<sup>17</sup> and fit a SVM curve to a cosine function with additive random noise. Assess the effects of changing the value of  $\sigma$  using a Gaussian kernel.
6. Use the cross-validation procedure listed at the MATLAB www site<sup>18</sup> for fitting the noisy cosine function.

## 7.15 Computational Labs

1. Go to the CoLab site<sup>19</sup> and execute the Python code for classifying a small set of  $2 \times 1$  vectors. This code was written by G. Varoquaux to perform regularized and unregularized SVM for different kernels.
2. Go to the soft-margin SVM lab<sup>20</sup> and explain the different results for different values of  $\gamma$  and  $C$ .
3. Go to the SVM lab in *LAB1/Chapter.Book.SVM/Chapter.SVM1/lab.html* and run the SVM lab. This MATLAB code computes the decision boundaries that separate signal from noise in migration images, and then mutes the noise in the migration image. The performance of the SVM is compared to that of the neural network and logistic regression algorithms.
4. Go to the CoLab<sup>21</sup> and compare the clustering effectiveness on two clouds of noisy points, each one hugging one of the two neighboring cosine functions.

## 7.16 Appendix: Defining the Dual Problem with a Lagrangian

Finding the optimal  $\mathbf{x}$  that minimizes or maximizes a quadratic functional  $f(\mathbf{x})$  subject to  $N$  inequality constraints  $g(\mathbf{x})^{(n)} \geq 0$  can be recast as the solution to the dual problem. The objective function for the dual problem is formed by adding  $N$  weighted inequality constraints onto  $f(\mathbf{x})$  to form what is known as the Lagrangian:  $L = \frac{1}{2}f(\mathbf{x}) - \sum_n \alpha_n g(\mathbf{x})^{(n)}$  subject to the simpler inequality

<sup>16</sup><https://colab.research.google.com/drive/1RyYIRJLqECz8ACglY1Tlz9Mx-gekCaD0?usp=sharing>

<sup>17</sup><https://www.mathworks.com/help/stats/fitrsvm.html>

<sup>18</sup><https://www.mathworks.com/help/stats/fitrsvm.html>

<sup>19</sup><https://colab.research.google.com/drive/1RyYIRJLqECz8ACglY1Tlz9Mx-gekCaD0?usp=sharing>

<sup>20</sup><https://colab.research.google.com/drive/1-SzbxFhuRgPyYdxaT1hvXLv1LAVtFwWx?usp=sharing>

<sup>21</sup><https://colab.research.google.com/drive/1WB5GCIbrJnIHUP7ED9vNWYHXCqIDm68l#scrollTo=QvpQJqA11sE0>

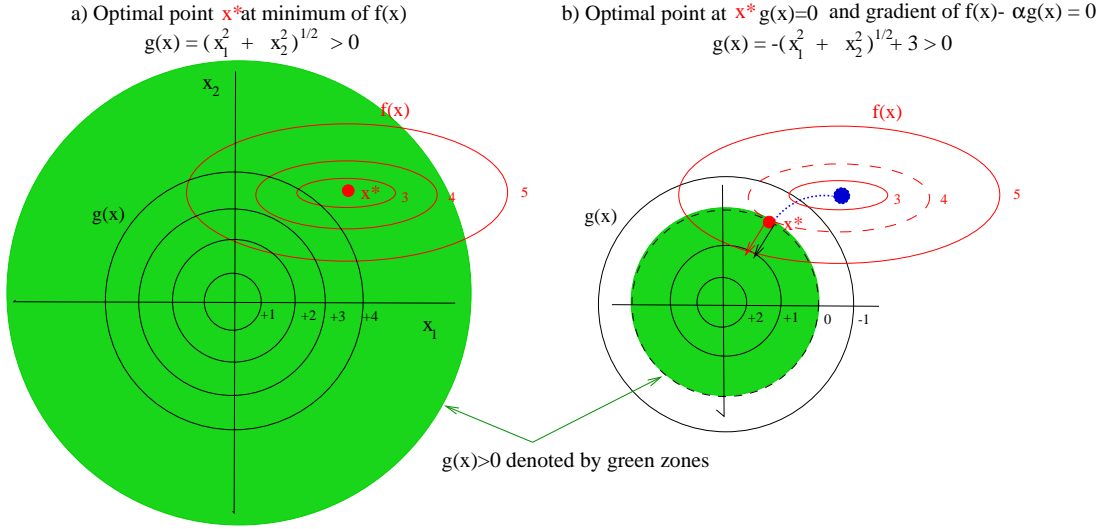


Figure 7.13: Red and black contours for the functions  $f(\mathbf{x})$  and  $g(\mathbf{x})$ , respectively; the green areas define the points  $\mathbf{x} \in V_o$  that satisfy the inequality constraint  $g(\mathbf{x}) \geq 0$ . In a) the constraint  $g(\mathbf{x}) = \sqrt{x_1^2 + x_2^2} \geq 0$  is satisfied everywhere so the minimum of  $f(\mathbf{x})$  is at the red dot  $\mathbf{x}^*$ . In b),  $g(\mathbf{x}) = -\sqrt{x_1^2 + x_2^2} + 3 \geq 0$  so the possible, i.e. feasible, solution points  $\mathbf{x}$  are restricted to be inside the small green disk. The points  $\mathbf{x} \in B_o$  on the dashed circular boundary satisfy  $g(\mathbf{x}) = 0$ , and the optimal point where  $L$  is stationary and a maximum is at the "kiss" point  $f(\mathbf{x})$  on  $B_o$  where  $\nabla L(\mathbf{x}, \alpha) = \nabla f(\mathbf{x}) - \alpha \nabla g(\mathbf{x}) = 0$ . The points along the dashed blue line are defined to be the ones where the elliptical and circular contours "kiss" each other, i.e. they satisfy the stationary condition  $\nabla f(\mathbf{x}) - \alpha \nabla g(\mathbf{x}) = 0$  where  $\alpha = |\nabla f(\mathbf{x})|/|\nabla g(\mathbf{x})|$ .

constraints  $\alpha_n \geq 0$ . There is an unknown weight  $\alpha_n$ , also known as the Lagrange multiplier, for each constraint so that there are  $N$  more unknowns to be determined. This dual problem is sometimes computationally less expensive to solve than the *primal* problem, as discussed in section 7.5. We now give a heuristic and intuitive derivation of a Lagrangian function for one inequality constraint, i.e.  $n = 1$ .

Consider the following minimization problem with one inequality constraint:

$$\begin{aligned} & \underset{\mathbf{x}}{\text{minimize}} && f(\mathbf{x}) \\ & \text{subject to} && g(\mathbf{x}) \geq 0, \end{aligned} \quad (7.47)$$

Figure 7.13a shows a 2D example where  $f(\mathbf{x})$  describes the red elliptical contours and  $g(\mathbf{x}) = \sqrt{x_1^2 + x_2^2}$  describes a circle for the inequality constraint. In this case all points in the infinitely-extended green region in Figure 7.13a satisfy the inequality constraint  $g(\mathbf{x}) \geq 0$  so that the solution to equation 7.47 is at the center point  $\mathbf{x}^*$  of the red ellipse. An unconstrained steepest descent method can then be used to search for  $\mathbf{x}$  that satisfies  $\nabla f(\mathbf{x}) = 0$ .

If  $g(\mathbf{x}) = -\sqrt{x_1^2 + x_2^2} + 3 \geq 0$  then the feasible points  $\mathbf{x}$  are only in the small green disk in Figure 7.13b. Which of these feasible points minimizes  $f(\mathbf{x})$ ? It is geometrically clear that the dashed red ellipse with the contour value  $f(\mathbf{x}) = 4$  just *kisses*<sup>22</sup> the boundary of the green disk at

<sup>22</sup>The kiss point is the point where the tangents of the  $g(\mathbf{x})$  and  $f(\mathbf{x})$  contours are parallel to one another.

$\mathbf{x}^*$ . No other ellipse contour with a *smaller* value intersects the green region. Therefore,  $\mathbf{x}^*$  must be the solution that minimizes  $f(\mathbf{x})$  and also satisfies the inequality constraint in equation 7.47. As discussed below,  $L$  achieves a maximum at this stationary point.

Notice that the circular contour at  $\mathbf{x}^*$  in Figure 7.13b is tangent to the red dashed contour of the ellipse. This means that the normal vectors  $\nabla g(\mathbf{x})$  and  $\nabla f(\mathbf{x})$  at  $\mathbf{x}^*$  are parallel<sup>23</sup> to one another at the minimization point  $\mathbf{x}^*$ . This suggests that we can define a new function  $L$  that is the weighted sum

$$L(\mathbf{x}, \alpha) = f(\mathbf{x}) - \alpha g(\mathbf{x}), \quad (7.48)$$

which has the property of being stationary at the *kissing* point  $\mathbf{x}^*$ :

$$\begin{aligned} \nabla L(\mathbf{x}, \alpha)|_{\mathbf{x}=\mathbf{x}^*} &= [\nabla f(\mathbf{x}) - \alpha \nabla g(\mathbf{x})]_{\mathbf{x}=\mathbf{x}^*}, \\ &= 0, \end{aligned} \quad (7.49)$$

which implies  $\alpha = |\nabla f|/|\nabla g| \geq 0$ . The weight  $\alpha \geq 0$  because the gradients of both functions are parallel and have the same sign at  $\mathbf{x}^*$ , so  $\alpha \geq 0$  to satisfy equation 7.49. In the third quadrant of b), the gradients  $\nabla f$  and  $\nabla g$  at their kissing points are *anti-parallel* so  $\alpha \geq 0$  excludes the stationarity condition in equation 7.49.

Equation 7.49 is a necessary but not sufficient condition for  $\mathbf{x}^*$  to be a minimizer of  $f(\mathbf{x})$ , because equation 7.49 cannot be satisfied for multiple  $\mathbf{x}^*$ 's lying on the blue *kiss*<sup>24</sup> curve in Figure 7.13b. The blue kiss curve, can extend into the green disk in Figure 7.13, and can be parameterized as  $\mathbf{x}^*(\alpha)$  because along this curve, given a feasible  $\alpha$ , there exists a *kissing* point  $\mathbf{x}^*$  that satisfies equation 7.49. With this parametrization, the reduced Lagrangian  $\mathcal{L}(\alpha)$  is given by

$$\mathcal{L}(\alpha) = L(\mathbf{x}^*(\alpha), \alpha). \quad (7.50)$$

We therefore need another mathematical condition in addition to equation 7.49 to pick out the optimal *kiss* point  $\widehat{\mathbf{x}}^*$  at the dashed black boundary in Figure 7.13b. This new condition is that, if we let the optimal  $\widehat{\mathbf{x}}^*$  be associated with  $\widehat{\alpha}$ , namely,  $\widehat{\mathbf{x}}^* = \mathbf{x}^*(\widehat{\alpha})$ , then  $\mathcal{L}(\widehat{\alpha})$  is maximum over  $\alpha$  along the kiss curve  $\mathbf{x}^*(\alpha)$ . This is found by

$$\frac{d\mathcal{L}(\alpha)}{d\alpha} = \left[ \frac{d\mathbf{x}^*}{d\alpha} \right]^T \overbrace{\nabla L(\mathbf{x}, \alpha)|_{\mathbf{x}=\mathbf{x}^*}}^0 + \overbrace{\frac{\partial L(\mathbf{x}, \alpha)}{\partial \alpha}|_{\mathbf{x}=\mathbf{x}^*}}^{-g(\mathbf{x}^*)}, \quad (7.51)$$

$$= -g(\mathbf{x}^*), \quad (7.52)$$

which follows from equations 7.48 and 7.49. At the boundary point  $\widehat{\mathbf{x}}^* = \mathbf{x}^*(\widehat{\alpha})$  we have  $\frac{d\mathcal{L}}{d\alpha} = -g(\widehat{\mathbf{x}}^*) = 0$ . In case of  $\alpha > \widehat{\alpha}$ , the kiss point enters the green disk, and so the slope  $-g(\mathbf{x}^*(\alpha)) < 0$ . In the case of  $\alpha < \widehat{\alpha}$ , the kiss point exits the green disk, and so the slope  $-g(\mathbf{x}^*(\alpha)) > 0$ . This is the condition for  $\mathcal{L}(\alpha)$  to be a maximum at  $\widehat{\alpha}$  along the kiss curve.

Therefore, the two conditions that are necessary and sufficient for  $\mathbf{x}^*$  to minimize  $f(\mathbf{x})$  subject to the inequality constraints are

$$\begin{aligned} &\underset{\alpha}{\text{maximize}} \quad \mathcal{L}(\alpha), \\ &\text{subject to} \quad \nabla f(\mathbf{x}) - \alpha \nabla g(\mathbf{x}) = 0 \text{ and } \alpha \geq 0. \end{aligned}$$

For  $N$  inequality constraints, the Lagrangian takes the form of equation 7.14. The next section

<sup>23</sup>The values of the ellipse and circle are both increasing to the left at  $\mathbf{x}^*$ . Hence, the gradients are parallel to one another, not anti-parallel.

<sup>24</sup>The blue kiss curve is defined by the *kissing* points where  $\nabla L = 0$  for  $\alpha = |\nabla f|/|\nabla g|$

shows a 1D example for solving the Lagrangian dual problem.

### 7.16.1 Simple Example of a Dual Solution

For a simple 1D convex minimization problem with inequality constraints, the optimization problem is defined as

$$\begin{aligned} & \underset{x}{\text{minimize}} && f(x) \\ & \text{subject to} && g(x) \geq 0, \end{aligned} \quad (7.53)$$

For the example  $f(x) = x^2$  and one inequality constraint, equation 7.53 becomes

$$\begin{aligned} & \underset{x}{\text{minimize}} && f(x) = x^2 \\ & \text{subject to} && g(x) = 0.7x - 0.14 \geq 0, \end{aligned} \quad (7.54)$$

where  $f(x) = x^2$ ,  $g(x) = 0.7x - 0.14$ , and  $h(x) = f(x) - \alpha g(x)$  are plotted in Figure 7.14. The inequality constraint is  $g(x) = 0.7x - 0.14 \geq 0$  so that the feasible points are for  $x \geq 0.2$ . It is obvious that  $x^* = 0.2$  at the green star minimizes  $f(x)$  for the feasible points  $x \geq 0.2$ . In contrast, the global minimum of  $f(x)$  is at  $x = 0$  to the left of  $x^*$ , but  $x = 0$  will violate the inequality constraint  $x \geq 0.2$ .

An alternative to the above optimization problem is to find  $x$  that maximizes the reduced Lagrangian

$$\begin{aligned} & \underset{\alpha}{\text{maximize}} && L(x(\alpha), \alpha) = f(x) - \alpha g(x), \\ & \text{subject to} && \alpha > 0, \end{aligned} \quad (7.55)$$

where  $\alpha$  is the Lagrange multiplier. These two stationarity conditions are expressed as

$$\begin{aligned} \frac{\partial L}{\partial x} &= \frac{\partial f}{\partial x} - \alpha \frac{\partial g}{\partial x} = 0 \\ \frac{\partial L}{\partial \alpha} &= -g(x) = 0 \quad \text{subject to } \alpha > 0. \end{aligned} \quad (7.56)$$

For the equation 7.54 example,

$$L(x(\alpha), \alpha) = x^2 - \alpha(0.7x - 0.14), \quad (7.57)$$

so that equation 7.56 becomes

$$\frac{\partial L}{\partial x} = \frac{\partial x^2}{\partial x} - \alpha \frac{\partial(0.7x - 0.14)}{\partial x} = 2x - 0.7\alpha = 0 \rightarrow \alpha = x/0.35, \quad (7.58)$$

$$\frac{\partial L}{\partial \alpha} = -0.7x + 0.14 = 0 \rightarrow x = 0.2. \quad (7.59)$$

Plugging equation 7.59 into equation 7.58 gives  $\alpha = 0.2/0.35 = 0.57$ , which is associated with the dashed blue curve  $L(x, \alpha = 0.57)$  plotted in Figure 7.14. This value of  $\alpha = 0.57 > 0$  satisfies the positivity constraint and also balances out the gradients of  $df/dx$  and  $dg/dx$  so that the sum  $\partial L/\partial x = \partial f/\partial x - \alpha \partial g/\partial x = 0$  is stationary at  $x^* = 0.2$ .

At each value of  $x$  there is a unique value of  $\alpha = x/0.35$  (see equation 7.58) that allows  $\partial L/\partial x = 0$ . Setting  $\alpha \rightarrow \alpha(x) = x/0.35$  from equation 7.58 says that  $L(x, \alpha(x))$  is a maximum at  $x^* = 0.2$

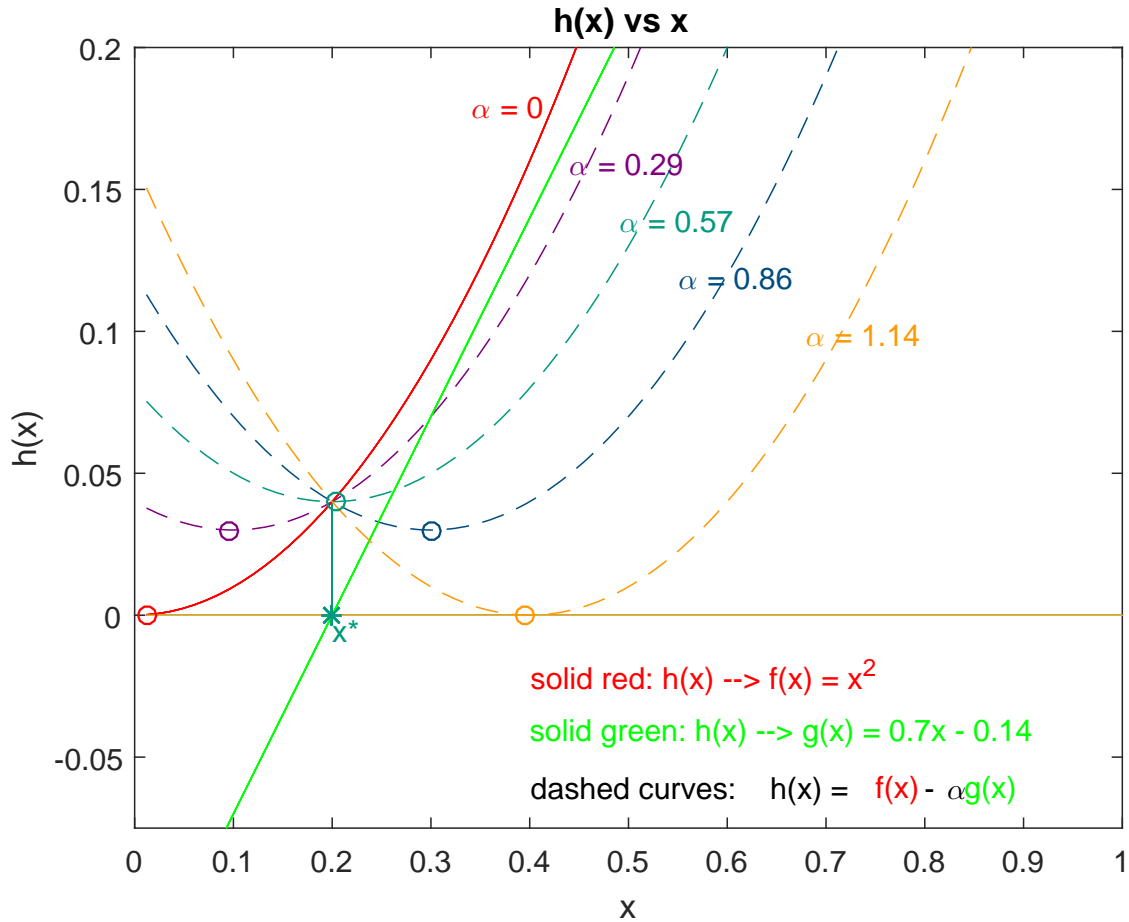


Figure 7.14: Plots of  $f(x) = x^2$ ,  $g(x) = 0.7x - 0.14$ , and  $f(x) - \alpha g(x)$ , where  $\mathbf{x}^*$  is the optimal solution that minimizes  $L$  subject to the inequality constraint. The Lagrangian  $L(x, \alpha) = f(x) - \alpha g(x)$  is plotted as dashed lines for various values of  $\alpha$ . Notice how the Lagrangian  $h(x)$  is first minimized for fixed values of  $\alpha \in \{0, 0.29, 0.57, 0.86, 1.14\}$  to give a list of  $h(x)$  values at the open circles. The maximum value in this list is the green open circle at  $x^*$ , which is the optimal solution.

because it satisfies the maximality conditions:  $\partial L / \partial \alpha = 0$  for  $x = 0.2$  and its second derivative

$$\begin{aligned} \frac{\partial^2 L(x = 0.35 * \alpha, \alpha)}{\partial \alpha^2} &= \frac{\partial^2 (0.1225\alpha^2 - 0.245\alpha^2 + 0.049\alpha)}{\partial \alpha^2}, \\ &= -.245 \end{aligned} \quad (7.60)$$

is negative<sup>25</sup>. Thus, the values of  $L(x, \alpha(x))$  plotted against  $x$  intersect the circled points in Figure 7.14 where  $L(x, \alpha(x))$  is a maximum at  $x^* = 0.2$ .

### 7.16.2 Concavity and Maximization of the Reduced Lagrangian

The previous section showed a simple example where a max-min optimization procedure is used to find the optimal solution to a quadratic function subject to a linear inequality constraint. The optimization procedure is used to find the minimum of  $L$  w/r to  $\mathbf{w}$  and  $b$  because  $\|\mathbf{w}\|^2$  is a convex function in  $\mathbf{w}$ . And then the maximum of  $L$  w/r to  $\alpha$  is found because  $L$  is a concave function in  $\alpha$ . We now show this procedure is valid for a more general Lagrangian function with many inequality constraints.

From equation 7.48, it follows that

$$\begin{aligned} L(\mathbf{x}, (1 - \beta)\alpha^0 + \beta\alpha^1) &= f(\mathbf{x}) - [(1 - \beta)\alpha^0 + \beta\alpha^1] g(\mathbf{x}), \\ &= [(1 - \beta) + \beta] f(\mathbf{x}) - [(1 - \beta)\alpha^0 + \beta\alpha^1] g(\mathbf{x}), \\ &= (1 - \beta) \{f(\mathbf{x}) - \alpha^0 g(\mathbf{x})\} + \beta \{f(\mathbf{x}) - \alpha^1 g(\mathbf{x})\}, \\ &= (1 - \beta) L(\mathbf{x}, \alpha^0) + \beta L(\mathbf{x}, \alpha^1), \end{aligned} \quad (7.61)$$

where the interpolation coefficient  $\beta \in [0, 1]$ . By evaluating this equation at  $\mathbf{x} = \mathbf{x}^*$  with  $\mathbf{x}^* = \arg \min_{\mathbf{x}} L(\mathbf{x}, (1 - \beta)\alpha^0 + \beta\alpha^1)$ , which depends implicitly on  $(1 - \beta)\alpha^0 + \beta\alpha^1$ , and by using equation 7.50 on the left-hand side (LHS) of equation 7.61, we arrive at

$$\mathcal{L}((1 - \beta)\alpha^0 + \beta\alpha^1) = (1 - \beta) \underbrace{L(\mathbf{x}^*, \alpha^0)}_{\text{to be bounded}} + \beta \underbrace{L(\mathbf{x}^*, \alpha^1)}_{\text{to be bounded}}. \quad (7.62)$$

We now show that the two Lagrangians  $L(\mathbf{x}, \alpha^0)$  and  $L(\mathbf{x}, \alpha^1)$  can be bounded from below w/r to variations in  $\mathbf{x}$ . For  $j =$  either 0 or 1, let  $\mathbf{x}^{j,*} = \arg \min_{\mathbf{x}} L(\mathbf{x}, \alpha^j)$ , namely,

$$\underbrace{L(\mathbf{x}^{j,*}, \alpha^j)}_{\mathcal{L}(\alpha^j)} \leq \underbrace{L(\mathbf{x}, \alpha^j)}_{L(\mathbf{x}^*, \alpha^j)}, \quad \forall \mathbf{x}, \text{ specifically } \mathbf{x} = \mathbf{x}^*, \quad (7.63)$$

where the left under-brace follows from equation 7.50. Equation 7.63 says that  $x^*$  is not as good as  $x^{j,*}$  at bringing  $L(\cdot, \alpha^j)$  to the minimum, because  $x^*$  is best at doing something else, namely, minimizing equation 7.61. Evaluating  $j = 1$  and  $2$ , in 7.63 gives

$$\mathcal{L}(\alpha^0) \leq L(\mathbf{x}^*, \alpha^0), \quad (7.64a)$$

$$\mathcal{L}(\alpha^1) \leq L(\mathbf{x}^*, \alpha^1). \quad (7.64b)$$

Inserting the right-hand sides (RHS's) of inequalities 7.64(a) and 7.64(b) into the RHS of equation 7.62 leads to

$$\mathcal{L}((1 - \beta)\alpha^0 + \beta\alpha^1) \geq (1 - \beta)\mathcal{L}(\alpha^0) + \beta\mathcal{L}(\alpha^1). \quad (7.65)$$

---

<sup>25</sup>The negative second-derivative is an alternative to the first-derivative maximum conditions we used earlier.

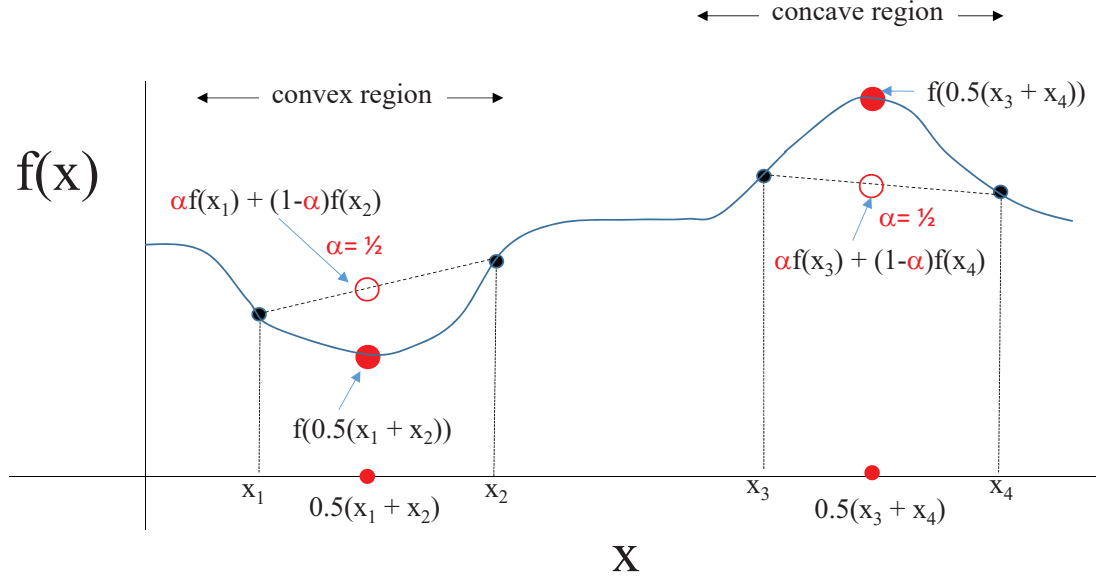


Figure 7.15: The function  $f(x)$  is convex for  $x_1 \leq x \leq x_2$  and concave for  $x_3 \leq x \leq x_4$ .

Equation 7.65 says that the reduced Lagrangian  $\mathcal{L}(\cdot)$  is concave. An example of a concave function is plotted on the right of Figure 7.15. It is known that a concave function has one global maximum. In case the  $\geq$  sign in equation 7.65 is flipped to  $\leq$ , then the function of interest would be convex, with an example plotted on the left of Figure 7.15.

## 7.17 Appendix: Karush-Kuhn-Tucker Conditions

We now define the Karush-Kuhn-Tucker conditions for the optimization problem with inequality constraints discussed in Appendix 7.16. If optimal solution  $\mathbf{x}$  is in the *inactive* region where  $g(\mathbf{x}) > 0$  then the inequality constraint  $g(\mathbf{x}) \geq 0$  is denoted as inactive and  $\alpha = 0$  for the solution to the Lagrangian problem. For example, the optimal solution is at the center of the green circle in Figure 7.13a where  $g(\mathbf{x}) > 0$ . This constraint is not needed when  $\mathbf{x}$  is in the inactive region so it makes sense to set  $\alpha = 0$  because it leads to a larger Lagrangian, i.e.

$$\mathbf{x} \in \text{inactive region}; \quad [f(\mathbf{x}) - \alpha g(\mathbf{x})]_{\alpha > 0} < [f(\mathbf{x}) - \alpha g(\mathbf{x})]_{\alpha = 0}. \quad (7.66)$$

However, if the optimal solution  $\mathbf{x}$  is somewhere on the boundary defined by  $g(\mathbf{x}) = 0$  then the constraint is active and  $\alpha > 0$ . Being active implies that the stationary condition is not just the unconstrained stationary condition  $\nabla f(\mathbf{x}) = 0$  but requires  $\nabla f(\mathbf{x}) - \alpha g(\mathbf{x}) = 0$ .

Assigning  $\alpha$  to be either 0 or  $> 0$  is compactly summarized by the condition  $\alpha g(\mathbf{x}) = 0$ , where  $\alpha > 0$  for  $\mathbf{x}$  on the boundary or  $\alpha = 0$  for  $\mathbf{x}$  in an inactive region where  $g(\mathbf{x}) > 0$ . Thus, this leads to three constraints for the Lagrangian problem known as the Karush-Kuhn-Tucker (KKT) conditions:

$$\begin{aligned} g(\mathbf{x}) &\geq 0, \\ \alpha &\geq 0, \\ \alpha g(\mathbf{x}) &= 0. \end{aligned} \quad (7.67)$$

If there are  $N$  inequality constraint such that  $g(\mathbf{x})^{(n)} \geq 0$  for  $n = [1, 2, \dots, N]$  then there are  $N$  Lagrange multipliers with  $\alpha_n \geq 0$  and  $g(\mathbf{x})^{(n)}\alpha_n = 0$  for  $n = [1, 2, \dots, N]$ . See Nocedal and Wright (1999) for more details.



## Part III

# Convolutional Neural Networks



## Chapter 8

# Convolution and Correlation

The convolution operation is the basis of a convolutional neural network. It effectively replaces the densely populated matrices  $\mathbf{W}^{[n]}$  in a fully connected NN with sparsely populated matrices. This reduces the amount of memory, computation and the tendency to overfit. This chapter presents the fundamentals of the convolution operation and its related cousin correlation.

### 8.1 Convolution

A system is defined to be any process that alters an input signal, such as the discrete  $N \times 1$  vector  $\mathbf{x} = \{x_1, x_2, \dots, x_N\}$ , to produce the response signal, such as the discrete  $M \times 1$  vector  $\mathbf{y} = \{y_1, y_2, \dots, y_M\}$  (Oppenheim et al., 1996). For example, the input is the transient source signal  $\mathbf{x}$  (see the stick-o-gram in Figure 8.1a) of a hammer striking the ground and the output signal is the recorded seismogram  $\mathbf{y}$  at a nearby geophone.

The *linear* discrete system response of the input  $\mathbf{x}$  gives the output  $\mathbf{y} = \mathbf{W}\mathbf{x}$  represented by the matrix-vector product

$$y_i = \sum_{j=1}^N \bar{w}_{ij} x_j, \quad (8.1)$$

where  $\mathbf{W}$  is the  $M \times N$  *impulse-response matrix* with coefficients  $\bar{w}_{ij}$ :

$$\begin{array}{c} \overbrace{\begin{bmatrix} y_1 \\ y_2 \\ \cdot \\ \cdot \\ y_M \end{bmatrix}}^{\text{Trace}} = \begin{array}{c} \overbrace{\begin{bmatrix} \bar{w}_{11} & \bar{w}_{12} & \dots & \bar{w}_{1N} \\ \bar{w}_{21} & \bar{w}_{22} & \dots & \bar{w}_{2N} \\ \cdot & & \cdot & \dots & \cdot \\ \cdot & & \cdot & \dots & \cdot \\ \bar{w}_{M1} & \bar{w}_{M2} & \dots & \bar{w}_{MN} \end{bmatrix}}^{\text{Each column=impulse response}} \overbrace{\begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_N \end{bmatrix}}^{\text{Source}} \end{array} \quad (8.2)$$

Each column of  $\mathbf{W}$  is considered to be the impulse response of the system. For example, if the input hammer signal is an impulse that only turns on with unit amplitude at the time index  $k$  then

$x_n = \delta_{nk}$  is written in vector notation as

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{k-1} \\ x_k \\ x_{k+1} \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \quad (8.3)$$

Plugging equation 8.3 into 8.2 yields the Earth's impulse response:

$$\begin{array}{c} \begin{array}{ccccccc} & & & \text{\textit{kth impulse response}} & & & \\ \overline{w}_{11} & \overline{w}_{12} & \dots & \overbrace{\overline{w}_{1k}} & \dots & \overline{w}_{1N} & \\ \overline{w}_{21} & \overline{w}_{22} & \dots & \overline{w}_{2k} & \dots & \overline{w}_{2N} & \\ . & . & \dots & \overline{w}_{3k} & \dots & . & \\ . & . & \dots & . & \dots & . & \\ . & . & \dots & . & \dots & . & \\ . & . & \dots & . & \dots & . & \\ \overline{w}_{M1} & \overline{w}_{M2} & \dots & \overline{w}_{Mk} & \dots & \overline{w}_{MN} & \end{array} & \begin{array}{c} \text{\textit{Input impulse at } } k\Delta t \\ \overbrace{\begin{bmatrix} 0 \\ 0 \\ . \\ 1 \\ . \\ . \\ 0 \end{bmatrix}} \end{array} & = & \begin{array}{c} \text{\textit{kth impulse response}} \\ \overbrace{\begin{bmatrix} \overline{w}_{1k} \\ \overline{w}_{2k} \\ . \\ . \\ . \\ \overline{w}_{Mk} \end{bmatrix}} \end{array} & = \mathbf{y}. \end{array} \quad (8.4)$$

Therefore the  $k$ th column vector of the  $M \times N$  matrix  $\mathbf{W}$  is interpreted as the impulse response of the system for an impulsive input  $x_n = \delta_{nk}$  excited at time  $k\Delta t$  at the hammer location. The output  $\mathbf{y}$  is the seismogram recorded at the specified geophone position. Obviously, the vibrations cannot be recorded at time  $n\Delta t$  until *after* the source is excited at time  $k\Delta t$  so a *causal system* demands  $w_{nk} = 0$  for  $n < k$ .

### 8.1.1 Time or Shift Invariance

A linear system is linear time invariant (LTI) if a  $k$ -shifted input leads to the original output, except shifted by  $k$  samples (Oppenheim et al., 2006). This means that the columns (or rows) of  $\mathbf{W}$  in equation 8.4 are shifted replicas of their neighbors. The column shift is by one time sample if it is an immediate neighbor, which leads to the time invariance property

$$\overline{w}_{ij} = w_{i-j}, \quad (8.5)$$

where  $w_{i-j} = 0$  for  $i < j$  is the defining property of a *causal LTI filter*. The LTI property implies that  $\overline{w}_{ij} = \overline{w}_{i+1,j+1} = w_{i-j}$ , where the coefficients  $w_{i-j}$  of the row vector  $\mathbf{w} = \{w_0, w_1, \dots\}$  will be denoted as filter, aka kernel<sup>1</sup>, coefficients. Note, the time subscript  $j$  for  $w_{i-j}$  can take on the value

<sup>1</sup>The term kernel is used in many branches of engineering and science ([https : //en.wikipedia.org/wiki/Kernel](https://en.wikipedia.org/wiki/Kernel)), but each has a specialized meaning. We will define the kernel function  $k(\mathbf{x}, \mathbf{y})$  in the sense of an integral transform:  $f(\mathbf{y}) = \int k(\mathbf{x}, \mathbf{y}) d\mathbf{x}$ , where  $\mathbf{y}$  and  $\mathbf{x}$  can be n-tuples  $\mathbf{x} \in R^M$ ,  $\mathbf{y} \in R^N$  and  $k(\mathbf{x}, \mathbf{y})$  is the kernel function. For example, the kernel of a 2D Fourier transform is  $e^{ik_1 x_1 + ik_2 x_2}$  where  $\mathbf{k} = \{k_1, k_2\}$ . This integral can be discretized to become a matrix-vector multiplication so that the coefficients in the matrix are the kernel, i.e. filter, coefficients.

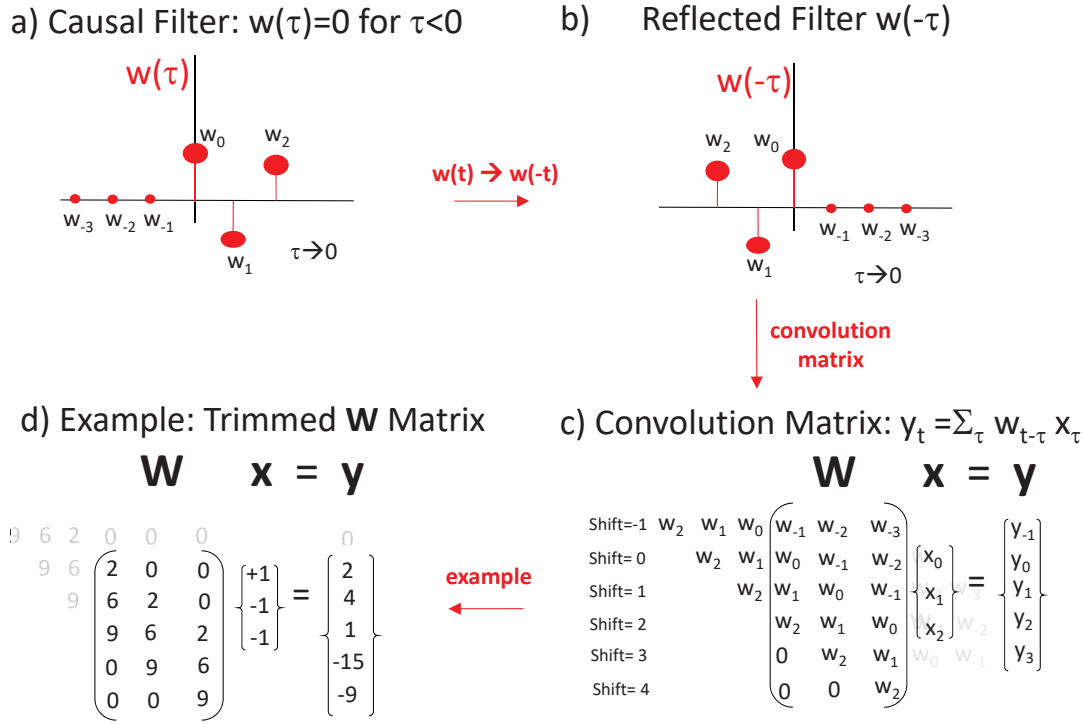


Figure 8.1: The a) causal filter  $w(\tau) = \{w_0, w_1, w_2\}$ , b) the filter  $w(-\tau)$  which is  $w(\tau)$  reflected about the origin  $\tau = 0$ , c) the convolution matrix  $\mathbf{W}$  where each output  $y_t$  is the dot product of the input vector  $\mathbf{x} = \{x_0, x_1, x_2\}$  with the row vector  $\mathbf{w}(t-\tau)$ ; here, the row vector is *shifted* by  $t$  samples. d) is an example of  $\mathbf{W}\mathbf{x} = \mathbf{y}$  for the filter  $\mathbf{w} = \{w_0, w_1, w_2\} = \{2, 6, 9\}$  and the input  $\mathbf{x} = \{x_0, x_1, x_2\} = \{1, -1, -1\}$ . The top row of zeros  $w_{-1} = w_{-2} = w_{-3} = 0$  in the matrix are eliminated to give a *trimmed*, i.e. compact, version of the matrix  $\mathbf{W}$ .

zero, which is often used as the reference time when the hammer strikes the ground.

Equation 8.5 says that the unit impulse response  $\bar{w}_{ij} = w_{i-j}$  does not depend on the absolute start time index  $j$  of the impulse or the recording time index  $i$ . It only depends on the time difference  $(i-j)\Delta t$  between the observation and excitation times, where  $\Delta t$  is the time-sampling interval.

For  $i = 0$ ,  $w_{i-j}|_{i=0} = w_{-j}$  is the original filter except  $w_j$  is reflected through the origin. As an example, Figure 8.1a plots the discrete filter  $w(\tau) = w_{\tau}$  for  $\tau \in \{-3, -2, -1, 0, 1, 2, 3\}$  and its reflected version  $w(-\tau) = w_{-\tau}$  in Figure 8.1b. If a time shift  $t$  is included then  $w(t-\tau) = w_{t-\tau}$  and the filter is shifted to the right by  $t$  samples. Otherwise it is shifted to the left for  $t < 0$ .

### 8.1.2 1D Convolution

An example of an LTI matrix  $\mathbf{W}$  for the 1D filter coefficients  $\mathbf{w} = \{w_0, w_1, w_2\}$  is obtained by substituting  $\mathbf{x} = \{0, 1, 0, 0, 0, 0, 0\}$  and equation 8.5 into equation 8.2 to get

$$\mathbf{W}\mathbf{x} = \begin{vmatrix} w_0 & 0 & 0 & 0 & 0 \\ w_1 & w_0 & 0 & 0 & 0 \\ w_2 & w_1 & w_0 & 0 & 0 \\ 0 & w_2 & w_1 & w_0 & 0 \\ 0 & 0 & w_2 & w_1 & w_0 \\ 0 & 0 & 0 & w_2 & w_1 \\ 0 & 0 & 0 & 0 & w_2 \end{vmatrix} \begin{vmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{vmatrix} = \begin{vmatrix} 0 \\ w_0 \\ w_1 \\ w_2 \\ 0 \\ 0 \\ 0 \end{vmatrix} = \mathbf{y}, \quad (8.6)$$

where  $\mathbf{w}$  is known as a 1D convolution filter (Oppenheim et al., 1996) that forms one of the building blocks of CNN. It is 1D in the sense that the filter only depends on a single variable, which is time in this example. Note, the storage requirement for the matrix  $\mathbf{W}$  is no larger than the number  $N_f$  of non-zero elements in the  $N_f \times 1$  vector  $\mathbf{w}$ . Moreover, the number of algebraic operations to compute the matrix-vector multiplication  $\mathbf{W}\mathbf{x}$  is  $O(N_f M)$ , where  $M$  is the number of rows.

In terms of a seismic experiment time invariance says that the outcome seismogram of the experiment is not determined by the day or hour it started, the outcome depends on the temporal difference between the start time of the source and the observation times in the seismogram. That is, the earth's impulse response is time invariant. For example, the seismogram amplitude observed 3 seconds after the hammer hits the ground is the same whether it hits the ground on a Monday or a Tuesday.

Substituting the LTI property of equation 8.5 into equation 8.1 gives the equation for convolution:

$$y_i = \sum_j w_{i-j} x_j \quad (\text{dot-product of reflected-shifted vector}), \quad (8.7)$$

which can be represented by the more compact notation

$$\mathbf{y} = \mathbf{w} \star \mathbf{x}, \quad (8.8)$$

where  $\star$  denotes convolution. More generally the upper and lower limits in the summation of equation 8.7 span the number of samples in  $\mathbf{x}$ .

The convolution operation for a fixed observation index  $i$  can also be described as a dot product of the shifted vector  $\mathbf{w}$  with the vector  $\mathbf{x}$ . If the units associated with the shift index  $i$  are something other than time, then the LTI system is referred to as a Linear Shift Invariant (LSI) system (Oppenheim et al., 1996). It also can be viewed as a weighted running average of the elements in the input vector, where the weighting elements are given by  $(\dots w_{-1}, w_0, w_1, \dots)$ .

An example of using 1D convolution is illustrated with the computation of the synthetic seismogram in Figure 8.2e. This seismogram represents the amplitude recorded as a function of 2-way propagation time for a wave propagating vertically in the earth and reflecting back to the surface from impedance discontinuities. In this case, the source is a point source on the surface with the time history  $w(t)$  shown in Figure 8.2d. The strengths of the upgoing reflections are governed by the reflection coefficients at each interface (see Chapter 27.1). The reflection coefficients are plotted as a function of depth in Figure 8.2a and two-way time in Figure 8.2b.

The time series  $r(t)$  in Figure 8.2b is also known as the impulse response of the medium. Convolution of this time series  $r(t)$  with the source wavelet  $w(t)$  gives the seismogram  $s(t) = r(t) \star w(t)$  in Figure 8.2e. Notice that there is a delay in the source wavelet in Figure 8.2d so the response in Figure 8.2e is also delayed by about the same amount of time. This highlights the property of a

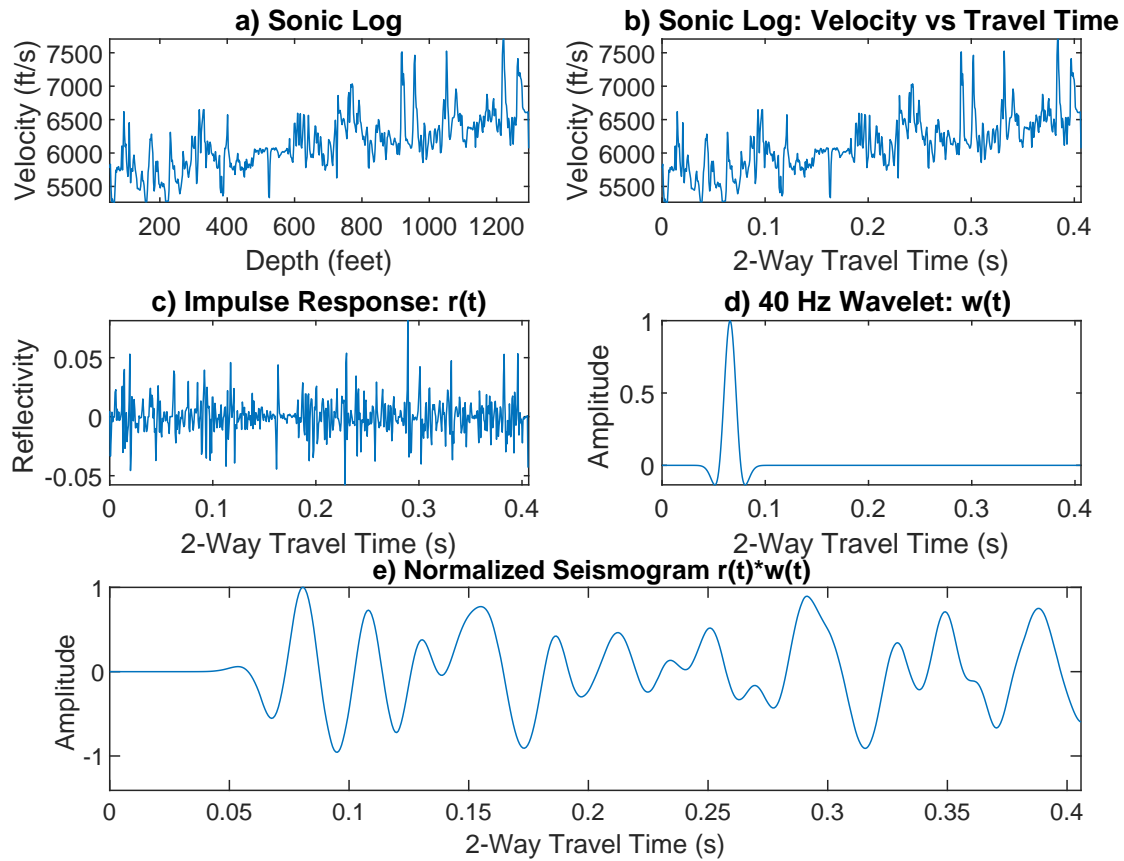


Figure 8.2: Plots of a) sonic log  $v(z)$  vs depth, b)  $v(t)$  converted from  $v(z)$ , and c) reflectivity computed from b) assuming density is constant. The 40 Hz source wavelet is in d) and the synthetic seismogram  $s(t) = r(t) \star w(t)$  is in e).

causal filter  $r(t)$ , i.e. the the seismogram  $s(t) = r(t) \star w(t)$  only records non-zero amplitudes at the same time or later after the source wavelet  $w(t)$  turns on. Mathematically, this causal property of the Earth is characterized by convolution which flips the  $r(t)$  across the time=0 origin before taking shifted dot products with  $w(t)$ . It is only when the positive time shift matches the delay time of the source will the seismogram record non-zero amplitudes. In this example the wavelet didn't start until about 0.04 seconds, which is the same start time of the seismogram. In contrast, correlation does not flip the reflectivity series so there will be non-zero dot products computed at the shift time=0 and even at negative shift times.

Distinguishing the difference between correlation and convolution is extremely important in geophysics. However, such a distinction is perhaps not so not important in machine learning. Small spatial shifts in correlated patterns compared to convolved patterns might not be so important for machine learning applications.

### 8.1.3 2D Convolution

For two-dimensional convolution, the input is a 2D array of, for example, intensity values of a grayscale photograph. The intensities on a  $\sqrt{N} \times \sqrt{N}$  grid are mapped, i.e. flattened, to a  $N \times 1$  vector  $\mathbf{x}$ . This is illustrated by the image in the blue box of Figure 8.3c being flattened into the  $\mathbf{x}$  vector in d).

The 2D convolutional filter  $\mathbf{w}$  that acts on  $\mathbf{x}$  is an array of weights  $\omega_i$  for  $i \in \{0, 1, 2, \dots, N_f - 1\}$  that occupy a 2D block in the image, such as the one in Figure 8.3a where  $N_f = 9$ . The equation for 2D convolution is

$$\bar{y}_{r,c} = \sum_{r'} \sum_{c'} w_{r-r', c-c'} \bar{x}_{r', c'}, \quad (8.9)$$

where the filter weights  $\omega_i$  depicted in Figure 8.3a are given by

$$\begin{aligned} w_{-1,-1} &= \omega_8; & w_{-1,0} &= \omega_7; & w_{-1,1} &= \omega_6; \\ w_{0,-1} &= \omega_5; & w_{0,0} &= \omega_4; & w_{0,1} &= \omega_3; \\ w_{1,-1} &= \omega_2; & w_{1,0} &= \omega_1; & w_{1,1} &= \omega_0. \end{aligned} \quad (8.10)$$

Each weight at a red point in Figure 8.3a is reflected through the origin  $\{r, c\} = \{0, 0\}$  to give the reflected filter  $w_{r-r', c-c'}|_{r=0, c=0}$  in Figure 8.3b for  $r = c = 0$ .

The blue box in Figure 8.3c contains the intensity coefficients of the 2D input image  $\mathbf{x} = \{x_0, x_1, \dots, x_{N-1}\}$ , and the reflected filter coefficients  $w_{-r, -c}$  are in the brown box. For  $w_{r-r', c-c'}$ , the  $r$  and  $c$  indices correspond to the row and column samples by which the filter  $w_{-r', -c'}$  is shifted. For example, at  $r = c = 0$  the filter  $w_{-r, -c}$  is unshifted and centered at the origin (red-filled circle) in the brown boxes of Figure 8.3b-8.3c.

The filled-red circle in the blue box is at  $\{r, c\} = \{1, 1\}$ , where the origin of the blue-box's coordinate system is in the center of the uppermost left pixel at the  $\{r, c\} = \{0, 0\}$  gridpoint. The output  $\bar{y}_{r,c}|_{r=1, c=1}$  in equation 8.9 is obtained by centering the  $3 \times 3$  filter  $w_{1-r', 1-c'}$  onto the red-filled circle at  $\{r, c\} = \{1, 1\}$ , and summing the products  $\sum_{r'} \sum_{c'} w_{1-r', 1-c'} \bar{x}_{r', c'}$  to get  $\bar{y}_{1,1}$ . The variable  $\bar{y}_{1,1}$  is redefined to be  $y_0$  and placed in the output yellow box on the right. This process of computing the dot products of the input with shifted filters  $w_{r-r', c-c'}$  can be repeated to get the values of  $\bar{y}_{1,2} = y_1$ ,  $\bar{y}_{2,1} = y_2$ , and  $\bar{y}_{2,2} = y_3$ . Here, the shifted filters are centered, respectively, at the blue ( $\{r, c\} = \{1, 2\}$ ), green ( $\{r, c\} = \{2, 1\}$ ) and purple ( $\{r, c\} = \{2, 2\}$ ) circles. The convolution filter in c) can be flattened and placed as shifted row vectors to form the convolution matrix in Figure 8.3d. This equation represents discrete 2D convolution of the input  $\mathbf{x}$  with the LSI filters in  $\mathbf{W}$  to get the output  $\mathbf{y}$ .

The  $4 \times 1$  dimension of the output vector  $\mathbf{y}$  in Figure 8.3d is smaller than that of the  $16 \times 1$



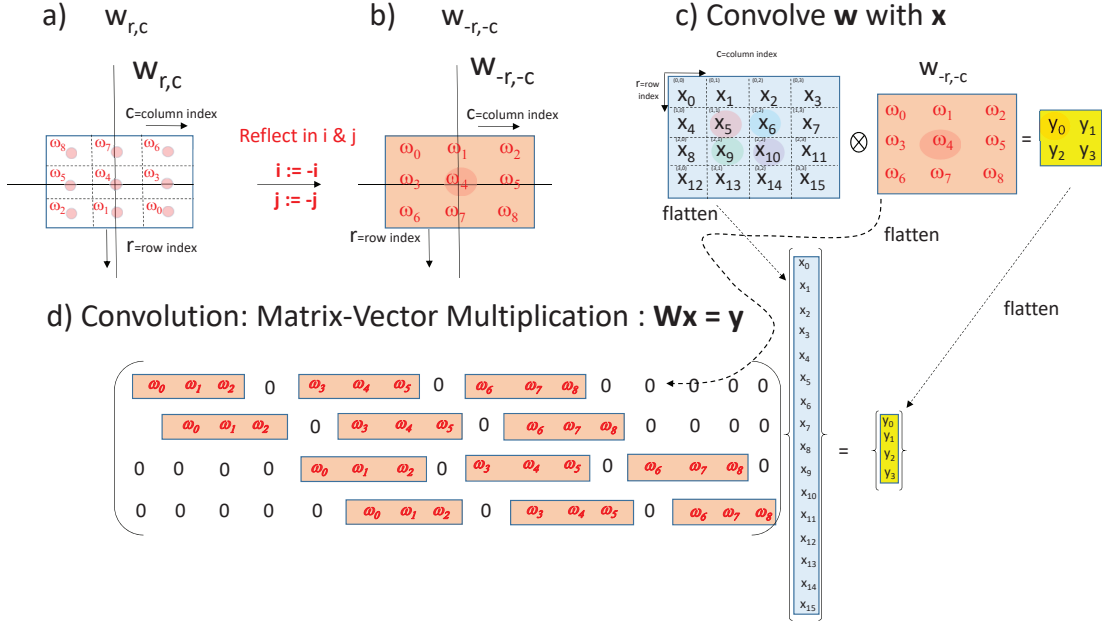


Figure 8.3: Plots of the filter a)  $w_{r,c}$  and its b) reflection  $w_{-r,-c}$  through the origin. The c) input image  $x$  is in the blue box and the reflected filter coefficients are plotted in the brown boxes. The corresponding convolution matrix is in d).

input vector. This can be inconvenient for coding multi-layer CNNs, so zero-padding is applied to the input  $x$  as illustrated in Figure 8.4. In this example, pixels with an intensity value of zero are padded to the outside of the blue box in Figure 8.3c to give the zero-padded input  $x$  in Figure 8.4. Centering the  $3 \times 3$  filter  $w$  at each of the filled-red circles in the blue box and computing its inner product with  $x$  gives the element values of  $y = \{y_0, y_1, y_2, \dots, y_{15}\}$  on the right. Now the dimension of  $y$  is the same as for  $x$ .

### 8.1.4 3D Convolution

The input can sometimes be a 3D image, such as the red, green and blue images in Figure 8.5. In this case the blocky  $3 \times 3 \times 3$  convolution filter is a 3D rather than a 2D filter. For the Figure 8.5 example the filter dimensions are  $3 \times 3 \times 3$  and the output is a  $4 \times 4$  image on the right. The filter is only centered at points in the input image  $x$  so that the filter stencil stays within the  $6 \times 6$  grid of the input image.

## 8.2 Correlation and Matched Filters

If

$$\overline{w}_{r+r',c+c'} = w_{r-r',c-c'}, \quad (8.11)$$

## Zero-Padded Convolution

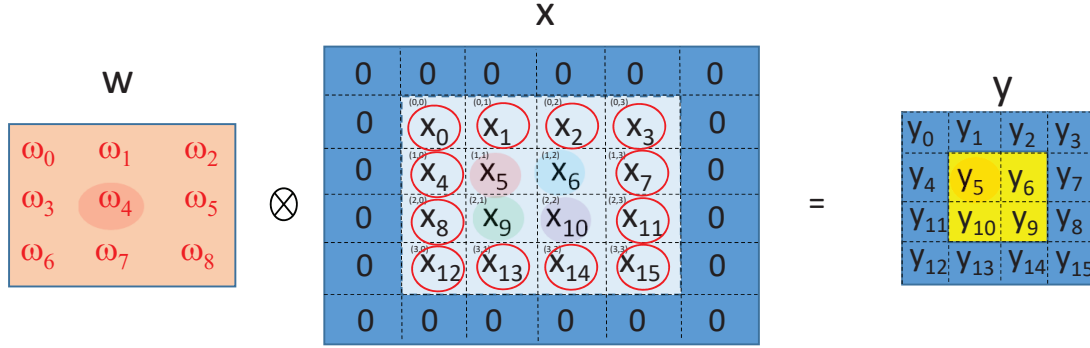


Figure 8.4: Zero-padded convolution  $\mathbf{y} = \mathbf{w} \star \mathbf{x}^{pad}$  pads zeros (dark blue rectangles in middle image) to the boundary of the input image  $\mathbf{x}$  so that the output dimension of  $\mathbf{y}$  is the same as the input dimension of the unpadded  $\mathbf{x}$  vector.

is plugged into equation 8.9 we get the cross-correlation between the input  $\mathbf{x}$  and the cross-correlation filter  $\overline{\mathbf{w}}$ :

$$\overline{y}_{r,c} = \sum_{\{r',c'\} \in B_{\{r,c\}}} \overline{w}_{r+r',c+c'} \overline{x}_{r',c'}, \quad (8.12)$$

where  $B_{\{r,c\}}$  are the set of coordinates that the filter coincides with for a specified shift pair of shift coordinate  $\{r, c\}$ . More compactly this correlation equation is denoted by

$$\mathbf{y} = \overline{\mathbf{x}} \otimes \mathbf{w}, \quad (8.13)$$

where  $\otimes$  is the symbol for cross-correlation. In fact, setting  $r = c = 0$  shows that

$$\overline{w}_{r',c'} = w_{-r',-c'}, \quad (8.14)$$

which says that the correlation wavelet  $\overline{w}_{r',c'}$  is a *flipped*, aka reflected, version of the convolution filter  $w_{r',c'}$ . For example, Figure 8.1b is the 1D correlation wavelet and Figure 8.1a is the 1D convolution wavelet. One is the flipped version of the other. Another example, Figure 8.3b is the correlation filter and Figure 8.3a is the convolution filter. Many papers in machine learning mislabel the correlation filter as the convolution filter. The convolution filter is only the same as the correlation filter if the filter coefficients are symmetrically distributed about the origin, which is generally not the case for most applications.

The correlation filter  $\overline{\mathbf{w}}$  can also be considered as a *matched* filter ([https://en.wikipedia.org/wiki/Matched\\_filter](https://en.wikipedia.org/wiki/Matched_filter)). Assume the image in Figure 8.6c

$$i(x, y) = w(x, y)_f + w(x, y)_s + 0.9n(x, y), \quad (8.15)$$

is obtained by adding the *fat*  $w(x, y)_f$  and *skinny*  $w(x, y)_s$  signals in Figure 8.6b to the weighted random noise  $0.9n(x, y)$  uniformly distributed between  $-0.5$  and  $0.5$ . Here,  $w(x, y)_s$  is 15 times skinnier than  $w(x, y)_f$ . The 2D Fourier transform  $\mathcal{F}[\cdot]$  applied to  $\overline{w}(x, y) \otimes \{s(x, y) + n(x, y)\}$  gives

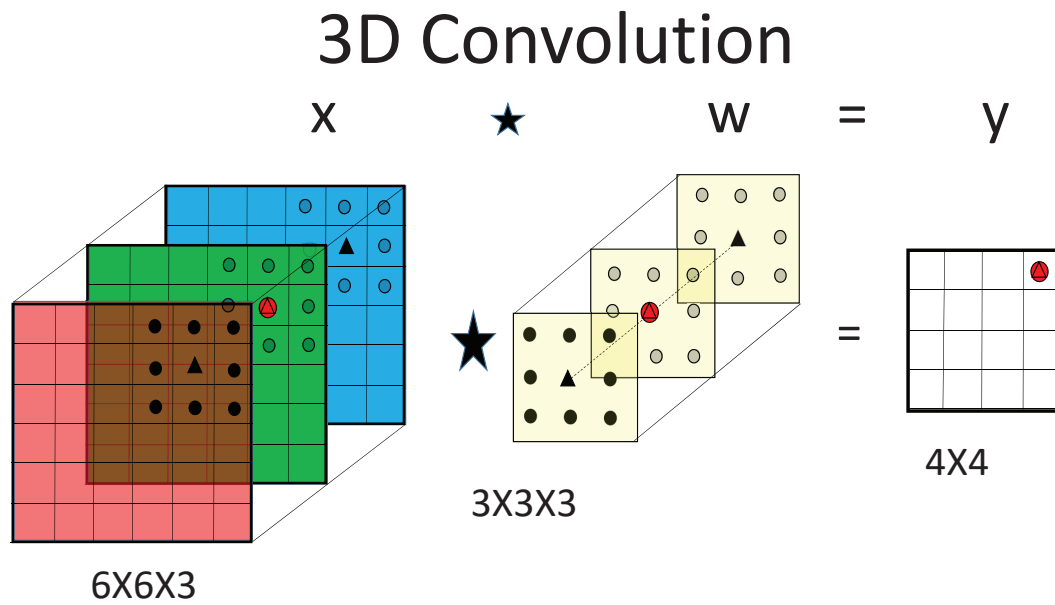


Figure 8.5: An example of 3D convolution filter is the  $3 \times 3 \times 3$  filter  $\mathbf{w}$  convolved with the  $6 \times 6 \times 3$  input  $\mathbf{x}$  to give the  $4 \times 4 \times 1$  output  $\mathbf{y}$ . The yellow  $3 \times 3 \times 3$  filter  $\mathbf{w}$  is centered at the red triangle in the  $6 \times 6 \times 3$  input  $\mathbf{x}$  on the left. A dot product between the filter and input weights  $\mathbf{x}$  is computed, and the result is placed in the red triangle of  $\mathbf{y}$  on the right. The correlation filter  $\mathbf{w}$  is centered at another interior gridpoint of  $\mathbf{x}$ , the dot product is computed and the result is placed in the appropriate gridpoint of  $\mathbf{y}$ . This process is repeated for all interior gridpoints in  $\mathbf{x}$  to fill up the  $4 \times 4$  grid in  $\mathbf{y}$ . If the input  $\mathbf{x}$  has more than three panels then the output  $\mathbf{y}$  will be a 3D block of numbers. Most convolutional filters in ML are 3D but will only output a 2D panel of numbers because the dimension of the 3D filter in the out-of-the-page axis is the same as for the input block  $\mathbf{x}$ .

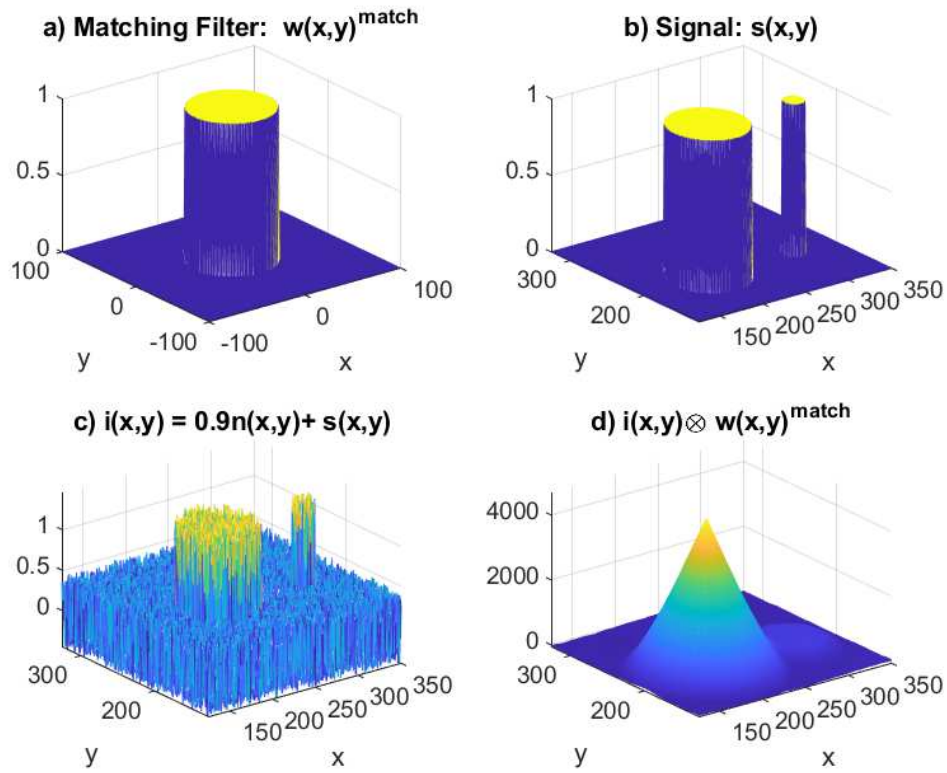


Figure 8.6: a) Fat matched filter  $w^{match}(x,y) = w(x,y)_f$ , b) fat and skinny coherent signals  $s(x,y) = w(x,y)_f + w(x,y)_s$ , c) noise+signal to give the image  $i(x,y) = 0.9n(x,y) + s(x,y)$ , and d) the result after correlating  $w^{match}(x,y) = w(x,y)_f$  with  $i(x,y)$ . In this example, the skinny signal has all but disappeared in d).

$$\begin{aligned}
\mathcal{F}[\bar{w}(x, y) \otimes \{s(x, y) + n(x, y)\}] &= \overbrace{W(k_x, k_y)_f}^{W(k_x, k_y)_f^*} [ \underbrace{W(k_x, k_y)_f + W(k_x, k_y)_s}_{S(k_x, k_y)} + N(k_x, k_y) ] \\
&= |W(k_x, k_y)_f|^2 + \underbrace{W(k_x, k_y)_f^* W(k_x, k_y)_s}_{\text{attenuated}} + \overbrace{W(k_x, k_y)_f^* N(k_x, k_y)}^{\approx 0}, \\
&\approx |W(k_x, k_y)_f|^2 + \underbrace{W(k_x, k_y)_f^* W(k_x, k_y)_s}_{\text{attenuated}}, \tag{8.16}
\end{aligned}$$

where  $\mathcal{F}[a(k) \otimes b(k)] = A(k)^* B(k)$  and uppercase letters denote the spectrum of the function in the  $k_x - k_y$  domain (see Exercise 8.7). Correlating the *fat* matched filter  $\bar{w}(x, y) = w(x, y)_f$  with  $i(x, y)$  gives the denoised result in Figure 8.6d. Here, the noise has been suppressed, but so has the correlation  $\mathbf{w}_f \otimes \mathbf{w}_s$ . This is because the low-wavenumber spectrum  $W(k_x, k_y)_f$  of the fat signal is only alive around the origin, while that of the skinny spectrum  $W(k_x, k_y)_s$  is alive over a much wider band. Multiplying these two spectra together results in a low-wavenumber spectrum and significantly attenuates the amplitude of  $\bar{w}(x, y) \otimes w(x, y)_s$  in Figure 8.6d.

In contrast, if the wideband matched filter  $\bar{\mathbf{w}} = \mathbf{w}_s$  is correlated with  $i(x, y)$  it results in the image shown in Figure 8.7d. Here, the relative amplitudes of both the fat  $\mathbf{w}_f$  and skinny  $\mathbf{w}_s$  signals are preserved because the correlation filter  $\bar{\mathbf{w}} = \mathbf{w}_s$  is wideband. A significant observation is that the *small correlation filter will not only pass both high and low wavenumbers of the input, but it is invariant to the spatial locations of the cylindrical patterns*. No matter where the patterns are located, the filter will output the response at these locations.

The spectra of  $W(k_x, k_y)_f N(k_x, k_y)^*$  is nearly zero in equation 8.16, because the correlation value of  $n(x, y) \otimes w(x, y)_f$ , for a specified shift value, is

$$\begin{aligned}
n(x, y) \otimes w(x, y)_f &= \sum_{\{x', y'\} \in B} n(x', y') w(x + x', y + y')_f, \\
&= \sum_{\{x', y'\} \in B_{\{x, y\}}} n(x', y'), \tag{8.17}
\end{aligned}$$

which is the sum of the random noise series in a round window defined by the points in  $B_{\{x, y\}}$  for each specified pair of shift coordinates  $B_{\{x, y\}}$ . This dot product incoherently adds out-of-phase random numbers between  $-0.5$  and  $0.5$ . Compared to a coherent signal added over the same number of points, the signal-to-noise ratio grows as  $\sqrt{N}$  where  $N$  is the number of points in the circle.

### 8.2.1 Multilayer and Multifilter Convolution

The FCNN in Chapter 6 employed many layers in order to accurately approximate the relationship between the input data and the target data. The same must be done with the convolutional neural network except it needs to increase the number of different filters in each layer. Only one set of shifted filters per layer does not have the capability of detecting widely varying patterns in the same image. Therefore, the convolutional neural network employs many filters per layer as illustrated in Figure 8.8. Each  $3 \times 3$  filter  $\mathbf{w}_k$  for  $k = \{1, 2, 3, 4\}$  is associated with a different dipping pattern. The output pre-feature map  $\mathbf{y}_k$  is the correlation of the  $\mathbf{w}_k$  filter with  $\mathbf{x}$ , which gives the location and correlation strengths of the patterns with similar dip patterns.

An example of a  $3 \times 3$  filter that extracts horizontal dips from the image is given by

$$\mathbf{w}_{horiz.} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{bmatrix}, \tag{8.18}$$

which is a finite-difference approximation to the derivative  $d/dy$ . A horizontal line in an image

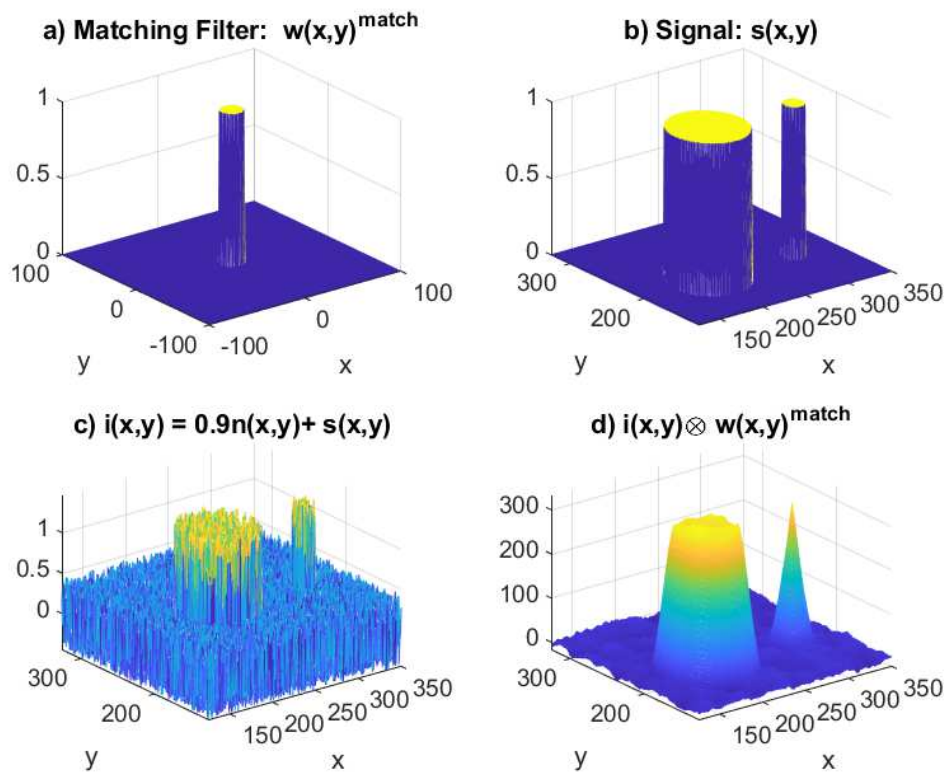


Figure 8.7: Same as Figure 8.6 except the matched filter is the skinny signal  $w(x,y)_s$ .

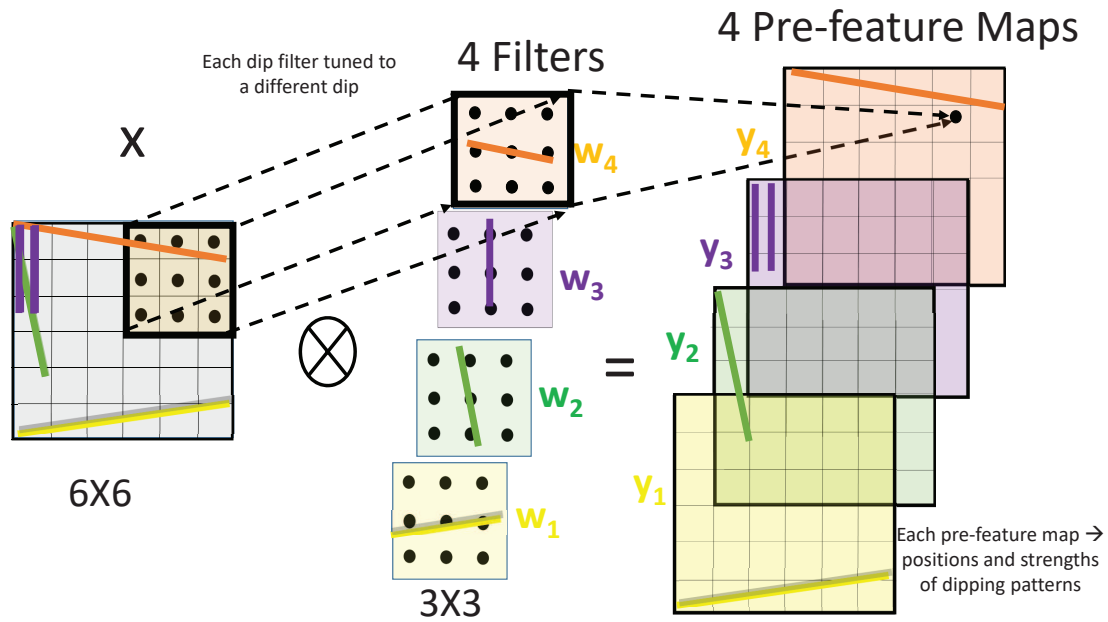


Figure 8.8: Multiple filters  $w_k$   $k \in \{1, 2, 3, 4\}$  in the same layer. Each one is tuned to a different dipping pattern which can capture, i.e. correlate, with a different feature, i.e. pattern, with the same dip in the input image. The output of the correlation of  $w_k$  with  $x$  gives the pre-feature map  $y_k$ . We denote  $y_k$  as a pre-feature map because the threshold function has not yet been applied to suppress noisy cross-talk.

represents a strong change in intensity along the vertical axis. Correlating the above filter with an image with horizontal lines will produce strong horizontal patterns in the pre-feature map. This is illustrated by the gently dipping yellow line in the yellow pre-feature map of Figure 8.8.

In general, equation 8.18 is a special case of a dip filter that is strongly activated by strong vertical variations in intensity. For example, a fat horizontal bar where the intensity values of the pixels are strongly discontinuous across the horizontal edges of the bar. This filter can be tuned to any dip angle by using a finite-difference approximation to a directional derivative.

Unfortunately,  $\mathbf{w}_{horiz.}$  will moderately correlate with lines that might have a gentle to moderate dip, which is a noisy distraction that might result in a moderately dipping line as a horizontal line. Therefore a threshold filter, such as a sigmoid function, needs to be applied to each of the elements in  $\mathbf{y}_{horiz.}$  to window out such noise. Thus, applying a threshold filter  $\sigma(\cdot)$  to a multitude of pre-feature maps  $\mathbf{y}_k$  gives, ideally, the output  $\mathbf{a}_k$  of features with only a small variation of dip angles around  $0^\circ$ . This output  $\sigma(\mathbf{y}_k)$  image is known as a feature map and forms the basis of the convolutional neural network architecture. The next chapter explains the CNN architecture and workflow in much greater detail.

### 8.3 Summary

The convolution operation is defined for 1D, 2D and 3D filters. They all can be recast as matrix-vector products where the input image has been flattened into the vector  $\mathbf{x}$ . Any row is a shifted replica of a neighboring row for a regular numbering scheme. Therefore, convolution can be described as a series of shifted dot products between the reflected filter and the input data. The convolution matrix is sparsely populated and so reduces the amount of memory, computation and the tendency to overfit compared to a FCNN. It's maximum correlation value is also invariant to the location of the correlated pattern in the input image. For the FCNN, each different location of the cylindrical object might demand a completely new row of coefficients in the  $\mathbf{W}$  matrix. This is not the case with the convolutional matrix where each row is a shifted copy of the filter that is tuned to a different location in the image. If it is small enough, one set of coefficients can strongly correlate with objects with different scales and locations.

The convolution kernel  $w_{r,c}$  is a flipped version of the correlation kernel  $\bar{w}_{-r,-c}$ . Therefore, we can also think of convolution as applying the matching filter  $\bar{w}_{-r,-c}$  to the input  $\mathbf{x}$ . If the patterns in  $\mathbf{x}$  are similar to those in  $\bar{w}_{-r,-c}$  then there will be a high value of the correlation value. If the pattern is different then there will be a low correlation value. If patterns are to be detected by a neural network it is better that the first layer has a wide-band filter in order to allow for a wideband response of the neural network.

### 8.4 Exercises

1. Your first convolutions.

- Use MATLAB's *stem* subroutine to plot  $\mathbf{x} = [x(0) \ x(1) \ x(2)] = [1 \ .5 \ .25]$ , where  $x(n) = x_n$  and  $n$  is the time sample index. Now plot  $x(-n+3)$ ,  $x(n-3)$ ,  $x(-n-3)$  and  $x(n+3)$  w/r  $n$ . Which is the delayed and which is the advanced version of  $x(n)$ ? Make sure you plot against the correct time values, including negative time if necessary.
- Plot  $x(-n)$  and explain why we say  $x(-n)$  is a reflected version of  $x(n)$ . Explain why we say that  $x(n-3)$  is a reflected-shifted version of  $x(n)$ . Is it shifted to the right or to the left of the origin?

2. Convolve  $[h(-1) \ h(0)] = [-1 \ 1]$  with  $\mathbf{x}$  (i.e.,  $y = h \star x$ ). Use MATLAB's *conv* subroutine, and make sure you properly label the time axis.



3. Now convolve  $\mathbf{x}$  with  $[h(-1) \ h(0)]$  (i.e.,  $y' = x \star h$ ), where the elements of  $\mathbf{x}$  comprise the matrix and  $\mathbf{h}$  is the vector in the matrix-vector multiply. Note that  $y^T = y$ , which is true for any LTI system; the property of  $\mathbf{x} \star \mathbf{h} = \mathbf{h} \star \mathbf{x}$  is the commutative property of any LTI systems. Note that, in general, matrix-matrix multiplication is not commutative.
4. How long is the output vector  $\mathbf{y}'$ ? What is the length of the output of convolving an  $N \times 1$  vector  $\mathbf{x}$  with the  $M \times 1$  vector  $\mathbf{h}$ ? Explain.

## COMMUTATIVE PROPERTY OF CONVOLUTION

$$\begin{array}{c}
 \text{-----} \\
 \text{I} \quad \text{I} \\
 \text{x} \text{ -----} \rightarrow \begin{array}{|c|c|} \hline \text{I} & \text{H} \\ \hline \text{I} & \text{I} \\ \hline \end{array} \text{I} \text{ -----} \rightarrow \text{y} \quad \text{or} \quad \text{h} \text{ -----} \rightarrow \begin{array}{|c|c|} \hline \text{I} & \text{X} \\ \hline \text{I} & \text{I} \\ \hline \end{array} \text{I} \text{ -----} \rightarrow \text{y} \\
 \text{-----}
 \end{array}$$

5. In general, multiplying an  $M \times 1$  vector  $\mathbf{x}$  by a dense  $M \times M$  matrix costs  $O(M^2)$  algebraic operations. Explain why it costs  $O(hM)$  to convolve the  $M \times 1$  vector  $\mathbf{x}$  with a short  $h \times 1$  vector  $\mathbf{h}$ . Here,  $h \ll M$ .
6. Convolve  $\mathbf{h} = [h(-10) \ h(-9)] = [-1 \ 1]$  with  $\mathbf{x} = [\mathbf{x}(0)] = [1]$  (i.e.,  $\mathbf{y} = \mathbf{h} \star \mathbf{x}$ ). Display the results. Is  $\mathbf{h}$  a causal or acausal filter.
7. For two continuous functions  $a(x)$  and  $b(x)$ , show that  $\mathcal{F}[a(x) \otimes b(x)] = A(k)^* B(k)$ , where  $\mathcal{F}[a(x)] = \int_{-\infty}^{\infty} a(x) e^{ikx} dx$ .
8. Assume that  $f(t)$  and  $g(t)$  are evenly sampled functions with the sampling index given by the integer  $t$  for  $t \in \{1, 2, 3, \dots\}$ . Show by substitution of variables ( $t' = t + \tau$ ) that

$$\begin{aligned}
 (f \otimes g)(\tau) &= \sum_{t=-\infty}^{\infty} f(t)g(t+\tau), \\
 &= \sum_{t'=-\infty}^{\infty} f(t'-\tau)g(t').
 \end{aligned} \tag{8.19}$$

Notice that, unlike convolution, the shift variable  $\tau$  has a minus in front of it in the argument of  $f(t' - \tau)$ . T



## Chapter 9

# Convolutional Neural Networks

The theory and practice of convolutional neural networks (CNNs) are introduced where the dense matrices of a fully-connected neural network (FCNN) are replaced by the sparse correlation matrices. In a convolutional neural network, the densely-populated rows of the  $O(M \times N)$  FCNN matrix  $\mathbf{W}^{[n]}$  are replaced by sparsely-populated rows of  $N_f$  non-zero elements. Most often,  $N_f$  is no more than a few dozen and  $N_f \ll N$ . Moreover, each row of non-zero elements is, roughly, a shifted replica of the previous row of elements, except at the boundaries of the image. Thus, the CNN matrix represents a spatially invariant convolution filter with  $O(N_f)$  memory<sup>1</sup> and matrix-vector computation requirements compared to  $O(MN)$  for each layer of a FCNN. Hence, the CNN is typically much less expensive for storage and computation of the matrix-vector multiplications. Moreover, the local pattern-matching nature of convolutional filters is often more effective at detecting localized patterns in the data than a FCNN. For these reasons and the successes of CNN since the early 2010s, CNN and its variants are arguably the most popular ML algorithms in use today.

### 9.1 Introduction

Chapter 8 discussed the motivation for replacing the matrices of a fully connected neural network by those of convolution (or matching filter) matrices. Figure 9.1 illustrates the key differences between the matrix-vector operation of a FCNN and a CNN, which result in CNN's three most important advantages.

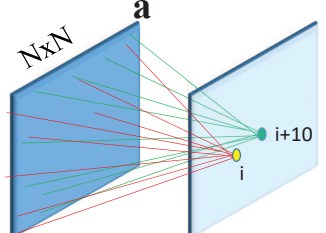
1. **Reduced Computational and Storage Costs=Convolution.** The CNN only requires  $O(N_f)$  memory locations for storing the  $N_f$  non-zero filter coefficients in any row vector of  $\mathbf{W}$ . In addition, only  $O(MN_f)$  algebraic operations are needed for the matrix-vector multiplication  $\mathbf{W}\mathbf{x}$ . This compares to  $O(MN)$  storage sites and computational operations for a FCNN layer with a densely-populated matrix  $\mathbf{W}$ .
2. **Local Pattern Matching=Small CNN Filters.** The number  $N_f$  of the CNN filter coefficients is much smaller than the number  $O(N)$  of non-zero coefficients in any row vector of  $\mathbf{W}$ , so the CNN filter captures localized patterns in the input. Such small patterns might be all that is needed to be resilient to different scales, rotations or translations of the same object<sup>2</sup>. In addition, the shift-invariance property of the matching filter provides location

---

<sup>1</sup>Irregular numbering of pixels will require the locations of the filter coefficients for each row. In this case, the locations must be stored for each row so that the memory requirement is  $O(MN_f)$ .

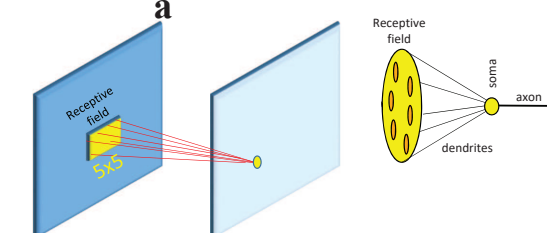
<sup>2</sup>The CNN filter in the first layer should be fairly small, being no larger than a few dominant wavelengths associated with the input image. To estimate the smallest significant wavelength in the input image, apply a 2D Fourier transform (Yilmaz, 2001) to the input picture  $d(x, z)$  to get  $D(k_x, k_z)$ , where the wavenumber

## Fully Connected Neural Network

$$\sum_{j=1}^M w_{ij}^{[n]} a_j^{[n-1]} = z_i$$


$\mathbf{W}\mathbf{a} \rightarrow O(N^4)$

## Convolutional Neural Network

$$z_i^{[n]} = \sum_{j \in B} \hat{w}_{i-j}^{[n]} a_j^{[n-1]} = z_i$$


$\mathbf{W}\mathbf{a} \rightarrow O(N^2)$  Faster Less memory

Figure 9.1: Matrix-vector multiplication for a (left) FCNN where the output value of  $z_i$  at the yellow circle is computed by summing together every weighted input value in  $\mathbf{a}$ . In comparison, the CNN on the right computes the values of  $z_i$  at the yellow circle from a few input components of  $\mathbf{a}$  in the yellow  $5 \times 5$  rectangle, also known as the *receptive field*. Furthermore, the weights in the correlation filter are the same for all shift values of  $i$ .

resilience with a CNN that is lacking in the FCNN. As will be discussed later, a large number of convolution filters can be incorporated into the first layer so that a large number of local features in an image can be detected.

3. **Regional Pattern Matching=Pooling+Deeper Layers.** Patterns much larger than the size of the filter can be detected by including more layers in the CNN and downsampling the output of these layers. This downsampling operation, known as pooling in section 9.2.1, typically halves the number of pixels along any one side of the picture, which automatically doubles the gridpoint spacing  $\Delta x$ . This means that the, e.g. 2 centimeter, gridpoint interval in the small convolutional filter is doubled to 4 centimeters to capture larger, i.e. regional, features. Multilayer downsampling can also provide scaling resilience when the object size varies from input picture to picture.

The combination of small matching filters, activation functions, pooling, and multilayer CNNs defines basic operations of the convolutional neural network (CNN). Another important aspect of a CNN is regularization, which will be discussed in section 9.2.6.

### 9.1.1 Brief History of CNN

CNN partly owes its inspiration to the biology experiments in the 1950-1960s where Hubel and Weisel (1959, 1962) discovered that a cat's visual cortex greatly responded to different orientations of a localized object. Figure 9.2 depicts the recorded responses of a cat's visual neurons, which are most responsive to horizontal or oblique orientations of a bar. This suggests that a cluster of neurons in the cat's brain is significantly responding to certain orientations of a localized object.

---

components are  $\mathbf{k} = (k_x, k_z)$  space. The large values of  $|D(k_x, k_z)|$  associated with the largest wavenumbers  $k^{max} = \sqrt{k_x^2 + k_z^2} = 2\pi/\lambda^{min}$  usually define the most important small-scale features  $\lambda^{min}$  of the object of interest.

From a digital processing point of view, this response is similar to the actions of a *localized* 2D dip filter (see Figure 8.8) that only lights up for certain dips of an object.

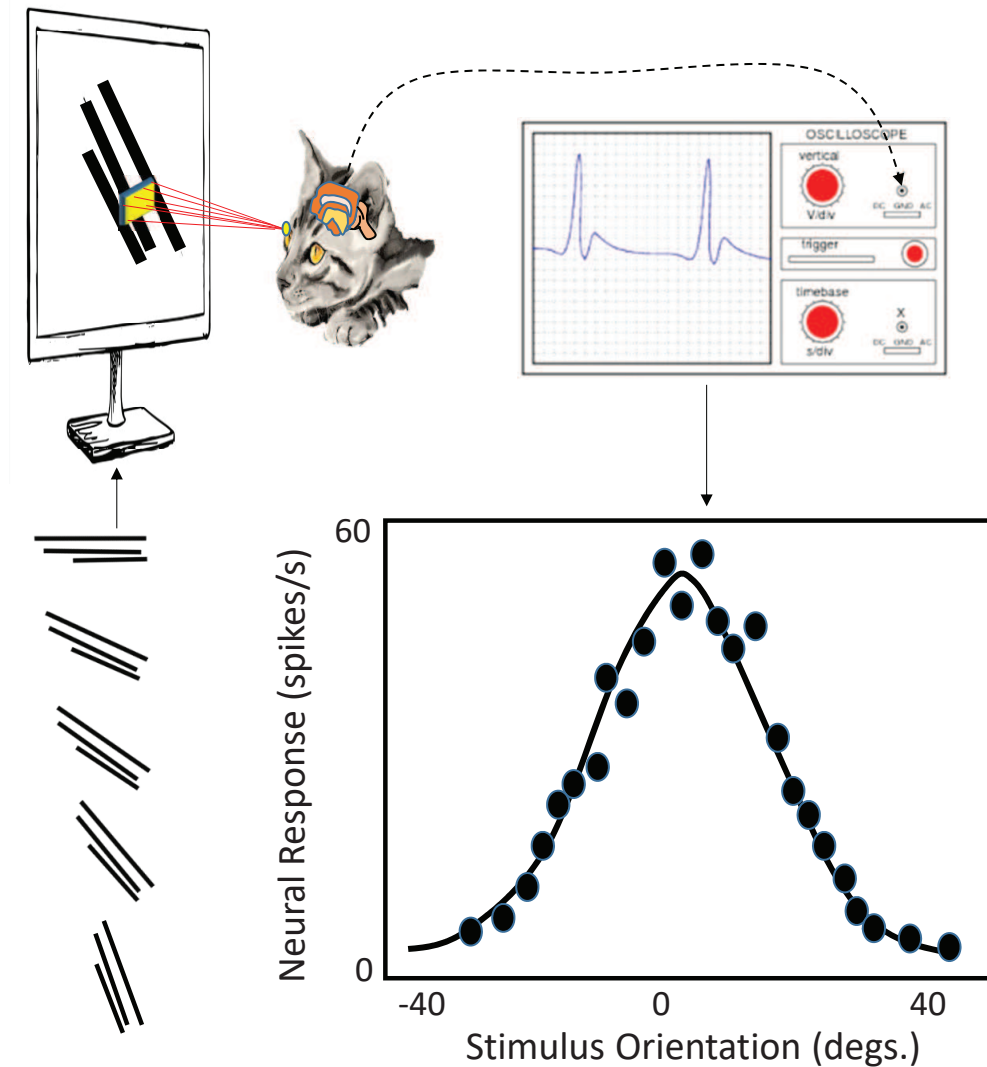


Figure 9.2: (Top) Cat and (bottom) plot of the cat's neural visual response for different orientations of the black bar. Figure roughly adapted from Hubel and Weisel (1959), who discovered that different neurons fired rapidly when lines were oriented at different angles. Other neurons detected edges no matter where they were located, just like a dip filter. Their work on establishing a foundation for visual neurophysiology and ocular columns was most deservedly rewarded by a Nobel Prize in Medicine in 1981.

In the 1980s, Fukushima (1980) proposed a spatially invariant operation, the summation of a localized set of input values with neural weights, to explain the neural mechanisms of a cat's visual pattern recognition. This is a classical neural network model except the number of correlation weights is much less than the number of input nodes and only scanned a small *receptive* field of the

input image. Similar to the horizontal matching filter in equation 8.18, the inner product  $(\mathbf{w}_k, \mathbf{x})$  is the correlation of a small matching filter with the input image. The size of the scanned field in the input image is known as the receptive field, which is also the size of the matching filter in a CNN. Fukushima (1980) pointed out that the response of the filter is invariant to a shift in the location of the pattern. It is also invariant to small changes in the shape or the size of the pattern.

Later, LeCun et al. (1989) presented one of the first convolutional networks trained by a gradient method. In their application, the goal was to recognize hand-written numbers for zip code identification on letters. Later, LeCun et al. (1998) developed the LeNet-5 CNN architecture illustrated in Figure 9.3 where the input  $32 \times 32$  image is a hand-written letter. The goal is to identify its identity in the alphabet.

1. In the first layer, six  $5 \times 5$  filters are applied to the input image to generate the six feature maps (FMs). Each FM has a dimension of  $28 \times 28$ . No zero-padding is used so the  $5 \times 5$  filters are centered no closer than two gridpoints from the edge of the image. This means that there are only  $28 \cdot 28 = 784$  gridpoints which allow centering of the  $5 \times 5$  filter so that no part of the filter stencil falls off the image.

In the context of the Figure 9.1 illustration we have the output of the matrix-vector multiplication given by

$$z_i^{[n]} = \sum_{j \in B_i} w_{i-j}^{[n]} a_j^{[n-1]}. \quad (9.1)$$

Here,  $w_j$  for  $j \in [1, 2, 3 \dots N_f]$ ,  $N_f \ll N$ , and  $B_i$  is a small set of integers that characterize the non-zero weights  $w_i$  for the shift index  $i$ .

2. After convolution, the activation function is applied to  $\mathbf{z}$  to get the components of the activation vector  $\mathbf{a}^{[n]}$ :

$$a_i^{[n]} = g(z_i^{[n]}), \quad (9.2)$$

where  $g(\cdot)$  is the activation function, which can be a ReLU, sigmoid or one of the many discussed in section 4.2.

3. The data are subsampled from the first layer to the second layer in Figure 9.3 by a factor of two, which can be accomplished by a max pooling operation. However, a common means for doing this is by using a correlation filter that is shifted by every two samples rather than one. This is known as a filter with stride 2. The combination of filtering input values, application of the activation function and subsampling the FMs comprises the operations of one filter for one layer.
4. Instead of using six 2D matching filters, sixteen 3D filters with the dimension of  $5 \times 5 \times 6$  are used for the second layer<sup>3</sup> at  $S2$  in Figure 9.3. This produces 16 FMs at  $C3$ .
5. The  $C5$  layer uses a  $120 \times 400$  matrix  $\mathbf{W}$  to produce the  $120 \times 1$  output vector  $\mathbf{z}$  in Figure 9.3.
6. The last layer used a softmax-like function to produce probabilities for the 10 possible class types.

The immediate application of the LeNet-5 network was to recognize hand-written numbers written on a check. Since then, there have been many successful applications of sophisticated CNN architectures to medicine, engineering, astronomy and the sciences for supervised pattern classification

---

<sup>3</sup>The small filters in Figure 9.3 appear to be two dimensional with small open rectangles in one feature map that telescope down to a point in the neighboring layer. However, the filter is actually three dimensional, where the small rectangular area is shifted laterally to occupy a 3D portion of each feature map in that layer. See the  $3 \times 3 \times 8$  filter in Figure 9.5 for a more accurate rendering of a 3D filter.

# LeNet-5

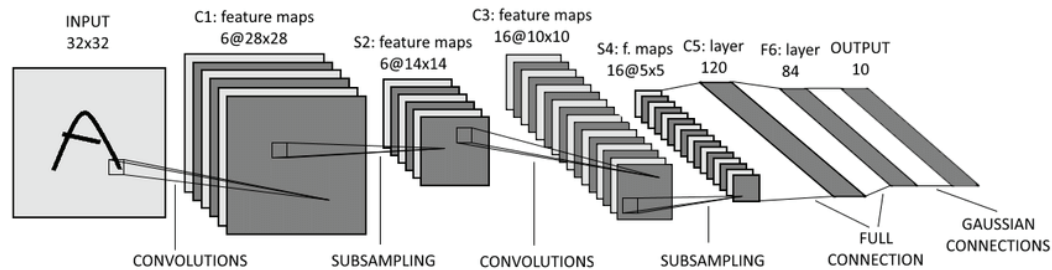


Figure 9.3: LeNet-5 CNN architecture (LeCun et al. 1998). Each plane is a feature map. This work was preceded by that of Zhang et al. (1988) who developed convolutional networks for image character recognition. Each filter symbol with the small rectangle appears to be 2D, but this filter is actually 3D where the rectangle of weights extends (perpendicular to the plane of the feature maps) to all the feature maps in a layer (see bright-yellow 3D filter in 1st layer of Figure 9.5). Figure from Golovko et al. (2017).

Oh and Jung (2004) achieved a computational breakthrough in the practical use of CNNs when they showed that a large number of GPUs could achieve more than a 10-fold speedup of CNN calculations compared to traditional CPU chips. This was followed by many demonstrations of the significant speedups offered by GPUs in solving for the weights in a CNN. Between 2011 and 2012, GPU codes won at least four image recognition contests ([https://en.wikipedia.org/wiki/Convolutional\\_neural\\_network#cite\\_note-0-2](https://en.wikipedia.org/wiki/Convolutional_neural_network#cite_note-0-2)). A deep CNN with more than 100 layers won the ImageNet 2015 contest (He et al., 2016), and a history of some competitions is shown in Figure 9.4.

CNN and its variants have become the most popular form of supervised learning since the early 2015's. It efficiently takes advantage of the hierarchical patterns in data and assembles more complex patterns from smaller and simpler patterns ([https://en.wikipedia.org/wiki/Convolutional\\_neural\\_network](https://en.wikipedia.org/wiki/Convolutional_neural_network)).

## 9.2 Building Blocks of CNN

The workflow in section 4.1.5 for estimating the weights in a neural network is also valid for a CNN. However, there are some new additions to form the building blocks of the CNN workflow. We will now describe these new additions using a Keras code<sup>4</sup> that implements the five-layer CNN architecture in Figure 9.5a. Figure 9.5b depicts the forward modeling Keras code in b), and a summary of the parameters is in c). The Keras code for the AlexNet architecture is in <https://engmrk.com/alexnet-implementation-using-keras/>.

<sup>4</sup><https://colab.research.google.com/drive/1ZZmYiFuSnVIlczmJRa3ncaer2sSu1GQA?usp=sharing#scrollTo=Lo3S63zGQ6Y>

# Training Error vs CNN Depth

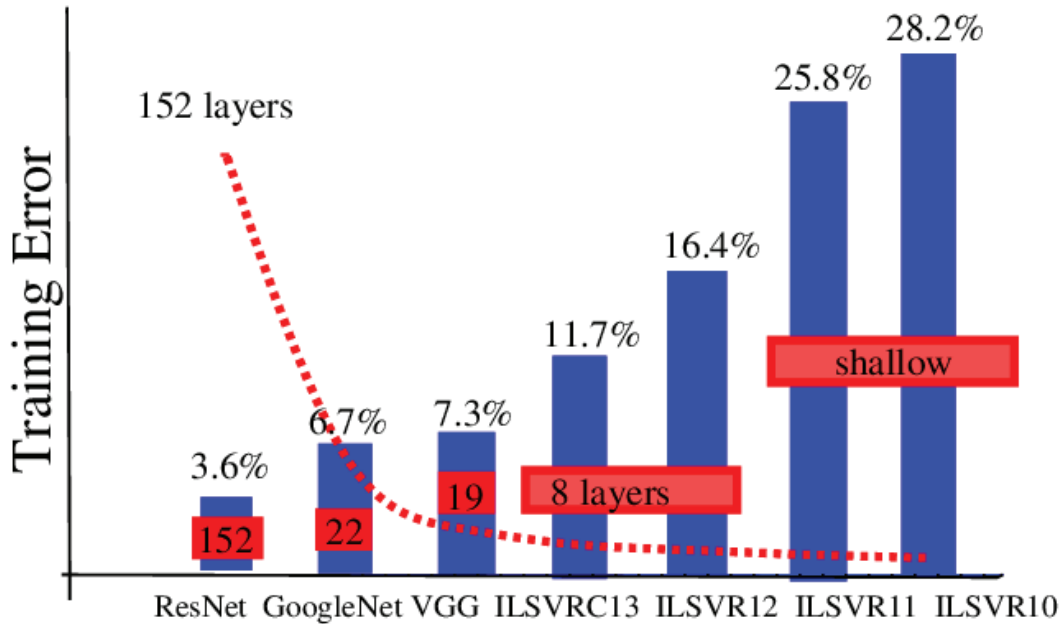


Figure 9.4: Training error vs calendar years for different CNN architectures developed over the last decade. Figure adapted from Youtube video (<https://www.youtube.com/watch?v=u6aEYuemt0M> accessed 11/03/2021) of Deep Learning for Computer Vision by Andrej Karpathy.

## 9.2.1 CNN Forward Modeling

The CNN forward modeling Keras code in Figure 9.5b is listed in Code 9.2.1.

**Code 9.2.1.** Keras Code for CNN Forward Modeling in Figure 9.5.

```

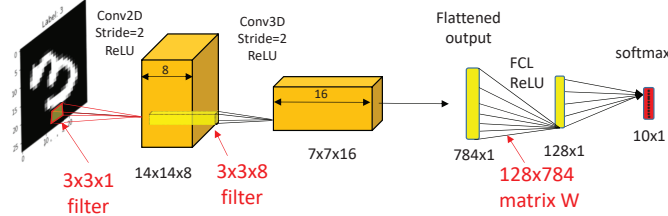
model = Sequential()
Line 1....model.add(Input(shape=(X_train.shape[1], X_train.shape[2], 1)))
Line 2....model.add(layers.Conv2D(filters=8, kernel_size=3, strides=(2, 2),
padding='same', activation="relu"))
Line 3....model.add(layers.Conv2D(filters=16, kernel_size=3, strides=(2, 2),
padding='same', activation="relu"))
Line 4....model.add(layers.Flatten())
Line 5....model.add(layers.Dense(128, activation='relu'))
Line 6....model.add(layers.Dense(10, activation='softmax'))

Line 7....model.summary()

```



## a) Five-Layer CNN Architecture for Classifying MNIST Number Images



## b) Keras Code

```

model = Sequential()
Line 1: model.add(Input(shape=(X_train.shape[1], X_train.shape[2], 1)))
Line 2: model.add(layers.Conv2D(filters=8, kernel_size=3, strides=(2, 2), padding='same', activation='relu'))
Line 3: model.add(layers.Conv2D(filters=16, kernel_size=3, strides=(2, 2), padding='same', activation='relu'))
Line 4: model.add(layers.Flatten())
Line 5: model.add(layers.Dense(128, activation='relu'))
Line 6: model.add(layers.Dense(10, activation='softmax'))

```

## c) Summary

```

model.summary()
Model: "sequential_5"

```

Layer (type)	Output Shape	Param #
Line 1: conv2d_6 (Conv2D)	(None, 14, 14, 8)	80
Line 2: conv2d_7 (Conv2D)	(None, 7, 7, 16)	1168
Line 3: Flatten_3 (Flatten)	(None, 784)	0
Line 4: dense_5 (Dense)	(None, 128)	100480
Line 5: dense_6 (Dense)	(None, 10)	1290
Total params: 103,018		
Trainable params: 103,018		
Non-trainable params: 0		

Figure 9.5: a) Forward modeling architecture for a five-layer CNN, b) Keras code for forward modeling with this architecture, c) summary of the filter characteristics and parameters. The input is a black-white image of a single-digit hand-written number and the goal of the CNN is to classify it as one of the numbers between 0 and 9.

Each operation in the CNN model is defined with the function call `model.add(·)`, where `(·)` represents another function along with its specified hyperparameters. The functions in Code 9.2.1 are explained below.

1. **Convolution Layer:** `layers2D(filters=8, kernel_size=3, strides=(2,2), padding='same', activation='relu')`. The input image in Figure 9.5a consists of a  $28 \times 28$  grid of intensity values that depicts a hand-written number. For the first convolution layer, the filter size  $3 \times 3 \times 1$  is specified with `kernel_size=3` for the coefficients  $w_i^{[1]}$  in equation 9.1. In this case there is only one channel for the input image, but with a color picture there would be the red, green and blue channels so the Keras code would automatically provide a  $3 \times 3 \times 3$  filter in the first layer. The 2D portion of this 3D filter is denoted as the kernel, which in this case is  $3 \times 3$ .

Taking the dot product of the  $3 \times 3 \times 1$  filter with different patches of the input image gives rise to the specified, i.e. `filters=8`, eight feature maps (FMs) in the leftmost orange block in Figure 9.5a. Each FM has the dimensions of  $14 \times 14$  because the convolution has a stride of 2 by the command `strides=(2,2)`. A stride `skip = 2` convolution only centers the correlation filter at every `skip=2` gridpoints along the horizontal and vertical coordinates. Therefore, the  $14 \times 14$  FM is half the size of the  $28 \times 28$  input image. An example of size reduction with `strides=(2,2)` is depicted in Figure 9.6a.

Zero-padding<sup>5</sup> is specified by the command `padding='same'` so that the filter can be centered

<sup>5</sup>If no zero-padding was specified by the command `padding = valid`, then the FMs would be slightly

all the way to the edge of the computational grid to give the same size output if **skip=1**. In general, the dimensions of the FM are determined by the size of the filter kernel, the FM size of the previous layer, and the stride<sup>6</sup>.

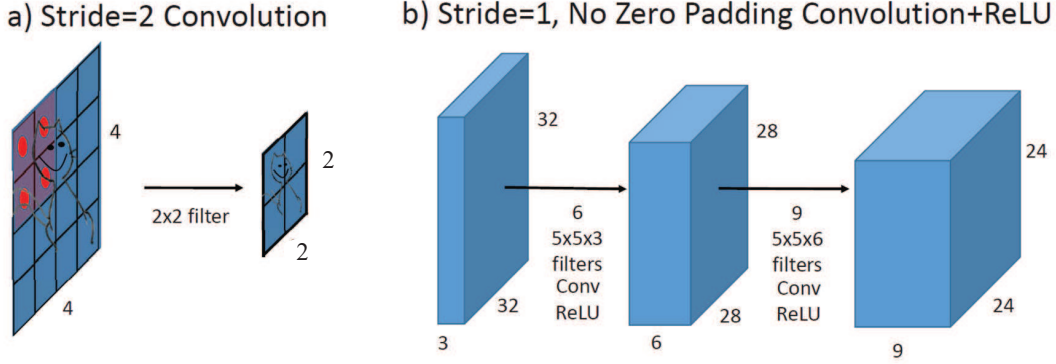


Figure 9.6: Examples of reduction in size of FMs by a) stride=2 convolution and b) sequential convolutions with  $5 \times 5 \times n_{ch}$  filters with stride  $n_s = 1$  and  $n_{ch} = 3, 6$ . The stride 2 convolution filter centers the  $2 \times 2$  filter (red square with red dots) at every other gridpoint. In b), the first layer has three FMs, the second one has 6 and the third one has 9 FMs. The width and height of the blue blocks have decreased by 4 gridpoints because the  $5 \times 5$  kernel cannot be centered at two or fewer gridpoints from an edge, otherwise the arms of the filter stencil fall off the computational grid. This problem can be remedied by zero padding around the boundaries of the blocks prior to convolution.

The convolution operations in the second layer in Figure 9.5a are specified to output the  $7 \times 7 \times 16$  block of FMs. This block consists of 16 feature maps specified by the Line 3 command **filters=16** in Figure 9.5b. As before a  $3 \times 3$  kernel is specified except now the 3D filter automatically has dimension  $3 \times 3 \times 8$  because the input is the  $14 \times 14 \times 8$  block with 8 FMs<sup>7</sup>. See Figure 8.5 for an example of a 3D filter.

2. **Activation Function.** Activation functions are used to eliminate noisy or unnecessary parts of the result after matched filtering. This concept is illustrated in Figure 8.8 and discussed in

less than half the size of the  $28 \times 28$  input image. This is because the filter cannot be centered at the edges of the input image, otherwise part of the  $3 \times 3$  filter stencil will fall off the grid. The closest a  $3 \times 3$  kernel can be centered next to the edge is just one gridpoint. If the filter size is  $5 \times 5$  then 3 gridpoints is the closest a filter center can be to the edge. In this case the output FM will be 4 points shorter and 4 points thinner than the original image. See Figure 9.6b.

<sup>6</sup>In general, if the input image has size  $n \times n \times n_{ch}$  then the filter size is  $n_f \times n_f \times n_{ch}$  and the output feature map dimension is  $n_c \times n_c$ . Here,

$$n_c = \left\lceil \frac{n - n_f + 1}{n_s} \right\rceil + n_p, \quad (9.3)$$

and  $n_p$  is the width and height of the zero-padding added to the feature map. Here,  $n_{ch}$  is the number of channels,  $n \times n$  is the dimension of the input image, and  $n_s$  is the stride of the filter which is equal to the number of pixels the filter is shifted between each dot product of the dot-product operation. In Figure 9.6b,  $n_s = 1$ ,  $n = 32$ ,  $n_p = 0$  and  $n_f = 5$  to give  $n_c = 28$ .

<sup>7</sup>A general 3D convolution operation will have shifting along all three coordinate axes. However, the 3D filter applied to the 2D FMs will not have a shift along the FM axis.

section 4.2.

For the first layer, the ReLU (see equation 4.31) activation function is specified with the Line 2 command `activation=relu` in Figure 9.5b and Code 9.2.1. To change the activation function to a sigmoid simply replace `relu` by `sigmoid` in the activation part so the Line 2 command contains `activation=sigmoid`.

The ReLU function is often preferred over most other activation functions but can lead to the exploding gradient problem discussed in section 4.2. This problem can be mitigated by applying an ad hoc threshold to the output of the ReLU functions, as discussed in Chapter 12. Another possibility is to use the Swish activation function in equation 4.36.

3. **Feature Maps.** There are eight feature maps (FMs) in the first layer of Figure 9.5a. These FMs result from applying eight different  $3 \times 3 \times 1$  filters  $\mathbf{w}_i$  for  $i \in [1, 2, \dots, 8]$  to the input image. The role of each filter is to, hopefully, decompose the original image into eight independent images, each one associated with one of the eight local features associated with the filters. For example, two of the filters might be  $\mathbf{w}_1$  and  $\mathbf{w}_2$ :

$$\mathbf{w}_1 = \begin{bmatrix} 0 & -1 & 1 \\ 0 & -1 & 1 \\ 0 & -1 & 1 \end{bmatrix}; \quad \mathbf{w}_2 = \begin{bmatrix} 0 & 0 & 0 \\ -1 & -1 & -1 \\ 1 & 1 & 1 \end{bmatrix}, \quad (9.4)$$

then the FM for  $\mathbf{w}_1$  would be largely composed of the vertical features of the number image 3 while the FM for  $\mathbf{w}_2$  would largely consist of the horizontal feature. These filters act as *matching* filters and extract parts of the image that are similar to the patterns in the filters.

4. **Subsampling Layer.** Figure 9.5a depicts the blocks of FMs that are subsampled by 1/2. This is accomplished by using a stride of 2 in the filtering operation, which is specified in Line 2 of the Figure 9.5b Keras code. One benefit of this operation is that it reduces computational and storage costs. See section 9.2.2 for an example.

In practice, the number of filters is typically doubled after 1/2 subsampling, which nullifies the potential benefit of reduced computations. However, reducing the size of the image provides some resilience to different scales, rotations and translations of an object, as illustrated in Figure 9.7. This is similar to the multiscale approach used in seismic inversion, where model parameters associated with large-scale features in the data are computed separately from the parameters associated with finer-scale features (Bunks et al., 1995). The separation between different scales can be accomplished by appropriate transforms such as the Fourier or Wavelet transforms (Yilmaz, 2001; Chanerley and Alexander, 2002).

Subsampling also doubles the spacing between neighboring gridpoints. This in turn allows a  $3 \times 3$  filter to be twice as tall and wide as in the previous FM. For example, the cat in Figure 9.6a is the same height before and after stride=2 convolution. Larger filters can be tuned to larger patterns in the image, which characterizes filters that are deeper in a network with sequences of convolution, activation and subsampling operations.

An older subsampling strategy is to replace the intensity value at a pixel by the, for example, average value of its nearest neighbors. A related strategy is known as max pooling and is illustrated in Figure 9.8. Here, the maximum value of the four pixels in the  $2 \times 2$  patch replaces the intensity value of the central point. The  $2 \times 2$  patch is slid over by two pixels, i.e. a stride of 2, and the max-pool process is repeated. In this example, the  $4 \times 4$  patch of pixels is reduced to a  $2 \times 2$  patch. Typically, a pooling filter is no larger than about  $2 \times 2$ , otherwise high-wavenumber information gets lost.

The combination of the operations *convolution*, *activation* and *subsampling* is considered one layer. A sequence of such layers is repeated with different hyperparameters for each layer until we get to the last few layers. The last few layers are often, not always, FCNNs in order

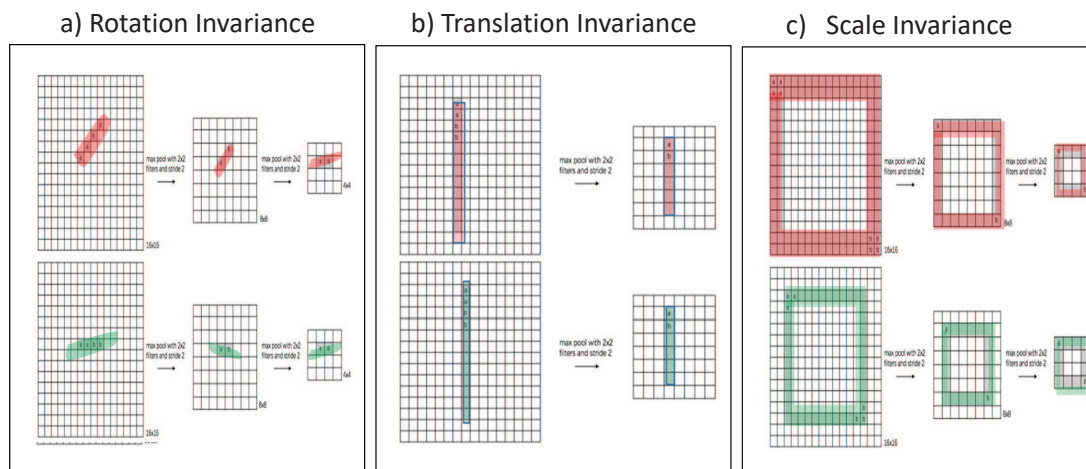


Figure 9.7: Subsampling the input image provides some resilience to changes in the image a) rotation, b) translation, and scale. Notice that the red and green patterns are the same after several subsampling operations.

to combine information from all the FMs. The hyperparameters are the design parameters, such as the number of filters, filter size, number of layers, and other parameters that don't get updated in the iterative backprojection operations. In contrast, the weights  $w_{i-j}^{[n]}$  get updated every iteration and are denoted as model parameters.

### $\frac{1}{2}$ Subsampling $\mathbf{S}$ and its Approximate Inverse $\mathbf{S}^T$

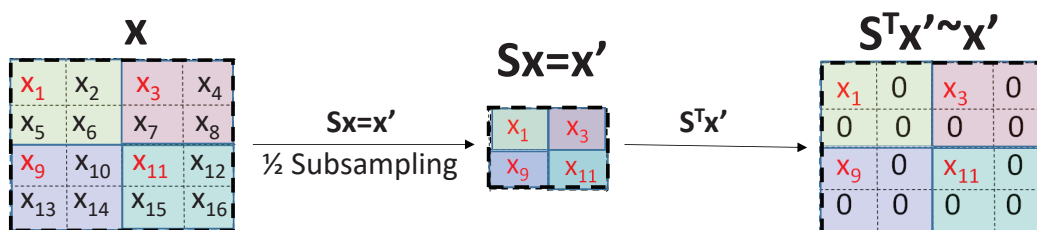


Figure 9.8: Grid of numbers  $\mathbf{x}$  subsampled by  $\mathbf{S}\mathbf{x} = \mathbf{x}'$  to get the  $2 \times 2$  grid in the middle. The subsampled  $4 \times 1$  vector  $\mathbf{x}'$  is upsampled by applying the  $16 \times 4$  subsampling matrix  $\mathbf{S}^T$  in equation to  $\mathbf{x}'$  to get the  $4 \times 4$  grid of numbers on the right.

5. **Fully-Connected Layer.** The last few layers combine all the local FMs into a *fully-connected* image by concatenating all of the vectors associated with each FM into one tall vector. This flattening is carried out by the Line 4 command `model.add(layers.Flatten())` in Figure 9.5b. This flattened vector is now the  $784 \times 1$  input into the fully-connected layer

(FCL) with 128 nodes. The command for the FCL is `model.add(layers.Dense(128, activation='relu'))`. The fully-connected layer uses the input as the  $784 \times 1$  flattened vector from the previous layer's FMs to form the  $\mathbf{W}$  matrix with dimension  $128 \times 784$ . It brings together many localized features from different FMs to form the  $128 \times 1$  vector  $\mathbf{W}\mathbf{a} = \mathbf{z}$ . Now the network consists of both the sparse convolutional matrices and the dense matrices for FCNN operations. This hybrid network is still considered as a CNN.

6. **Softmax Layer** In image classification there might be dozens of classes, not just two as in binary classification. In this case the softmax activation function  $g(\mathbf{z})$  in equation 4.55 is used to classify the input image. Since there are ten possible single-digit numbers the softmax function is designed to output a  $10 \times 1$  vector, as specified by the Line 6 command `layers.Dense(10, activation='softmax')`.
7. **Parameter Count.** Line 1 in Figure 9.5c lists 80 unknown filter coefficients  $w_{ij}^{[1]}$  for the first layer. In this case the number of unknown filter coefficients for the eight  $3 \times 3$  filters is only  $3 \cdot 3 \cdot 8 = 72$ . However, there are 8 filters so there are an additional 8 bias terms  $b$ , one for each filter, that must be computed. Therefore, the total number of unknown parameters is  $72 + 8 = 80$ .

The back-propagation algorithm determines the patterns that will be extracted, i.e. the weights for these filters  $w_{ij}^{[n]}$ , so it is a mystery as to what types of filters will emerge. This mystery is partially solved by the recognition that the CNN problem can be recast as an optimization problem that minimizes the misfit objective function subject to the  $L^1$  norm of the weights. This subject is discussed in Chapter 24.

What is not mysterious is that these patterns are for small features in the first layer because the filters are specified to be small and localized compared to the size of the input image. They are somewhat equivalent to high-wavenumber pass filters.

### 9.2.2 Model Update by Back-propagation

The coefficients in the filters specified for the Figure 9.5a model and the forward modeling code in Figure 9.5b are updated using a stochastic gradient descent procedure. The formulas for the CNN gradient are the same as for the NN gradient described in Chapters 4 and 6. However, the inverse to the subsampling operation does not exist. That is, it is straightforward to transform a  $4 \times 4$  grid of numbers into a  $2 \times 2$  grid of numbers, but there is no unique mapping from a  $2 \times 2$  grid of numbers to a  $4 \times 4$  grid of numbers. Thus the inverse does not exist. Instead, various approximations are made to the transpose of the sub-sampling operator. This lacks a rigorous mathematical foundation but it seems to work for CNN.

#### Transpose of the Subsampling Operator

Why are we interested in the transpose of the sub-sampling (or pooling operator)? Recall that the  $L^2$  minimization problem demands the gradient of the misfit  $\epsilon = 1/2 \|\mathbf{L}\mathbf{m} - \mathbf{d}\|^2$ :  $\nabla \epsilon = \mathbf{L}^T(\mathbf{L}\mathbf{m} - \mathbf{d})$ . This means that if the forward modeling operator is represented by the rectangular matrix  $\mathbf{L}$  then the gradient takes the transpose of the forward modeling matrix and applies it to the residual vector. In a deep neural network there is a long chain of matrix-matrix operations and squashing functions, so the transpose will be a long chain of multiplicative matrix transposes interspersed with the transpose of squashing functions. Some of the matrix transposes will be those for the pooling operators. Thus the transpose of that operator needs to be constructed.

An example of a pooling operator that sub-samples the  $4 \times 4$  grid of numbers in Figure 9.8 into

a  $2 \times 2$  grid is the matrix-vector operation  $\mathbf{S}\mathbf{x}$

$$\overbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}}^{\mathbf{S}} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{15} \\ x_{16} \end{bmatrix} = \begin{bmatrix} x_1 \\ x_3 \\ x_9 \\ x_{11} \end{bmatrix}, \quad (9.5)$$

where  $\mathbf{x} = \{x_1, x_2, \dots, x_{16}\}$  is the  $16 \times 1$  vector after flattening the  $4 \times 4$  grid of numbers. The rightmost image in Figure 9.8 depicts the result after applying the transpose matrix  $\mathbf{S}^T$  to the  $4 \times 1$  vector  $\{x_1, x_3, x_9, x_{11}\}$ :

$$\overbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}^{\mathbf{S}^T} \begin{bmatrix} x_1 \\ x_3 \\ x_9 \\ x_{11} \end{bmatrix} = \begin{bmatrix} x_1 \\ 0 \\ x_3 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ x_9 \\ 0 \\ x_{11} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad (9.6)$$

where the values of  $x_1$ ,  $x_3$ ,  $x_9$  and  $x_{11}$  are repositioned to their original locations in the  $4 \times 4$  grid in Figure 9.8. The values at the other gridpoints are zero, which is sometimes known as bed-of-nails upsampling (Mishra, 2020).

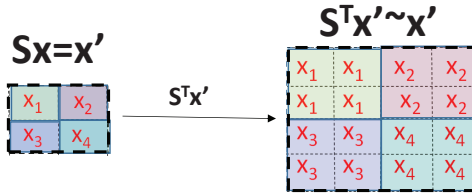
If the maximum values in each  $2 \times 2$  grid of numbers on the left of Figure 9.8, for example the green square, is used to fill up the corresponding colored squares in the  $2 \times 2$  grid of  $\mathbf{S}\mathbf{x} = \mathbf{x}'$ , then this subsample procedure is known as max pooling (Goodfellow et al., 2016). The unpooling operation is the same as seen on the rightside of Figure 9.8, except the squares with non-zero values are the same as the ones with maximum values on the left side. For example, if  $x_1$ ,  $x_3$ ,  $x_9$ , and  $x_{11}$  happen to be the maximum values in their  $2 \times 2$  neighborhoods then Figure 9.8 describes max pooling and max unpooling. Here, the maximum positions in the pooling layer determine where the non-zero values are in the max unpooling layer.

The average values in each  $2 \times 2$  colored square in  $\mathbf{x}$  on the leftside of Figure 9.8 can be used to give the values in the corresponding colored squares in  $\mathbf{S}\mathbf{x}$ . This procedure is known as average-pooling.

Two other unpooling operations are nearest neighbor interpolation in Figure 9.9a and bi-linear interpolation in Figure 9.9b. In bi-linear interpolation, the 4 nearest pixel values of the input pixel are used to perform a weighted average based on the distance of the four nearest cells from the input pixel. Both operations are used in upsampling networks such as an autoencoder discussed in Chapter 14.

If the stride of the muting filter is equal to 1, then the transpose matrix approximates the inverse

## a) Nearest Neighbor Upsampling



## b) Bi-linear Interpolation

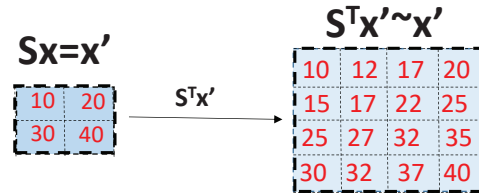


Figure 9.9: Upsampling from a smaller grid to a larger grid by a) nearest neighbor interpolation and b) bi-linear interpolation. In nearest neighbor interpolation, the input value  $x_1$  is copied to its nearest K-neighbors.

operation<sup>8</sup>. As noted in Chapter 8, the transpose to a convolution matrix is a correlation matrix, and vice versa.

### 9.2.3 Keras Code

Figure 9.5 defines the multiclass classification problem discussed in section 4.4 where each input image is labeled with just one of the 10 single-digit labels. The true labels of the output  $10 \times 1$  vector are all zero, except for the element position that defines the class type. The Keras code in Code 9.2.2 updates the CNN weights by SGD.

<sup>8</sup>If  $\mathbf{S}\mathbf{x} = \mathbf{x}'$ , then the inverse operation in the least squares sense is  $\mathbf{x} = [\mathbf{S}^T\mathbf{S}]^{-1}\mathbf{S}^T\mathbf{x}' \approx \mathbf{S}^T\mathbf{x}'$ . Of course  $[\mathbf{S}^T\mathbf{S}]^{-1}$  does not exist for a subsampling operation so it must be heavily regularized to give the mathematical approximation by  $\mathbf{x} \approx \mathbf{S}^T\mathbf{x}'$ .

**Code 9.2.2.** *Keras Code for Updating CNN Weights by SGD*

```

Line 1....opt = tf.keras.optimizers.SGD(learning_rate=0.001)
Line 2....model.compile(optimizer=opt, loss='sparse_categorical_crossentropy',
                        metrics=['accuracy'])
Line 3....history = model.fit(x=X_train, y=y_train, validation_data=
                            (X_val, y_val), shuffle=True, epochs=10, batch_size=128)

```

## Reported Results

Epoch 1/10

```

329/329 [=====] - 5s 16ms/step - loss: 1.5448 -
        accuracy: 0.7469 - val_loss: 0.6181 - val_accuracy: 0.8374

```

Epoch 2/10

```

329/329 [=====] - 5s 16ms/step - loss: 0.4603 -
        accuracy: 0.8697 - val_loss: 0.4556 - val_accuracy: 0.8683

```

Epoch 3/10

```

329/329 [=====] - 5s 16ms/step - loss: 0.3433 -
        accuracy: 0.8992 - val_loss: 0.3528 - val_accuracy: 0.8977

```

Epoch 4/10

For the number-classification problem in Figure 9.5, the multiclass cross-entropy objective function

$$\epsilon = \sum_{k=1}^K \sum_{i=0}^9 \bar{y}_i^{(k)} \ln y_i^{(k)}, \quad (9.7)$$

is used for training (see section 4.4). Here,  $\bar{y}_i^{(k)}$  for  $i \in \{0, 1, \dots, 9\}$  is the true label for the  $k^{th}$  example image and  $y_i$  is the one predicted by the softmax function at the  $i^{th}$  output node for  $i \in \{0, 1, \dots, 9\}$ . This loss function is specified by the Line 2 command **loss='sparse\_categorical\_crossentropy'**. The step length, aka the learning rate, for the stochastic gradient descent method is specified to be equal to 0.001 in Line 1 as

**refopt = tf.keras.optimizers.SGD(learning\_rate=0.001)**. The batch size of 128 and the number of epochs is 10 as specified by the Line 3 commands **epochs=10** and **batch\_size=128**. Each batch of 128 images is randomly reshuffled as specified by the Line 3 command **shuffle=True**.

After 10 epochs, the training is finished with the training images in the training set, which is 80% of the total images in the labeled data. The remaining 20% comprises the images in the validation set.

The trained CNN model is then validated on the unused validation images. The commands for plotting cross-entropy and accuracy as a function of epoch number for both the training data and the validation data are shown in Figure 9.10. The Keras commands for plotting these figures is in Code 9.2.3.



**Code 9.2.3.** *Keras Codes for Plotting Cross-entropy and Accuracy vs Iteration Number*

```
# summarize history for accuracy
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Loss')
plt.ylabel('Binary Crossentropy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

# summarize history for accuracy
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Loss')
plt.ylabel('Binary Cross-entropy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```

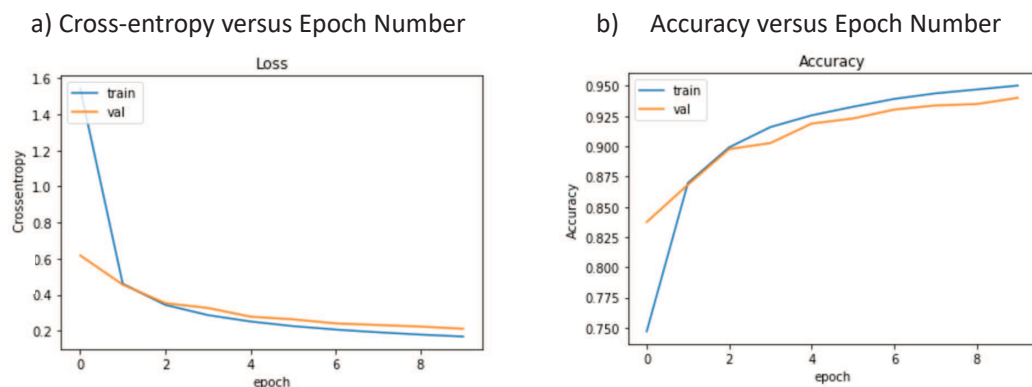


Figure 9.10: Plots of a) cross-entropy and b) accuracy versus epoch number for the five-layer CNN model in Figure 9.5.

The input images and the predicted labels are in Figure 9.11 and the Keras script is in Code 9.2.4.

**Code 9.2.4.** *Keras Code for Plotting Images and Labels*

```

To view the labels next to the input images
plt.figure(figsize=[6,6])
pred=model.predict('X_test')
for i in range(4):
    k = np.random.randint(X_test.shape[0])
    plt.subplot(2,2,i+1)
    plt.title("Predicted Label: %i"%np.argmax(pred[k]))
    plt.imshow(X_test[k].reshape([28,28]), cmap='gray');

```

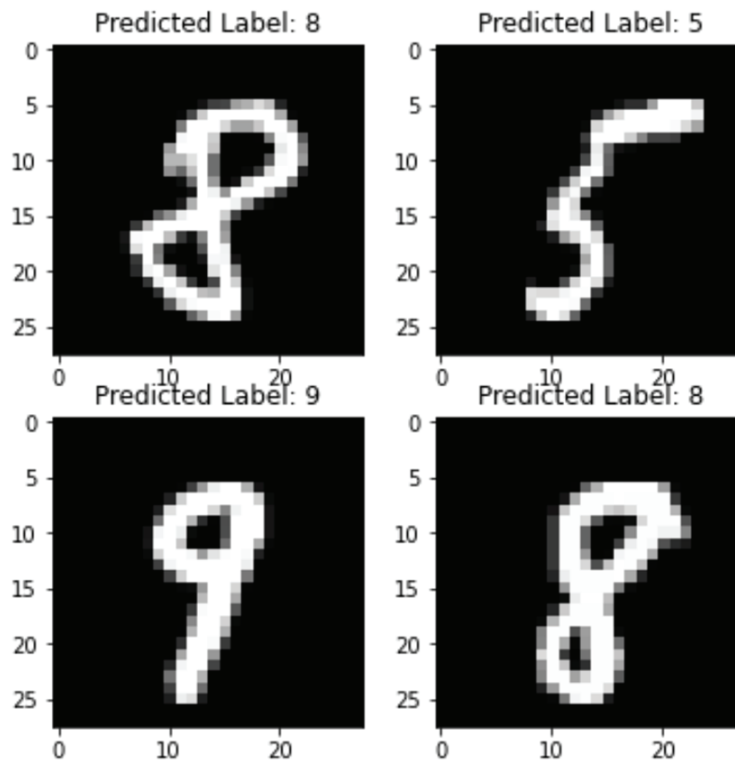


Figure 9.11: Images of hand-written numbers and the labels predicted from the trained CNN.

### 9.2.4 Visualizing the Filters as Correlation Functions

We now demonstrate that the CNN filters act as matching filters that tune themselves to the dominant features of the input images. There will be two classification examples, the first will be for images of sticks oriented at different angles (see Figure 9.12a) and the other will be for MNIST

images (see Figure 9.13a). The CNN will consist of two layers, a hidden layer with large filters and the last layer consists of a softmax function. The output vector has the same dimension as the number of classes. These networks are considered to be extreme because their filter size is the same as the input image. There will be no zero-padding so there exists only one lag value, which is zero lag, that produces the  $1 \times 1$  feature map.

The first test is for classifying  $40 \times 40$  images of sticks oriented at 25 different angles, as shown in Figure 9.12a. The CNN architecture is shown at the top of Figure 9.12, and consists of 25  $40 \times 40$  filters that are the same size as the input pictures. In this case, the filters act as a dictionary with the same sized grids as the input image and the same number of words as the vocabulary of the input pictures. The input training set consists of 25 images, each with a unique orientation as shown in Figure 9.12a. After training with 20 epochs and batch sizes of 5 with an Adam step length, the results are shown in Figure 9.12b. The learned filters strongly resemble the actual input images, partly because they are of the same size and same number as the input images. This is a severely undetermined problem because there are 25 filters with  $40 \times 40 = 1600$  unknowns per filter, so the total number of unknowns is  $25 \times 1600 = 40,000$ . However, there are only 25 input images, which is the same as saying that there are only 25 equations of constraint, more than three orders-of-magnitude fewer equations than the number of unknowns.

In practice, the filters should be much smaller than the input images in order to avoid creating a huge dictionary of filter words. Instead we want the CNN to create a smaller number of alphabet filters that recognize letters in the first layer, recognize words in the second layer and so on. When smaller-sized filters were tested, say  $5 \times 5$  leading to fewer unknowns, the accuracy in identifying the orientation of the sticks was more than 95%. Unlike Figure 9.12b, the filter shapes were not easily identifiable with different stick orientations.

Figure 9.12b shows that each image is surrounded by ghost-like images of the other sticks. This is denoted as cross-talk noise because there is some correlation of the filter with sticks of different orientations.

The feature maps (FMs) are of size  $1 \times 1$  because only the zero-lag correlation is allowed in the filter. An example of the numbers in the 25  $1 \times 1$  FMs for an input stick with an orientation of  $60^\circ$  is given below. Here, the red number is the one for the FM associated with the filter orientation of  $60^\circ$ . Underneath these 25 numbers are the 25 numbers for the  $25 \times 1$  softmax output. As expected, the softmax output is highest at the 9<sup>th</sup> element position, which indicates the class of sticks with  $60^\circ$  orientation.

Feature map for the filter at 60 degrees:(correlation result)

```
[-3.9313195 -3.8512936 -3.882589 -3.7395825 -3.694348 -3.5521138
-3.3681073 -0.96085596 2.8906903 -1.1699747 -3.167439 -3.5620592
-3.7346756 -3.8448954 -3.9283197 -3.952573 -3.9429069 -3.968153
-4.0101204 -3.9551105 -3.974689 -3.955679 -3.9627805 -3.9612474
-3.9414587 ]
```

25x1 Softmax Output Vector

```
[0.00102189 0.00110703 0.00107292 0.00123787 0.00129515 0.00149311
0.00179475 0.01992785 0.93791986 0.01616744 0.00219358 0.00147833
0.00124396 0.00111414 0.00102496 0.0010004 0.00101012 0.00098494
0.00094446 0.00099787 0.00097852 0.0009973 0.00099024 0.00099176
0.00101158]
```

The second example is for 60,000 images from the MNIST data. Here, a two-layer CNN with 10 filters is used for training, where the input pictures are grayscale images on a  $28 \times 28$  grid, and the size of each filter is  $28 \times 28$  gridpoints. A softmax function is applied to the  $10 \times 1$  output from the first hidden layer to give the  $10 \times 1$  output vector that classifies the label for the input image. The total number of unknown filter coefficients is  $28 \times 28 \times 10 = 7,840$ . The goal is to create filters

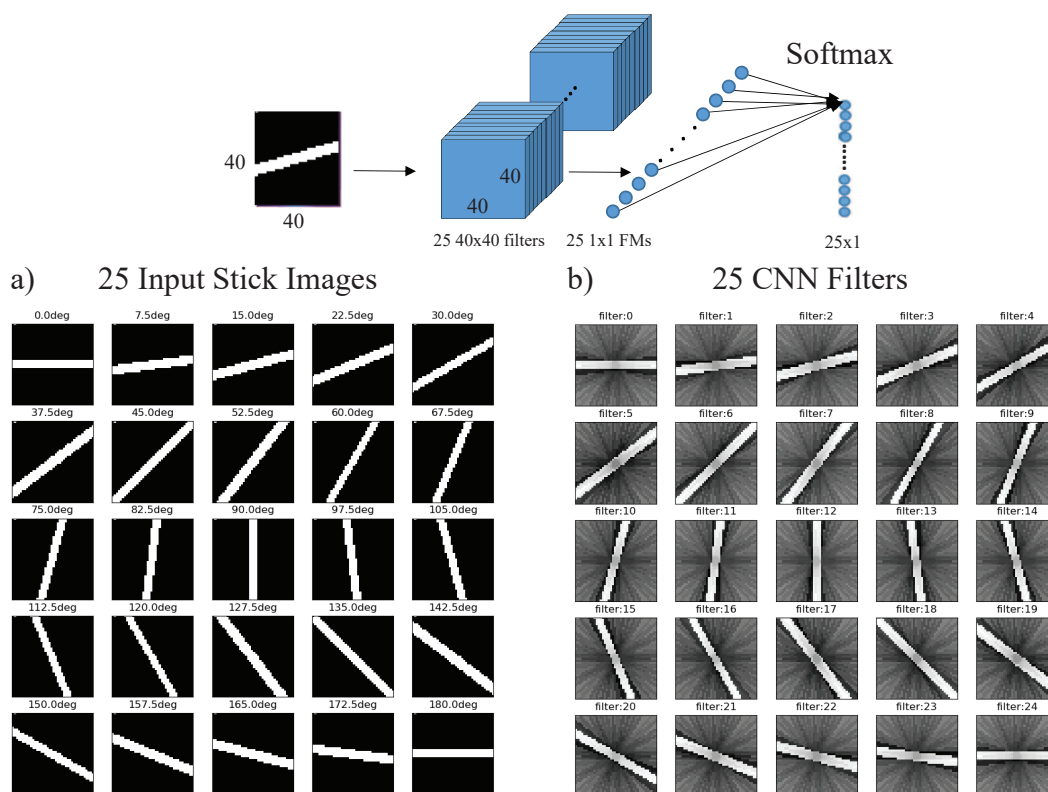


Figure 9.12: Images of a) sticks with different orientations and b) CNN filters learned by back-propagation with the architecture at the top.

that, hopefully, can be interpreted as matching filters. See the top of Figure 9.13 for the network diagram.

The results after 20 epochs, with a batch size of 100, provides an accuracy of about 91%, which is less than that for the deeper network used in the previous MNIST example. Figure 9.13b shows nine of the *learned* filters, some of which roughly resemble the numbers of interest. These results are consistent with the claim that the CNN filters can act as matching filters tuned to different features in the input data.

### 9.2.5 Visualizing the Images Associated with Feature Maps

Which portion of a feature map is related to which object in the input image? To answer this question (Zeiler and Fergus, 2013; Simonyan and Zisserman, 2014; Simonyan et al., 2014), assumed a well-trained network so that the  $k^{th}$  FM in the  $n^{th}$  layer is given by  $\mathbf{a}_k^{[n]} = f_n^{[n]}(\mathbf{x})$ , where  $f_n^{[n]}(\mathbf{x})$  represents the forward modeling operations in the CNN that compute this feature map from the input image. The goal is to find  $\mathbf{x} \approx \mathbf{x}_k^{[n]} = [f(\mathbf{x})_n^{[n]}]^{-1}(\mathbf{a}_k^{[n]})$ , where  $\mathbf{x}_k^{[n]}$  is the portion of the input image which is largely responsible for the feature-map values  $\mathbf{a}_k^{[n]}$ , and  $[f(\mathbf{x})_n^{[n]}]^{-1}(\cdot)$  represents the approximation to the inverse of the forward modeling operator  $f(\mathbf{x})_n^{[n]}(\cdot)$ . In practice, the inverse is approximated by the adjoint operations used in back-propagation of residuals, except only the values of the selected FM are back-propagated. No batch normalization is used in the adjoint operations

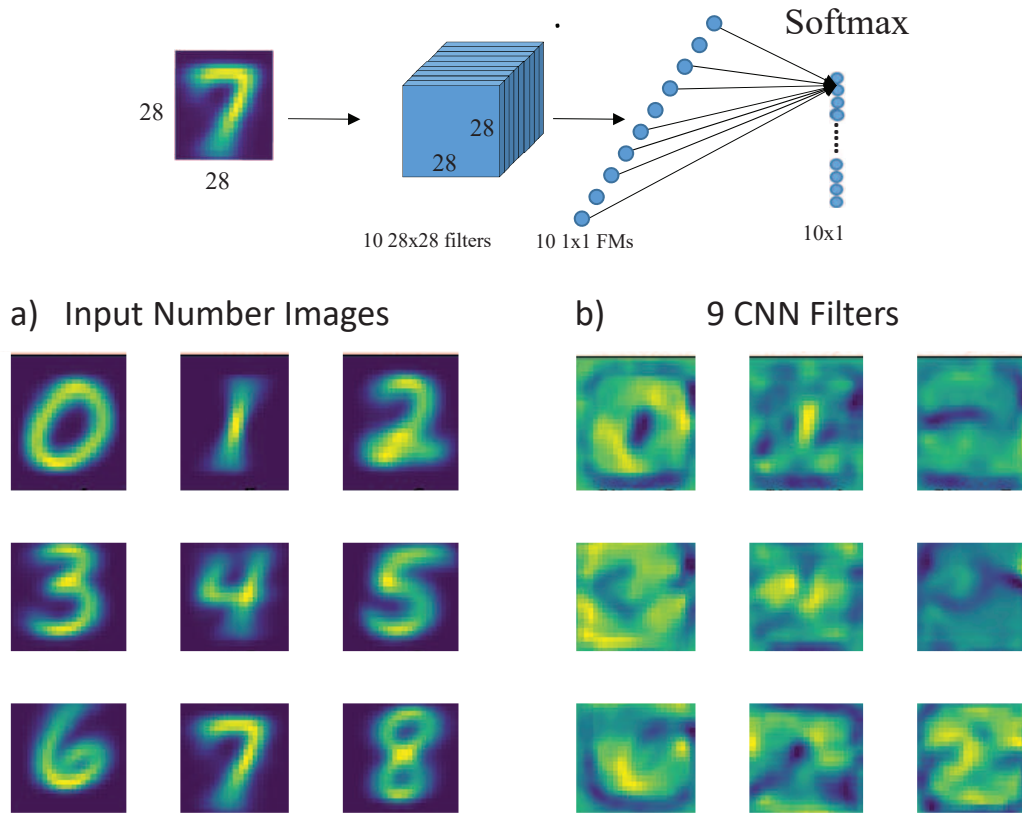


Figure 9.13: Images of a) hand-written numbers and b) CNN filters. The architecture of the network is at the top.

(Zeiler and Fergus, 2013).

As an example, Figure 9.14 depicts the CNN architecture and the input image of a Pomeranian dog. The CNN is well trained with many different animal pictures and the FM with the strongest weights is extracted from the 5th layer for the Pomeranian image. All the other FMs in this layer are muted except for this FM, which is back-propagated to the input layer to give the grayscale rendering at the top left. This image resembles an upside-down rendering of the dog-face, which is important for classification. Different portions of the input image can be masked to determine how they affect this FM. Systematically occluding portions of the input and forward propagating the input to the specified FM will reveal its influence there. This back-propagation procedure is used to answer the question about which portion of a feature map is related to which object in the input image.

The back-propagation procedure can be used to show that low-wavenumber portions of the input image are represented at the deeper portions of the network. It can also show that smaller details of the input image are captured by the shallower layers. This is not surprising: repetitive use of the pooling operations force the filters to be lower wavenumber filters.

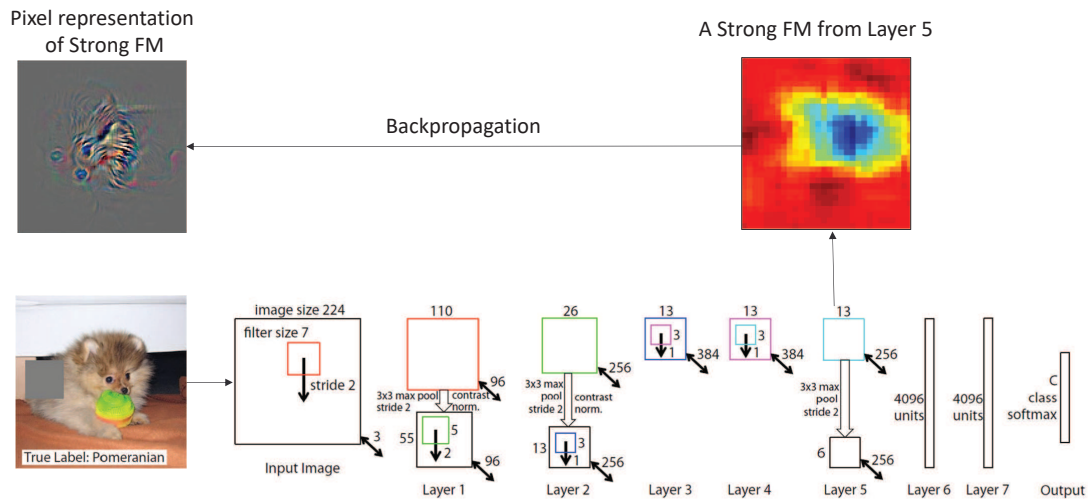


Figure 9.14: The dog image is forward modeled and the FM at the 5<sup>th</sup> layer is extracted to give the colorful image at the top right. This FM is backprojected back to the input space location to give the grayscale image at the top left. Figure adapted from Zeiler and Fergus (2013).

### 9.2.6 Regularization

Too many fully-connected layers and model parameters compared to the number of constraint equations can lead to overfitting, examples of which are shown in Figures 4.27-4.28. Thus, regularization methods such as damping are needed to stabilize the solution.

There are many regularization methods, some of them are listed below and many details for regularizing autoencoders are discussed in Chapter 14.

1. **More data.** Increase the number and variety of examples in the training data.
2. **Early stopping.** Stopping the iterations early so the solution does not fit noise or include superfluous basis functions. This type of regularization is also used for conjugate gradients (Nocedal and Wright, 1999).
3. **Marquardt damping.** A penalty function is added to the objective function so that too many activated neurons (Ng, 2013; Zhou, 2018), non-smoothly varying weights or large weights are penalized. Figure 2.5 illustrates its effect on the search direction.
4. **Dropout.** A successful way to prevent overfitting is to perform dropout on the network's nodes (Nitish et al., 2014) at each iteration. Units are randomly removed at every hidden layer from the network at each iteration, which can be seen as a form of adding noise.

An example might be to randomly remove half the number of nodes in the hidden layers and 1/5 in the input layer; it is not used in the output layer (Brownlee, 2018). Removing nodes in the hidden layers suppresses the activity of some nodes that tend to overfit the input data. Wager et al. (2013) and Srivastava et. al. (2014) mathematically showed that randomly eliminating nodes in a layer is equivalent to regularizing the objective function. This is proven for the linear case in section 14.8.1.

Srivastava et. al. (2014) suggest that if a unit is retained with probability  $p$  during training, the outgoing weights of that unit are multiplied by  $p$  at test time. A typical probability

of  $p = 0.5$  approximates the optimal value for a wide range of networks (Brownlee, 2018). Brownlee (2018) summarizes many practical tips for implementing dropout, including the use of weight constraints if the weights after dropout become too large.

A plausible explanation for the success of dropout regularization is that at each iteration a new network is used to find the next update direction, similar to the red arrows in Figure 4.24. This update direction is likely to explore a much different area of model space compared to the model which is stuck with the same nodes that are memorizing the data, aka as co-adaption (Brownlee, 2018). The narrow area explored by co-adaption might be similar to the narrow region associated with the highest long black arrow in Figure 4.24.

The final weights at the last epoch can be considered as the average model for many network models which have explored a wide area in model space. At the final epoch, the last weights used for every node are the final weights for the test model.

Dropout can significantly improve training speed, even for deep neural nets. The technique seems to reduce node interactions, leading them to learn more robust features that better generalize to new data. Almost all deep networks use dropout.

A better strategy than randomly dropping more than half the nodes during training is to selectively eliminate the parts of the feature map that have a value less than some specified threshold value (Han et al., 2015). In this way the relatively inactive parts of the network are eliminated and mitigates the tendency for overfitting .

Finally, a recent study (Frankel and Carbin, 2019) shows with empirical examples that many nodes with small weights can be eliminated, aka pruned, from the original dense network. The pruned network with up to 90% fewer nodes than the original network can be retrained to *give better accuracy and is more generalizable than the original network*. The initial weights for the original network are the starting weights for the original dense network. Apparently, the location of the active nodes in the trained dense network are sensitive to the initial random weights. The pruned network is much less likely to overfit than the dense network.

5. **Image normalization.** Assume that a CNN classifier is only trained on darkened black-white pictures of apples and bright pictures of pears, where the apple's intensity values are between 15 and 30 and those of the pears are between 30 and 255. Now assume that the CNN is only trained on these pictures. For inference, insert into this trained network a new picture of an apple taken in bright daylight. The apple might be misclassified as a pear because the intensity values range between 0 and 255 and provide details never before seen in an apple.

To overcome this problem, the input pictures in the network are demeaned and normalized by the variance at every iteration of a mini-batch update. For example, the mean  $\bar{x}$  and standard deviation  $\sigma$  of the apple picture are given by

$$\bar{x} = \frac{1}{I} \sum_{i=1}^I x_i; \quad \sigma = \sqrt{\frac{1}{I-1} \sum_{i=1}^I (x_i - \bar{x})^2}, \quad (9.8)$$

and the demeaned and normalized pixel values are given by

$$\tilde{x}_i = \frac{x_i - \bar{x}}{\sigma}, \quad (9.9)$$

where  $I$  is the number of pixels in the input picture. Now, the statistical distributions of the pears and apples will be roughly similar to one another and so the network will tune itself to the geometrical and textural features that distinguish one from the other. More simply, if the sum of the squared residuals in the loss function can be divided between the large residuals of well-lit pears and the small residuals of dark apples, then it will devote its gradient values to reducing the pear-related residuals.

6. **Batch normalization.** A similar procedure is used to normalize the distributions of weights in a layer. This is because as the iterations proceed, the statistical distributions of the weights can dramatically vary from the starting weights of a uniform distribution of a zero-mean Gaussian noise with a standard deviation of 1. Such a large range of weight values at later iterations can slow down convergence and shut down the activation of certain nodes.

## Batch Normalization

**Input:** Values of  $\mathbf{a}$  over a mini-batch:  $\mathbf{B}=\{\mathbf{a}^{(1)}, \mathbf{a}^{(2)}, \dots, \mathbf{a}^{(64)}\}$   
Parameters to be learned  $\gamma, \beta$

**Output:**  $\mathbf{a}_i = \text{BN}(\{\mathbf{a}_i\})$

$$\mu_j = \frac{1}{64} \sum_{i=1}^{64} a_j^{(i)} \quad : \text{mini-batch mean}$$

$$\sigma_j^2 = \frac{1}{64} \sum_{i=1}^{64} (a_j^{(i)} - \mu_j)^2 \quad : \text{mini-batch variance}$$

$$\hat{a}_j^{(i)} = \frac{a_j^{(i)} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}} \quad : \text{normalize}$$

$$\bar{a}_j^{(i)} = \hat{a}_j^{(i)} \gamma + \beta \quad : \text{scale and shift}$$

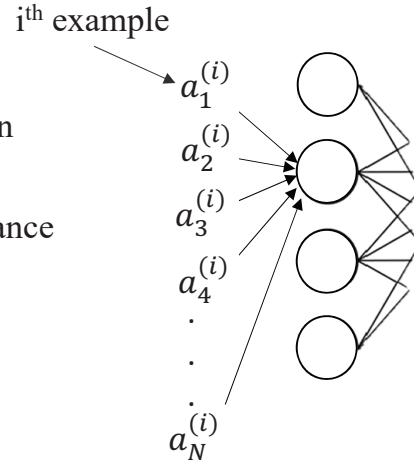


Figure 9.15: Workflow for batch normalization. <https://www.youtube.com/watch?v=nUUqwaxLnWs>.

Figure 9.15 illustrates the strategy of *batch normalization*, where the input values  $a_j^{(i)}$  into a layer's node from a previous node can have a wide distribution of values. These inputs can all have different means and variances, so they will be demeaned and normalized by averaging over the mini-batch of examples. In this case, the  $i^{(i)}$  example is given by  $a_j^{(i)}$  and there are 64 examples in the mini-batch. To insure all the examples are ideally demeaned and normalized, the parameters  $\gamma$  and  $\beta$  are learned. The normalization scalars are applied to the model weights, which is the same as using a diagonal preconditioner as discussed in section 2.1.5. It is also similar to the illumination compensation procedure to correct for loss of amplitudes due to defocusing and geometric spreading (Rickett, 2003).

Batch normalization (Ioffe and Szegedy, 2015) is regularly used to speedup convergence, including the labeling of cracks in Chapter 11. This speedup can sometimes be more than order-of-magnitude.

The regularization methods of *batch normalization* and *dropout* can be added to the Keras code by the commands `model.add(layers.BatchNormalization())` and `model.add(layers.Dropout(0.2))` in Code 9.2.5.



**Code 9.2.5.** *Keras Code with Dropout and Batch Normalization*

```
model = Sequential()
model.add(Input(shape=(X_train.shape[1], X_train.shape[2],
1)))
model.add(layers.Conv2D(filters=8, kernel_size=3,
strides=(2,2), padding='same', activation="relu"))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.2))
model.add(layers.Conv2D(filters=16, kernel_size=3,
strides=(2, 2), padding='same', activation="relu"))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.2))
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.2))
model.add(layers.Dense(10, activation='softmax'))

model.summary()
```

The summary of the parameter values for Code 9.2.5 is in Code 9.2.6.

**Code 9.2.6.** *Keras Summary of Code 9.2.5*

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 14, 14, 8)	80
batch_normalization (Batch Normalization)	(None, 14, 14, 8)	32
dropout_5 (Dropout)	(None, 14, 14, 8)	0
conv2d_4 (Conv2D)	(None, 7, 7, 16)	1168
batch_normalization_1 (Batch Normalization)	(None, 7, 7, 16)	64
dropout_6 (Dropout)	(None, 7, 7, 16)	0
flatten_1 (Flatten)	(None, 784)	0
dense_3 (Dense)	(None, 128)	100480
batch_normalization_2 (Batch Normalization)	(None, 128)	512
dropout_7 (Dropout)	(None, 128)	0
dense_4 (Dense)	(None, 10)	1290
Total params: 103,018		
Trainable params: 103,018		
Non-trainable params: 0		

**9.2.7 Mini-Batch, Step Length, Training and Data Augmentation**

The best practices for using mini-batches, training and step lengths are discussed in section 4.5 and Chapter 4. Typically, mini-batch sizes of 64 or 128 are used, but this will vary from problem to problem.

The training strategy is summarized in Figure 4.10. We first randomly select about 80% of the labeled data for training, and the remaining 20% should be used for validation. Using a gradient descent code, the network trains on the training data and then tests its inference accuracy on the validation set. The training data should encompass all of the characteristics associated with any future data that will be input into the network.

For a classification problem, the number of examples for each class of data in the training set should be about the same. This results in a balanced training set. For example, if dog pictures must be distinguished from cat pictures then the sum of the squared residuals from the dog pictures should roughly be about the same as that from the cat pictures. If there are many more dog pictures then the gradient method will devise update directions that mainly reduce the residuals associated with dogs and largely ignore reducing the cat residuals. Chapter 11 addresses this issue with labeling cracks in sandstones. One possible remedy is to augment the training data. The training set of a few cat pictures can be augmented by applying rotations, scaling, stretching and displacement operations to the existing cat pictures to create new ones. Other procedures such as interpolating the latent space variables in autoencoders is discussed in Chapter 14.

## 9.3 Architectures of CNN

There are many CNN architectures and their performances for some of them are graphed in Figure 9.4. These different architectures appear to have evolved by trial and error, where some intuition is used to help improve the design. Undoubtedly, new architectures and paradigms will emerge to replace the most popular ones today (Roman, 2020).

### 9.3.1 AlexNet

The AlexNet architecture has eight layers, 5 convolutional and 3 fully connected, and is shown in Figure 9.16. It was introduced by Krizhevsky et al. (2012) and won an ImageNet Classification Challenge (ILSVRC) in 2012 by a large margin because the competing algorithms were not based on deep learning. The AlexNet Keras code is in <https://engmrk.com/alexnet-implementation-using-keras/> and has a few more layers than the pioneering LeCun architecture (LeCun et al., 1998). AlexNet is one of the first Deep Neural Networks (DNNs) which classify large numbers of images. It consists of over  $6010^6$  parameters, and is the first to implement the ReLU activation function. At that time the other competing algorithms were not based on Deep Learning. This architecture has had a significant impact on the domain and many practical networks still use this architecture.

The intuition behind this network is that each 2D convolutional layer learns a more detailed representation of the images than the previous one. For example the first layer is able to recognize very simple forms or colors, and the last one more complex forms.

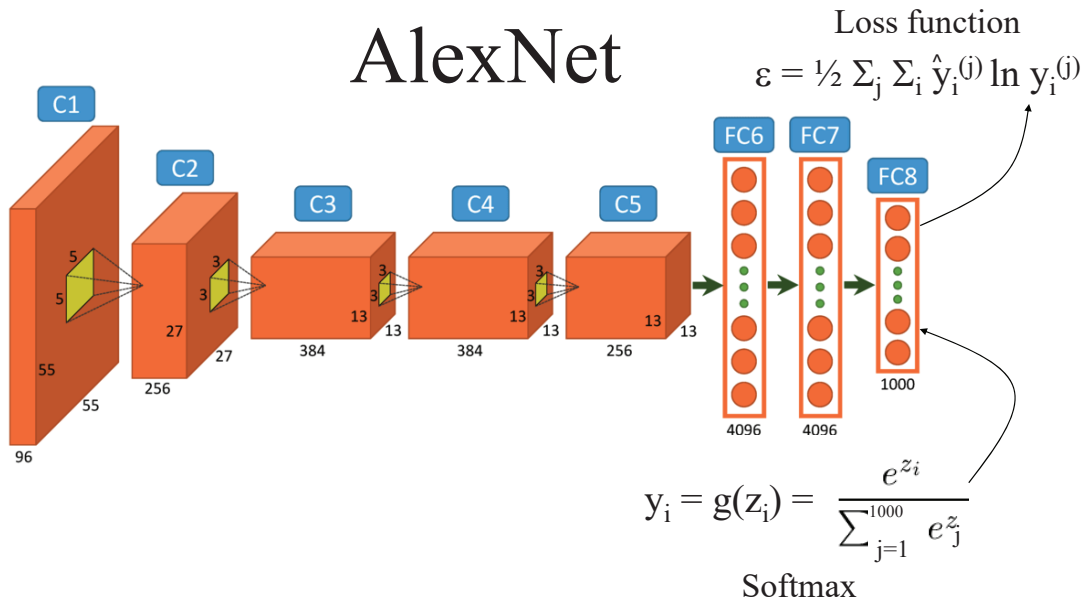


Figure 9.16: AlexNet architecture with a softmax function at the output and the cross-entropy loss function for a 1-out-of-1000 classifier. Figure adapted from <https://www.saagie.com/blog/object-detection-part1>.



a linear mapping according to He et al. (2016). He et al. (2016) shows skip layers every two layers of a standard conv-net network.

ResNet-152 introduced the concept of residual learning in which the subtraction of features is learned from the input of that layer by using shortcut connections (directly connecting input of  $n^{th}$  layer to some  $(n+x)^{th}$  layer, which is shown as curved arrow). It has proven that the residual learning can improve the performance of model training, especially when the model has deep network with more than 20 layers, and also resolve the problem of degrading accuracy in deep networks. There are variations of ResNet with a different number of layers, but the most used is ResNet50, which consists of 50 layers with weights. The ResNet also contains up to 5 times fewer parameters than a VGG of equivalent depth because it replaces the last few layers with layers denoted as GlobalAveragingLayers (Ramon, 2020).

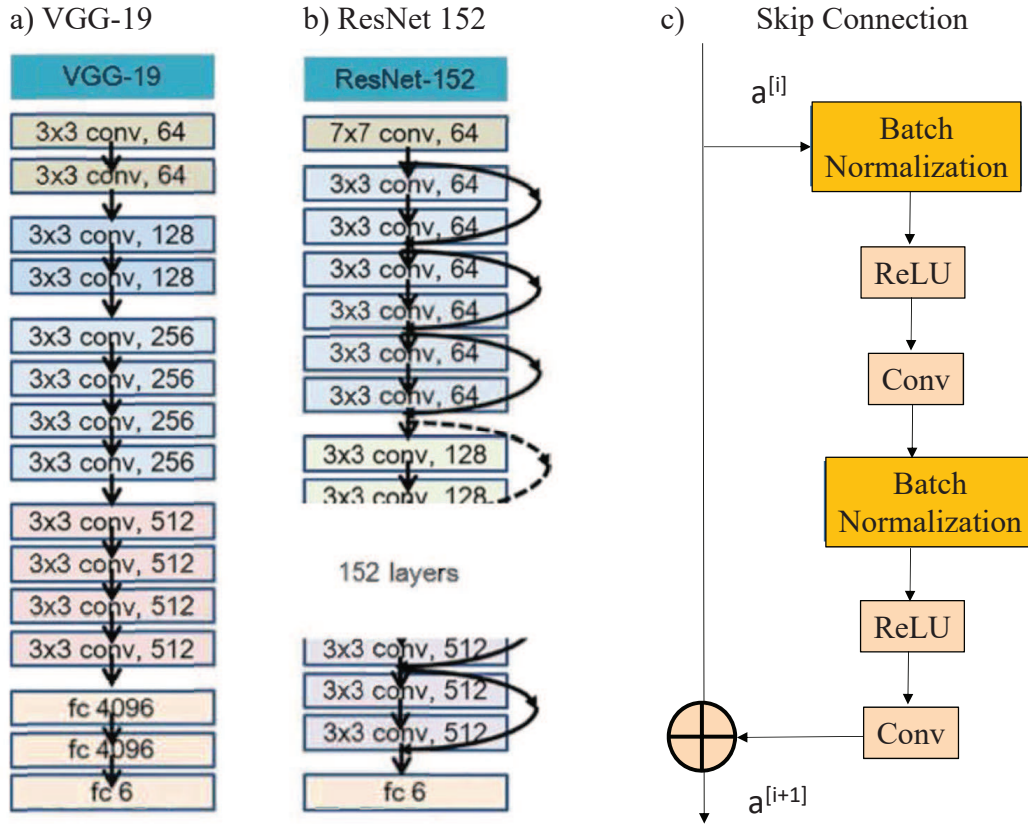


Figure 9.18: Architectures for a) VGG-19, b) ResNet and a c) skip connection. Figure adapted from He et al. (2016).

A dramatic demonstration of how the skip connection smooths the loss function around the minimizer is illustrated by Li et al. (2018a) in Figure 9.19. The loss function is plotted in two dimensions near the minimizer point for two types of deep networks: a 110-layer network without skip connections in Figure 9.19a and the 121-layer network with skip connections in Figure 9.19b. It is obvious that the skip connections lead to a smoother loss landscape in Figure 9.19b so that the convergence rate will be faster than the one without skip connections. This is similar to the effect of regularization on an ill-conditioned system of equations: the long-narrow valleys of the misfit

function tend to become more round and lead to faster convergence (see Chapter 2) and there is less of a tendency for overfitting (see Figure 4.27).

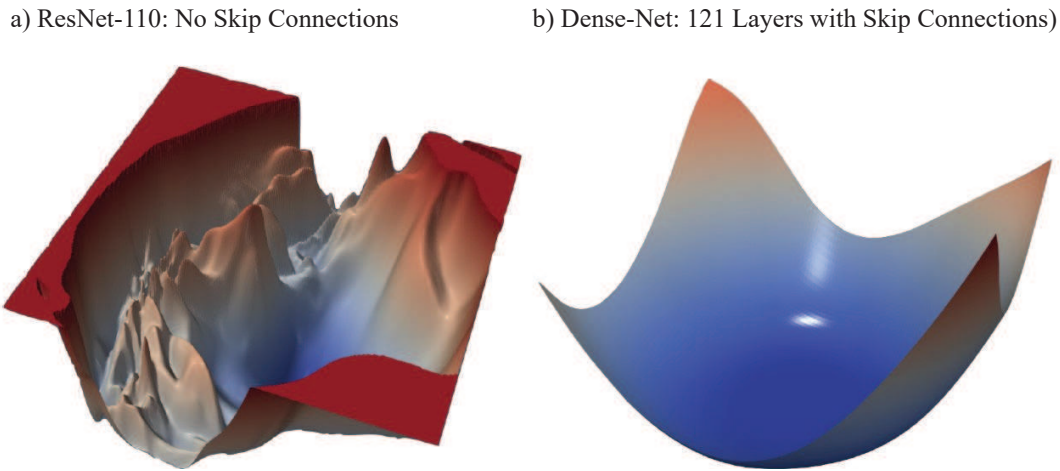


Figure 9.19: 2D Loss functions for a) ResNet-110 without skip connections and , b) DenseNet-121 with skip connections. Figure adapted from Li et al. (2018a). The origin of the coordinate system is defined as the model weight vector  $\mathbf{w}^0$  that minimizes the loss function, and the two coordinate axis are obtained by choosing two random directions  $\mathbf{w}^1$  and  $\mathbf{w}^2$  from the origin in the higher dimensional space of weights. The points along the plane defined by  $\mathbf{w}^1 - \mathbf{w}^0$  and  $\mathbf{w}^2 - \mathbf{w}^0$  are used to evaluate the loss function as plotted above. Image adapted from Li et al. (2018a).

Other networks that are popular are Xception (Chollet, 2016) and Inception (Szegedy et al., 2014). The Xception breaks up the 2D convolutions into concatenations of 1D convolutions, which reduces computation time. The Inception strategy is to use concatenated filters of different sizes to extract features with different scales.

Which CNN architecture should one use and should one try to innovate with new ones? The answer according to Hinton, Karpathy and Ng is to "not be a hero". Use one of the latest ones that work best with the ILSVRC contest, perhaps delete some parts. Karpathy says to play with the regularization strength and dropout rates.

## 9.4 Deep Learning Software

Karpathy, former Stanford Professor and currently AI manager at Tesla, is reported to have a high regard for Pytorch (<https://pytorch.org/tutorials/>), which powers the self-driving capabilities of the Tesla vehicles. To quote an article from *analyticsindiamag.com*<sup>10</sup>: *When it comes to machine learning frameworks, TensorFlow and PyTorch are widely popular with the practitioners. No other framework comes even remotely close to what these two products of Google and Facebook respectively, have in store. These frameworks have slowly found their niche within the AI community. PyTorch, especially has become the go-to framework for machine learning researchers..*

<sup>10</sup><https://analyticsindiamag.com/tesla-pytorch-self-driving-computer-vision-karpathy-elon-musk-ai/>

There are a number of CNN software packages, and Keras (<https://keras.io/> or <https://www.datacamp.com/community/tutorials/deep-learning-python>) is one of the easiest to use, similar to Python. It is a high level language based on the software Torch/Tensorflow and Pytorch. Once you master Torch, you can develop your own special functions. For some reason Karpathy doesn't list Caffe as one of his top three favorites, but he notes its nice features. An overview of Caffe is at <https://caffe.berkeleyvision.org/tutorial/>. He doesn't list Theano or Lasagne as his favorites.

What about MATLAB? A free 30-day license is at <https://www.mathworks.com/products/get-matlab.html>.

A systematic search for the optimal hyperparameters is always recommended. This goal can be achieved more easily using the software at github<sup>11</sup> and the keras site<sup>12</sup>.

I also recommend Andrew Ng's excellent Youtube videos on Deep Learning, too many to list. Andrew is a master of ML and its exposition, who deeply understands the benefits of sipping water, rather than guzzling it.

## 9.5 Seismic Fault Detection by CNN

Araya-Polo et al. (2017), Huang et al. (2017), Guitton (2018), Xiong et al. (2018), Shi et al. (2019), Wu et al. (2019a, 2019b), Wu et al. (2020a, 2020b), Qi et al. (2020), Gao et al. (2021), Li et al. (2021), Shi et al. (2021b), Wrona et al. (2021) and many others used CNN architectures to detect different geologic signatures in seismic sections, such as faults, seismic horizons, salt flanks, river channels and other geologic bodies. Instead of training on field data, Shi et al. (2019, 2021b), Gao et al. (2021), Wu et al. (2019a, 2019b), Wu et al. (2020a, 2020b), Li et al. (2021) and others trained the CNN on synthetic data. This has the benefit of not having to label the faults since the fault locations are already known by defining the synthetic fault model. However, the potential drawback is that synthetics sometimes do not provide a fully realistic rendering of a fault seen in seismic sections, especially if the training synthetics do not undergo a realistic 3D prestack migration. Thus, some subtle fault signals might be missed by the CNN detector. However, efficient transfer training on the real data set such as that employed in Shi et al. (2021a) can be used to smarten advanced versions of the CNN model.

As an example of CNN fault detection, Xiong et al. (2018) used a CNN system to identify faults in seismic data. Unlike some of the previous attempts that used features manually labeled or extracted from the seismic data as input (Zhang et al., 2014), Xiong et al. (2018) used three slices from the migration image as the input (see Figure 9.20) and used an automatically labeled attribute image as the output image (see Figure 9.20). In the attribute image the faults were automatically labeled by a coherency method.

Xiong et al.'s (2018) schematic for the CNN architecture is shown in the middle of Figure 9.20 and some examples from the training set are given in Figure 9.21. The target output values in the training set at each point  $O$  can be interpreted as faults or not by interpreters or auto-picking algorithms. In their paper, they classified eight different 3D seismic image cubes of real data using a coherence-based method where faults are roughly delineated (Qi et al., 2017). After annotation, every point in the training cubes is labeled as either a fault or non-fault using a threshold strategy: the points with coherence values larger than a predefined threshold are classified as faults, otherwise they are labeled as non-faults.

Before the three slices are fed into the network, they are normalized by subtracting the mean and dividing by the standard deviation. Similar to the classical CIFAR-10 classification problem (Krizhevsky and Hinton, 2009; Google, 2017), the network consists of two convolutional (Conv) layers and two fully-connected (FC) layers followed by a softmax classifier, which gives the predicted labels as the output (Bishop, 2006). The Conv layers both have 64 filters with sizes of  $5 \times 5$ . There

<sup>11</sup>[github.com/keras-team/keras-tuner](https://github.com/keras-team/keras-tuner) Documentation

<sup>12</sup><https://keras-team.github.io/keras-tuner/>

are 384 and 192 nodes in the two FC layers, respectively. After every Conv layer and FC layer, they applied the ReLU activation function. A  $2 \times 2$  max-pooling with a stride of 2 and batch normalization are used after both Conv layers. The final softmax classifier produces a probability indicating the likelihood of a fault being present in the center point of the input. The prediction of the label is then generated by applying a threshold (0.5) to the output likelihood values, such that any probability value greater than the threshold value is labeled as a fault, otherwise it is not a fault.

The network is trained from scratch by initializing the weights of all the layers as in the CIFAR-10 tutorial of Tensorflow (Google, 2017; Abadi et al., 2016). A gradient descent optimizer is implemented in Tensorflow with the default parameters and the initial learning rate, i.e. step length, being 0.1. The training samples are randomly shuffled before they are fed into the network for every epoch, which is a critical step in order to obtain good network performance. The model is saved when the loss function

$$\epsilon = \frac{1}{n} \sum_{i=1}^n \tilde{y}_i \ln y_i, \quad (9.11)$$

reaches a plateau during the optimization process, and it is then validated with the validation dataset. Here,  $y_i$  is the  $i^{th}$  predicted output from the softmax function and  $\tilde{y}_i$  is the actual label for the  $i^{th}$  example. The saved model achieved classification accuracies of about 73% on both the training and validation datasets.

The weights for the trained model are applied to the testing cube comprising a 3D seismic image volume of real data. The cube size in gridpoints is  $1000 \times 655 \times 1083$ . The top, front and side panels show, respectively, the time section ( $T = 312$ ), inline section ( $Y = 423$ ) and cross-line section ( $X = 417$ ). The locations of the sections are marked by the black lines in the figure. One of the eight cubes is randomly selected for validation and will not be seen by the network during the training process, while the other seven cubes are used to generate the training dataset. The training dataset is constructed by randomly selecting 50000 points from each cube labeled as faults (which is approximately 0.2% of all fault points in the cubes), and another 50000 points labeled non-fault. The number of non-fault points is much larger than fault points in the cubes. For this real data cube, the CNN obtains about 74% classification accuracy, which is close to that obtained for the training and validation datasets.

Figure 9.21 shows some representative fault and non-fault samples randomly selected from the training dataset. There appears to be a systematic difference between the two classes. Non-fault samples show more continuous seismic events. A validation dataset of the same size is constructed in the same way as the training dataset, which is used to monitor the training process and determine the termination of training.

The trained CNN model applied to the test cube consists of a 3D seismic image volume of real data as shown in Figure 9.22. The cube size is  $1000 \times 655 \times 1083$ . The top, front and side panels show the time ( $T=312$ ), inline ( $Y=423$ ) and cross-line ( $X=417$ ) sections, respectively. Locations of the sections are marked by the black lines in the figure. For this real data cube, the CNN obtains about 74% classification accuracy; close to that obtained for training and validation datasets.

The fault probability cube output by the CNN is shown in Figure 9.22a along with results from a seismic coherence cube (Figure 9.22b). Seismic coherence is a well-known and widely used attribute to highlight discontinuities in seismic image (Bahorich and Farmer, 1995). As we can be seen in Figures 9.22 and 9.23, the CNN results show a higher resolution compared to the coherence volume. The seismic faults as well as channels are clearer in the CNN results. Recalling that the fault probability calculation is made independently for different points, the clear continuous outlines of discontinuities in the fault probability images show that the trained network performs robustly in the presence of noise.

For the test with the 3D image cube of size  $1000 \times 655 \times 1083$ , it took about 2.5 hours to obtain



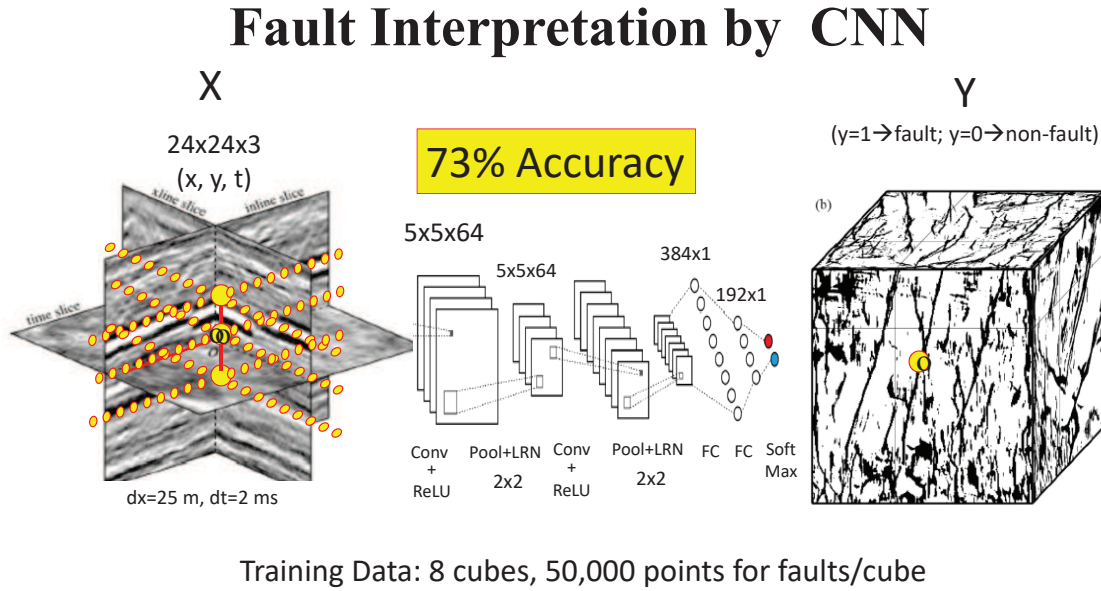


Figure 9.20: Three slices on the left are taken from a 3D migration cube and used as the input data to the CNN. The output is the point  $y_i$  at  $o$  on the right which is taken to be either a fault ( $y_i = 1$ ) or not ( $y_i = 0$ ). The  $24 \times 24 \times 3$  filter is shifted over to a new position  $o'$ , and the dot product of the filter with the image points is computed again. Repeating this for all input points gives the first feature map. Note, the filter is not shifted up or down and only shifted along the lateral directions for this training example. A new training example will have the center point of the filter shifted up or down. Figure adapted from Xiong et al. (2018).

the fault probability result using a computer cluster with 20 nodes (40 CPU cores in each node). So this method is still practical even without possible improvements by reducing duplicate calculations in adjacent locations.

## 9.6 Summary

The characteristics of a CNN and its associated Keras code are described. A key advantage of CNN over classical neural networks is that its convolutional nature leads to a significant increase in computational efficiency by more than several orders of magnitude. Empirical evidence suggests that it is more efficient at detecting localized patterns in input data because it roughly emulates processing procedures employed by a brain's neurons. The CNN uses a weighted sum of local pixel values to give the input to the  $j^{th}$  node of the next layer. The set of  $N$  weights  $w_i$  for  $i \in [1, 2 \in N]$  are shared for computing the input into any node of the next layer.

Another advantage of CNN is that it tends to be resilient to translations of the object of interest in the input image. This is not surprising because each row of the convolution matrix is a shifted copy of a neighboring row. In addition, CNNs tend to be more resilient to object rotations and object scaling as networks become deeper with more subsampling operations such as pooling.

An example is presented for CNN identification of faults from a migration image. The predicted faults appear to be more accurate than those interpreted from a coherency cube. This is a promising

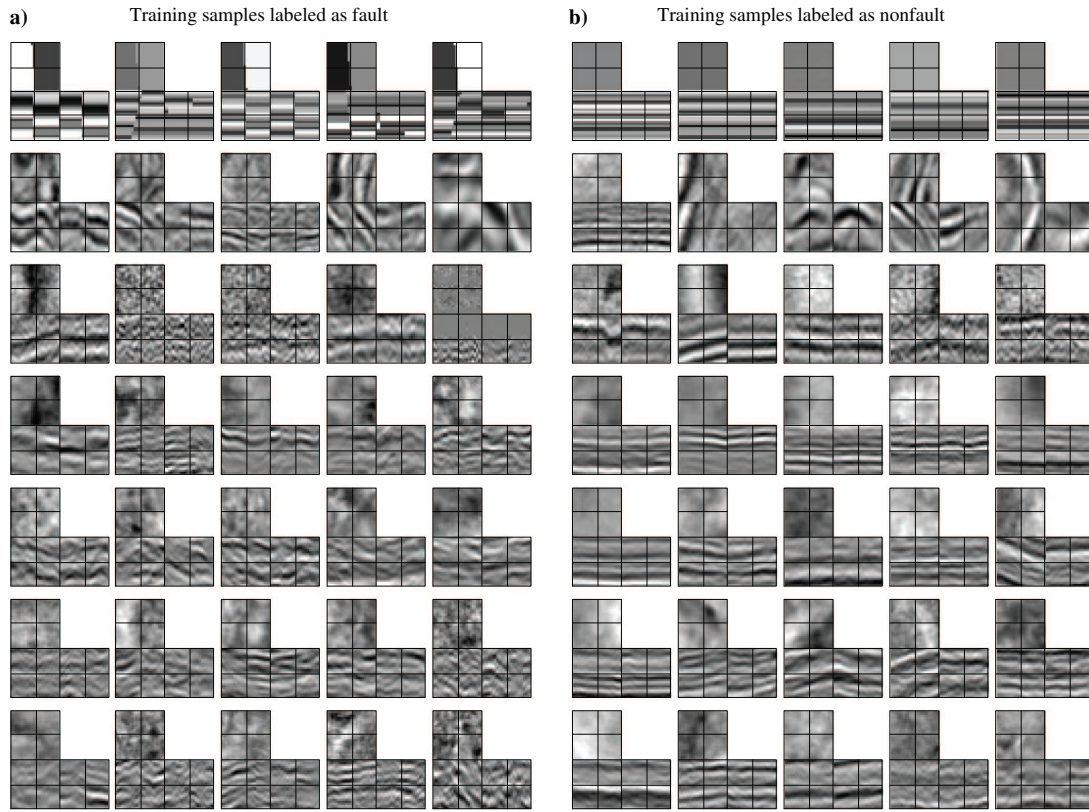


Figure 9.21: Samples labeled as (a) fault and (b) non-fault. All samples are randomly selected from the training dataset, except the first row showing synthetic samples. Each sample is composed of front, side and top panels which are inline, cross-line and time slices, respectively. The size of each slice is  $24 \times 24$ . The spatial grid size is 25 meters while the vertical time sampling interval is either 0.002 or 0.004 seconds for different cubes. Figure adapted from Y. Xiong et al. (2018).

result and gives hope that CNN's will be a useful tool for many aspects of exploration geophysics. However, the next chapter shows that U-Net architecture is the preferred semantic-segmentation CNN for salt-boundary picking and for fault identification. U-Net is the gold standard for precise semantic segmentation of an input image.

## 9.7 Exercises

1. Go to [#scrollTo=Lo3S63zGQ6Y7](https://colab.research.google.com/drive/1ZZmYiFuSnVIlczmJRa3ncaer2sSu1GQA?usp=sharing). Compare the convergence rate for a two-layer CNN with batch normalization to one with batch normalization. Which one is faster? Explain.
2. Same as Exercise 1, except deepen the network to 6 convolution layers.
3. Same as Exercise 1, except compare effects of dropout on convergence rate for a 6-layer network.

## Cubes of Data, CNN & Coherency Images

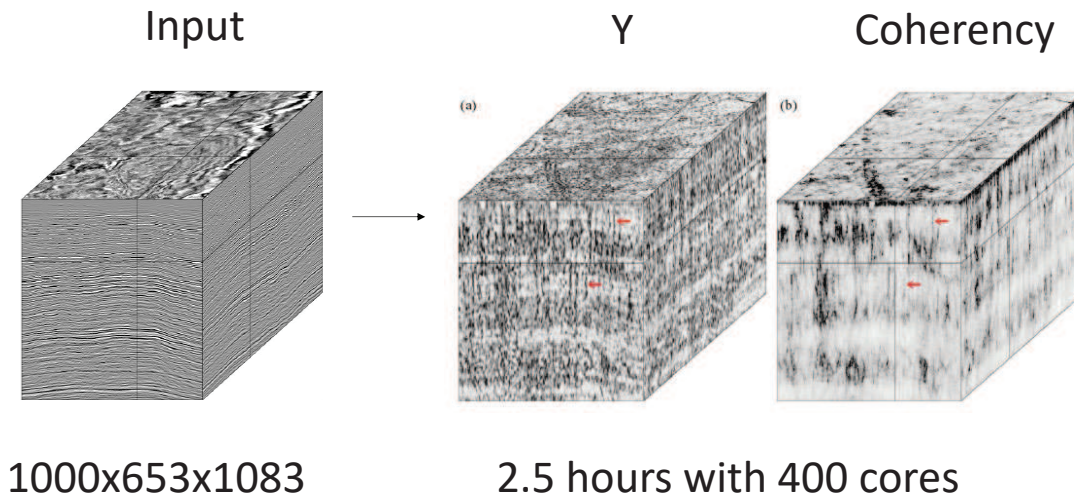


Figure 9.22: Real data example showing the fault probability cube from the CNN prediction, compared with the corresponding coherence cube. The cube size is  $1000 \times 655 \times 1083$ . The spatial grid size is 25 meters while the vertical time sampling interval is 0.002 seconds. Figure adapted from Xiong et al. (2018).

4. The first layer of a CNN supposedly decomposes an image into smaller components associated with edges oriented at certain angles. How can one use local damped regularization and local slant stacking to encourage the filters in the first layer to output FMs with edges oriented in different directions?
5. Adjust the code in Exercise 9.7.1 so that it uses the smart dropout procedure described in Frankel and Carbin (2019). Compare the accuracy of the CNN with smart dropout to the one without dropout. In this case, eliminate around 90% of the nodes.
6. Use the codes at <https://github.com/keras-team/keras-tuner> and <https://keras-team.github.io/keras-tuner/> to find the optimal hyperparameters for the previous problem.

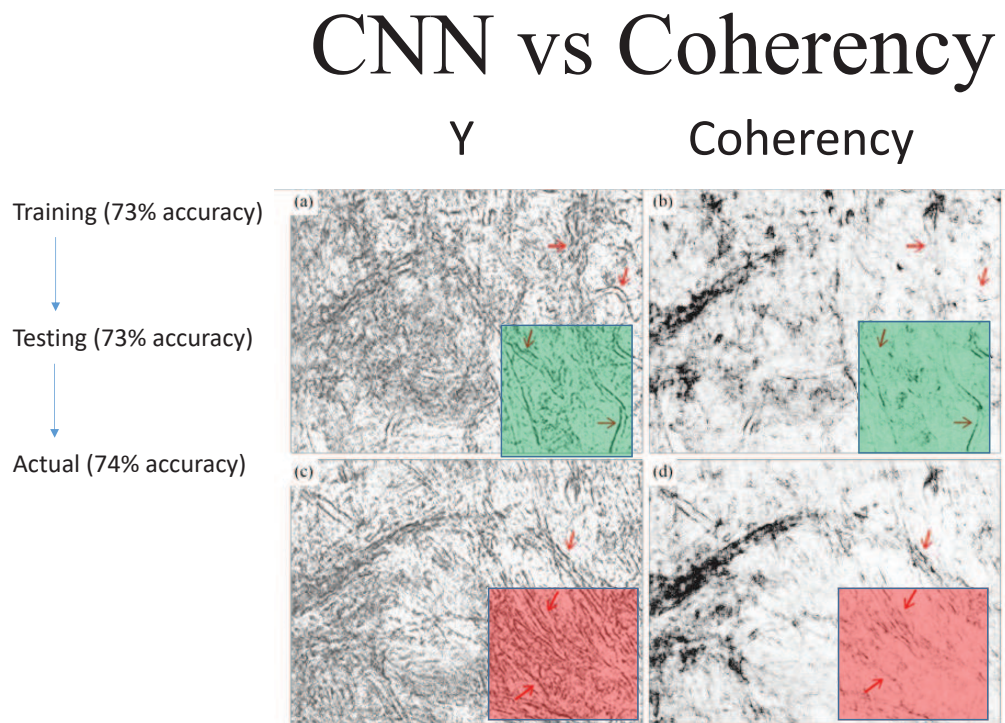


Figure 9.23: Time slices of fault probability [(a) and (c)] from CNN prediction, compared with seismic coherence [(b) and (d)], at two different time slices,  $T = 312$  (top row) and  $T = 387$  (bottom row), respectively. The section size is  $655 \times 1083$ . The spatial grid size is 25 meters. Figure adapted from Xiong et al. (2018).

## Chapter 10

# Object Identification by CNN

We now describe several tasks of computer vision for identifying objects in images: image classification, classification+localization, object detection, semantic segmentation, and instance segmentation. Of these categories, semantic segmentation, which classifies each pixel in an image, is one of the most important tools for classifying images in biomedicine. Examples are presented for CNN classification in seismic, optical and ground penetrating radar images. The CNN architectures of AlexNet, U-Net and Region-based CNN (R-CNN) are employed for processing images in this chapter.

### 10.1 Introduction

Interpreting huge volumes of images obtained by remote sensing devices can be time consuming, error prone and expensive. An example is medical diagnosis (Biswas et al., 2019) of images obtained by X-ray machines, CT scans, MRIs, PET scans or microscopes. Instead of a time-consuming and error-prone analysis of these images, CNN can quickly reach reliable conclusions that can be vetted by the expert. For example, CNN can be used to characterize plaques in arteries from CT scans, automatically segment ventricles from MRI images, detect breast cancer tumors in mammography images, detect cancerous cells in blood samples, and characterize lung disease in X-rays images (Biswas et al., 2019). Neural network methods have also been used for computer vision in many industries (Kivinen et al., 2014) including automatic detection of cracks in roads (Zhang et al., 2016), fault detection in seismic data (Zhang et al., 2014), and analysis of well log data (Zhang et al., 2018). Their impressive performances highlight the effectiveness of deep neural networks, which are likely to replace or complement the conventional manual interpretation of features (LeCun et al., 2015)

For exploration geophysics, seismic imaging of the subsurface often requires the recording of hundreds of terabytes of digital data and its processing into huge 3D volumes. Each pixel in the volume can have many attributes (Chopra and Marfurt, 2006) such as local dip, intensity, instantaneous phase and other features that might indicate faults, the type of lithology or the presence of oil. Successful interpretation sometimes requires combining attributes decided by cross plotting and the input from the experienced interpreter (Taner, 2001). Fortunately, neural networks can be used to assist the interpreter in analyzing large groups of attributes (AlRegib et al., 2018) and large volumes of seismic data.

Figure 10.1 illustrates six types of computer vision tasks for analyzing images based on increasing complexity in the image ([cs231n.stanford.edu/slides/2017/cs231n\\_2017\\_lecture11.pdf](https://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf)).

1. Image classification: Given an image, the computer outputs a discrete class, which is the main object in the image. For example, the image in Figure 10.1a is classified as a Karst image.

2. **Classification+Localization:** Identify an object, classify it and localize it. Typically the localization is accomplished by defining the smallest bounding box around it as in Figure 10.1b. This procedure assumes images with just one object.
3. **Object Detection:** For images with multiple objects of different types, classify and localize all objects in the image. Assign a class to each object and draw a bounding box around it as in Figure 10.1c.
4. **Semantic Segmentation:** Classify every pixel in the image. We do not know how many objects there are in the image, we only know the class of each pixel.
5. **Instance Segmentation:** Classify every pixel in the image; different objects of the same class will be assigned as a unique instance of the object. The number of instances with the same class will be equal to the number of objects in that same class.

This chapter now describes the methods for accomplishing the above computer vision tasks. The pixel-based classification scheme, also called semantic segmentation, will be used to interpret three types of geoscience images: seismic images, ground penetrating radar (GPR) data and camera photos of rock slices. For the rock photos and GPR data we use the simple AlexNet architecture (Long et al., 2015), and use the U-Net architecture for the seismic data. The U-Net is used because it is one of the most successful networks for identification of medical images (Biswas et al., 2019). The region-based CNN, that is R-CNN, will be used to localize, window and classify objects in different regions of the photo. The actual algorithm we use for an example is known as Faster R-CNN.

## 10.2 Image Classification

The three size scales for classifying the contents of an image are global, object-sized localization, and pixel-based.

### 10.2.1 Global Picture-size Classification

Classifying a picture as to whether it contains, e.g. a karst<sup>1</sup> in the photo of Figure 10.1a, is an example of global classification that can be accomplished by a properly trained neural network. Each picture of a, say, geological outcrop gives rise to equations of the form

$$\sigma(\mathbf{W}^{[N]}\sigma(\mathbf{W}^{[N-1]}(\dots(\mathbf{W}^{[1]}\mathbf{z}^{[0]})\dots))) = y, \quad (10.1)$$

where  $\sigma$  denotes an activation function,  $\mathbf{z}^{[0]}$  is the input image, and  $\mathbf{W}^{[n]}$  represents the convolutional matrix of filter coefficients in the  $n^{th}$  CNN layer as well as any max-pooling and batch normalization operations. Here,  $y = 1$  for the presence of a, say karst, and  $y = 0$  otherwise. Typically, there are many thousands of unknown matrix coefficients so we need many more thousands of pictures in the training set to properly train the CNN network.

### 10.2.2 Object Detection: Localize and Classify a Single Object in an Image

We assume a single dominant object in an image. Localizing an object with a bounding box, e.g. the green box in Figure 10.1b, and identifying the class of the bounded object is denoted as localized

---

<sup>1</sup>Karst is a topography formed from the dissolution of soluble rocks such as limestone, dolomite, or gypsum. It is characterized by underground drainage systems with sinkholes and caves (<https://en.wikipedia.org/wiki/Karst>).



## Summary: Object, Instance and Semantic Segmentations

a) Classification 1 Object



Karst

b) Classify+Localize: 1 Object



Karst

c) Semantic Segmentation



Karsts

d) Instance Segmentation



Karst1, Karst2

e) Object Detection: Classify+Detect many Objs.



Car, Karst

Figure 10.1: Six types of computer vision tasks ([cs231n.stanford.edu/slides/2017/cs231n\\_2017\\_lecture11.pdf](https://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf)). Photo courtesy of Sherif Hanafy.

## Bounding Box Coordinates & Class

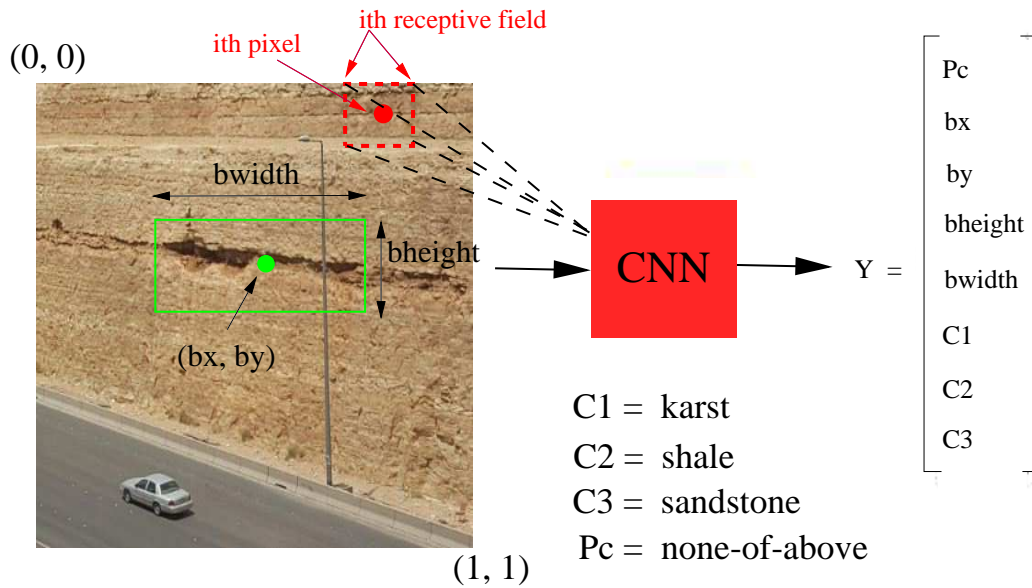


Figure 10.2: Photo of the road cut as the input image and the desired output is the  $8 \times 1$  target vector  $\mathbf{y} = (p_c, b_x, b_y, b_{height}, b_{width}, c_1, c_2, c_3)$ . For a sufficient number of training examples the CNN should learn how to enclose objects of interest with boxes. Photo courtesy of Sherif Hanafy.



windowing and classification. For training, we define the coordinates of the smallest bounding box<sup>2</sup> which enclose the object as shown by the green box in Figure 10.2. The scale of the input photo is normalized so that the upper left corner is at the origin  $(0, 0)$  while the lower right corner is at  $(1, 1)$ . The center-point coordinates  $(b_x, b_y)$  and the width and height of the red box are given, respectively, by the terms<sup>3</sup> in

$$(b_x, b_y, b_{width}, b_{height}). \quad (10.2)$$

The optimal values of these coordinates define the bounding box with the smallest area, which is a regression problem.

This is also a classification problem if the object inside the box is to be classified. For this example, if the object is either a karst ( $c_1 = 1$ ), a shale ( $c_2 = 1$ ) or limestone ( $c_3 = 1$ ) in the bounding box (BB) then  $p_c = 1$ , otherwise  $p_c = 0$ . The rock class  $c_i$  is a one-out-of-three classification scheme so that there is, in the groundtruth labeling, only a single element equal to one for one of the three elements  $(c_1, c_2, c_3)$  in the classification part of the target vector  $\mathbf{y}$ , and the remaining elements are zero. Predicting this rock classification vector demands a softmax activation function at the output of the last layer. The  $p_c$  is a metric which is used to eliminate a proposed bounding box if  $p_c \approx 0$ .

Therefore, the output of the CNN model is the  $8 \times 1$  target vector shown on the right of Figure 10.2. This is a pixel-based classification-regression problem so that there will be an equation of constraint for each pixel in the input image:

$$\sigma(\mathbf{W}^{[N]})\sigma(\mathbf{W}^{[N-1]}(\dots(\mathbf{W}^{[1]}\mathbf{z}^{[0]})\dots))_k = \mathbf{y}_k, \quad (10.3)$$

where  $\mathbf{y}_k$  is the target vector for the  $k^{th}$  pixel and the convolution filter in the first layer is centered at the  $k^{th}$  pixel of the input image. This is a mixed regression-classification problem because half the  $8 \times 1$  target vector is for classification and the other half is for the coordinate values of the BB. Each input picture might contain more than a million pixels, so there are an equal number of equations of constraint for each picture.

The four coordinates in equation 10.2 and the classification vector are collected into one target vector  $\mathbf{y}$  as shown in Figure 10.2. The training set consists of  $N$  pictures, each with a target vector  $\mathbf{y}^{(i)}$  for  $i = [1, 2, \dots, N]$ , and the objective function  $\epsilon$  for one bounding box can be the following:

$$\begin{aligned} \epsilon = & \frac{1}{2} \sum_{i=1}^N [(b_x^{(i)} - \tilde{b}_x^{(i)})^2 + (b_y^{(i)} - \tilde{b}_y^{(i)})^2 + (b_{height}^{(i)} - \tilde{b}_{height}^{(i)})^2 + (b_{width}^{(i)} - \tilde{b}_{width}^{(i)})^2] + \\ & -\lambda \sum_{i=1}^N \ln(p(\tilde{\mathbf{y}}^{(i)})), \end{aligned} \quad (10.4)$$

where the tilde indicates the predicted data and  $\lambda$  is a term that balances the regression and classification parts of the objective function. If there is more than one proposed BB then there is a summation over the different bounding boxes. In practice for R-CNN, the coordinate residuals can be defined by defining a *learnable* (Girshick et al., 2016) transformation that maps a proposed BB to a ground-truth BB.

The probability  $p(f_k)$  is computed by the softmax operation  $p(f_k) = e^{f_k} / \sum_{k'=1}^C e^{f_{k'}}$ , where  $f_k$  is the network's input into the softmax function at the correct rock class channel denoted as  $k$ . To reduce the sensitivity to outliers and exploding gradients, the  $L_2$  loss for the coordinates (Girshick et al., 2016) is replaced by an  $L^1$  loss function for the Fast RNN (Girshick, 2015). The number of classes is equal to  $C$ .

<sup>2</sup>Labeling in this example means both assigning a class and defining the coordinates of the smallest bounding box (BB) for the object in image.

<sup>3</sup>Alternatively, the coordinates of the upper-left and lower-right corners of the ROI can be specified.

## Intersection of Union

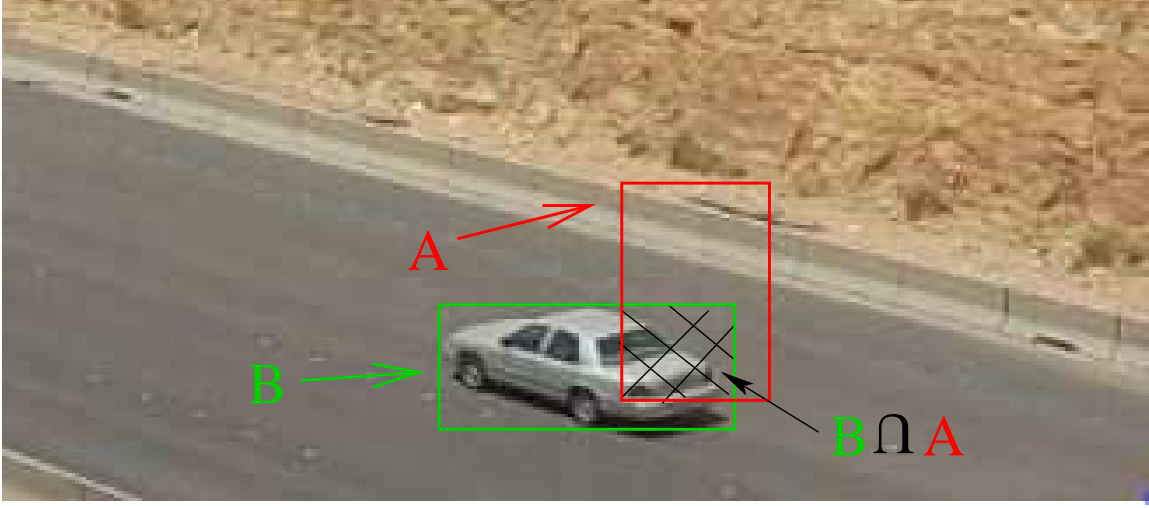


Figure 10.3: Proposed and actual BBs in red and green, respectively. The intersection area between A and B is given by the cross-hatched area, and the IOU is given by equation 10.5.

### 10.2.3 Judging the Accuracy of the Predicted BB: IOU

Training a network to find the bounding boxes of objects suggests a need for quantitatively judging the accuracy of the coordinates  $(\tilde{b}_x^{(i)}, \tilde{b}_y^{(i)}, \tilde{b}_{width}^{(i)}, \tilde{b}_{height}^{(i)})$  for our proposed bounding box (BB). This accuracy score is the sum of squared coordinate residuals in equation 10.4, which is not very effective when the starting BB has a much different size or aspect ratio compared to the actual BB. A better BB metric that tells us how close the initial BB is to the actual one is the intersection-of-union (IOU) defined below.

Assume that your starting BB is the red one in Figure 10.3 with area  $A$ , and the groundtruth BB is the green one with area  $B$ . The IOU formula is defined as

$$IOU = \frac{A \cap B}{A \cup B}, \quad (10.5)$$

where IOU measures the similarity of two boxes, or the overlap of the two boxes. If  $IOU = 1$  then the boxes perfectly coincide with each other so that the starting BB is perfect. Smaller IOUs indicate less accuracy in the proposed BB. The IOU can be used as a more effective loss function for training the classifier, and typically an  $IOU > 0.7$  is considered to be an effective BB (<https://www.youtube.com/watch?v=ANlzQ5G-XPE>). The IOU is especially effective in estimating starting BBs for the selective search strategy discussed in Appendix 10.17.

### 10.2.4 Object Detection: Localize and Classify Multiple Objects in an Image

If an image contains many objects then finding the coordinates of the smallest bounding box for each instance of an object and classifying it is a much more difficult task than the detection+localization of a single dominant object. Without knowing the number of objects, how do we define the number of BB coordinates per image in equation 10.4? For example, Figure 10.1e has

	Mean Average Precision	Frame per second
Detection Framework	mAP	FPS
Faster RCNN - <u>VGG16</u>	73.2	7
Faster RCNN - ResNet	76.4	5
YOLO	63.4	45
SSD 500	76.8	19
YOLO v2 ( <u>416x416</u> image size)	76.8	67
YOLO v2 (480x480 image size)	77.8	59

Figure 10.4: Comparison of run times (right column) and accuracy (left column) for R-CNN and YOLO methods. Table from Redmon and Farhadi (2016).

three objects (two karsts and a car) and each requires finding a minimally sized bounding box and a class type. This task is even more challenging if the objects are located in shadows or are occluded by other objects. There are two types of CNN solutions to this problem: region-based CNN (R-CNN) and you-only-look-once (Redmon et al., 2015). Compared to R-CNN which analyzes distant regions separately from one another at each iteration, you-only-look-once (YOLO) simultaneously analyzes all parts of the image for objects at each iteration. According to <https://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/>, YOLO is very fast and accurate, and some tests show that YOLO2 is almost an order-of-magnitude faster than Faster R-CNN (see Figure 10.4 and Ren et al. (2016)). YOLO.v4 (Bochkovskiy et al., 2020; Anka, 2020) is optimized to efficiently train and run on a single GPU using small mini-batch sizes.

**R-CNN.** Region-based CNN methods are denoted as R-CNN (Girshick et al., 2015), Fast R-CNN (Girshick, 2015; Girshick et al., 2016) and Faster R-CNN (Ren et al., 2016). They were developed to localize with BBs and classify multiple objects in a single image. The R-CNN algorithm relies on the strategy illustrated in Figure 10.5.

- Create a set of rectangular windows that approximately locate all possible objects in the input image; these windows are known as region proposals or *regions of interest (ROIs)*. These ROIs can be determined by a classic computer vision algorithm, such as an edge detector, a shape detector or selective search (see Appendix 10.17). A few thousand ROI's can sometimes be generated for an input image (<https://deepsense.ai/region-of-interest-pooling-explained/>), and the selective search strategy of Uijlings et al. (2013) is the one described in Appendix 10.17.
- The intersection of the input image with the proposed ROI is defined as an ROI's sub-image. Each sub-image is warped into a  $J \times J$  image, where  $J$  is an integer smaller than the smallest width or height of an ROI. For example,  $J = 7$  but the object in this coarsened representation should still be recognizable. These sub-images lead to more efficient computations by the

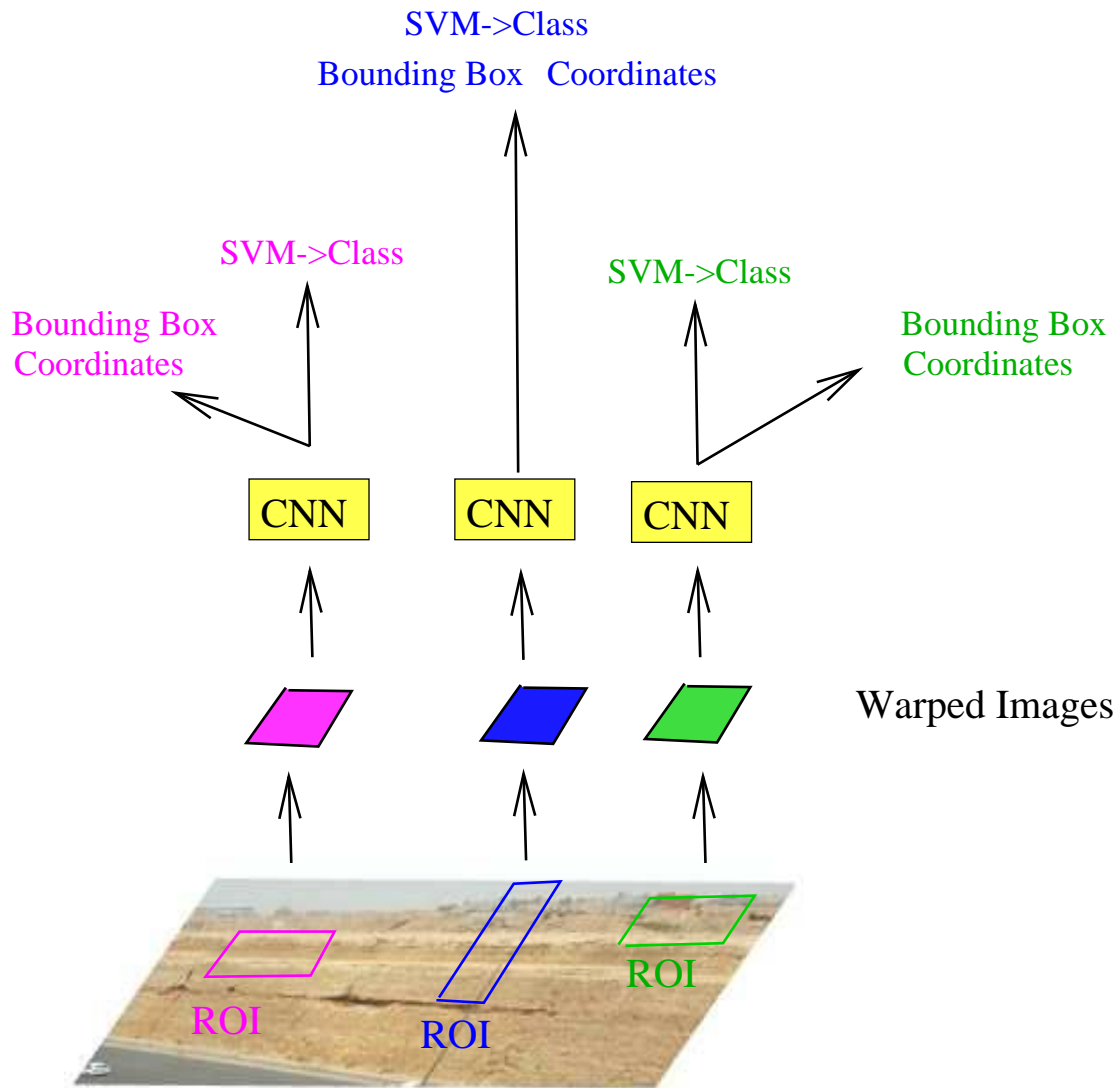


Figure 10.5: R-CNN workflow, where regions of interest (ROIs) in the input image are selected by some classical computer vision method.

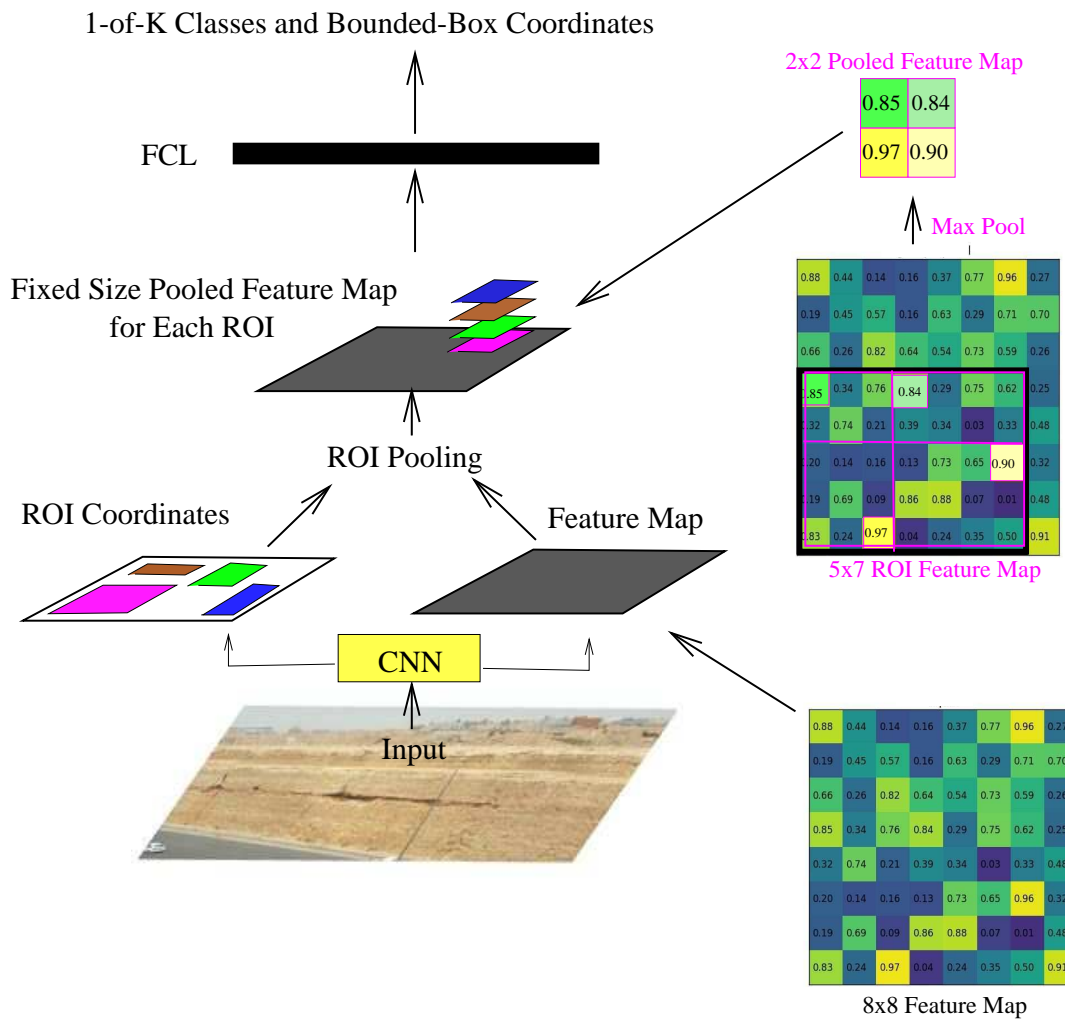


Figure 10.6: Fast R-CNN workflow. Illustration adapted from <http://deepsense.ai/region-of-interest-pooling-explained/> and Girschick (2015).

CNN.

- Each warped image is input into a separate SVM to classify the objects and the background in the associated ROI.
- Once the objects are classified in each ROI, then a linear regression model is used to estimate the coordinates of the bounding box for each object in a ROI.

The problem with R-CNN is that it can be very time consuming because each CNN and SVM is used to detect, find the bounding box coordinates and classify objects in the corresponding ROI. The inefficiency arises because of these many multiple steps and "...from the fact that each ROI may have a very large receptive field, often spanning the entire input image. Since the forward pass must process the entire receptive field, the training inputs are large (often the entire image)." (Girschick, 2015).

**Fast R-CNN.** To speed up R-CNN, the Fast R-CNN algorithm (Girschick, 2015; Girschick et al.,

2015; Girshick et al., 2016) passes the entire image into a *single* CNN which generates regions of interest. At the same time it extracts feature maps from the ROIs, pools them into smaller images, and then classifies objects and estimates the coordinates of their bounding boxes. The Fast R-CNN workflow is illustrated in Figure 10.6, and consists of the following steps.

1. **Assume there are  $N$  specified rectangular ROIs.** The coordinates of these ROIs can be determined by a selective search procedure. The key properties of the ROIs are described by a  $N \times 5$  matrix. The first column of the  $N \times 5$  matrix represents the index numbers associated with each ROI and the remaining four columns contain the coordinates of the upper-left corner and the height and width of the ROI.
2. **Feature Map Generation.** The image is input into the deep ConvNet, and the output consists of the ROIs and the feature maps illustrated by the gray "Feature Map" rectangle in Figure 10.6. Each specified ROI region in a feature map is saved and denoted as a ROI feature map, as illustrated by the  $5 \times 7$  black rectangle on the right of Figure 10.6.
3. **Pooled Feature Maps.** The ROI feature maps are then reduced in size by pooling, as denoted by the  $2 \times 2$  pooled feature map in Figure 10.6. These reduced-size maps will be denoted as pooled feature maps.
4. **Fully-Connected Neural Network Layer.** The pooled feature maps are then flattened into a tall vector, which is then input into a sequence of FCLs.
5. **Output.** There are two types of outputs from the FCLs: one is a regression output that refines the 4 bounding box coordinates of each bounding box and the other produces softmax probability estimates of the  $K$  classes of objects, as well as a catch-all "background" class denoted by the element  $p_c$  in Figure 10.2.
6. **Faster R-CNN.** Faster R-CNN (Ren et al., 2016) separately finds the ROIs with a Region Proposal Network (RPN). The feature maps are first extracted from the input image using CNN and then these feature maps are inserted into the RPN, which returns object proposals or ROIs. The feature maps in each ROI are reduced to the same small size by pooling. These pooled feature maps are then passed to a sequence of FCLs to classify the objects and estimate the coordinates of the bounding boxes. A good tutorial on Faster RNN is at <https://tryolabs.com/blog/2018/01/18/faster-r-cnn-down-the-rabbit-hole-of-modern-object-detection/>.

Training the Fast R-CNN can be much faster than the more conventional R-CNN, and Faster R-CNN is faster than any of them.

### You-Only-Look-Once (YOLO)

A very fast and popular object detection method is YOLO, or you-only-look-once. As indicated in Figure 10.4, YOLO can be both faster and more accurate than the R-CNN methods. YOLO partly avoids false positives because it sees the complete image at once as opposed to R-CNN methods which only analyze the generated region proposals. However, a YOLO limitation is that it typically only predicts one type of class in one grid, so it struggles with very small objects (<https://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/>). A partial remedy for this is to divide the input picture into smaller cells, but a longer runtime is the price you pay.

There are four main steps for training and prediction of a YOLO network.

1. *Divide the image into  $2 \times 3$  cells, assign an anchor box to each cell and compute the class of an object ( $c_i$  for  $i \in \{1, 2, 3\}$ ) in each cell, as depicted in Figure 10.7a.* Here, the groundtruth is that there are two types of objects, i.e. soccer players and open grass areas, each surrounded by the dashed bounding boxes with the appropriate color. For labeling, assign the coordinates of a dashed BB around each object and also assign the center-point coordinates to each BB

as well as the object's class. For Figure 10.7a, the colored disks denote the center points of the objects, and a cell cannot contain the center point of more than one object.

2. Define the target tensor with dimension  $2 \times 3 \times 8$  as depicted in Figure 10.7b. Each cell proposes  $N_a$  possible anchor boxes, aka proposed BBs, so there are a total of  $S \times N_a$  possible BBs (see Figure 10.7b) where  $S$  is the number of cells. At most, only one of these BBs per cell is selected at the end of training, and only if it contains an object's center point<sup>4</sup>.
3. Specify  $N_a$  anchor boxes/cell. Two or more anchor boxes/cell are typically assigned in order to possibly achieve faster convergence. It is hoped that the shape of one of the proposed anchor boxes is nearly the same shape as the groundtruth BB for that cell, which will promote fast convergence.

For the Figure 10.7c example, two anchor boxes per cell are proposed so each cell has a  $16 \times 1$  predicted target vector  $\mathbf{y}$ . The elements in  $\mathbf{y}$  consist of the six predicted class labels, the predicted anchor box coordinates and a confidence factor  $p_c$  for each anchor box; the first 8 elements  $(p_c, b_x, b_y, b_{width}, b_{height}, c_1, c_2, c_3)$  in the  $16 \times 1$  vector  $\mathbf{y}$  are for the first proposed anchor box. The remaining 15 elements are for the second anchor box in that cell. The confidence factor  $p_c$

$$p_c = Pr(Object) \times IOU, \quad (10.6)$$

for each anchor box indicates the confidence value  $0 \leq p_c \leq 1$  we have of an object in that BB and its overlap, if any, with the groundtruth BB. According to Dharmi (2019), only one anchor box associated with a cell having a center point is retained: the one with the highest IOU, while the other anchor boxes for that center point cell are discarded. This is known as non-max suppression where the IOU can only be used during training, not prediction of an unlabeled input picture. Non-max suppression ensures that the YOLO algorithm detects each object only once in the training phase.

The term  $Pr(Object)$  is the softmax output in the classifier portion of the predicted target vector  $\mathbf{y}$ . If there is no groundtruth object in the parent cell of the anchor box then the groundtruth  $Pr(Object) = 0$  and the coordinates for this anchor box are ignored during training. Eliminating anchor boxes with a predicted  $p_c$  below a threshold value is how we reduce the number of anchor boxes.

4. Training the CNN with a large labeled data set finds the weights that can localize (i.e., predict the BB coordinates of the actual objects) and classify objects in each BB. The input pictures are labeled with the BB coordinates of the objects, their class values, and the confidence values of an object's center point in the cell. The detection network has 24 convolutional layers followed by two FC layers as illustrated in Redmon (2015). The CNN weights are found by a gradient descent method with the YOLO loss function given by the summation over  $S=6$

---

<sup>4</sup>In Redmon's original paper, each of the anchor boxes for one cell shared the same set of class variables  $c_i$  for  $i \in \{1, 2, \dots, N_c\}$  where  $N_c$  is the number of class variables.

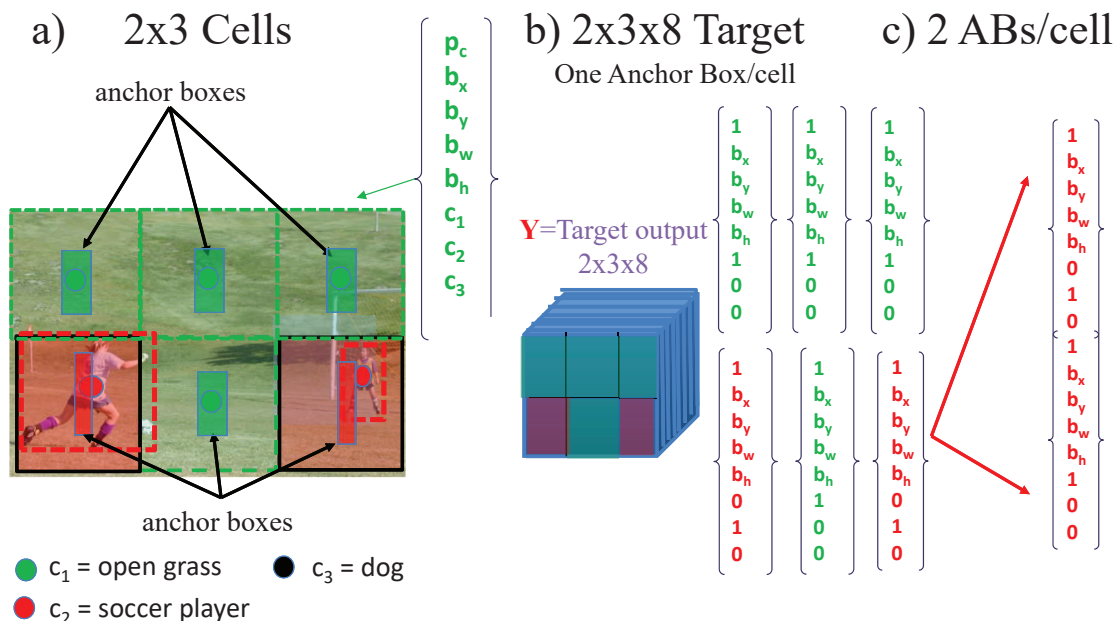


Figure 10.7: a) Image divided into 6 cells, with one anchor box/cell, and the colored disks are the center points of different objects. The red disks are the center points for the soccer-player object while the green disks are the center points for open-grass areas. The bounding boxes surrounding the soccer players (open grass) are denoted by dashed red (green) lines. There is an  $8 \times 1$  target vector/cell. The cell size is usually selected to be small enough so that there is only one type of object in the cell. b) The  $2 \times 3 \times 8$  target tensor is depicted as the multi-colored cube, where there is an  $8 \times 1$  target vector/cell; one for each of the 6 cells. c) If two anchor boxes/cell are proposed then the target tensor will have the dimensions  $2 \times 3 \times 16$ . The elements of a cell's  $16 \times 1$  target vector consist of three possible classes, 2 confidence factors  $p_c$  and two sets of anchor box coordinates for the two proposed anchor boxes/cell. The confidence values  $p_c$  for many of the proposed BBs will be low so all but two of the final anchor boxes for soccer players will be used and will coincide with the dashed groundtruth BBs denoted by dashed red rectangles in a).



cells for the Figure 10.7 example:

$$\lambda_{coord} \sum_{i=0}^S \sum_{j=0}^B \mathbb{L}_{ij}^{obj} [(b_x^{(i)} - \tilde{b}_x^{(i)})^2 + (b_y^{(i)} - \tilde{b}_y^{(i)})^2] \quad (10.7)$$

$$\begin{aligned} &+ \lambda_{coord} \sum_{i=0}^S \sum_{j=0}^B \mathbb{L}_{ij}^{obj} [(\sqrt{b_{width}^{(i)}} - \sqrt{\tilde{b}_{width}^{(i)}})^2 + (\sqrt{b_{height}^{(i)}} - \sqrt{\tilde{b}_{height}^{(i)}})^2] \\ &+ \sum_{i=0}^S \sum_{j=0}^B \mathbb{L}_{ij}^{obj} (c_i - \tilde{c}_i)^2 + \lambda_{noobj} \sum_{i=0}^S \sum_{j=0}^B \mathbb{L}_{ij}^{noobj} (c_i - \tilde{c}_i)^2 \\ &+ \sum_{i=0}^S \mathbb{L}_i^{obj} \sum_{c \in classes} (p_c^{(i)} - \tilde{p}_c^{(i)})^2, \end{aligned} \quad (10.8)$$

where tilde symbol denotes the predicted target variables, and  $\mathbb{L}_{ij}^{obj}$  takes the value of 1 if there is an object's center point in the  $i^{th}$  cell that is also contained by the  $j^{th}$  anchor box for that cell. Otherwise  $\mathbb{L}_{ij}^{obj} = 0$ .

If there is no object's center point in a cell then the proposed anchor-box coordinates for that cell are not computed during training, but the  $\tilde{c}_i$ s for that anchor box will still be computed because the term multiplying  $\lambda_{noobj}$  in equation 10.8 will be non-zero.

The term  $p_c^{(i)}$  is the groundtruth confidence factor in equation 10.6 for the  $i^{th}$  cell. Here,  $\mathbb{L}_i^{obj} = 1$  denotes if an object's center point is in the  $i^{th}$  cell, otherwise  $\mathbb{L}_i^{obj} = 0$ . At the end of training, hopefully the predicted  $\tilde{c}_i$ 's and  $\tilde{p}_c^{(i)}$  will match the actual ones.

As Redmon (2015) states: *Note that the loss function only penalizes classification error if an object is present in that grid cell.... It also only penalizes bounding box coordinate error if that predictor is "responsible" for the ground truth box (i.e. has the highest IOU of any predictor in that grid cell).* He also states that the absence of objects in cells can lead to model instability. *To remedy this, we increase the loss from bounding box coordinate predictions and decrease the loss from confidence predictions for boxes that don't contain objects. We use two parameters,  $\lambda_{coord}$  and  $\lambda_{noobj}$  to accomplish this. We set  $\lambda_{coord} = 5$  and  $\lambda_{noobj} = 5$ .*

The details for localization and classification are explained in <https://www.analyticsvidhya.com/blog/2018/12/practical-guide-object-detection-yolo-framework-python/>. Excellent tutorial videos by Andrew Ng are at [https://www.youtube.com/watch?v=9s\\_FpMpdYW8](https://www.youtube.com/watch?v=9s_FpMpdYW8) and [https://www.youtube.com/watch?v=9s\\_FpMpdYW8&list=PLkDaE6sCZn6Gl29AoE31iwdVwSG-KnDzF&index=30](https://www.youtube.com/watch?v=9s_FpMpdYW8&list=PLkDaE6sCZn6Gl29AoE31iwdVwSG-KnDzF&index=30).

Andrew Ng's tutorials note that Redmon's (2015) paper is very difficult to understand, partly because the terms in equation 10.8 are not clearly or completely defined. For example, the coordinates of the anchor boxes should be indexed not only by the parent cell index  $i$  but also by the anchor box index  $j$  for that cell. Apparently, the bounding box index is silent in the formulas and only one anchor box is used at the final iteration. The concept of each groundtruth object having a center point (CP) is not clearly explained, but later publications describe the use of the CP coordinates as an indicator of whether the coordinates of an anchor box are computed during training.

5. *Prediction.* The trained model is applied to unlabeled data and the relevant anchor boxes and classifications are computed. IOUs are not available because the groundtruth is not known, so  $\tilde{p}_c$  and  $\tilde{c}_i$  are computed by the network for all the anchor boxes. If two anchor boxes originate from the same cell, then non-max suppression is used to choose the anchor box with the highest value of  $p_c$ . Anchor boxes with  $p_c$  below a certain threshold are eliminated. This

highlights a limitation of the original YOLO algorithm: there can only be one center point of an actual object for any one cell.

### 10.2.5 Pixel-size Classification: Semantic Segmentation

If each pixel in a photo is classified then this is denoted as semantic segmentation (<https://www.topbots.com/semantic-segmentation-guide/>). This type of classification scheme uses a sliding window approach where the receptive field (see red-dashed box in Figure 10.2) of the 1st-layer's convolution operator is centered at the  $i^{th}$  pixel. The  $i^{th}$  lag value of the convolution operator give rise to the  $i^{th}$  equation of constraint denoted as

$$\sigma(\mathbf{W}^{[N]}\sigma(\mathbf{W}^{[N-1]}(\dots(\mathbf{W}^{[1]}\mathbf{z}^{[0]})\dots))_i = \mathbf{y}_i \text{ for } i \in \{1, 2, \dots, N^2\}, \quad (10.9)$$

for an  $N \times N$  input image. Here,  $\mathbf{y}_i$  is the multinomial classification vector for the  $i^{th}$  lag value associated with the  $i^{th}$  receptive field. Therefore, an  $N \times N$  image can theoretically give rise to as many as  $N^2$  equations of constraint. This can be computationally burdensome so a stride of more than one can be used to reduce computational expense at the expense of less spatial resolution in the classification image.

Semantic segmentation has a wide variety of uses, including interpretation of x-rays or MRI scans for anomalies, self-driving cars that require knowledge of where every object of interest is located in real time, and geological interpretation of seismic images or optical photos such as Figure 10.1. Figure 10.1d is an example of *instance segmentation* which treats multiple objects of the same class as distinct individual objects (or instances). This compares to semantic segmentation which treats multiple objects of the same class as a single entity. Typically, instance segmentation is more difficult to accurately achieve than semantic segmentation. Chapter 11 presents an example of detecting cracks in photos of a rock face by semantic segmentation.

## 10.3 Labeling Tools

Labeling photos is the most time-consuming part of preparing the supervised data for training. If a GAN (Generative Adversarial Network) described in Chapter 18 is insufficient then efficient labeling software is needed. Photoshop can click on a few points of an object and fill in the object with a uniform color. It can also connect points with an irregular line that follows a contour of intensity, or values of maximum pixel intensity. MATLAB has a labeling tool called *Imagelabeler* at <https://www.mathworks.com/help/vision/ug/label-pixels-for-semantic-segmentation.html>. In Figure 10.8, a few mouse clicks on the border of a rock formation fills in the rock formation with a uniform color.

GIMP (<https://www.gimp.org/downloads/>) and Inkscape are both useful as imaging tools not to be confused with data analysis tools that have strong imaging components. GIMP is primarily useful for raster images, Inkscape SVG (Scalable Vector Graphics) is primarily useful for vectors such as polygonal outlines and undulating text strings although the path function in GIMP is also vector based. GIMP vectors are useful for identifying components to operate on just like selection by color or color threshold.

Figure 10.9 shows a screen dump of the GIMP interface displaying a picture of cracks on a dry lake bed and after labeling of the cracks with a color of orange, otherwise the color is black. When saving paths in either program, the transform from SVG format and Bézier-curves to some other program input is not as easy as using simple raster I/O. In GIMP, selection and save as raster is useful if the control points are not needed. Note that Photoshop, like GIMP, also has path selection tools but taking data in/out of Adobe products might not be as useful as using common tools like GIMP. GIMP is open source software and your own labeling software can be imported into it. It is the favorite labeling program for some professional labelers and we used it for labeling the drone photos shown in Chapter 11.

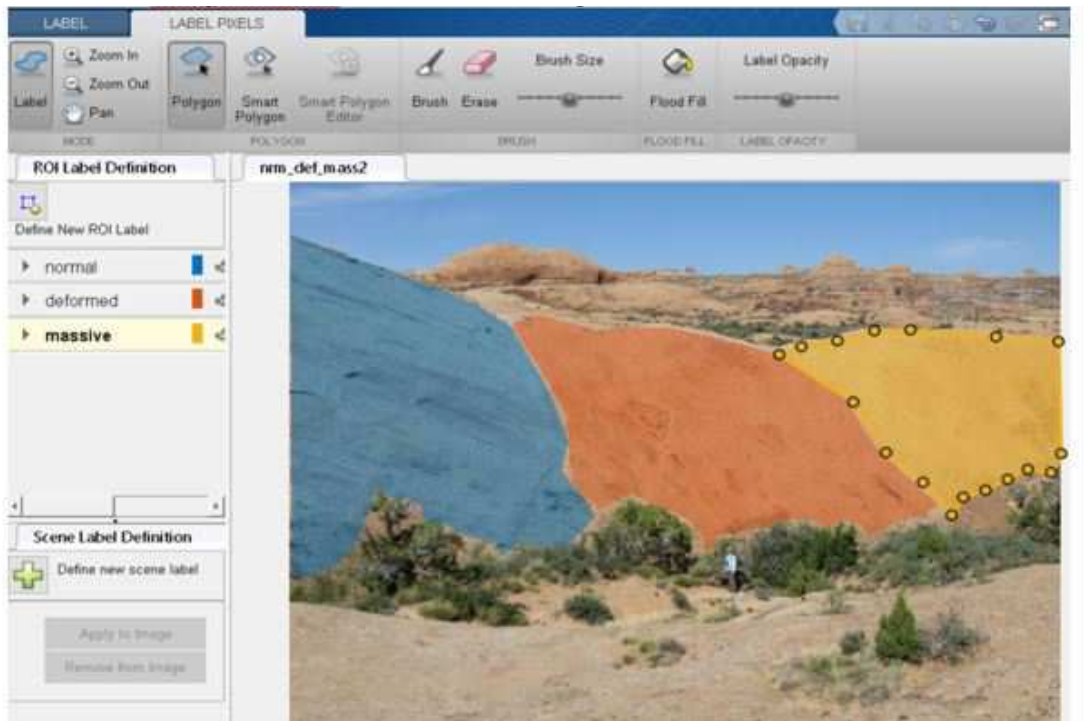


Figure 10.8: Screen dump of MATLAB Imagelabeler tool.



Figure 10.9: Screen dump of GIMP GUI used for labeling images of cracks in a dried-out lake bed. Image courtesy of Yunsong Huang.

Table 10.1: CNN Parameters for Salt Body Classification

Image Size	Image Patch Size	Number of Channels	Conv. Patch Size	Number of Conv. Layers	Size of Max Pool	Number of Iterations
$586 \times 740$	$39 \times 39$	6	$2 \times 2$	3	$2 \times 2$	50

Another labeling tool is *SymFD v1.1* which uses a wavelet transform to efficiently detect edges, ridges, and cracks in an image. It consists of a GUI with tunable parameters that can be interactively set to detect cracks with a specified range of crack widths, lengths, and intensity values. The download instructions are below.

1. Download SymFD.zip from <http://www.math.uni-bremen.de/cda/software.html>
2. Extract the folders.
3. Open MATLAB→Home→Set Path.
4. Add with subfolders→Load all v1.1 folders.
5. Save.

and the execution steps are

1. Load image.
2. Pick any generator, adjust values of the parameter alpha, orientation operator, and minimal contrast. The shear features include ridges and lines. Only display negative values in the edge-detected image.
3. Detect features and export them.

In the next sections we demonstrate the use of semantic segmentation in seismic migration images, GPR data and optical photos.

## 10.4 Salt Body Detection by AlexNet

A 2D seismic migration image (Yilmaz, 2001) is shown in Figure 10.10a. In seismic imaging of salt basins, identifying and picking salt boundaries is the first step for velocity model building for oil exploration (Yilmaz, 2001). Therefore, the goal is to use the CNN with the AlexNet architecture in Figure 10.11 to automatically identify the boundary of the salt body without using time-intensive manual picking.

We will first use a monoparameter CNN method to pick the salt boundary, and then use a multiparameter CNN method where important salt features at each pixel are also used as input data. The multiparameter features will be the local dip angle in Figure 10.10b and the instantaneous frequency in Figure 10.10c (Yilmaz, 2001) at each pixel. These parameters are selected because they appear to uniquely distinguish the seismic properties in the salt from those outside. Approximately 700  $39 \times 39$  positive patches (i.e.  $y = 1$ ) and 700 negative patches (i.e.  $y = 0$ ) of the same size are picked manually in the images. These input patches are interpolated to be  $32 \times 32$  patches. Around 80% of the patches are used as the training set and the others are used as the validation set. The CNN parameters for salt body classification are listed in Table 10.1.

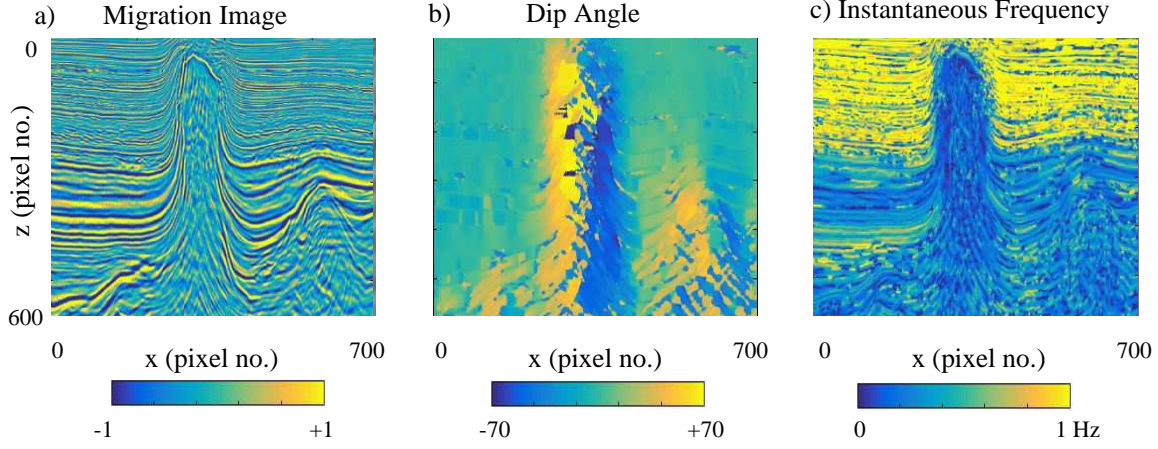


Figure 10.10: a) Seismic migration, b) local-dip angle and c) instantaneous frequency images. Scales are normalized.

#### 10.4.1 AlexNet CNN Architecture

Figure 10.11 depicts the AlexNet architecture, where semantic segmentation is used to predict the label of each pixel. The network is composed of three convolution (Conv) layers and two FC layers followed by a softmax classifier. There are 6 convolution filters with size  $2 \times 2$  in the first Conv layer. The number of channels is doubled in the deeper convolution layers. Rectified linear activation (ReLU) and a  $2 \times 2$  max-pooling are applied after each Conv layer. There are 128 nodes in the first FC layer and we dropout 50% of them in the second FC layer. The softmax classifier gives the probability of the center point being a reflection event or not. The probability of two classifiers are compared and the one with the higher probability is selected as the prediction label. All convolutional filter kernel elements are trained from the data in a supervised fashion by learning from the labeled set of training examples. A training set  $S = (\mathbf{x}^{(i)}, y^{(i)})$  is given which contains  $n$   $32 \times 32$  image patches, where  $\mathbf{x}^{(i)}$  represents the  $i^{th}$  image patch and  $y^{(i)}$  is the class label of either  $y^{(i)} = 1$  or  $y^{(i)} = 0$ . The cross-entropy cost function is given by

$$L = -\frac{1}{N} \sum_{i=1}^N \ln p(Y = y^{(i)} | \mathbf{x}^{(i)}), \quad (10.10)$$

where  $p(Y = y^{(i)} | \mathbf{x}^{(i)})$  is the probability that the label of  $\mathbf{x}^{(i)}$  is  $y^{(i)}$ .

#### 10.4.2 Numerical Results

The results of monoparameter and multiparameter CNN applied to the seismic data are now presented.

#### 10.4.3 Monoparameter CNN

The architecture of the monoparameter CNN is illustrated in Figure 10.11a and only the migration image is used as the input data. The inference accuracies of the training and the validation data sets are 96.5% and 95.0%, respectively. The predicted CNN labels in the migration image are shown in Figure 10.12a. The points in yellow and blue denote the predicted salt body and the non-salt sediments, respectively. The salt classification has been mostly accurate, except around the

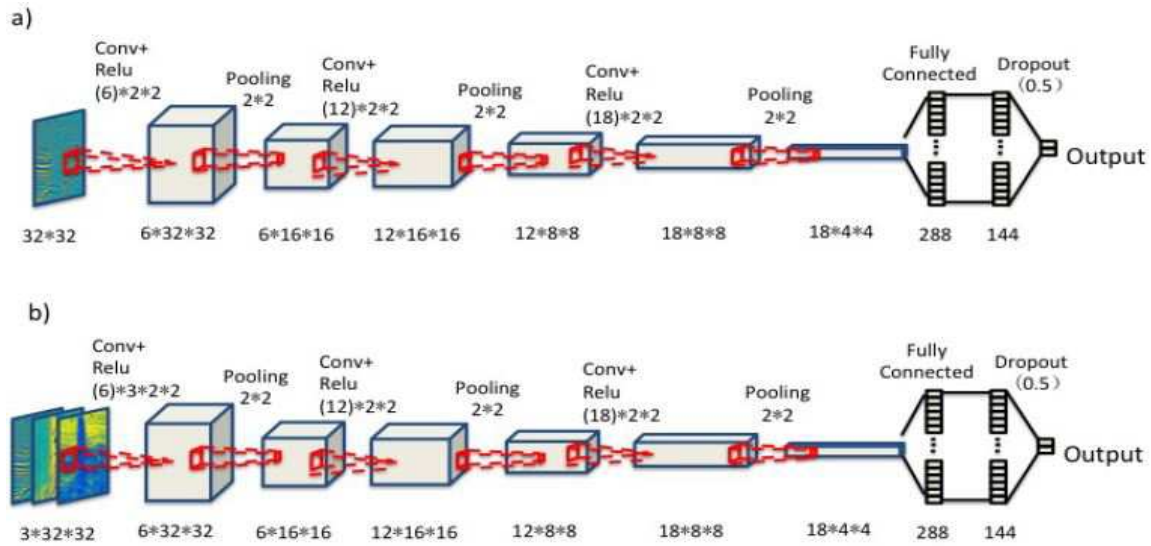


Figure 10.11: Illustration of a) the monoparameter CNN architecture and b) the multiparameter CNN architecture.

boundary between the salt and sediments.

#### 10.4.4 Multiparameter CNN

To improve the prediction accuracy around the salt-sediment boundary we now use a multiparameter CNN. The input data for the CNN architecture in Figure 10.11b consist of the migration image as well as the images with dip angle and instantaneous frequency. After training and testing, the accuracy for both the training and validation data sets is over 98%.

Figure 10.12b shows class predictions from the multiparameter CNN. The misclassifications around the salt body have been greatly reduced and the salt boundary is acceptably accurate in many locations. This suggests that adding more physical features such as dip and instantaneous frequency in the input might noticeably improve the performance of CNN. However, there is still a blurry resolution of the salt body boundary. This problem will be mitigated by the U-Net architecture that will be introduced in sections 10.8 and 10.9.

### 10.5 Detection of Rock Cracks by AlexNet

Detecting the density and orientation of rock cracks in boreholes is an important means for identifying breakout zones, permeable zones and lithology. For example, a borehole televiewer tool (see Figure 10.13a) can be used to produce panoramic images of the borehole wall, where the cracks correspond to the dark horizontal features in Figure 10.13b.

However, the problem with determining crack orientations and density is that, for large data sets, it often involves labor-intensive detection efforts that can take days to complete. In this case it is important to automate the crack-detection process. As an example, Figure 10.14a-10.14b show the photos of a cracked rock in grayscale and RGB, respectively. Either the gray scale or RGB images can be used as input to classify each pixel as either a crack ( $y=1$ ) or not a crack ( $y=0$ ).

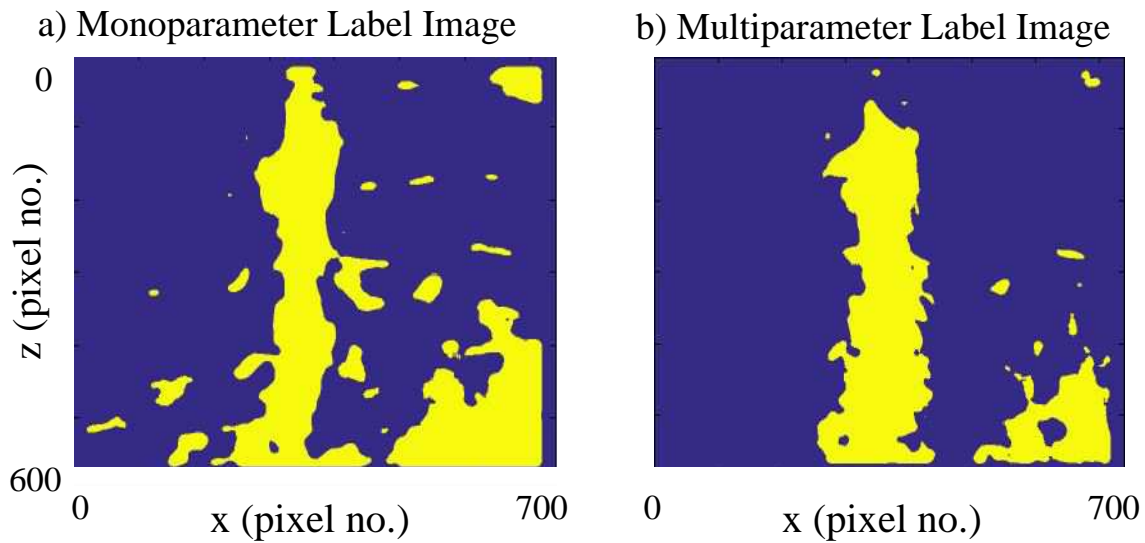


Figure 10.12: The labeled images obtained from the a) monoparameter CNN and b) multiparameter CNN.

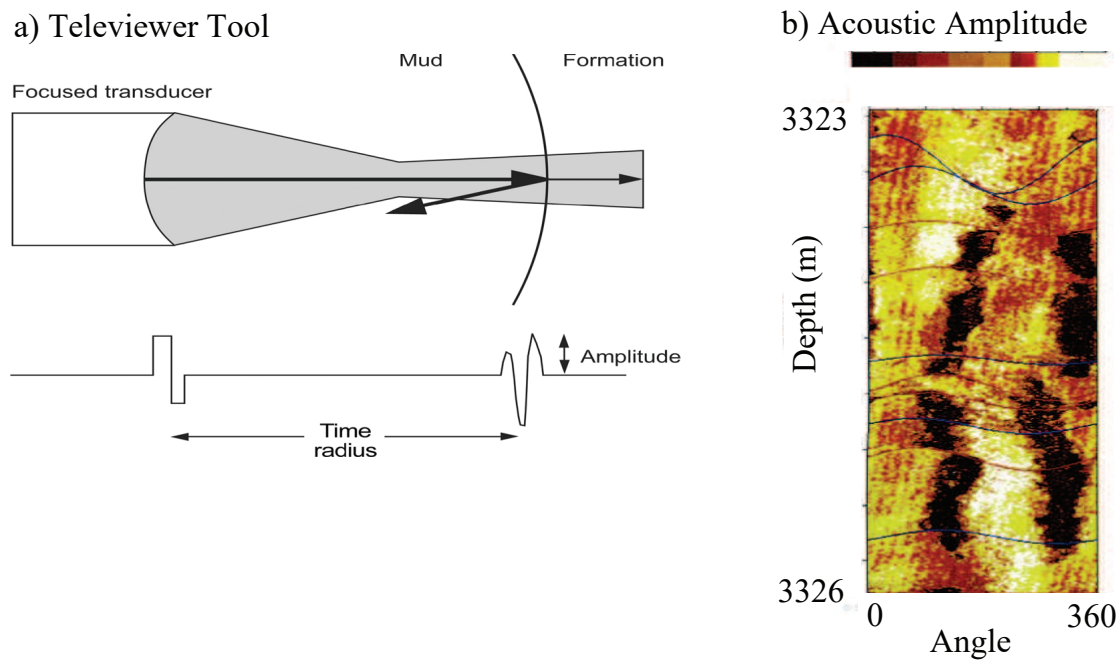


Figure 10.13: a) Ultrasonic borehole imager (UBI). The UBI measures reflection amplitudes of acoustic waves and estimates the radial distance by using traveltimes and a direct measurement of mud velocity. b) Example of breakout detection using an ultrasonic borehole televiewer. Breakouts are indicated by the low acoustic amplitude of the reflected signal, shown here as darker areas. The breakouts are rotated because of a drilling-induced slippage of localized faults. Image and caption from [www.petrowiki.org/Borehole\\_imaging](http://www.petrowiki.org/Borehole_imaging).



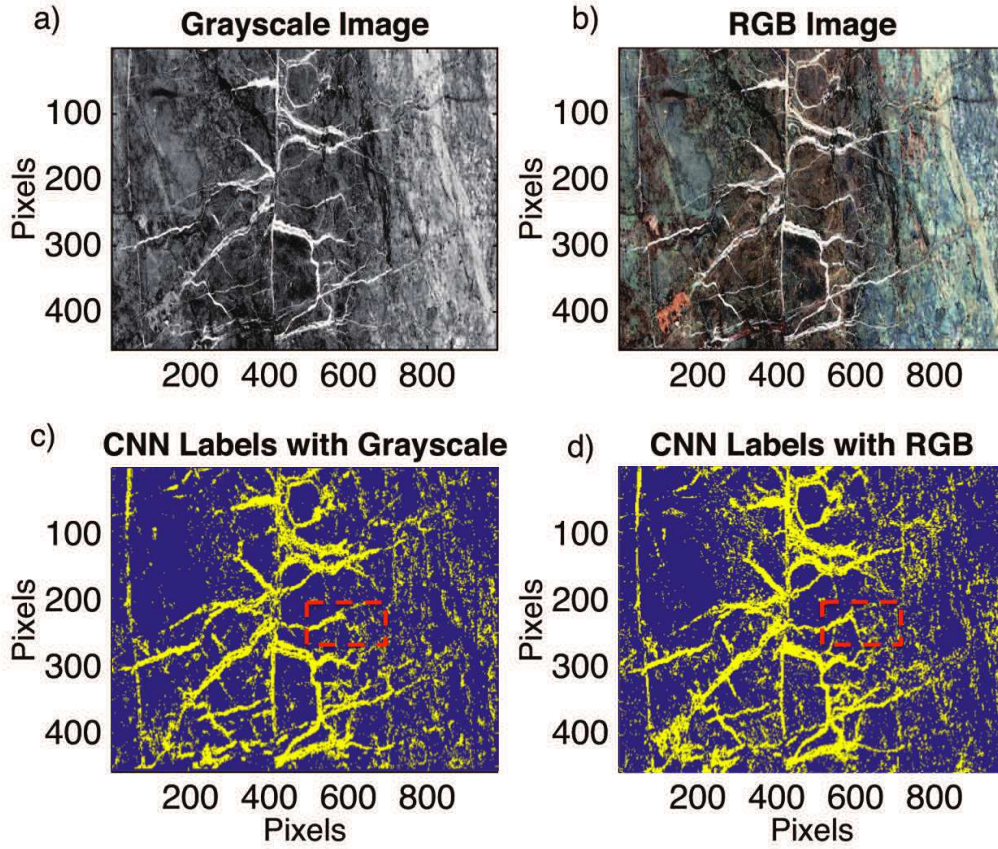


Figure 10.14: a) Grayscale and b) RGB images of the rock cracks. c) and d) are the CNN labeled images where yellow is the color of a crack.

### 10.5.1 Unbalanced Crack Data

In many cases, cracks represent a small fraction of the occupied space in a photo. This means that for any photo the number of semantic segmentation equations with  $y = 1$  is small compared to the number of equations for  $y = 0$ . This will give rise to a severely unbalanced set of equations that can lead to slow convergence in training and limitations in accurately labeling cracks. To overcome this problem, the density of cracks can be increased by extracting cracks from the images and artificially inserting them into crack images. This can produce an augmented data set that is well balanced with a nearly equal number of training equations for both cracks and background. Alternatively, a GAN method can be used to generate cracks (Hu, 2019) and can also generate both artificial pictures with both cracks and labels (<https://www.frontiersin.org/articles/10.3389/frobt.2018.00066/full>), or each equation with a crack can be heavily weighted while the background equations are downweighted (Lin et al., 2018 and <https://www.topbots.com/semanticsegmentationguide/>). This last approach is used for detecting cracks in Chapter 11.

### 10.5.2 Numerical Results

The AlexNet architecture in Figure 10.15 will be used for rock crack identification and its parameters are listed in Table 10.2. For training, 400  $17 \times 17$  positive patches ( $y = 1$ ) and 400



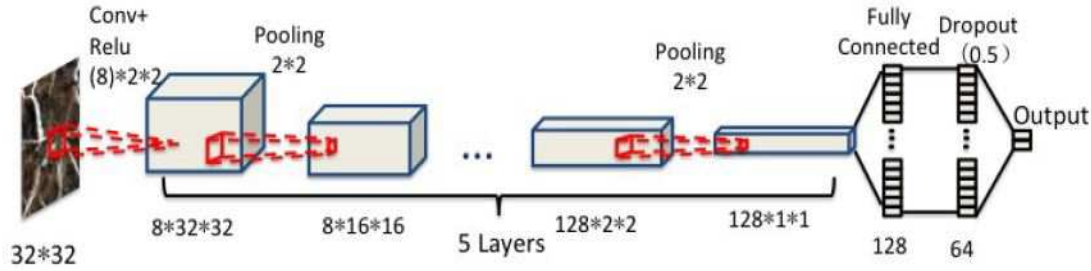


Figure 10.15: CNN architecture for detecting rock cracks.

$17 \times 17$  negative patches ( $y = -1$ ) are randomly selected from the image. More than 80% of the patches are used as the training set and the others are used as the validation set.

The MATLAB code for the AlexNet CNN code is given in Appendix 10.18, and a slice of the code is in Code 10.5.1. Note how the size of the  $17 \times 17$  input images are defined at the beginning, and are interpolated to be  $32 \times 32$  images with an even number of pixels along each side of the image. This is necessary because, unlike the Python-based pooling subroutines, there is no automatic padding for pooling in MATLAB code. Pooling for MATLAB requires an even number of input pixels along each side of the input feature map. After pooling the size of the image shrinks by  $1/2$ , so with a  $32 \times 32$  input we can have about 5 layers before the output becomes too small.

#### Code 10.5.1. MATLAB Code for AlexNet.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                               Inputs                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ntrain=200; % The number of training set

wdx=8;
wdz=8;
image_x=wdx*2+1; % Original Image Patch Size X
image_z=wdz*2+1; % Original Image Patch Size Z

% Interpolate image to make sure size before pooling is even
image_x_new=32; % Interpolated Image Patch Size X
image_z_new=32; % Interpolated Image Patch Size Z

opts.alpha = 1; % Learning Rate
opts.batchsize = 20; % Minibatch Size
opts.numepochs = 10; % Iteration Number

%%% AlexNet CNN structure %%%%
cnn.layers = {
    struct('type', 'i') %input layer
    struct('type', 'c', 'outputmaps', 6, 'kernelsize', 5) %convolution layer
    struct('type', 's', 'scale', 2) %sub sampling layer
    struct('type', 'c', 'outputmaps', 12, 'kernelsize', 5) %convolution layer
    struct('type', 's', 'scale', 2) %sub sampling layer
};
```

In Figure 10.16a, 400  $17 \times 17$  positive patches and 400  $17 \times 17$  negative patches are randomly

Table 10.2: CNN Parameters for Rock Crack Classification

Image Size	Image Patch Size	Number of Channels	Conv. Patch Size	Number of Conv. Layers	Size of Max Pool	Number of Iterations
457×977	17×17	8	4×4	5	2×2	250

selected to give a well-balanced data set. The training set consisted of 80% of the patches and the other patches are used for the validation set. In the first test, only the big cracks are picked in the training set. Figure 10.16b shows the crack detection results with big cracks only, where the pixels in yellow and blue denote the cracks and the non-cracks. The big rock cracks are identified correctly from the background. The cracks in the left side of the image are not included in the training set, so they cannot be recognized in the testing set. To identify all the cracks in the image, both the small and big cracks are included in the training set. Figure 10.16c shows the crack detection results with all the cracks.

To improve the classification results, the RGB image is used as the input image instead of the grayscale image. The input of the network for the RGB image has three RGB channels, then the convolution filters in the first Conv layer also have three channels.

Figures 10.14c and 10.14d compare the crack detection results for the grayscale and RGB images. Since the rock cracks in the images are mostly white, both the RGB and grayscale values are high for the rock cracks. The identification of the rock cracks for both cases show acceptable accuracy. The RGB image contains more information than the grayscale image, so the CNN result for the RGB image is slightly better than for the grayscale image, such as in the red boxes of Figure 10.14c-10.14d. If the targets are colored, the RGB value of the target could be high but the grayscale value is an average value. In this case, the RGB image should be more easily classified than the grayscale image.

## 10.6 Detection of Basalt in GPR Images

A GPR dataset was collected in the Republic of Georgia in 2018 to detect the topography of the shallow basalt-sediment boundary. This boundary is at the Dmanisi archaeological site (<https://en.wikipedia.org/wiki/Dmanisi>), which is important because in 2010 hominid fossils were discovered at this boundary about 100 meters from the GPR survey. The white lines in Figure 10.17 correspond to the survey lines.

The GPR data sets were recorded with center frequencies of 500 MHz and each trace is effectively a zero-offset trace even though the source and receiver are separated by 4 cm. A vertical seismic profile (VSP) transmission survey was carried out over one of the holes in Figure 10.17, and the resulting GPR velocity model is shown in Figure 10.18a where the blue color corresponds to the velocity of 7.2 cm/ns and the other color is 7.8 cm/ns. There are 15 survey lines along the x direction, each with a crossline spacing of about 0.5 meter. The data were migrated using least squares migration (Schuster, 2017) and the resulting image is shown in Figure 10.18b. The goal is to train an AlexNet CNN to identify the basalt boundary so that it can be automatically detected for use by the paleoanthropologists. Low spots in the basalt boundary might act as fossil collection points and therefore be the target for future excavations.

The architecture of our CNN is illustrated in Figure 10.19. The network takes 32×32 patches from the migration image as input and the label at the center point is the label at the binary output. There are two convolution (Conv) layers and two FC layers followed by a softmax classifier to output a 2 × 1 vector of label information. There are 8 convolution filters with size 2×2 in the first Conv layer and 16 convolution filters in the second layer. The operations of rectified linear activation

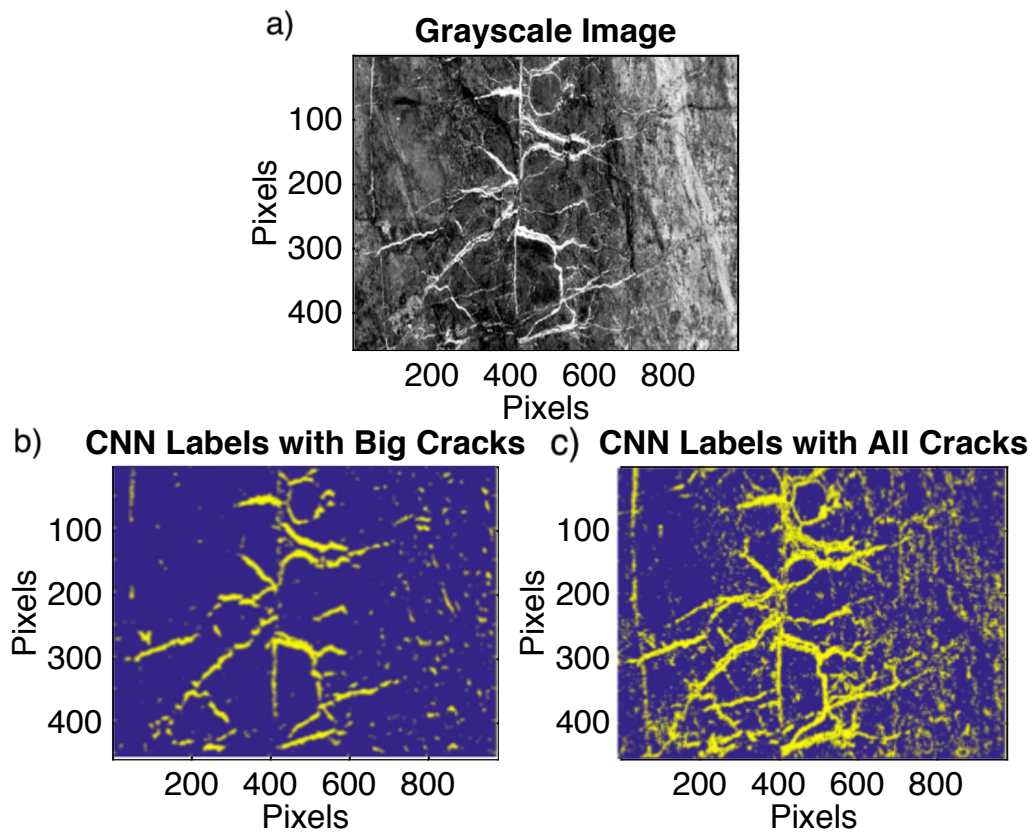


Figure 10.16: a) Grayscale image of the rock cracks. The CNN labels for the b) big cracks only and c) all the cracks.

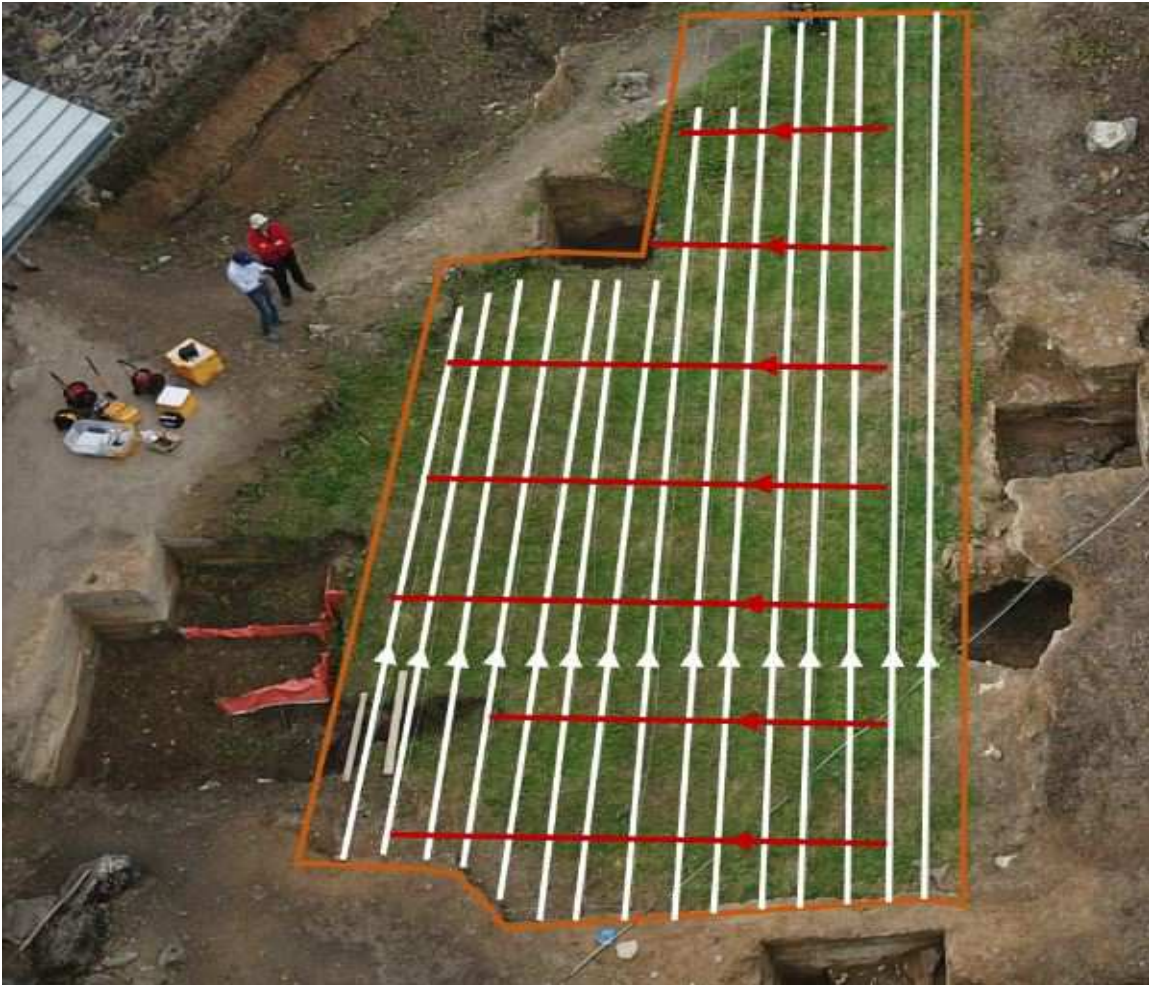


Figure 10.17: GPR lines at Dmanisi survey site in the Republic of Georgia. Photo courtesy of Sherif Hanafy.

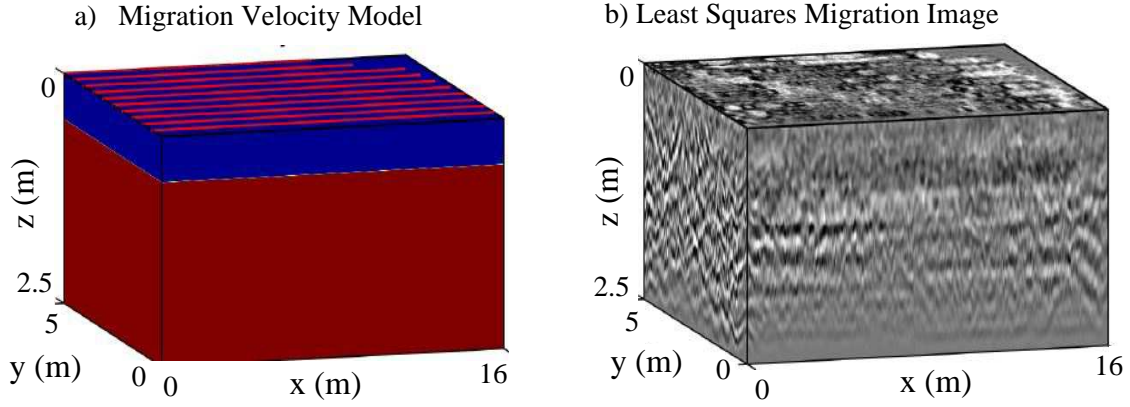


Figure 10.18: a) Migration velocity model (red-blue pattern on the surface are the survey lines) and b) 3D Kirchhoff least squares migration image.

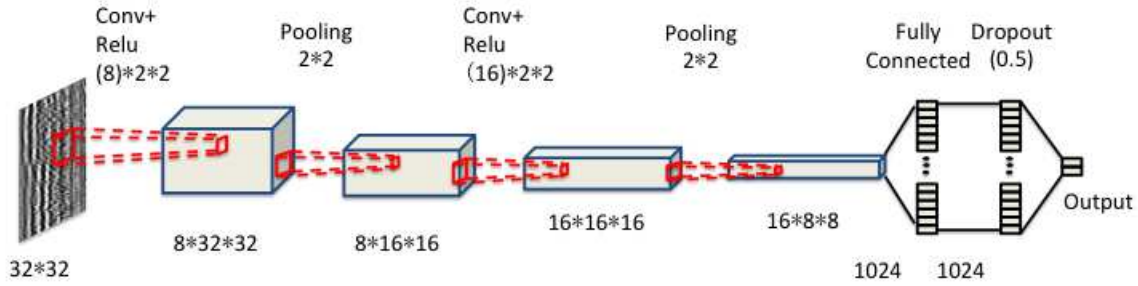


Figure 10.19: The CNN architecture used for labeling the GPR image.

(ReLU) and Max-pooling with size  $2 \times 2$  are applied after each Conv layer. There are  $32 \times 32 = 1024$  nodes in the first FC layer and we dropout 50% of them in the second FC layer. The softmax classifier gives the probability of the center point being a reflection event or not. The probability of two classifiers are compared with one another and the one with higher probability is selected as the prediction label.

All convolutional filter kernel elements are trained from the data in a supervised fashion by learning from the labeled set of training examples. A training set  $S = \{\mathbf{x}^i, y^i\}$  is given which contains  $N$  image patches, where  $\mathbf{x}^i$  is the  $i^{th}$  image patch and  $y^i \in \{0, 1\}$  is the corresponding class label. Then the corresponding cross-entropy cost function is given in equation 10.10.

### 10.6.1 Numerical Results

The image slice at 0.5 m shown in Figure 10.20a is used for training. 500 positive patches are picked manually as shown in Figure 10.20b and 500 negative patches are selected randomly from this image slice. 80% of the patches are used for the training dataset and the others are used as the validation dataset.

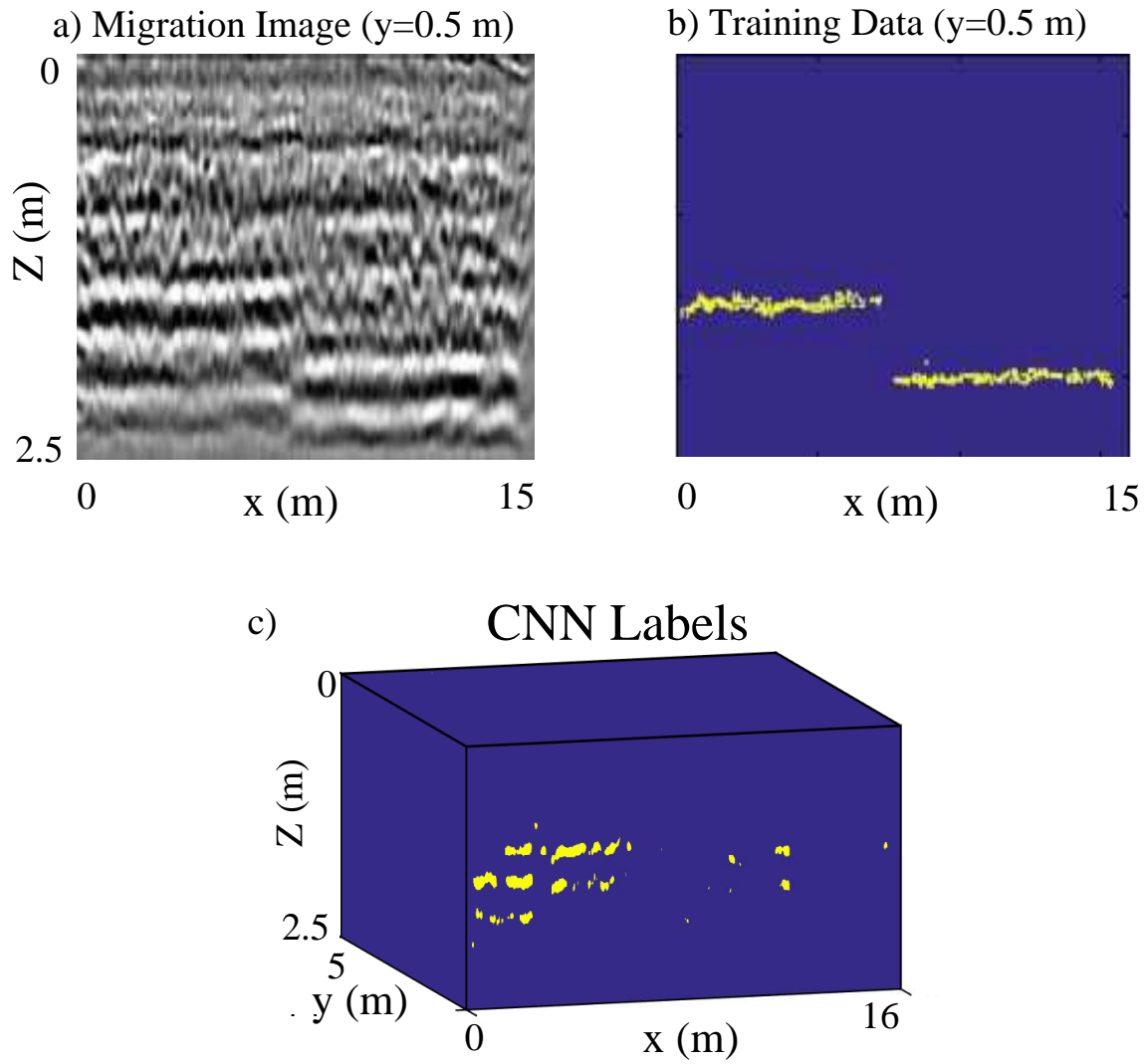


Figure 10.20: a) Migration image, b) CNN labels from the training data and c) labels (yellow is  $y = 1$ , purple=0) for training.



Table 10.3: CNN Parameters for Basalt Boundary Detection

Image Size	Image Patch Size	Number of Channels	Conv. Patch Size	Number of Conv. Layers	Size of Max Pool	Number of Iterations
500×1600	9×9	8	2×2	2	2×2	100

The CNN parameters for identifying the top basalt boundary are listed in Table 10.3. The accuracy of the training set and the validation set is 95.9% and 95.0%, respectively. The trained CNN structure is applied to the migration image and the CNN labels are shown in Figure 10.20. The points in yellow and in blue denote the reflection events and the non-reflection events.

Figure 10.21 shows the slices of the migration image and their CNN labels from 0 to 0.87 *m*. The CNN labels show that the depth of the basalt is around 1.2 *m* and suggests a fault around  $X = 7$  *m*. Figure 10.22 shows the slices of the migration image and their CNN labels from 1 *m* to 5 *m*, where the signal-to-noise ratio (SNR) in these migration slices is higher than the SNR in the training set. The CNN labels are affected by the noise and the reflection events are not continuous. However, we still notice that the reflection events are flat, the depth of the basalt is still around 1.2 *m* and there are no faults in these images.

## 10.7 Fault Detection by a Fully Convolutional Neural Network and Deconvnet

The problem with the previous example is that there is a poor resolution of the salt boundary picked by the trained ML. This problem is largely a result of the maxpool downsampling in the architecture: maxpool operations halve the resolution of a layer's output image. Maxpool operations remove precise *where* information in the migration image. This problem is illustrated with the FMs shown in Figure 10.23b-10.23d. Many of the FMs in the deeper layers are almost blank, or only show low-wavenumber features associated with the input seismic section shown in a). In this case, the objective is to pick the faults shown in a) where much of the high-waveumber information is lost after the downsampling operations.

To overcome this downsampling problem, Long et al. (2015) used communication between the CNN layers and the final output layer. They accomplished this with upsampling operations that recovered the high-frequency detail of FMs in the shallower layers. The architecture of their network resembles that of Figure 10.24, where the upsampling operations are carried out by the transpose operation, such as that shown in Figure 10.28. However, the upsampling operation is followed by a weighted convolution operation where the weights are learned during the training as illustrated in Figure 10.25.

A more interpretable image associated with each feature map can be constructed by upsampling the FM at each layer to the original size of the image. This can be done by inserting a transpose convolutional operator after the last convolution of each convolutional block. This is known as a deconv layer that recovers the downsampled feature maps to the original size, where an example of a deconv architecture is shown in Figure 10.24. For example, assume that the  $32 \times 32$  FM image after downsampling becomes a  $4 \times 4$  FM after maxpooling by a factor of 8. The deconv layer is denoted as "deconv×8". The filter size of this transposed convolutional layer is  $16 \times 16$ . The filter sizes of other convolutional layers are shown in Figure 10.24, where the fuse layer (see Figure 10.24b) consists of a  $1 \times 1$  filter that fuses the four feature maps from different deconv layers into the final result where the final FM is now the same size as the input image. Figure 10.26 shows the FMs after the deconvolution operations.

The FMs from a layer are upsampled, deconvolved and combined into just one FM at the

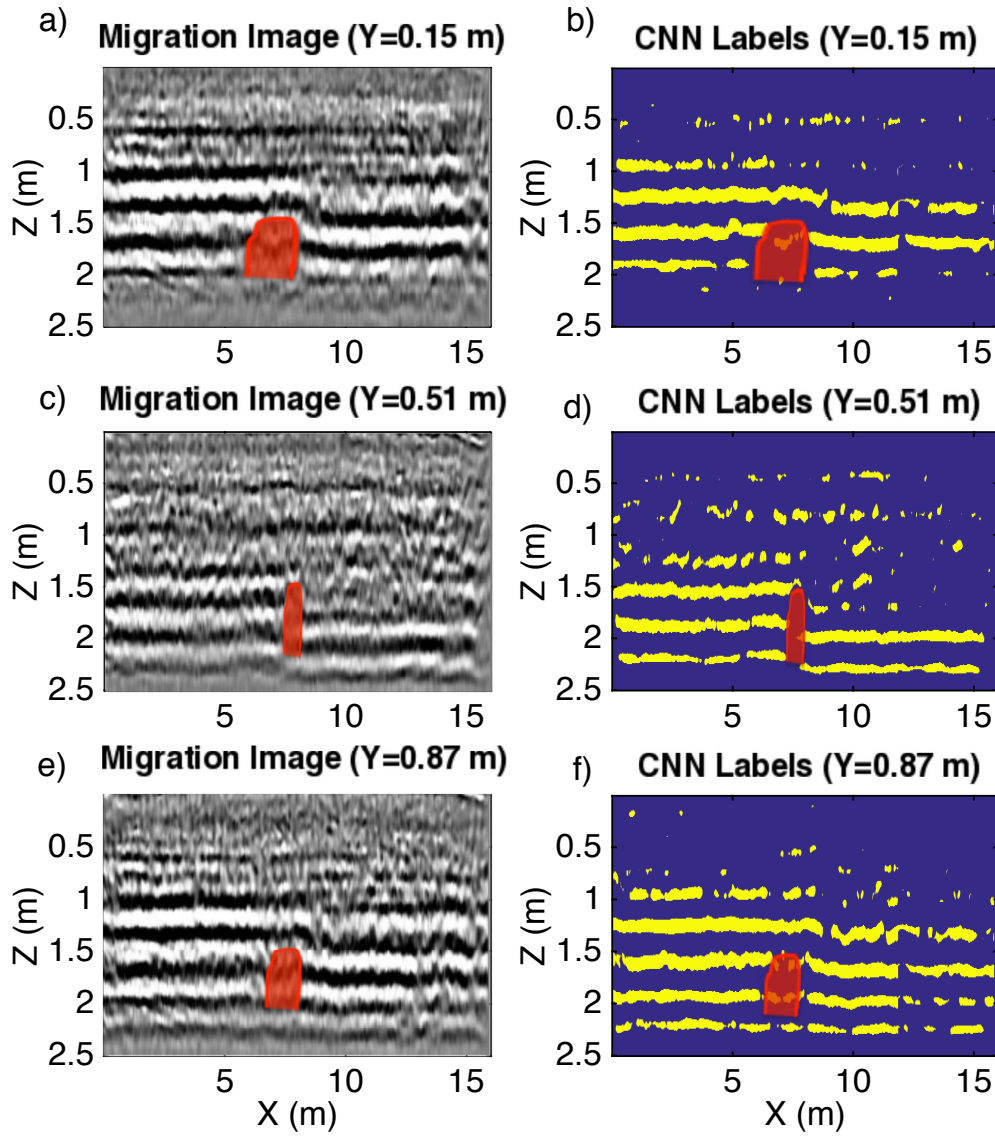


Figure 10.21: Migration images at a)  $Y=0.15$  m, c)  $Y=0.51$  m and e)  $Y=0.87$  m. The CNN labels at b)  $Y=0.15$  m, d)  $Y=0.51$  m and f)  $Y=0.87$  m. The red zones indicate the location of the faults.



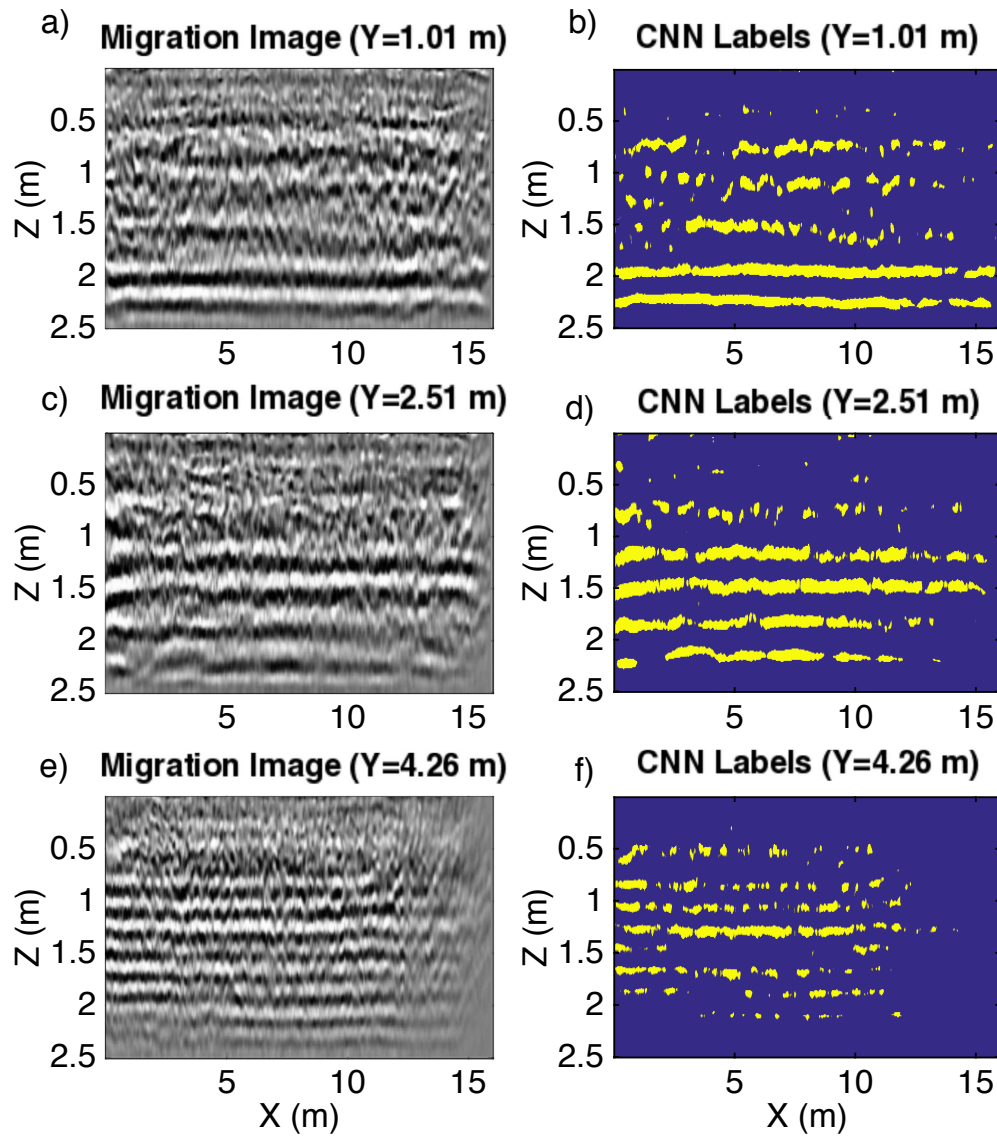


Figure 10.22: Migration images at a)  $Y=1.01$  m, c)  $Y=2.51$  m and e)  $Y=4.26$  m. The CNN labels at b)  $Y=1.01$  m, d)  $Y=2.51$  m and f)  $Y=4.26$  m.

a) Architecture

b) Feature Maps of 2<sup>nd</sup> Conv. Layer

c) Feature Maps of 4<sup>th</sup> Conv. Layer

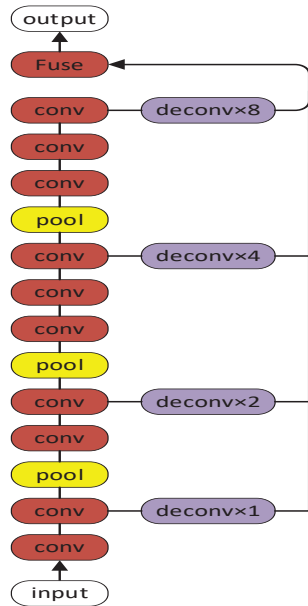
d) Feature Maps of 7<sup>th</sup> Conv. Layer

final deconvolved image in Figure 10.26. This final FM is added to other *final* deconvolved images from deeper blocks. For example, the FM4 image in Figure 10.26 is quite coarse compared to the input seismic image because it is not combined with the higher-resolution images deconvolved from shallower layers. In comparison, FM1 has very high resolution because it has not undergone more than one maxpooling operation and is combined with the deconvolved FMs from deeper blocks to give an accurate fault prediction.

## 10.8 Salt Boundary Detection by U-Net

As mentioned in Chapter 9, max pooling (or down-sampling) is somewhat insensitive to the location of the object of interest, so some spatial resolution is lost as to where the object is. In other

a) CNN+Deconv Architecture



b) Filter Parameters for each Layer

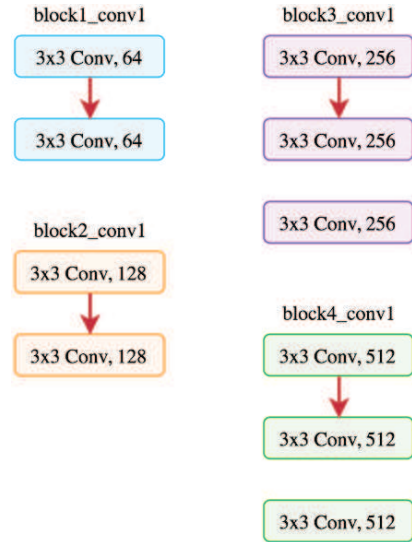


Figure 10.24: a) Architecture of CNN and deconv operations for detecting faults and b) filter parameters. The final FM images are only the result of deconvolution operations and do not get combined into a fully-connected neural network. Thus, this type of architecture is known as a fully convolutional network (FCN).

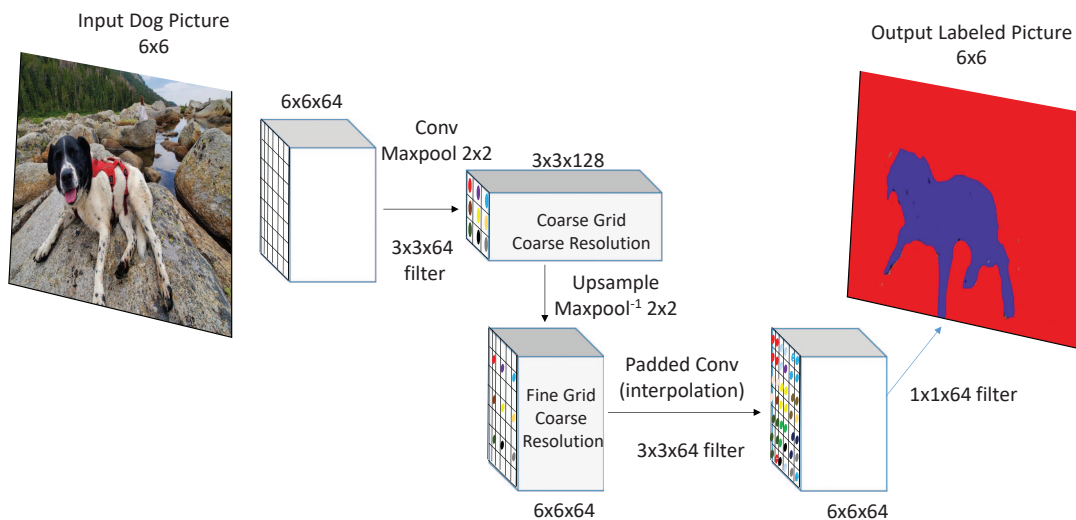


Figure 10.25: Upsampling operation associated with one block of the Figure 10.24 CNN model.

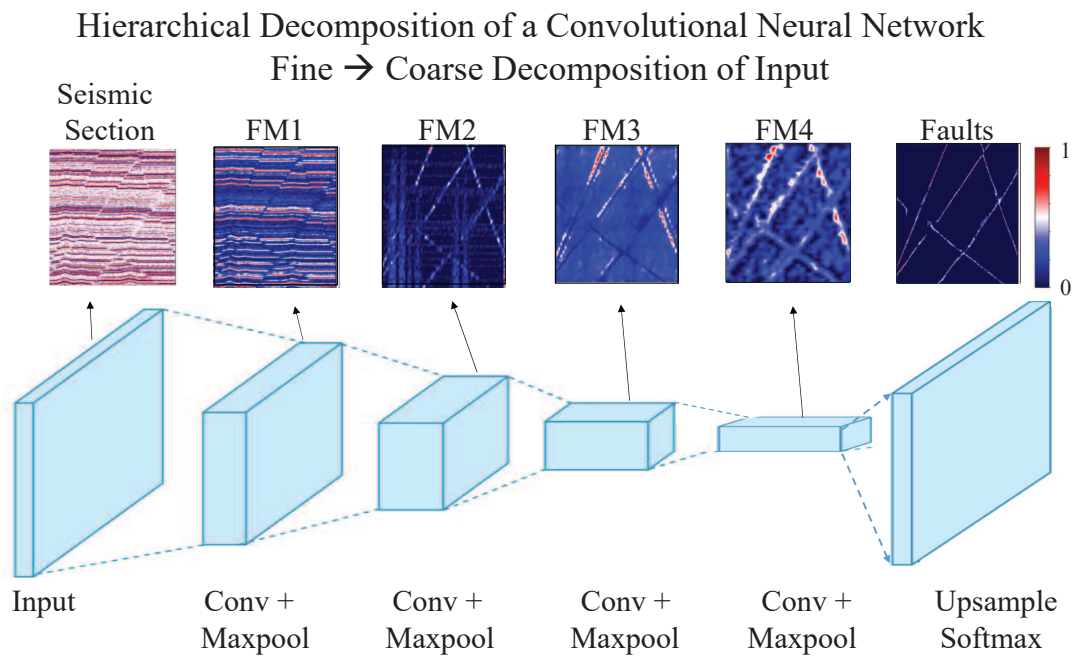


Figure 10.26: Block diagram of the convolutional neural network (CNN) architecture that decomposes the input picture of a seismic section into fine-scale features feature maps (FM1 and FM2) and coarser-scale feature maps (FM3 and FM4). The pixels in the far-right image are labeled  $0 < y < 1$ , where  $y = 1$  indicates that the pixel is a fault and  $0 \leq y \leq 1$  indicate the probability of being a fault. The FMs shown above are the same size as the input image because they have been deconvolved. Figure inspired partly from Dertat (2017) and Shi et al. (2018).

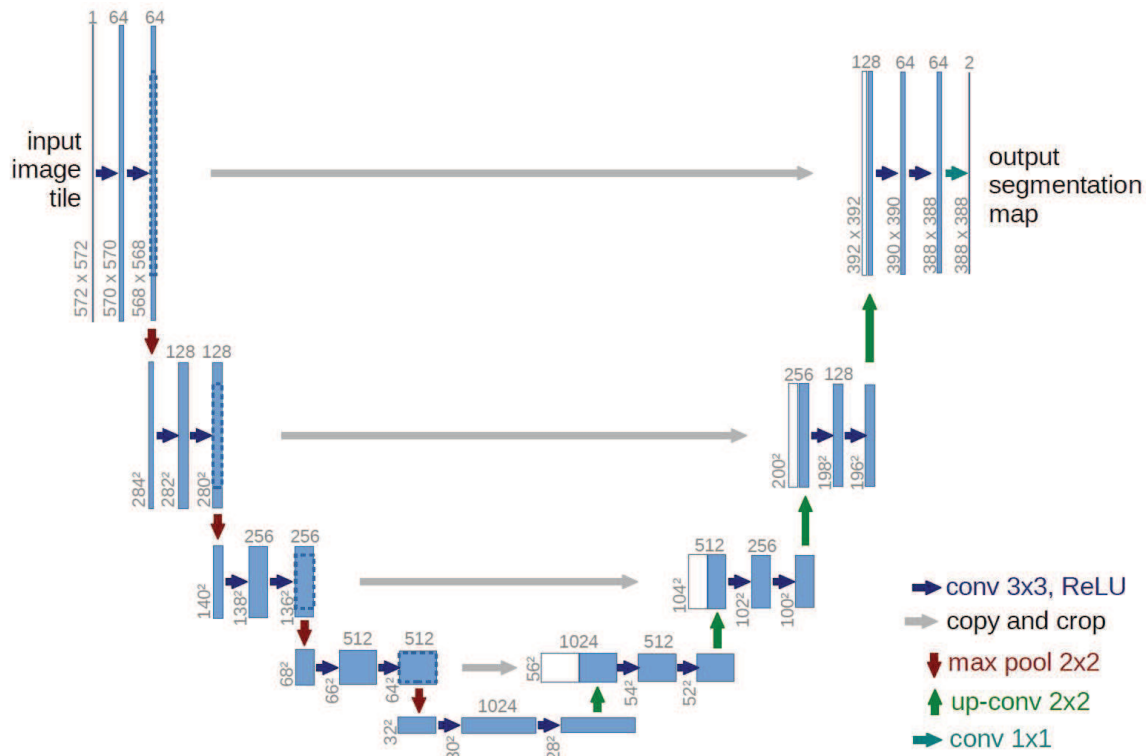


Figure 10.27: U-Net architecture. Figure from Ronneberger et al. (2015).

words, AlexNet knows what the object is but not precisely where it is after several max pooling operations. Therefore, it is not easy to transform a tall vector at the end of the network into a picture the same size as the input image with the pixels precisely classified.

To mitigate this precision problem, the U-Net architecture in Figure 10.27 was proposed by Ronneberger et al. (2015) that connects a contracting network to a symmetric expanding network to give a precise classification of each pixel. The U-Net CNN has proven to be very popular in biomedical applications such as precise detection of anomalies in microscope images of blood cells, CT scans of internal organs or MRI images. Precision is extremely important for cancer detection.

The U-Net architecture is in the form of the letter U, and some of its salient characteristics are the following.

- The up-sampling operator of the U-Net is selected to be the transpose of the convolution operation where, for example, the transpose of the  $4 \times 16$  convolution matrix in Figure 10.28b is the  $16 \times 4$  matrix in Figure 10.28. Applying the  $16 \times 4$  matrix to a  $4 \times 1$  input vector transforms it into a longer  $16 \times 1$  vector. The weights in the transposed convolution are learnable, so there is no need to use an interpolation method to inflate the size of a feature map. See [https://github.com/vdumoulin/conv\\_arithmetic](https://github.com/vdumoulin/conv_arithmetic) for many different animated examples of transpose operations.
- The expanding part of the U-Net is sometimes known as the decoding or deconvolution set of operations, while the contracting part is known as an encoder<sup>5</sup>. As we know, the least squares

<sup>5</sup>The autoencoder discussed in Chapter 14 consists of a symmetrical arrangement of a contracting encoder and an expanding decoder.

deconvolution operator is in the form of  $[\mathbf{W}^T \mathbf{W}]^{-1} \mathbf{W}^T$ , and  $\mathbf{W}^T$  is only an approximation if  $[\mathbf{W}^T \mathbf{W}]$  is diagonally dominant.

- The horizontal gray arrows in Figure 10.27 connect the leftside blocks of layers to those on the right. Appending, or concatenating, the output from the encoder's layers to the inputs of their opposite facing decoders is claimed to allow for, after training, precise prediction of the to-be-labeled pixels. This is done at every sub-block before the locations become further blurred before another round of max pooling in the contracting blocks. Empirical results suggest that this concatenation of information from the encoding block can increase the resolution of semantic segmentation.
- There are no fully-connected layers and the network is a fully-convolutional neural network (FCN). This avoids the possible difficulty of transforming a vector to an image with precise locations of the correctly labeled pixels.
- A transposed convolution with stride 2 or more can generate checkerboard artifacts in the up-sampled image. The article by Odena et al. (2016) recommends an up-sampling operation (i.e., an interpolation method) followed by a convolution operation to reduce such issues. Figures 10.29 and 10.30 show examples of 1-stride of an input  $2 \times 2$  grid of numbers and 2-stride upsampling of an input  $3 \times 3$  grid of numbers to get the larger shaded green grids on the top. This is the upsampling procedure for correlation of the filter (gray grids) with the blue grid numbers (see Figure 10.28).

### 10.8.1 Keras U-Net Code

The Keras U-Net code and salt classification lab are at *LAB1/Chapter.CNN.Unet/U\_Net\_Lab.html* and in Exercise 4. The subroutine in the Keras code for defining a layer's convolution, batch normalization, and activation functions is given at Code 10.8.1 (<https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47>).

#### Layer Subroutine

**Code 10.8.1.** Keras Code for "conv2d\_block" in U-Net.

```
#define the convolution layer
def conv2d_block(input_tensor, n_filters, kernel_size = 3, batchnorm = True):
    """Function to add 2 convolutional layers with the parameters passed to it"""
    # first convolution layer
    x = Conv2D(filters = n_filters, kernel_size = (kernel_size, kernel_size),\
               kernel_initializer = initializers.he_normal(seed=1), padding = 'same')(input_tensor)
    if batchnorm:
        x = BatchNormalization()(x)
    x = Activation('relu')(x)

    # second convolution layer
    x = Conv2D(filters = n_filters, kernel_size = (kernel_size, kernel_size),\
               kernel_initializer = initializers.he_normal(seed=1), padding = 'same')(x)
    if batchnorm:
        x = BatchNormalization()(x)
    x = Activation('relu')(x)

    return x
```

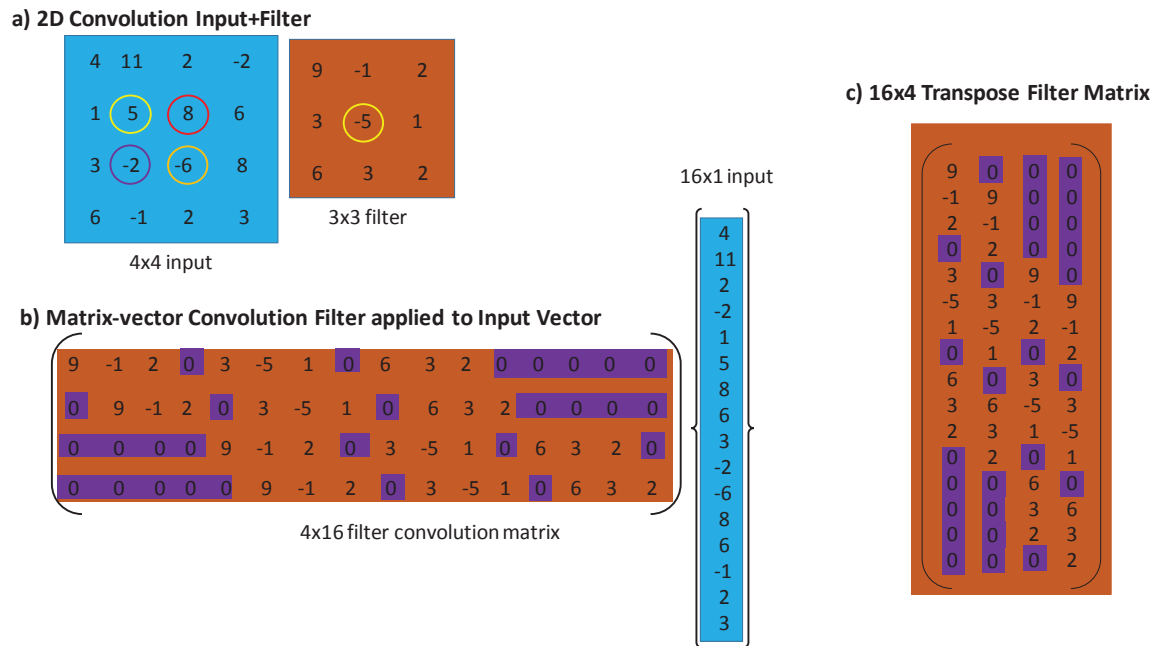


Figure 10.28: a) 2D filter and input image, b) corresponding matrix-vector operation, and the c) transpose of the convolution matrix in b). The only permissible lags in a) are the four lag values denoted by circles where the yellow circle corresponds to the zero-lag position of the convolution operator. Here, the convolution coefficients in the orange filter have already been mirrored across the horizontal and vertical axes with the origin at the yellow circle. In b), the output of the matrix-vector multiplication at 0 lag position is computed by taking the dot product of the top row vector with the blue column vector. The dot product for lag=1 is computed by shifting the orange box in a) so that its yellow circle coincides with the red circle in the blue box, and then taking the dot product of the coincident numbers. This is equal to the dot product of the second row vector in b) with the blue column vector to the right. The transpose of the contracting  $4 \times 16$  convolution matrix in b) is the expanding  $16 \times 4$  matrix in c). Partly adapted from Shibuya (2016).



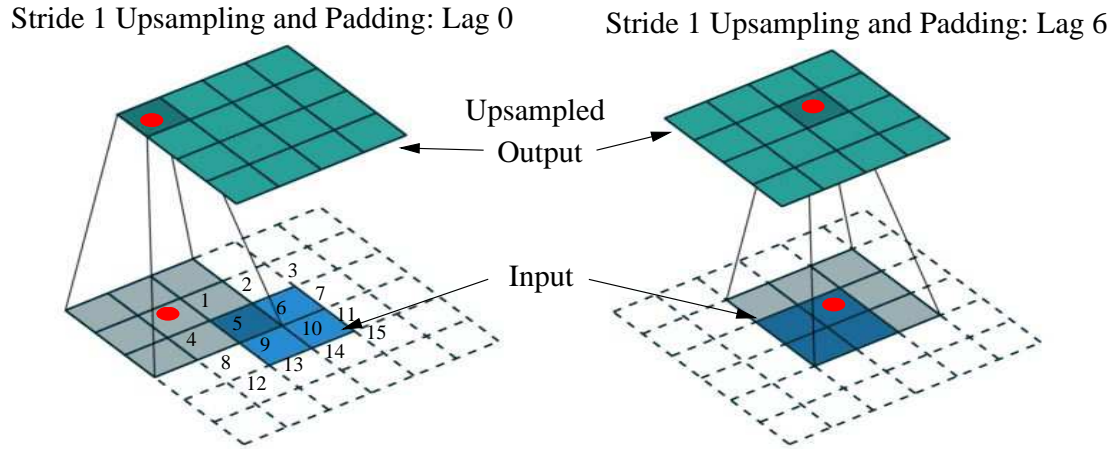


Figure 10.29: Stride 1 upsampled  $4 \times 4$  green grid on the top for lag 0 on the left and lag 6 on the right. The input  $2 \times 2$  grid of blue squares is at the bottom. The red dots correspond to the center point of the gray filter and the numbers in the leftside image are the lag numbers of the filter. Image adapted from <http://datascience.stackexchange.com/questions/6107/what-are-deconvolutional-layers> and Dumoulin and Francesco (2016).

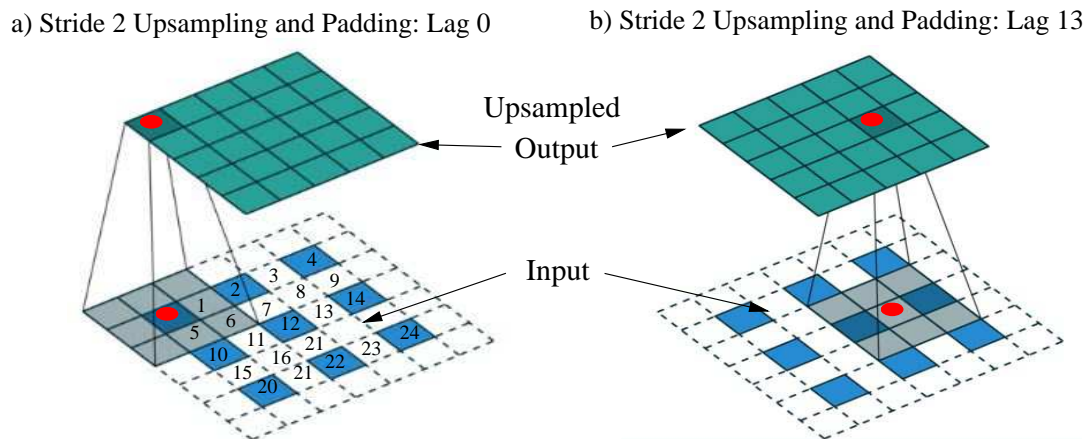


Figure 10.30: Similar to the previous figure except the filter stride is 2, there is padding, and the filter and input dimensions are  $3 \times 3$ . The lag associated with b) is 13 and the output grid is  $5 \times 5$  rather than the  $3 \times 3$  input grid. The  $3 \times 3$  input grid of blue blocks has been inflated to a padded  $7 \times 7$  grid to accommodate the stride of 2 in the filter; the number in any white block is zero. See [https://github.com/vdumoulin/conv\\_arithmetic](https://github.com/vdumoulin/conv_arithmetic) for animations of stride 2 upsampling operation.



This subroutine can now be used to define the U-Net architecture in Code 10.8.2.

**Code 10.8.2.** *Keras Code for U-Net CNN.*

```
# define U_net architecture
def get_unet(input_img, n_filters = 16, dropout = 0.1, batchnorm = True):
    """Function to define the UNET Model"""
    # Contracting Path
    c1 = conv2d_block(input_img, n_filters * 1, kernel_size = 3, batchnorm = batchnorm)
    p1 = MaxPooling2D((2, 2))(c1)
    p1 = Dropout(dropout)(p1)

    c2 = conv2d_block(p1, n_filters * 2, kernel_size = 3, batchnorm = batchnorm)
    p2 = MaxPooling2D((2, 2))(c2)
    p2 = Dropout(dropout)(p2)

    c3 = conv2d_block(p2, n_filters * 4, kernel_size = 3, batchnorm = batchnorm)
    p3 = MaxPooling2D((2, 2))(c3)
    p3 = Dropout(dropout)(p3)

    c4 = conv2d_block(p3, n_filters * 8, kernel_size = 3, batchnorm = batchnorm)
    p4 = MaxPooling2D((2, 2))(c4)
    p4 = Dropout(dropout)(p4)

    c5 = conv2d_block(p4, n_filters = n_filters * 16, kernel_size = 3, batchnorm = batchnorm)

    # Expansive Path
    u6 = Conv2DTranspose(n_filters * 8, (3, 3), strides = (2, 2), padding = 'same')(c5)
    u6 = concatenate([u6, c4])
    u6 = Dropout(dropout)(u6)
    c6 = conv2d_block(u6, n_filters * 8, kernel_size = 3, batchnorm = batchnorm)
    u7 = Conv2DTranspose(n_filters * 4, (3, 3), strides = (2, 2), padding = 'same')(c6)
    u7 = concatenate([u7, c3])
    u7 = Dropout(dropout)(u7)
    c7 = conv2d_block(u7, n_filters * 4, kernel_size = 3, batchnorm = batchnorm)

    u8 = Conv2DTranspose(n_filters * 2, (3, 3), strides = (2, 2), padding = 'same')(c7)
    u8 = concatenate([u8, c2])
    u8 = Dropout(dropout)(u8)
    c8 = conv2d_block(u8, n_filters * 2, kernel_size = 3, batchnorm = batchnorm)

    u9 = Conv2DTranspose(n_filters * 1, (3, 3), strides = (2, 2), padding = 'same')(c8)
    u9 = concatenate([u9, c1])
    u9 = Dropout(dropout)(u9)
    c9 = conv2d_block(u9, n_filters * 1, kernel_size = 3, batchnorm = batchnorm)

    outputs = Conv2D(1, (1, 1), activation='sigmoid')(c9)
    model = Model(inputs=[input_img], outputs=[outputs])
    return model
```

Changes to the input arguments can be made to test the sensitivity of the architecture to changes in the hyperparameters such as the filter sizes.

## 10.8.2 Numerical Results

The U-Net CNN is applied to a public-domain seismic data that is labeled and contains salt and sedimentary features. The goal is to detect the salt in the images, which is similar to that in the AlexNet salt detection example. However, U-Net has better spatial resolution than AlexNet so the predicted salt boundary should be more highly resolved.

Figure 10.31a depicts a typical  $125 \times 125$  patch of seismic data and Figure 10.31b depicts the

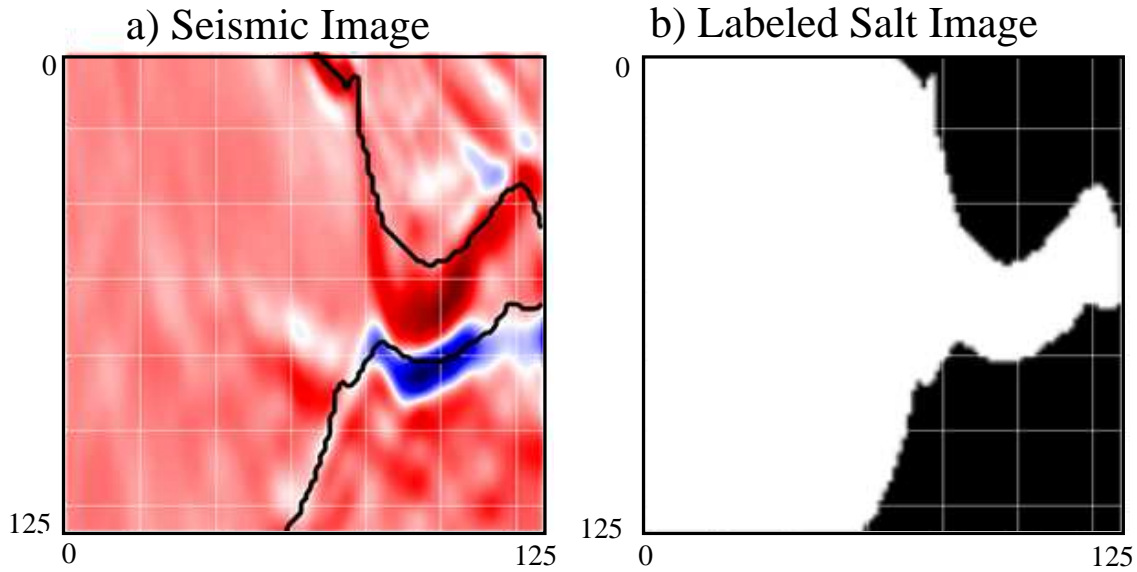


Figure 10.31: (a) Input seismic image and b) labeled image where white corresponds to salt ( $y = 1$ ) and black corresponds to non-salt ( $y = 0$ ).

labeled image. Here, white corresponds to the salt and black is the non-salt background. After training, a sample of the resulting prediction is shown in Figure 10.32. The input seismic image is in a), the groundtruth is in b) and the predicted labels are in c). It is clear that the U-Net CNN has achieved much better resolution of the salt boundary than given by the AlexNet from the previous section. The learning rate curves are shown in Figure 10.33, where the accuracies for the U-Net CNN training and validation data sets are close to one another. Better accuracy could have been achieved if more training examples were used.

The training for 4000 examples takes about 5 hours on a conventional laptop computer. To lessen the training time, the number of training examples is reduced to 400. Several examples of the validation results are shown in Figure 10.34, which has reduced accuracy compared to Figure 10.32. This is expected since the number of training examples has been reduced by an order-of-magnitude. The corresponding learning rate and accuracy curves are in Figure 10.35.

Different step lengths and learning rate values were used to assess the rate of convergence in Figure 10.36. The Adam method seemed to perform the best in Figure 10.36a, although this depends on the value of the learning rate parameter as shown in Figure 10.36b.

As discussed in Chapter 9, batch normalization just before each activation function U-Net is an effective means for speeding up the convergence rate and getting more accurate predictions. It has been reported that some users employ batch normalization after the activation function, *not before*. The effectiveness of using batch normalization after an activation function is now tested with the 400-example training set.

The accuracy curve for using batch normalization after the activation functions is shown in Figure 10.37a, while the results that did not have batch normalization are shown in Figure 10.37b. It is obvious that batch normalization leads to better convergence. Batch normalization was also used just before the activation function, and the accuracy curve was similar to that in Figure 10.37a, except it was moderately more well behaved.

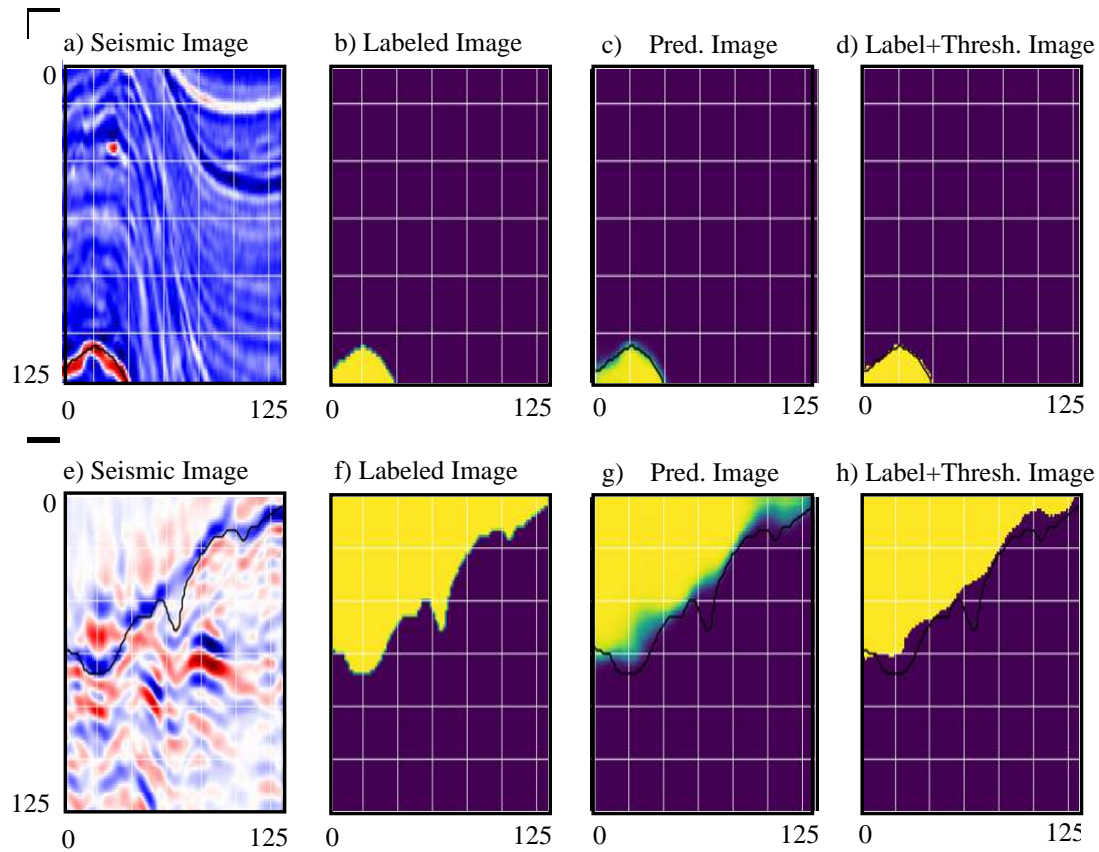


Figure 10.32: Validation examples. Top Row: a) Input seismic image, b) true salt labels, c) predicted salt probabilities and (d) predicted salt labels. Bottom row is the same except for another validation example. Figure courtesy of Abdullah Alali.

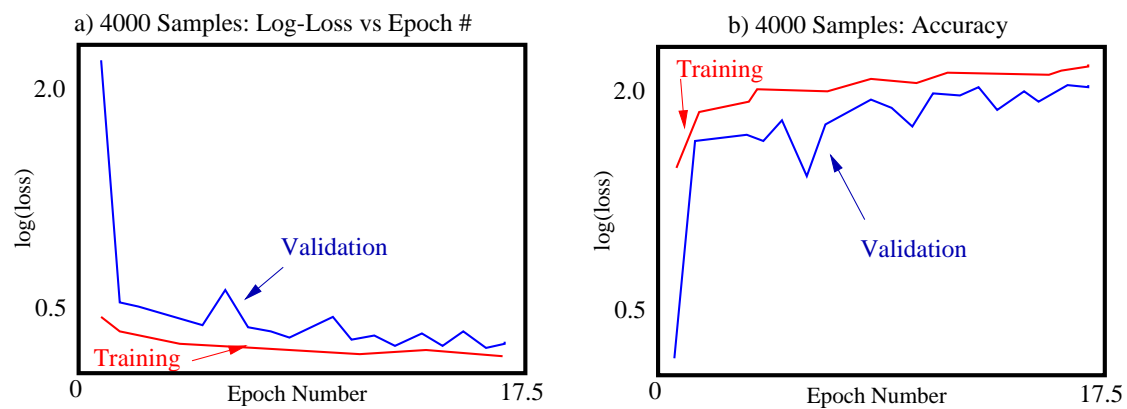


Figure 10.33: a) Learning and b) accuracy curves for the 80% training (red) and 20% validation (blue) examples, where there are 4000 examples in the training set. Figure courtesy of Abdullah Alali.

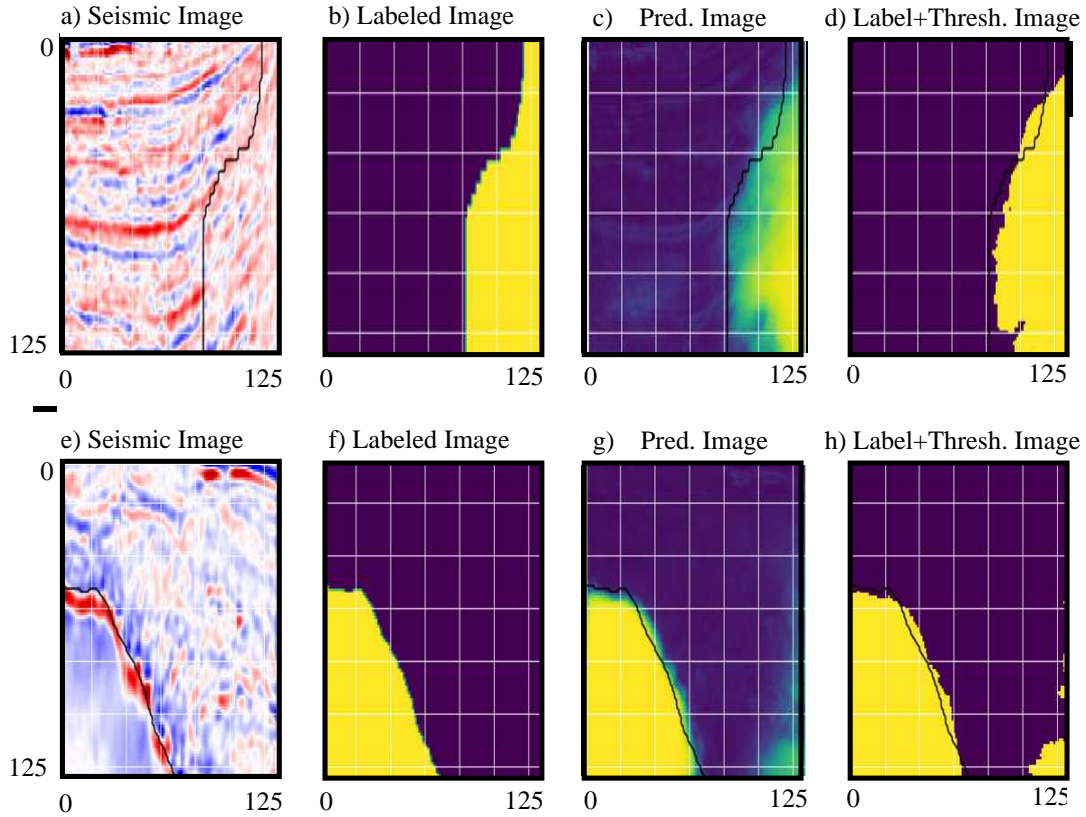


Figure 10.34: Similar in training strategy used for Figure 10.32 except there are only 400 training examples and different parts of the seismic sections are labeled. Figure courtesy of Abdullah Alali.

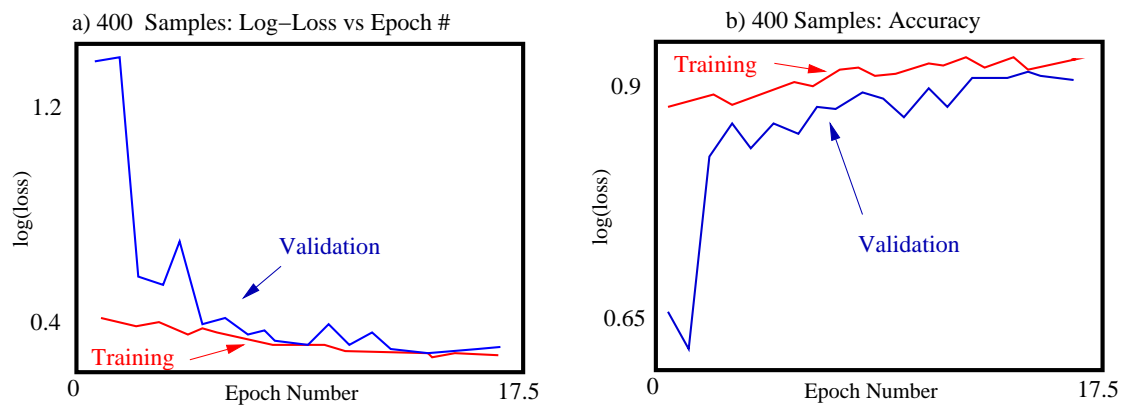


Figure 10.35: Similar to Figure 10.33 except a different part of the migration image is used and there are only 400, rather than 4000, training examples. Figure courtesy of Abdullah Alali.

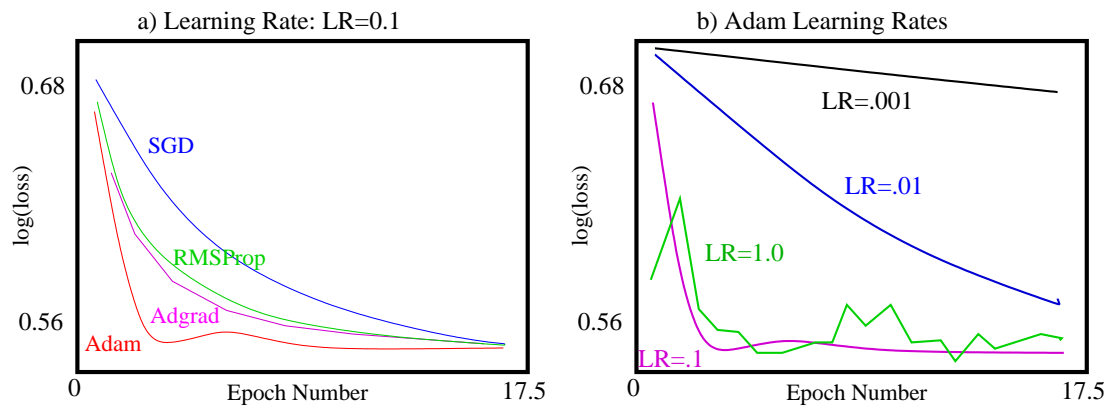


Figure 10.36: Convergence curves for a) different types of step length methods and for b) the Adam step length method. Figure courtesy of Abdullah Alali.

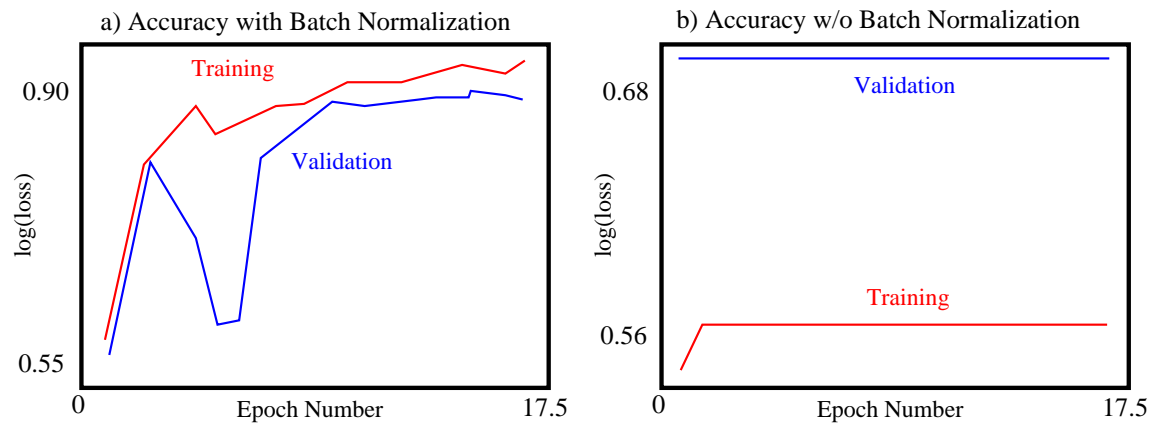


Figure 10.37: Accuracy curves for U-Net a) with batch normalization after each activation and b) no batch normalization. Figure courtesy of Abdullah Alali.

**Code 10.8.3.** Keras Code for Activating BatchNormalization.

```

if batchnorm:
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

to

if batchnorm:
    x = Activation('relu')(x)
    x = BatchNormalization()(x)

```

## 10.9 Fault Detection by U-Net

Some of the previous examples trained the CNN on labeled field data, which requires a great deal of manual labor if picked by humans. This is one of the main problems in applying CNN to geoscience data (Bergen et al., 2019).

To avoid manual labeling of field data, Xi et al. (2020) constructed synthetic models of faulted layers and trained a U-Net model to detect these faults. The synthetic faults were automatically labeled by the algorithm so no manual picking was required.

The first two steps for training are depicted in Figure 10.38, where the synthetic sections are similar to those computed by convolving seismic wavelets with reflectivity sections (see Appendix 27) with curved boundaries and faults. The faults are automatically labeled by the computer program, and the trained network is then applied to seismic sections migrated from field data. Figures 10.38-10.39 compare the results for picking the faults by a standard image processing method based on coherency and by the U-Net architecture. The green zones in Figure 10.38c highlight some areas where the U-Net detector is more accurate while the red zones show where the image processing method is more accurate. The widths of the dark lines are proportional to the thickness of the probability values for the fault identifier.

Both the standard method and the U-Net did not accurately detect listric faults compared to the human-interpreted section in Figure 10.38e. The failure of the CNN is an example of incomplete training with synthetic data that fails to account for surprise features in the field data. Xiong et al. (2020) conclude that *The image processing fault probability exhibits a better performance in detecting vertical normal faults in a higher signal-to-noise dataset. The CNN method performs better than the image processing method in detecting high angle dipping faults. Both methods failed to completely map listric faults. We anticipate that augmenting the training data with a suite of listric fault training models will improve the CNN performance.*

It is not surprising that the U-Net missed some faults that were detected by the coherency-based method in Figure 10.38: if such faults are not present in the synthetic training data then the CNN will be incompletely trained. The U-Net might have performed better if it employed transfer training where a large portion of the CNN model from synthetic training is combined with training on a small subset of labeled field data. Details for implementing transfer training are described in Chapter 11.

A more successful example of fault detection is shown in Figure 10.40 where the synthetic models were tuned (Wu et al., 2020) to the type of faults expected in the data. Unlike the listric faults in Figure 10.39c, this field area was dominated by the steeply dipping faults seen in the images. Therefore, as long as the synthetic training faults are accurate renditions of the actual geology at the field site, fault detection and geobody detection by CNN appears to greatly improve the efficiency of interpreting large volumes of seismic images.

## 10.10 Blood Cell and Microfossil Windowing and Labeling by Faster R-CNN

Blood samples are placed on microscope slides and analyzed for their content. In the case of cancer patients, white and red blood cell counts are especially important in monitoring the current health of the patient as well as determining the optimal dosage of medicine. Such measurements should be both accurate and timely. As an example, Figure 10.41a depicts the windowed and labeled blood cells computed by the Faster R-CNN algorithm in the Computational Labs section. On the right are microscope images of rocks samples containing sandstone, granite and fossils. Geologists often need to detect, label, and provide quantitative estimates of rock properties such as size distribution and types of fossils, porosity, mineral content and distributions. Such tedious tasks might be accomplished by an effective CNN method, such as the Faster R-CNN or YOLO2 algorithms for labeling the blood samples.

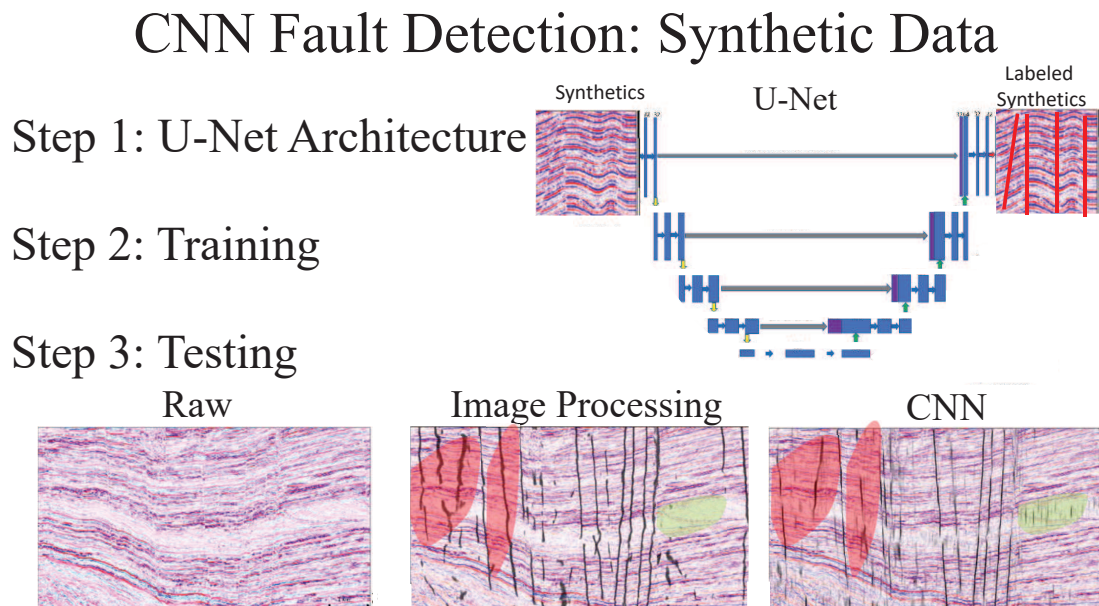


Figure 10.38: The steps for training the U-Net detection of faults in seismic migration sections. The first two steps are for training with synthetic data, and the third step is to apply the trained CNN to field data. The green zones show where the CNN performed more accurately and the red zones showed where the image processing method performed better. Figure from Xi et al. (2020).



## CNN Fault Detection: Synthetic Data

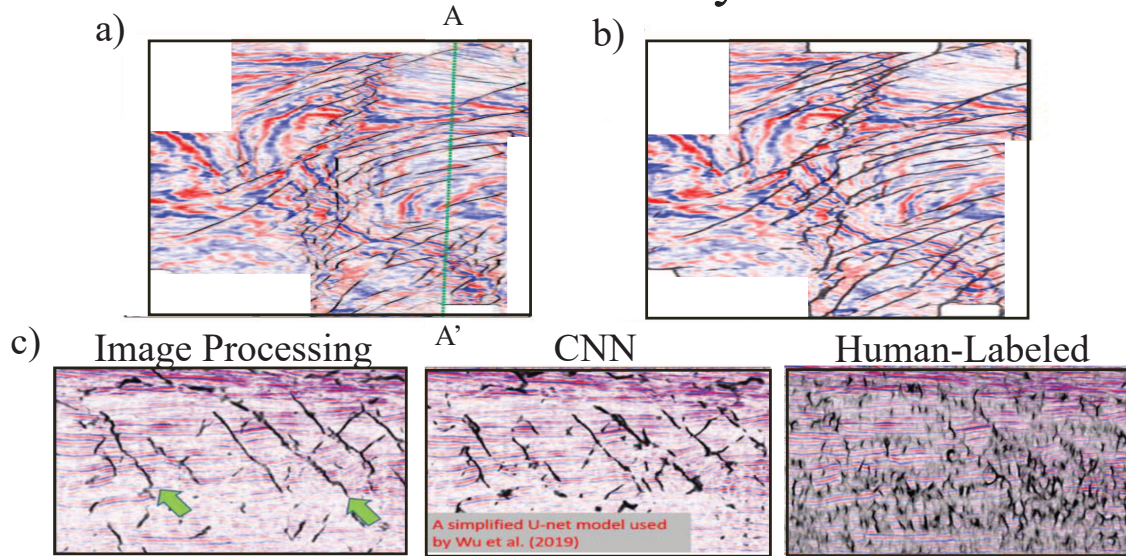


Figure 10.39: Depth section of a 3D seismic image cube with faults interpreted by an a) image processing method and b) by the CNN. The vertical section along the line AA' is displayed in c), where the human-labeled listric faults (dark lines with gray background) on the far right showed the limitations of auto-picked faults by CNN and image processing. This is a salient example of incomplete CNN training on synthetic data which fails to identify the desired features in the field data. Figure from Xi et al. (2020).

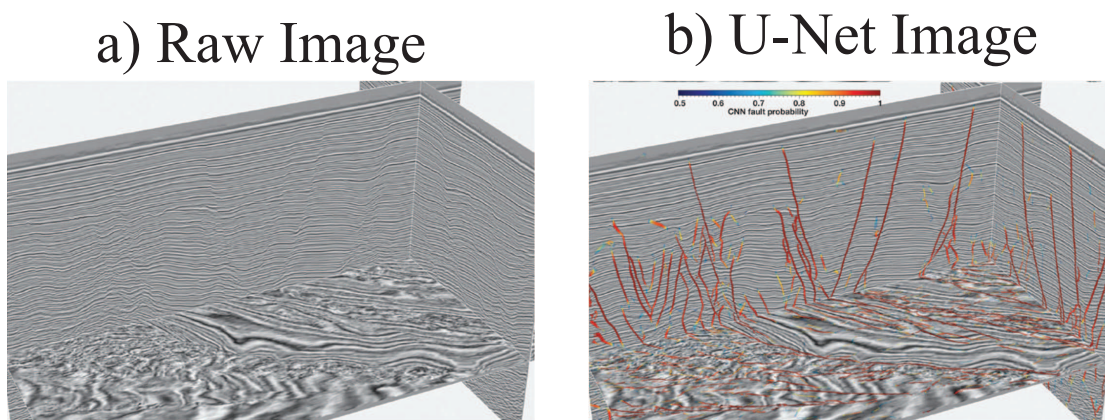
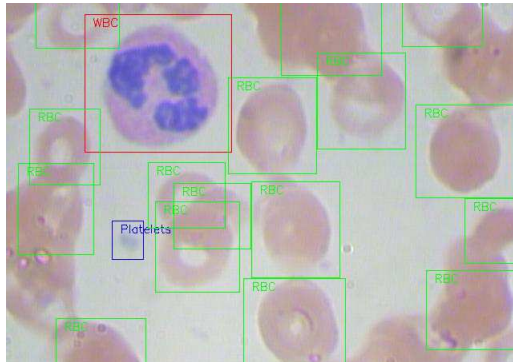


Figure 10.40: Fault detection in a subset of the Opunake-3D seismic data using the CNN trained by only synthetic data sets. Figure from Wu et al. (2020b).



a) Labeled Blood Cells with Bounding Boxes



b) Microscope Images of Rocks

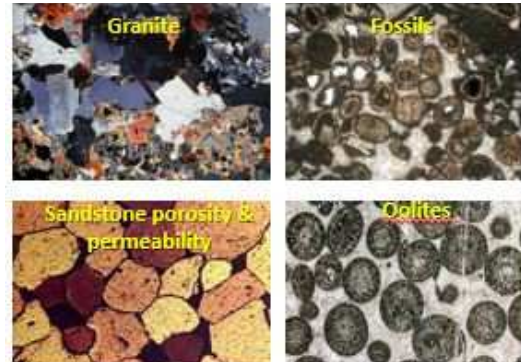


Figure 10.41: a) Windowing and labeling of microscope images of a blood sample by Faster R-CNN and b) image of a thin section from a rock. The white and red blood cells are labeled in a) while the fossils should be labeled in b).

As an example, Figure 10.42a depicts an image of microfossils extracted from a well core. The weights used to train a YOLO2 Keras code for detecting and labeling blood cell samples were used to compute the bounding boxes of the microfossils to give the results shown in Figure 10.42b. The bounding boxes for many of the microfossils are accurately computed, but no effort was made in labeling the microfossils. The old blood cell labels were used. The detection accuracy can be improved by labeling BBs and the class types in a new training set of microfossil images. Then the weights from the blood cell example can be used with transfer learning to detect and label all of the microfossils. This task is performed in the lab at [/LAB1/Chapter.Yolo//LAB1/Chapter.Yolo/Yolo.html](#).

## 10.11 Small Object Detection in Drone Photographs by Superpixel-based CNN

It is often the case that a small object needs to be identified in a large digital photograph. The size of the object might be around  $10 \times 10$  pixels, but it is buried in a sea of many other objects in a large background. This presents two problems.

1. **Imbalanced Training Data.** Training a semantic-segmentation NN with photos containing these sparsely located objects is imbalanced, where most of the pixels are labeled as no object  $y = 0$ , and only a relatively much smaller number of pixels are labeled as the object  $y = 1$ . Such training often leads to an inaccurate identification of the small object in test photos.
2. **Data Augmentation.** Data augmentation should be used to balance the data by increasing the number of small objects in a photo, but this can be difficult by deliberate insertion of stretched or rotated images of small objects into a typical photograph. This can often lead to a marked intensity contrast between the pixels of the inserted small-object and their new neighboring pixels, which can lead to a mistrained NN that only identifies such artifacts as  $y = 1$ .

There are at least two categories of data augmentation, one augments training data in data space and the other in feature space (Shorten and Khoshgoftaar, 2019). In the data space, new data can



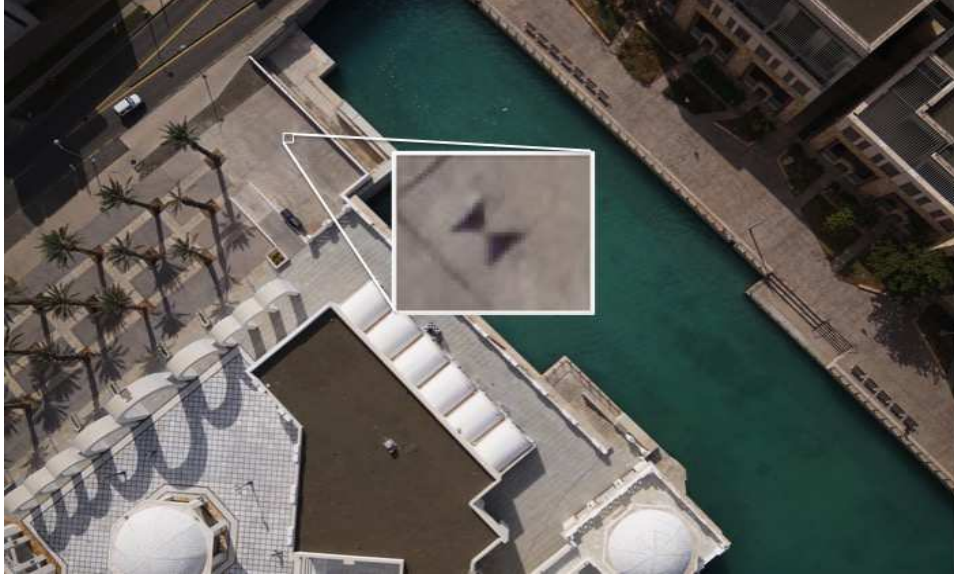


Figure 10.43: An example of the  $0.5 \times 0.5 \text{ m}^2$  GCP on the ground, which has a unique hourglass shape and its color is dark. Image adapted from Feng et al. (2021a).

### 10.11.2 Numerical Results

The superpixel-based CNN method is applied to the 2D aerial image shown in Figure 10.45, where the full image consists of  $3500 \times 5000$  pixels and the GCP has about  $25 \times 30$  pixels. The number of GCP patches is much less than the number of non-GCP image patches, and so these input data are extremely unbalanced. SLIC segments all the objects into superpixels so that the superpixels from the non-GCP objects have a similar shape and color while the GCP has a unique hourglass shape and dark color. Then the samples for the non-GCPs and GCPs can be easily balanced.

The aerial image is segmented into approximately 30000 superpixels using the SLIC algorithm and superpixel-based CNN is then applied to the SLIC-processed data. Since the hourglass shape and the dark color of the GCP are known, there is no need to extract positive patches from the aerial images. 3000 synthetic positive training samples are easily constructed by rotating, shrinking and expanding a dark-color hourglass shape superpixel as shown in Figure 10.46. 3000 negative samples are picked randomly from the non-GCP superpixels in the image. Only 10 epochs are needed to achieve an accuracy higher than 99% accuracy for both the training and validation examples. The trained CNN is then applied to all of the other superpixels in the image. Only 5 superpixels are classified as GCPs as shown in Figure 10.47 and all of them have similar hourglass shapes and dark colors.

A pre-trained sliding-window CNN can then be applied to these 5 positive patches to remove the false positives and the GCPs are correctly located. The same workflow is applied to the two new images shown in Figure 10.47b and 10.47c. The GCPs in these two images can be easily located from the final positive patches, respectively.

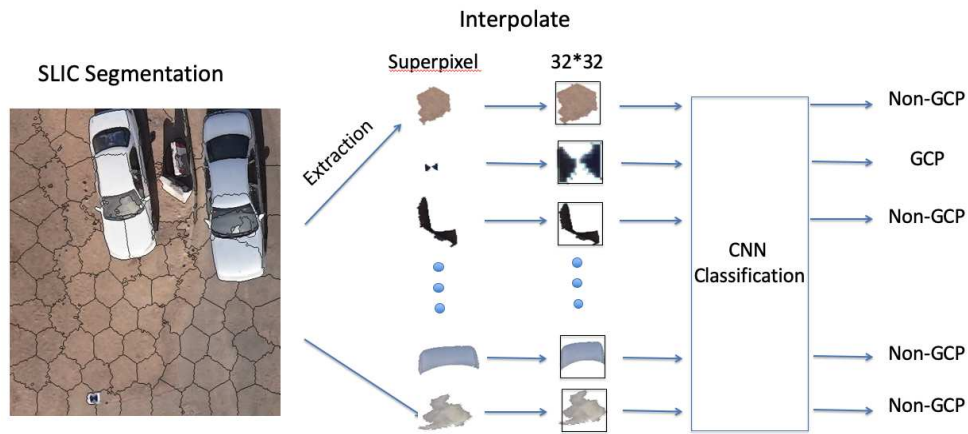


Figure 10.44: The workflow for the superpixel-based CNN method. The superpixels are extracted from the images using SLIC (see Appendix 10.19) and the superpixels are interpolated to be the same size for CNN classification. Then the superpixels are labeled by the CNN (Feng et al., 2021a).

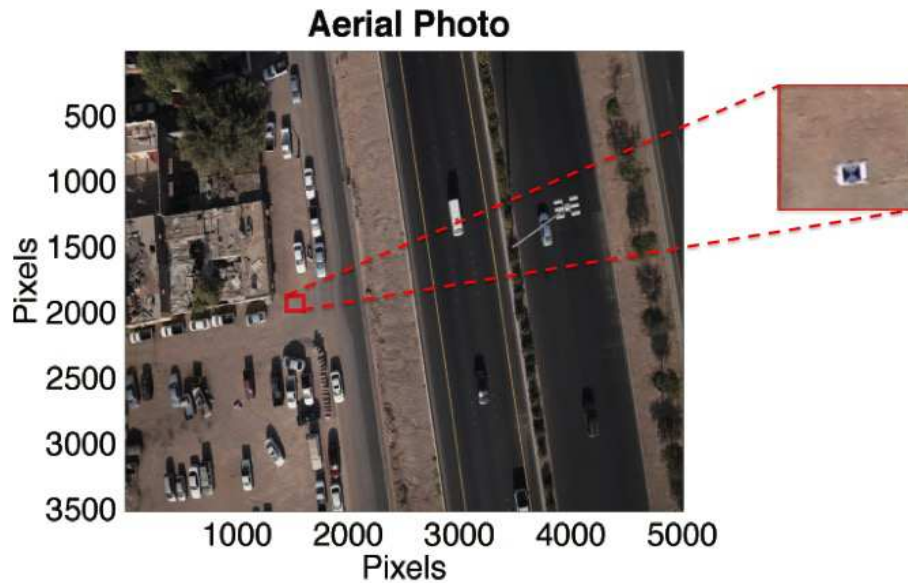


Figure 10.45: The input aerial photo. The red box indicates the location of the GCP. Image adapted from (Feng et al., 2021a).

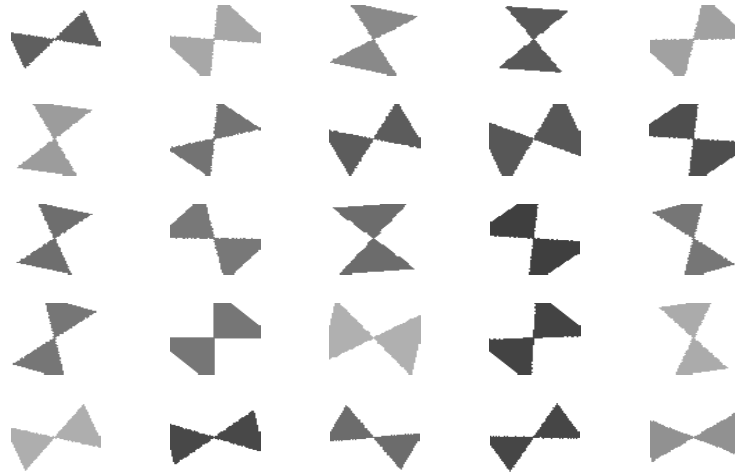


Figure 10.46: The synthetic positive patches created for superpixel-based CNN by rotating, shrinking and expanding a dark-color hourglass shape superpixel. Image adapted from Feng et al. (2021a).



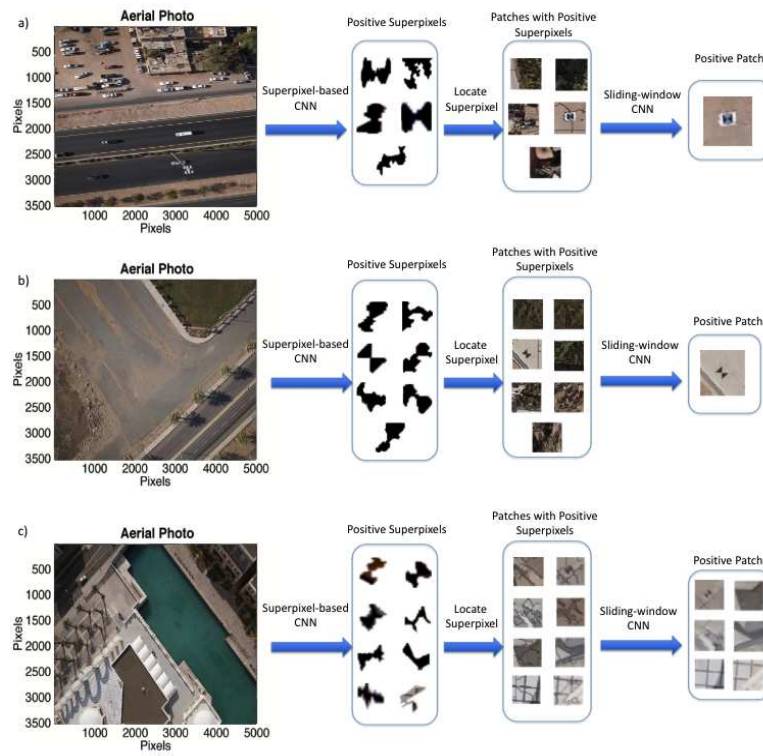


Figure 10.47: Superpixel-based and sliding-window binary classifications for a) Figure 10.45, b) test image 1 and c) test image 2. The positive superpixels are firstly found by superpixel-based CNN classification and the false positives among the positive superpixels are then removed by sliding-window CNN classification. Image adapted from (Feng et al., 2021a).

In the superpixel-based CNN, SLIC is applied to extract the unique shape and color of the GCP as superpixels and CNN is used to classify them. For our data, this procedure can quickly narrow the search to fewer than 10 superpixels labeled as GCPs. The false positives can be eliminated by an additional sliding-window CNN. The numerical tests indicate that this method is very efficient in locating the small GCP in a large drone image.

The success of the superpixel-based CNN depends on the efficacy of SLIC where all the pixels in the GCP must be partitioned into the same superpixel. However, there are several factors which can lead to an ineffective output from SLIC. The first factor is the presence of shadows in the drone image, where the color contrast of the GCP is not obvious when the GCP is located in the shadow shown in Figures 10.48a and 10.48b. In this case, the compactness variable in SLIC should be small so that it can detect small color contrasts. The second factor is the number of superpixels in SLIC. If the number is too small, the GCP may be partitioned into a large superpixel and the superpixel doesn't have a hourglass shape. If the number is too big, the GCP may be separated into several small superpixels as shown in Figure 10.48c and 10.48d. DBSCAN can merge these superpixel into a hourglass-shape superpixel. However, it will increase the diversity of the non-GCP superpixel and decrease the accuracy of the CNN classification.

Another limitation of this method is the high computational cost of SLIC for large images. To mitigate this problem, selective search (Uijlings et al., 2013) or a region proposal network (Ren et al., 2015) can be used as alternative to SLIC.

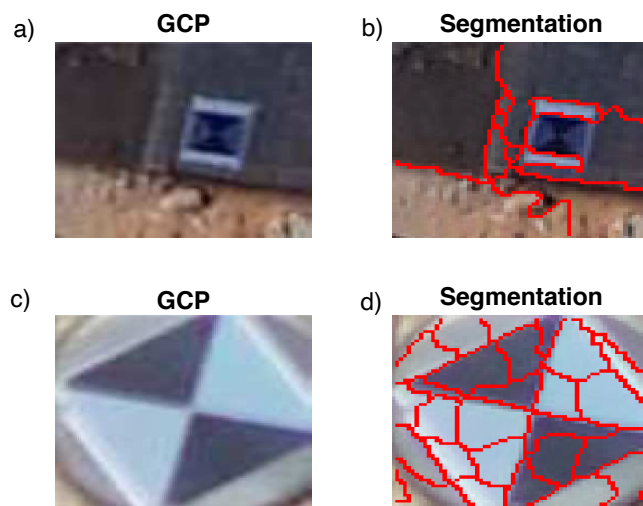


Figure 10.48: a) A GCP in the shadow and b) its segmentation. c) A big GCP and d) its segmentation. Image adapted from (Feng et al., 2021a).

## 10.12 Super-resolution Image Enhancement by CNN

Resolution improvement by CNN has led to an important class of enhancement techniques known as super-resolution (SR) enhancement of images (Guo et al., 2020; Han and Wang, 2020; Wu et al., 2021; Wurster et al., 2021). The goal of SR enhancement by CNN is to recover high-frequency details from a low-resolution image. One of the benefits of a successful SR method is that smaller-sized or low-wavenumber inputs can be computationally inexpensive to process or model, and the final

output can then be upsampled by SR to get the high-wavenumber result. This is much less expensive in terms of memory and computation costs compared to full processing of the high-wavenumber image. As an example, CNN has been shown to be a more efficient data compression scheme than JPG schemes (Wu et al., 2021).

Wang et al. (2021) provides a comprehensive survey of three categories of SR denoted as supervised SR, unsupervised SR and domain-specific SR. Figure 10.49 gives an overview of 4 CNN SR architectures. The iterative up-and-down sampling scheme (UDS) in d) resembles the encoder-decoder architecture of a U-Net, except UDS is iteratively repeated to correct for upsampling artifacts and does not have communication bridges. The upsampling innovation with CNN is that the transpose operations of a *deconvolution* (see Long et al., 2015) appear to be more accurate than the traditional upsampling methods such as bi-cubic interpolation (Wang et al., 2021). One of the benefits of this upsampling accuracy is that forward modeling simulations on coarse grids can be inexpensively computed, and then the SR procedure can upsample them to much higher accuracy. As an example, Li et al. (2021) and Ni et al. (2021) *...feed a machine learning algorithm with models of a small region of space at both low and high super-resolutions. The algorithm learns how to upscale the low-res models to match the detail found in the high-res versions of cosmological simulations.* (<https://www.sciencedaily.com/releases/2021/05/210504191602.htm>).

It is obvious that this kind of procedure can benefit many areas in geoscience imaging. One possibility is to use full-waveform inversion (FWI) for inverting small areas of the subsurface at high resolution, and larger areas at lower resolution. The ML algorithm can be trained to accurately upsample these areas to a common finer grid. Even an accurate downsampling might be possible, where the missing lower frequencies in the land seismic data prevents FWI from attaining good imaging quality. The ML algorithm can be trained to downsample seismic data to low-frequency data obtained from the well log. Then the recorded seismic data from a land survey can be downsampled to include the missing low frequencies. Similarly, the cost of 3D fluid flow modeling can be decreased by using this strategy of multiscale modeling and the training of a SR algorithm to upsample the super-resolution of the fluid-flow simulations.

### 10.13 Identification of Bird Types by CNN

Identifying and counting the bird types from photos taken over a long period of time can be beneficial for environmental studies of bird habitats. Long-term surveys by human observers are impractical, but they are possible if machine learning methods are coupled to thousands of weekly photos of bird environments (Mao et al., 2021). Such surveys can monitor, for example, how weather influences the living patterns of birds their migration schedules. Monitoring wildlife patterns in this way can be applied to any species of animals or insects.

To demonstrate the feasibility of using CNN to monitor the characteristics of birds, more than 280 photographs of birds were taken in the backyard of GTS (the primary author of this book). Fatme Ramadan (MS student at KAUST) and Shi Yongxiang (intern at KAUST, PhD student at Beijing University) devised a CNN model, described below, to detect the number and types of these birds.

The CNN models AlexNet and VGG16 are used to identify different bird types contained in 280 backyard photos of birds, some of which are shown in Figure 10.50. The goals of the CNN and its attendant lab in section 10 are the following.

- Employ AlexNet architecture for image identification.
- Apply the data augmentation technique to expand a dataset.
- Assess the effects of different hyperparameters on prediction accuracy.
- Employ transfer learning with the VGG16 network architecture.



### 10.13.1 Data Description and AlexNet model

The original dataset consists of 280 bird images taken in the backyard of GTS. The images belong to 6 different bird types, labeled Bird Type 1 to 6, described in the table below.

<i>Bird Type</i>	<i>Number of Images</i>
<i>Bird 0</i>	16
<i>Bird 1</i>	16
<i>Bird 2</i>	143
<i>Bird 3</i>	39
<i>Bird 4</i>	45
<i>Bird 5</i>	12
<i>Bird 6</i>	20

Here, bird type 0 is that for an unknown bird type, which is typically the case when there is no bird in the photograph. It is obvious that the input data are severely imbalanced in the number of different types of birds. Therefore, the pictures were balanced by data augmentation (see the right-side of Figure 10.50) where pictures were cropped, stretched, reflected or rotated to give a total of 650 pictures. There were about 90 pictures per bird type. Sample images of two different bird types and the corresponding labels are shown in the Figure 10.52. There are a total of 7 labels, where one of the labels is for, typically, photos that do not contain a bird.

There are 650 bird pictures, of which 20% were used for testing (130 images) and the rest were used for training (416 images) and validation (104 images). The following link: <https://colab.research.google.com/drive/1X7vRwDH6T-jN4BXFwXhtBxki0FWUQK-w?usp=sharing> connects to a Google Colab notebook that implements both the AlexNet (see the VGG16 architecture in Figure 10.51 for bird identification. Transfer training was employed with the VGG16 architecture.

### 10.13.2 Numerical Results

The results of training the AlexNet weights on the input data set are shown in Figure 10.52. Here, the accuracy of the training prediction at each epoch is in close agreement with the validation predictions, where the validation accuracy is 92.3% and the test data accuracy is 93%. This suggests that the model weights are not overfitted.

The top row of Figure 10.53 depicts the accuracy vs epoch plots using batch normalization (red) and not using batch normalization (blue) in the AlexNet architecture. Clearly, batch normalization is required for an acceptable convergence rate. The bottom row of plots depict the results using stochastic gradient descent (SGD) and the Adam (blue) step lengths. In this case, SGD presents a superior performance.

Different dropout rates were tested and the results in Figure 10.54a- 10.54b suggest that dropping out 70% of the weights is the lower limit one can accept. In this case there should be more than a 50% increase in computational efficiency with such a dropout rate.

Finally, transfer training was used as described in Figure 10.51, and the results are shown in Figure 10.54c- 10.54d. They show accuracy and loss factors that are nearly the same as those displayed for the AlexNet model in Figure 10.52. However, the transfer training model only had to train approximately  $12 \times 10^6$  weights associated with the green FCN's in Figure 10.52 compared to  $56 \times 10^6$  weights for the AlexNet model. For this example, transfer training reduced the computational costs by  $\approx 1/5$ .

## 10.14 Summary

Six types of computer vision tasks are defined: classification, semantic segmentation, classification+localization, object detection, and instance segmentation. Of these tasks, semantic segmentation seems to be one of the most important for seismic interpretation. However, this requires extensive labeling of images for training, which is quite tedious for tens of thousands of input examples. Therefore we might consider using efficient semi-automated labeling tools or unsupervised methods such as GANs.

Several examples were shown for labeling images with AlexNet, a simple and useful CNN method that starts with large images and contracts the size down to much smaller feature maps. AlexNet detects the class of the object but it can suffer from loss of spatial resolution because its architecture is that of a contracting encoder with maxpooling operations. This resolution can be improved by a U-Net architecture, which has both a contracting (encoding) part and an expanding (decoding part). Also, there is a communication bridge between the decoder and encoder networks that preserves the spatial information in the input image. This allows for precise location of the labels, compared to, e.g., AlexNet that only consists of an encoder network which loses location information after downsampling by maxpooling operations. The concatenation of feature maps from the contracting part to the expanding part tends to give location information of the object so as to get a precise location of the object. Results with U-Net detection of salt bodies and faults are consistent with this claim. These results are similar to those in the next chapter which successfully label cracks in a sandstone massif by the U-Net. U-Net is the current gold-standard for precise semantic segmentation of input images.

An example was provided for Faster R-CNN windowing and labeling many objects in a blood sample image. Such an algorithm can be used for object detection and labeling in many other scientific fields such as analysis of rock samples and aerial photos. If a small object is to be found in a large photo, then training can be successful by using the superpixel CNN described in the text. This allows data augmentation to be implemented to balance the training data.

## 10.15 Exercises

1. The intermediate terms in the softmax function might be very large due to the exponentials. This can lead to numerical instabilities, so it is important to use a normalization trick (<http://cs231n.github.io/linear-classify/#softmax>). If we multiply the top and bottom of the softmax fraction by a constant  $C$  and push it into the sum, we get the following (mathematically equivalent) expression:

$$\frac{C e^{f_i}}{C \sum_j e^{f_j}} = \frac{e^{f_i} + \log C}{C \sum_j e^{f_j} + \log C} \quad (10.11)$$

To improve stability a common choice for  $C$  is to set  $\log C = -\max_j f_j$ . Perform a numerical experiment to show that the softmax function can lead to numerical instabilities, and they can be mitigated with a properly chosen value of  $C$ .

2. Using the U-Net code listed in the computational labs, perform the following sensitivity tests for convergence rate and accuracy. Make sure the initialized weights are the same for all tests and convergence curves and accuracy values should be plotted as well.
  - **Sensitivity of convergence rate to batch normalization.** Run the U-Net code with and without batch normalization.
  - **Sensitivity to type of loss function.** Replace cross-entropy loss function with the means square error (MSE).

- **Sensitivity to unbalanced data.** Imbalance the data by using a preponderance of training examples with sediment only.
  - **Sensitivity to dropout percentage.** Try using 5%, 20% and 50% dropout rates.
3. Same tasks as the previous question except use AlexNet.
  4. Unbalance the data by only using a training set with 5% salt and the rest having no salt. Show how convergence rate slows down. Then read Lin et al. (2018) and <https://www.topbots.com/semanticsegmentationguide/> and attempt to balance the training data by overweighting the salt examples. Show results.
  5. Go to *Salt Detection with U-net Lab* in *LAB1/Chapter.CNN.Unet/U\_Net\_Lab.html* and run the Keras code. Change the initial weights for the U-Net subroutine in section 10.8.1 so that they are zero. This means that

```
# first convolution layer
x = Conv2D(filters = n_filters, kernel_size = (kernel_size, kernel_size),\
           kernel_initializer = initializers.he_normal(seed=1), padding = 'same')(input_tensor)
```

should be changed to

```
# first convolution layer
x = Conv2D(filters = n_filters, kernel_size = (kernel_size, kernel_size),\
           kernel_initializer = 'zeros', padding = 'same')(input_tensor)
```

6. For the previous exercise, test the sensitivity of convergence to the learning rate by commenting the "ReduceLROnPlateau(...)" line in

```
# Set input layer size
input_img = Input((im_height, im_width, 1), name='img')
# Set the parameters for Unet
model = get_unet(input_img, n_filters=16, dropout=0.05, batchnorm=True)
# Set optimizer, loss function. Set accuracy as metric
model.compile(optimizer=Adam(), loss="binary_crossentropy", metrics=["accuracy"])

#Visualize the structure of Unet
model.summary()

# Set criterias for the inversion procedure
# Early Stopping: If the validation set loss function does not decrease in
# 10 iterations, then early stop.
# ReduceLROnPlateau: If the validation set loss function does not decrease in 5 iteration,
# then multiply learning rate with factor.
# ModelCheckpoint: save_weights_only=True--Save only the weight,
#                  #save_best_only=True -- Only Save the best model with lowest validation loss
#                  #'model-tgs-salt.h5' -- file name

callbacks = [
    EarlyStopping(monitor='val_loss',patience=10, verbose=1),
    ReduceLROnPlateau(factor=0.1, patience=5, min_lr=0.00001, verbose=1),
    ModelCheckpoint('model-tgs-salt.h5', verbose=1, save_best_only=True, save_weights_only=True)
]
```

to become

```
callbacks = [
    EarlyStopping(monitor='val_loss',patience=10, verbose=1),
    # ReduceLROnPlateau(factor=0.1, patience=5, min_lr=0.00001, verbose=1),
    ModelCheckpoint('model-tgs-salt.h5', verbose=1, save_best_only=True, save_weights_only=True)
]
```

## 10.16 Computational Labs

1. Gimp labeling lab. Install `ja` [href=https://www.gimp.org/downloads/](https://www.gimp.org/downloads/) and go to *LAB1/Chapter.Neural/GIMP/GIMPlab.html*. Proceed to label photos per instructions.
2. Go to *Letter Classification by Convolutional Neural Network* lab in *LAB1/Chapter.Book.CNN/Chapter.CNNLetter/lab.html* and run the MATLAB code. It uses AlexNet CNN to classify the MNIST input images that are images of the digits 0-9. This is an example of a global classification of each picture, where each input picture represents an equation as in equation 10.1.
3. Go to *Identifying Rock Crack by Convolutional Neural Network* in *LAB1/Chapter.CNNCrack/CNN\_lab.html* and run the MATLAB code. It uses AlexNet CNN to classify the cracks in a photo of a rock. This is an example of pixel-sized classification of each pixel, where each pixel in a picture represents an equation as in equation 10.9.
4. Go to *Salt Body Identification by Convolutional Neural Network* in *LAB1/Chapter.Book.CNN/Chapter.CNNSalt/lab.htm* and run the MATLAB code. It uses AlexNet CNN to classify salt in a seismic migration image. This is an example of pixel-sized classification of each pixel, where each pixel in a picture represents an equation as in equation 10.9.
5. Go to *Salt Detection with U-net Lab* in *LAB1/Chapter.CNN.Unet/U\_Net\_Lab.html* and run the Keras code. It uses U-Net CNN to classify salt in seismic migration images. This is an example of pixel-sized classification of each pixel, where each pixel in a picture represents an equation as in equation 10.9.
6. Go to [https://colab.research.google.com/drive/1Ivil1U\\_BC0InKURUEMQ4BUZUZ8Q71Qdk?usp=sharing](https://colab.research.google.com/drive/1Ivil1U_BC0InKURUEMQ4BUZUZ8Q71Qdk?usp=sharing) and run the Keras code to detect faults in synthetic sections. It uses the fully-convolutional network described in Shi et al. (2018), and also employs upsampling *deconvolution* layers to assist in interpreting the feature maps.
7. Go to */LAB1/Chapter.CNN.RCNN/lab.html* and run the Keras code for R-CNN identification of cell types in blood samples. Plot out the training and validation accuracies as a function of iteration number. How much more training data must you use before you increase the accuracy by 10%. Read <https://tryolabs.com/blog/2018/01/18/faster-r-cnn-down-the-rabbit-hole-of-modern-object-detection/> for a description of Faster R-CNN. What CNN architecture does this code use?
8. Go to */LAB1/Chapter.Yolo/Yolo.html* and run the Keras code for YOLO2 identification of microfossils in microscope images. Plot out the training and validation accuracies as a function of iteration number. How much more training data must you use before you increase the accuracy by 10%.
9. Take Andrew Ng's course (<https://www.coursera.org/learn/convolutional-neural-networks>) on convolutional deep learning to better understand methods of object detection.
10. Go to <https://colab.research.google.com/drive/1YS8U8r69T5ON7ggbMFErE6Bi8KPPVQ7t?usp=sharing> and run the Colab Keras code to identify types of birds from pictures.

## 10.17 Appendix: Selective Search

Selective search is a method that searches for different objects in a sub-image and proposes the region of interest (ROI) for each object. For example, the karsts in Figure 10.1a are characterized by dark shadows and a texture not seen in other parts of the image. These karsts can be detected and classified by a sliding window approach

([https://results?search\\_query=sliding+windows+andrew+ng](https://results?search_query=sliding+windows+andrew+ng) and [https://www.youtube.com/results?search\\_query=sliding+windows+andrew+ng](https://www.youtube.com/results?search_query=sliding+windows+andrew+ng)).

For sliding windows, a window with different aspect-ratios and sizes is trained by a classifier to detect an object in the window. These windows are then applied to the test image and the classifier tells us if there is an object in the window. The problem with this approach is the huge computational cost in training a classifier for sliding windows with many sizes and aspect ratios.

A more practical approach is to extract hints from the image about the best bounding boxes. These hints can be found by looking for similarities in neighboring pixels, and grouping them as a potential object. These potential objects can then be combined with similar neighbors to become larger objects. This multiscale search is repeated until some similarity threshold can not be reached. In this case the BB of the potential objects can easily be found<sup>6</sup> and used as a proposed region of interest (ROI) in a test image. The BB coordinates for this ROI can then be used as the unknowns for training by a classifier; once trained, coordinates of a proposed ROI can be adjusted to agree with the groundtruth BB of a test image. This strategy is much more cost efficient compared to exhaustive search by the sliding window approach described in the previous paragraph.

An effective selective search strategy for object detection was proposed by van de Sande et al. (2011) and Uijlings et al. (2013). Their strategy looked for hints of objects by finding patches of pixels with strong similarities such as texture and color. They would then classify the similar pixels as one type, each associated with a BB. They would then grow the size of these regions by blending neighboring BBs with a similar similarity index. This repeated growth of BB size would stop once the neighboring patches were insufficiently similar to the current region. Once stopped the bounding boxes can easily be determined and used as the starting ROIs used for CNN training.

As an example, consider the picture in Figure 10.56f, where there are two green bounding boxes that enclose the woman and the television. The goal is to propose many smaller ROIs, and then gradually merge them to get a few that approximate the green boxes. Any extra that are left will eventually be eliminated by training the network. There are three steps for proposing ROIs by selective search (van de Sande et al., 2011; Uijlings et al., 2013).

1. **Propose hundreds or thousands of starting ROIs by breaking up image into sub-images.** This can be done by, for example, a clustering procedure<sup>7</sup> that proposes hundreds or thousands of cluster centers (see Chapter 17). Each cluster of pixels will be given a different color, as shown in 10.56a. A bounding box will be determined for each centroid, as illustrated in Figure 10.56b.
2. **Merge Clusters to Propose Larger Bounding Boxes.** Merge bounding boxes of clusters that are neighbors and have similar properties, such as texture, color, and size<sup>8</sup>. Figure 10.56c depicts the merger of smaller clusters into larger clusters, and Figure 10.56d describes the associated blue bounding boxes.
3. **Repeat** step 2 until the whole region becomes a single region. Figure 10.56e-10.56f depict the merged regions and bounding boxes at a later iteration.

See Figure 10.55 for an illustration and Figure 10.56 for a description of the selective search algorithm.

<sup>6</sup>Search for the maximum and minimum  $x$  and  $y$  coordinates of the BB for the contiguous pixels with the same similarity metrics.

<sup>7</sup>An alternative method is to use semantic segmentation.

<sup>8</sup>van de Sande et al. (2011) define the similarity of region  $a$  and  $b$  as  $S(a, b) = S_{size}(a, b) + S_{texture}(a, b)$ , where  $S_{size}(a, b)$  encourages small regions to merge into larger ones. The texture metric is defined in van de Sande (2010). Uijlings et al. (2013) incorporate the additional similarity metrics of color histograms and how well two ROIs fit within one another.

## 10.18 Appendix: MATLAB Code for AlexNet Crack Detection

The MATLAB code AlexNet detection is given below. This code is written by Shihang Feng and is used in the computational exercises that is a companion to this book.

```
clear all,close all

addpath ./CNN
addpath ./util
addpath ./data/

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                               Inputs                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ntrain=200; % The number of training set

wdx=8;
wdz=8;
image_x=wdx*2+1; % Original Image Patch Size X
image_z=wdz*2+1; % Original Image Patch Size Z
% Interpolate image to make sure size before pooling is even
image_x_new=32; % Interpolated Image Patch Size X
image_z_new=32; % Interpolated Image Patch Size Z

opts.alpha = 1; % Learning Rate
opts.batchsize = 20; % Minibatch Size
opts.numepochs = 10; % Iteration Number

%%%%% CNN structure %%%%%%%%%%
cnn.layers = {
    struct('type', 'i') %input layer
    struct('type', 'c', 'outputmaps', 6, 'kernelsize', 5) %convolution layer
    struct('type', 's', 'scale', 2) %sub sampling layer
    struct('type', 'c', 'outputmaps', 12, 'kernelsize', 5) %convolution layer
    struct('type', 's', 'scale', 2) %sub sampling layer
};
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                               Step 1 Prepare test and training set                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

load image_input.mat

%%%%%%%%%%%%% Load Prepared training set%%%%%%%%%%%%%
load train_new.mat

% %%%%%%%%%%%%%% Load Prepared training set%%%%%%%%%%%%%
% % Pick Train Examples
% % Pick Crack points
% figure(1)
% imagesc(image_input);
% title('Pick Crack Points','fontsize',22);
% train.label(1:ntrain)=1;
% train.label(2,1:ntrain)=0;
% [train.trainx(1:ntrain),train.trainz(1:ntrain)]=ginput(ntrain);
% pause(1)
% figure(2)
% imagesc(image_input);
%
% % Pick No-crack points
% title('Pick Non-Crack Points','fontsize',22);
% train.label(ntrain+1:2*ntrain)=0;
% train.label(2,ntrain+1:2*ntrain)=1;
% [train.trainx(ntrain+1:2*ntrain),train.trainz(ntrain+1:2*ntrain)]=ginput(ntrain);
```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                               Step 2 Convert Image into Image patches                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[nz,nx]=size(image_input);
image_pad=padimage(image_input,wdz,wdx);
%%% Obtain image batches from full image
image_slice=zeros(image_z,image_x,nz*nx);
image_slice_new=zeros(image_z_new,image_x_new,nz*nx);
[Xq,Zq]=meshgrid(1:(image_x-1)/(image_x_new-1):image_x,1:(image_z-1)/(image_z_new-1):image_z);
ii=1;
for ix=1:nx
    for iz=1:nz
        image_slice(1:image_z,1:image_x,ii)=image_pad(iz:iz+image_z-1,ix:ix+image_x-1);
        image_slice_new(:, :, ii)=interp2(image_slice(:, :, ii),Xq,Zq);
        ii=ii+1;
    end
end

%%% Obtain image batches from train set
train_slice=zeros(image_z,image_x,2*ntrain);
train_slice_new=zeros(image_z_new,image_x_new,2*ntrain);
[Xq,Zq]=meshgrid(1:(image_x-1)/(image_x_new-1):image_x,1:(image_z-1)/(image_z_new-1):image_z);
for in=1:2*ntrain
    train_slice(1:image_z,1:image_x,in)=...
        image_pad(train.trainz(in):train.trainz(in)+image_z-1,train.trainx(in):train.trainx(in)+image_x-1);
    train_slice_new(:, :, in)=interp2(train_slice(:, :, in),Xq,Zq);
end

clear image_slice train_slice

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                               Step 3 Train CNN structure                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Input Training and Test set
train_x =train_slice_new; % Training set
train_y =train.label;    % Training label
test_x =image_slice_new; % Test set

rand('state',0)
cnn = cnnsetup(cnn, train_x, train_y);
cnn = cnntrain(cnn, train_x, train_y, opts);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                               Step 4 Input full image for Classification                               %
%   (Separate The Image into 10 parts In case of out of memory) %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for ii=1:9
    disp(['Part ',num2str(ii),' Start']);
    uu = cnnff(cnn, test_x(:, :, 457*100*(ii-1)+1:457*100*ii));
    [vv,ll]=max(uu.o);
    ui=reshape(-ll+2,457,100); %Image-->457x100 patches
    un(:, 100*(ii-1)+1:100*ii)=ui(:, 1:100);
end

disp(['Part 10 Start']);
uu = cnnff(cnn, test_x(:, :, 900*457+1:977*457));
[vv,ll2]=max(uu.o);
ui2=reshape(-ll2+2,457,77);

```

```

un(:,901:977)=ui2(:,1:77);
disp(['Done...']);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure(1)
subplot(1,2,1)
imagesc(image_input)
title(['Input Image'], 'fontsize', 22, 'color', 'black')
set(gca, 'fontsize', 22)
xlabel('Pixels', 'fontsize', 22)
ylabel('Pixels', 'fontsize', 22)
subplot(1,2,2)
imagesc(un)
title(['CNN Labels'], 'fontsize', 22, 'color', 'black')
set(gca, 'fontsize', 22)
xlabel('Pixels', 'fontsize', 22)
ylabel('Pixels', 'fontsize', 22)
set(gcf, 'unit', 'centimeters', 'position', [12 5 30 13])
colormap(parula)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot the error curve
figure(2)
% plot(cnn.rL);
% title(['Misfit Curve'], 'fontsize', 22, 'color', 'black')
% set(gca, 'fontsize', 22)
% xlabel('Iteration', 'fontsize', 22)
% ylabel('Misfit', 'fontsize', 22)

```

## 10.19 Appendix: Simple Linear Iterative Clustering (SLIC)

Simple linear iterative clustering (SLIC) algorithm is a  $k$ -means clustering approach to efficiently generate superpixels from the image in the CIELAB color space (Achanta et al., 2012), where a superpixel is defined as a group of connected pixels with similar colors. CIELAB is a three-dimensional integer space  $(L^*, a^*, b^*)$  for the digital representation of colors, where  $L^*$  represents the lightness component. The lightness value  $L^*$  ranges from 0 to 100, where  $L^* = 0$  denotes black and  $L^* = 100$  denotes white. The symbol  $a^*$  represents the green-red component, with green in the negative direction and red in the positive direction. Similarly,  $b^*$  represents the blue-yellow color component, with blue in the negative direction and yellow in the positive direction. The  $a^*$  and  $b^*$  axes both range from -128 to 127. The clustering workflow for SLIC requires the following steps shown in Figure 10.57:

1. In the first step, the input image is partitioned into  $K$  approximately equally-sized superpixels. For an input image with  $N$  pixels, the size of each superpixel is about  $N/K$ . Hence, there is an initial superpixel center at every grid interval  $S = \sqrt{N/K}$  as shown in Figure 10.57a. The coordinates for the initial superpixel centers are  $[L_k, a_k, b_k, x_k, y_k]^T$  with  $k = \{1, 2, \dots, K\}$  at regular grid interval  $S$ .
2. If the initial center of a superpixel is placed on an edge or a noisy pixel, there are no pixels in the neighborhood belonging to the same cluster with the center. A noisy pixel is one in which the amplitude value varies by more than twice or less than half of the average amplitudes of the surrounding pixels. The magnitude of the image gradient is defined as:

$$G_{x,y} = \|\mathbf{I}_{x+1,y} - \mathbf{I}_{x-1,y}\|^2 + \|\mathbf{I}_{x,y+1} - \mathbf{I}_{x,y-1}\|^2, \quad (10.12)$$

where  $\mathbf{I}_{x,y}$  is the CIELAB vector  $[L_{x,y}, a_{x,y}, b_{x,y}, x, y]^T$  corresponding to the pixel at position  $(x,y)$ , and  $\|\cdot\|$  is the  $L_2$  norm. The calculation of the gradients takes into account both color and intensity information. If the center is located on the edge or a noisy pixel, its magnitude of the image gradient will be large. To prevent this situation, the magnitudes in the  $3 \times 3$  neighborhood around the initial



superpixel centers are calculated as shown in Figure 10.57b. Then the centers are moved to the locations corresponding to the points with the lowest magnitude.

3. The area of the search region for each superpixel center is  $2S \times 2S$  as shown in Figure 10.57c, where  $S$  is the grid interval. Each pixel  $i$  would be in the search regions of its surrounding superpixel centers and it should be associated with the nearest one. The dark green area in Figure 10.57c is composed of the pixels associated with the yellow superpixel center. Compared with the conventional  $k$ -means clustering that compares each pixel with all the clustering centers, this method greatly speeds up the algorithm by limiting the size of the search region.

The distance measure  $D_s$  between the pixel and superpixel center is defined as follows:

$$\begin{aligned} d_{lab} &= \sqrt{(L_k - L_i)^2 + (a_k - a_i)^2 + (b_k - b_i)^2}, \\ d_{xy} &= \sqrt{(x_k - x_i)^2 + (y_k - y_i)^2}, \\ D_s &= d_{lab} + \frac{m}{S} d_{xy}, \end{aligned} \quad (10.13)$$

where  $D_s$  is the sum of the CIELAB distance  $d_{lab}$  and the weighted  $xy$  plane distance, which is normalized by the grid interval  $S$ . A variable  $m$  is introduced for controlling the compactness of a superpixel.

4. After every pixel is associated with its nearest superpixel center, an update step adjusts new superpixel centers to be the mean  $[L, a, b, x, y]^T$  vector of all the pixels belonging to the superpixel shown in Figure 10.57d. The  $L_1$  norm is used to compute a residual  $E$  corresponding to the new superpixel center locations  $[L_{k_{new}}, a_{k_{new}}, b_{k_{new}}, x_{k_{new}}, y_{k_{new}}]^T$  and previous superpixel center locations  $[L_k, a_k, b_k, x_k, y_k]^T$ :

$$\begin{aligned} E &= \sum_k (|L_k - L_{k_{new}}| + |a_k - a_{k_{new}}| + |b_k - b_{k_{new}}|) \\ &+ \sum_k (|x_k - x_{k_{new}}| + |y_k - y_{k_{new}}|). \end{aligned} \quad (10.14)$$

5. Repeat steps 3 and 4 until the residual  $E$  is less than the specified threshold. This superpixel process is a cluster-like method.

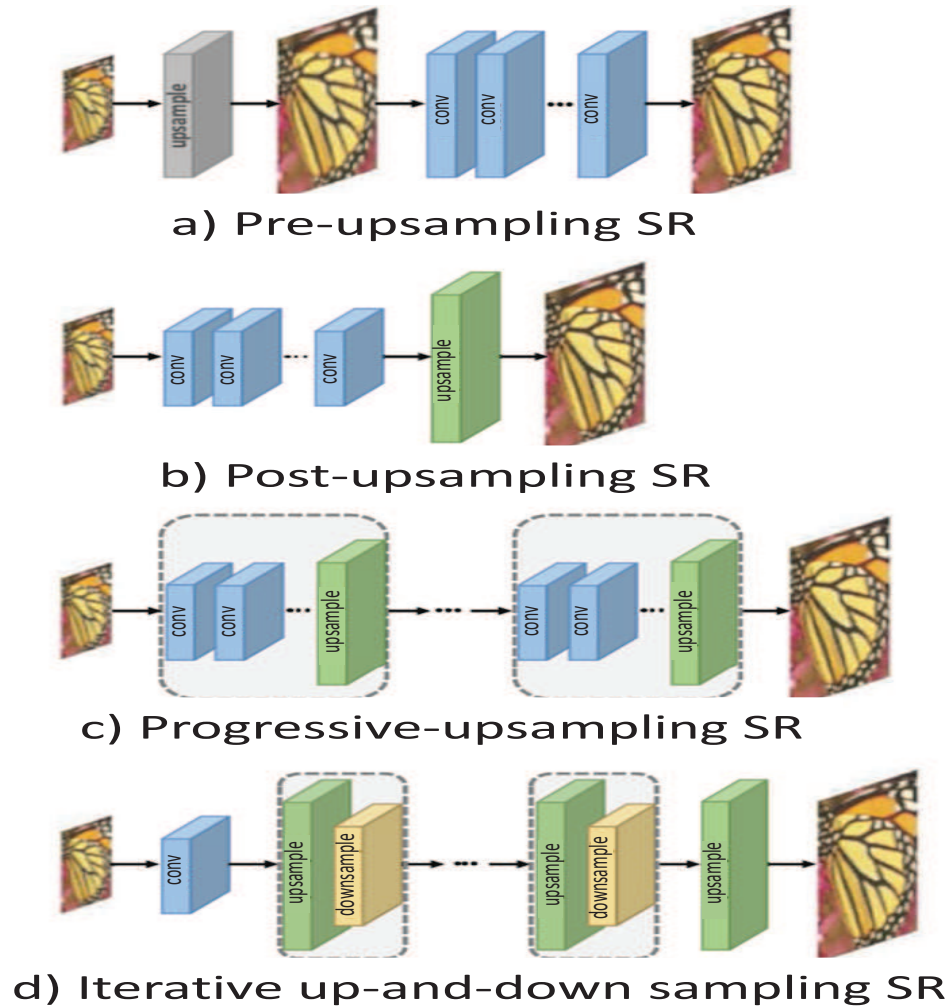


Figure 10.49: Super-super-resolution model frameworks based on deep learning. The cube size represents the output size. The gray ones denote predefined upsampling, while the green, yellow and blue ones indicate learnable upsampling, downsampling and convolutional layers, respectively. And the blocks enclosed by dashed boxes represent stackable modules. Figure from Wang et al. (2021).

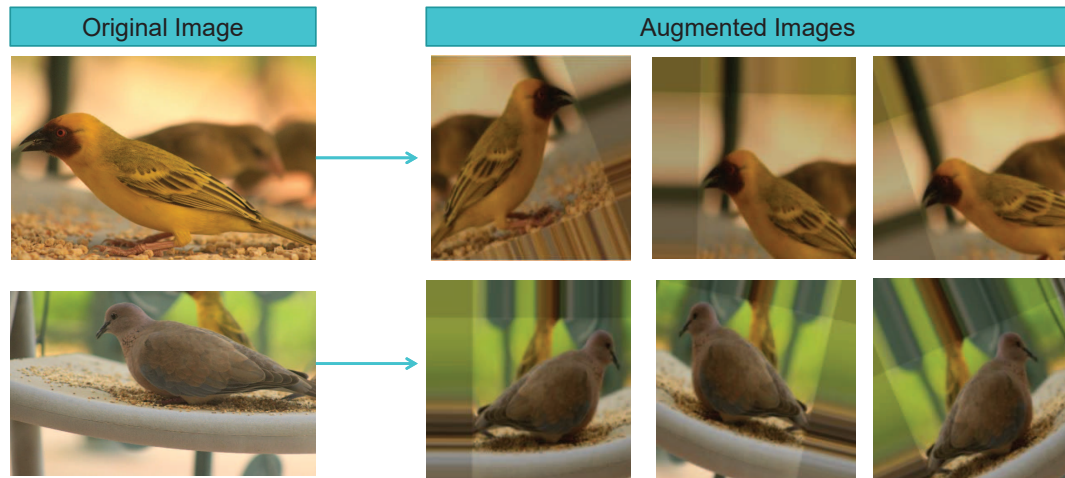


Figure 10.50: Two samples of the 280 bird pictures on the left, and some augmented (stretched, rotated, cropped) pictures on the right. Figure from Fatme Ramadan.

- We use as our base model the **VGG16** network and load the weights trained on ImageNet, a benchmark dataset for object recognition.

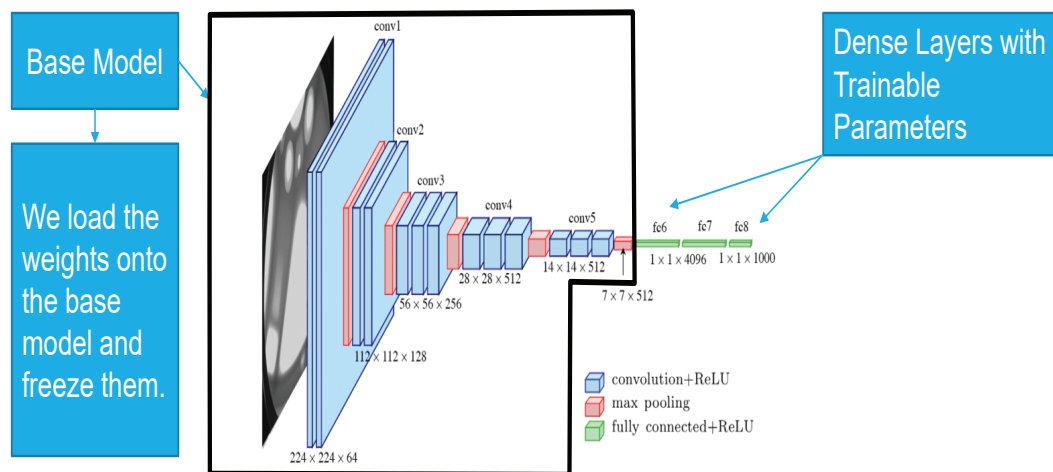


Figure 10.51: VGG architecture used for transfer learning. The weights for the architecture outlined by the black line were trained on a huge data set of images denoted as ImageNet. These weights were frozen, and the weights denoted by the green fully connected weights to the right of the black line were trained on the bird pictures. This is known as transfer training. Figure courtesy of Fatme Ramadan.

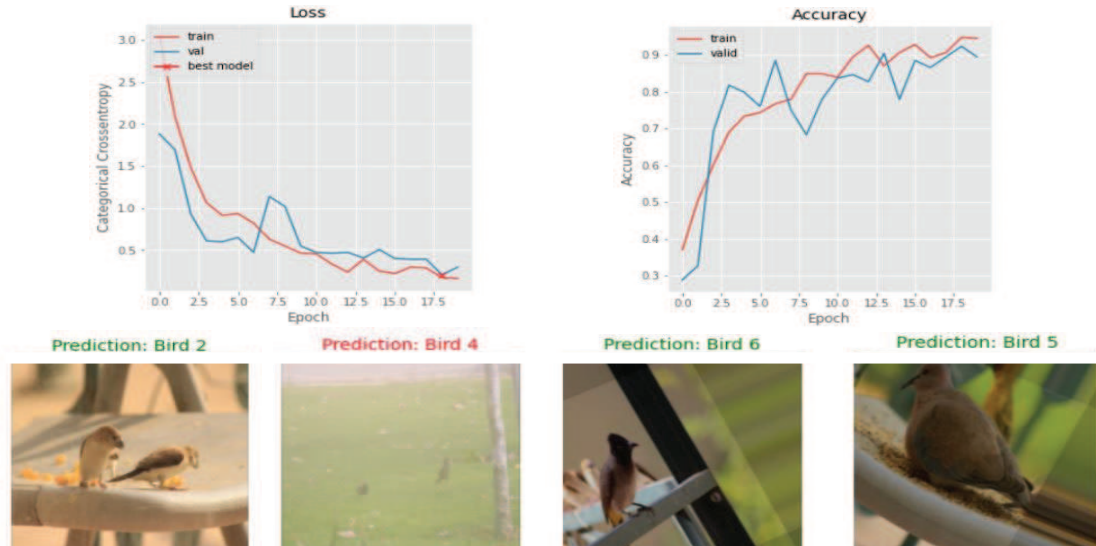


Figure 10.52: Loss and accuracy plotted against epoch number for the AlexNet prediction of bird types. The lower row of pictures are labeled with the predicted bird type, where the correct (incorrect) predictions are in green (red). Figure courtesy of Fatme Ramadan.

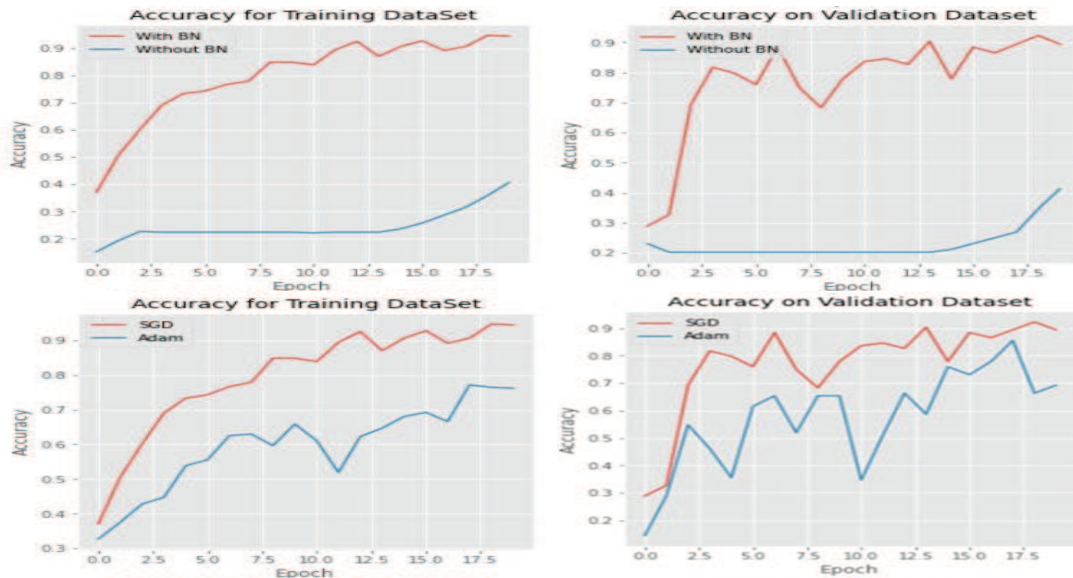


Figure 10.53: Accuracy vs epoch plots except top row is for training with (red) and without (blue) batch normalization (BN). The bottom row is for using an Adam (blue) and a stochastic gradient descent (red) step length. The left column of images is for the training data and the right column is for the validation data. Figure courtesy of Fatme Ramadan.

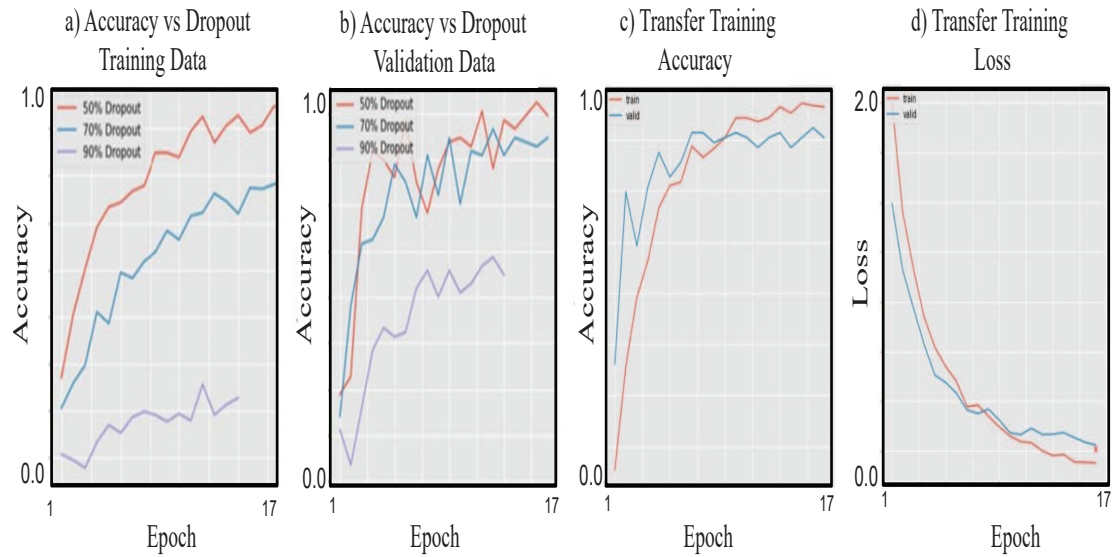


Figure 10.54: a) and b): Accuracy vs epoch plots for different dropout rates. c) and d) Accuracy and loss plots for transfer training. Figure courtesy of Fatme Ramadan.

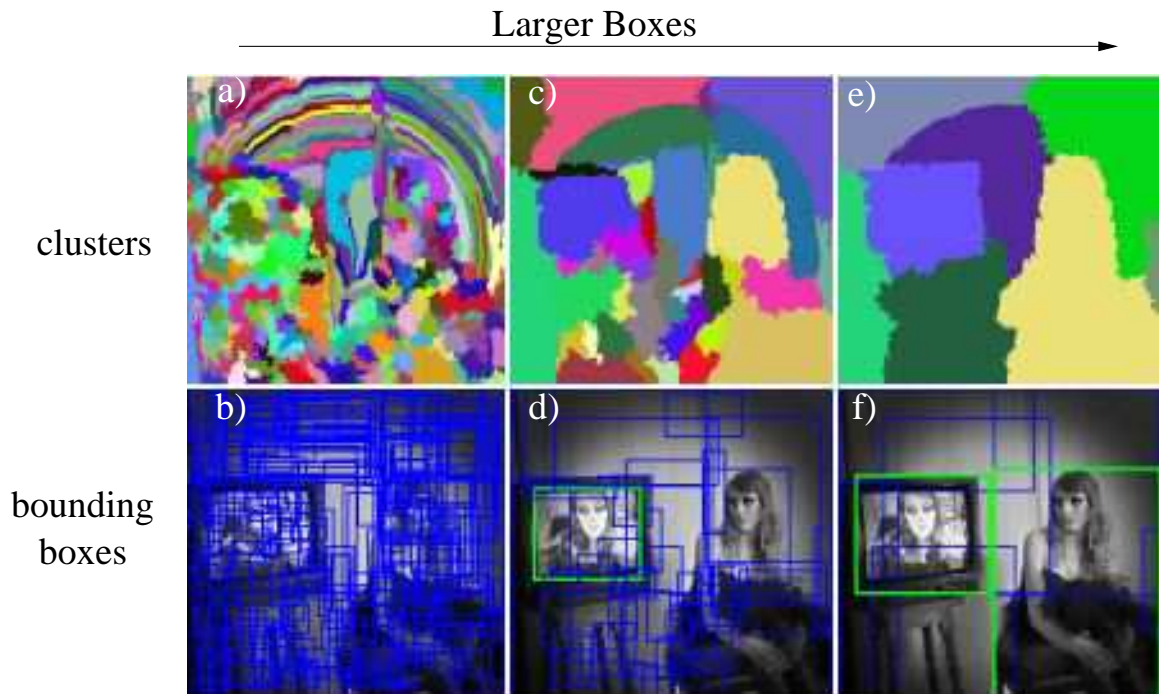


Figure 10.55: Photo in f) has two objects bounded by the green boxes (Felzenszwalb and Huttenlocher, 2004). Images a-f are the images and bounding boxes obtained by the selective search procedure described in Uijlings et al. (2013).



---

**Algorithm 1: Hierarchical Grouping Algorithm**


---

**Input:** (colour) image

**Output:** Set of object location hypotheses  $L$

Obtain initial regions  $R = \{r_1, \dots, r_n\}$  using [13]

Initialise similarity set  $S = \emptyset$

**foreach** *Neighbouring region pair*  $(r_i, r_j)$  **do**

    Calculate similarity  $s(r_i, r_j)$

$S = S \cup s(r_i, r_j)$

**while**  $S \neq \emptyset$  **do**

    Get highest similarity  $s(r_i, r_j) = \max(S)$

    Merge corresponding regions  $r_t = r_i \cup r_j$

    Remove similarities regarding  $r_i : S = S \setminus s(r_i, r_*)$

    Remove similarities regarding  $r_j : S = S \setminus s(r_*, r_j)$

    Calculate similarity set  $S_t$  between  $r_t$  and its neighbours

$S = S \cup S_t$

$R = R \cup r_t$

Extract object location boxes  $L$  from all regions in  $R$

---

Figure 10.56: Selective search algorithm described in Uijlings et al. (2013).

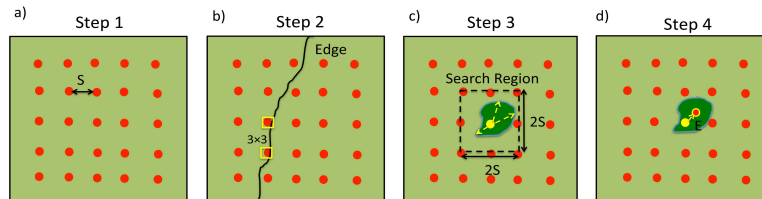


Figure 10.57: The workflow of SLIC. Step 1: initialize superpixel center with grid interval  $S$ . Step 2: move the centers away from the edge and noisy pixels. Step 3: cluster the pixels with a  $2S \times 2S$  search region. Step 4: Update the location of centers. Image adapted from (Feng et al., 2021a).

## Chapter 11

# Semantic Segmentation of Cracks in Photos

A modified version of the U-Net architecture for semantic segmentation is used to identify and label cracks in photos of sandstone massifs in the Middle East. The cracks occupy less than 1% of a photo, which leads to a huge number of CNN equations labeled as *non-crack*  $y = 0$  compared to the much smaller number of equations labeled as a *crack*  $y = 1$ . To avoid this unbalanced number of equations, each photo was segmented into approximately 300 sub-photos of much smaller size. These sub-photos concentrated the attention of the receptive fields around the large cracks and so significantly increased the ratio of crack-area/no-crack-area. In addition, a special weighting formula is used to overweight each equation associated with  $y = 1$  in the training data. The results lead to a more than 97% accuracy (by pixels number) in predicting cracks in the validation data.

### 11.1 Introduction

All solids weaken over time and develop a reduction in their mechanical strength. A sign of this weakness is the development of cracks, a quasi-linear physical separation of material on the surface and in the interior of a solid. For siting buildings on a rock foundation, assessing dam hazards, avoiding drilling hazards or for mining excavation, it is critical to assess the density and distribution of fractures in the rock mass. The weakening of the rock is due to many causes such as weathering, cooling after a period of heating, mechanical deformation due to stresses from local or tectonic stresses, and the weight of the overburden over time. Figure 11.1 shows overhead and side views of several rock massifs that will undergo development for tourism.

To develop this site, engineers must drill into portions of the rock massif to give a safety assessment of the rock's integrity. This integrity is related to the density of large cracks. To quantitatively estimate the density of cracks, more than 23,000 drone photos were taken of the two areas. Figure 11.2a shows a raw photo and Figure 11.2b depicts the cracks labeled in red by a human interpreter. There are too many photos to manually delineate cracks in a reasonable amount of time, and so a semantic segmentation algorithm is used to detect and label large cracks. Due to its outstanding performance in labeling salt boundaries and medical images, we selected the U-Net architecture shown in Figure 11.3 to detect the cracks.

### 11.2 Detection and Labeling of Cracks by U-Net

There are three phases to detecting the cracks in more than 23,000 photos (Shi et al., 2021): labeling cracks in fewer than 0.5% of the 23,000 photos, training the network on the labeled photos

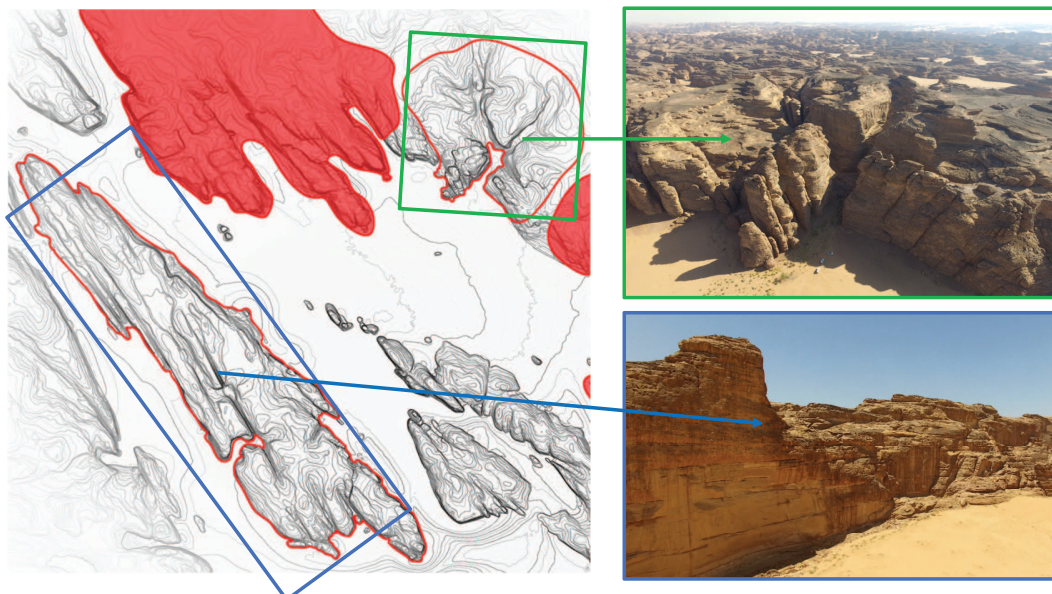


Figure 11.1: Topographic map and photos taken of the sandstone massif: the two structures of interest are referred to as the (bottom right) *Peak* (PE) and the (top right) *Flat* (FL). The horizontal width of the topographic map is about 1.5 km.

and then applying the trained network to detect and label cracks in the rest of the photos.

### 11.2.1 Labeling Cracks

The public-domain GIMP program (<https://en.wikipedia.org/wiki/GIMP>) was used to label cracks in  $194 = 127 + 67$  photos. We define a crack to be a separation between adjacent sides of a rock that typically appear as a dark shadow in the photographs. Only cracks longer than a meter were labeled, although this length estimate is a rough one because no absolute length scale is provided in the photos. The GIMP software allows the user to click between two points and then it draws a curve between the points. This was sufficient for a single user to label up to 10-20 photographs per day. Several groups of 50 or more photos were labeled, and then tests were conducted for training the U-Net. This experience suggested that we redo the labeling with thinner labels (about 6 pixels or so wide) and also provide a more complete labeling of all cracks. We mitigated the unbalanced-equation problem by replacing each photo with about 300 sub-photos dominated by cracks and by applying a  $9\times$  weighting to the equations with a labeled crack. An example of a labeled data set is shown in Figure 11.2.

**Phase I and Phase II Labeling.** There were two phases of labeling denoted as Phase I and Phase II. Phases I and II consisted of labeling and training of 127 photos and 67 photos, respectively. The Phase I labeling was primarily for the 57 photos of the open-side of the Flat facade and the peak (PE) facade (see images in Figure 11.1), while Phase II was for labeling and training for 67 photos of the top of the Flat. The other side of the Flat facade was not labeled because there was not enough room to closely photograph the walls with the drone.

It was found that training the network primarily for the facade cracks was insufficient for accurately labeling top-view cracks of the massifs. The top-view cracks were of a different nature than



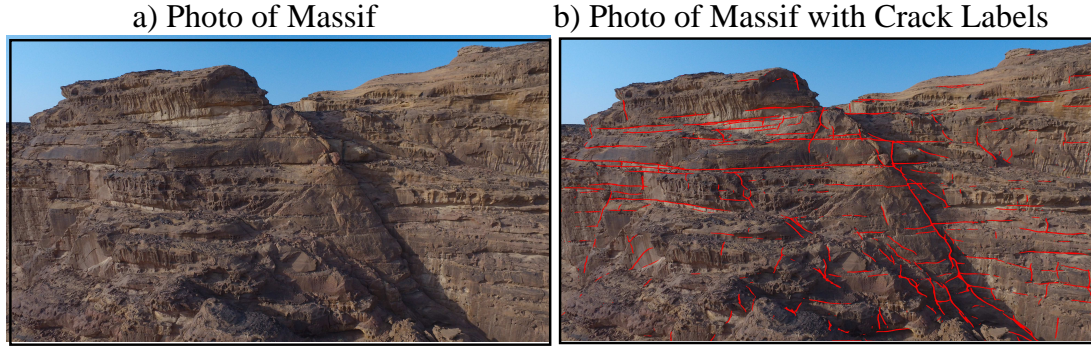


Figure 11.2: Facade and top-view photos before and after labeling of the cracks. Assortment of labels provided by Marco Ballesio, Yongxiang Shi and Dan Trentman.

those for the facade. This prompted the initiation of the labeling and training of 67 photos of the top of The Flat.

### 11.2.2 Phase I Training and Validation

A random sampling of  $118+9=127$  training photos was drawn from 23,000 photos of the sandstone complex taken by a drone; 118 photos were used for training and 9 for validation. There were two training sets, one set for side-views (aka facade-views) of the PE massif and the Flat (FL) facade, and one set for the their top views. For PE massif, there were 45 training photos for the facade pictures and 33 photos for the top-view photos. For the FL massive, we selected 9 facade photos and 31 top-view photos. The number of photos for training was determined by trial-and-error until our validation set achieved a prediction accuracy of more than 97%. Here, accuracy is defined as

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}, \quad (11.1)$$

where  $TP=\#$  of True Positives,  $TN=\#$  of True Negatives,  $FP=\#$  of False Positives and  $FN=\#$  of False Negatives<sup>1</sup>. The training was processed separately for the side and top views.

The U-Net architecture in Figure 11.3 was selected for the CNN. The output is a binary label per pixel for either a crack  $y = 1$  or non-crack  $y = 0$ . The zoom-view of the architecture for the individual convolutional blocks shown in Figure 11.3 is constructed for the following purposes:

- Additional convolutional branches: Most of the cracks in drone images are approximately horizontal or vertical, which means they are very long in one dimension and short in the other. Two additional convolutional branches with filter sizes  $1 \times 9$  and  $9 \times 1$  can help the network extract crack features by focusing on a specified dimension, without increasing many parameters compared to a square filter.
- Residual part: Our network depth is 40 layers, including 36 convolutional layers and 4 transposed layers. The skip connections in each block can mitigate difficulties with a vanishing gradient.

There were 54/64 photos used for the facade/top-view training sets. The area of the cracks in the photos comprised less than 1% of the each photo, which leads to an unbalanced set of equations

<sup>1</sup>Most parts in a photo consist of non-cracks, so the number of pixels without a crack is much bigger than the number of crack pixels. Therefore a high accuracy number is more indicative of accurately detecting non-crack pixels.

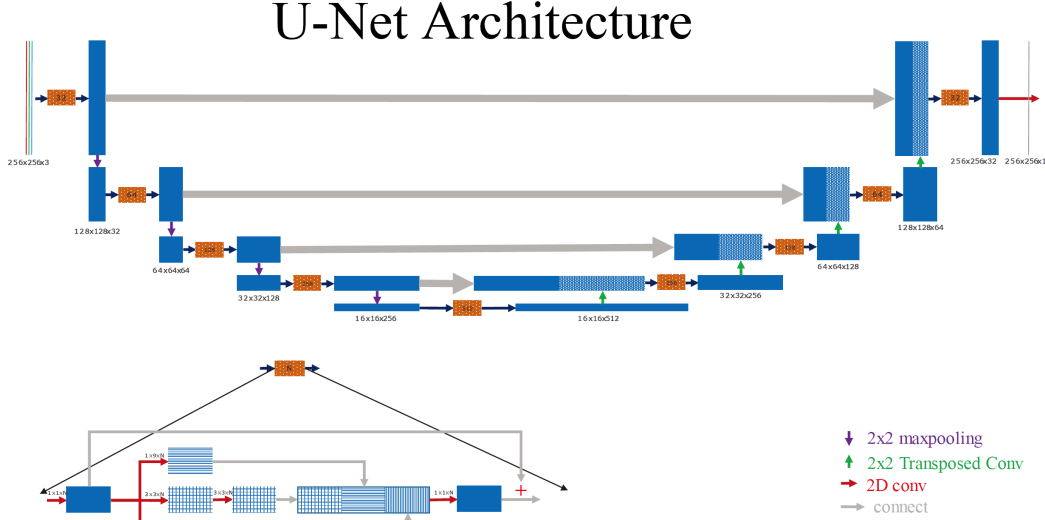


Figure 11.3: U-Net-like architecture of the CNN used for detecting cracks in photos of the sandstone massif. The blue arrows point to a new convolutional layer, the gray arrows indicate skip layers, and blue boxes denote feature maps. The *height*  $\times$  *width*  $\times$  *channel number* of the feature maps are indicated below the blue boxes (Ronneberger et al., 2015).

where most of the pixels have a label of  $y = 0$ . This will lead to poor convergence and large errors in inference labeling. There are many schemes for mitigating the problems of an imbalanced system of multiclass equations where, in pixel-level classification, the number of equations for one or more classes greatly exceed those for other classes. This is the case for cracks which occupy an area that is less than a few percent of the background area. To mitigate the imbalance problem, schemes such as the one proposed by Guerrero-Pena et al. (2018) who segmented the boundaries of cells by the using weights associated with the location, length of touching borders, and the geometry of cell contours. Another approach is to mask each object using R-CNN (He et al., 2017) and then apply a FCNN or CNN to detect the localized objects of interest.

To mitigate the problem of an unbalanced system of equations we implemented two remedies.

- Each  $3000 \times 4000$  photo is segmented into approximately 300 sub-photos of much smaller size of  $256 \times 256$ . These sub-photos concentrate the attention of the receptive fields around the large cracks and so significantly increased the ratio of crack-area/no-crack-area.
- We overweighted the equations in favor of the crack  $y = 1$  equations by a multiplicative factor 0.9 and underweighted the no-crack  $y = 0$  equations by 0.1. This also mitigated the unbalanced-equation problem.
- The adjusted binary loss function used for training is

$$Loss(y, p) = - \sum_i [0.9 \times y^{(i)} \ln(p^{(i)}) + 0.1 \times (1 - y^{(i)}) \ln(1 - p^{(i)})], \quad (11.2)$$

where the summation in  $i$  is over the indices of the examples.

Figure 11.4 depicts the loss versus iteration number for the facade and top-view images. Both

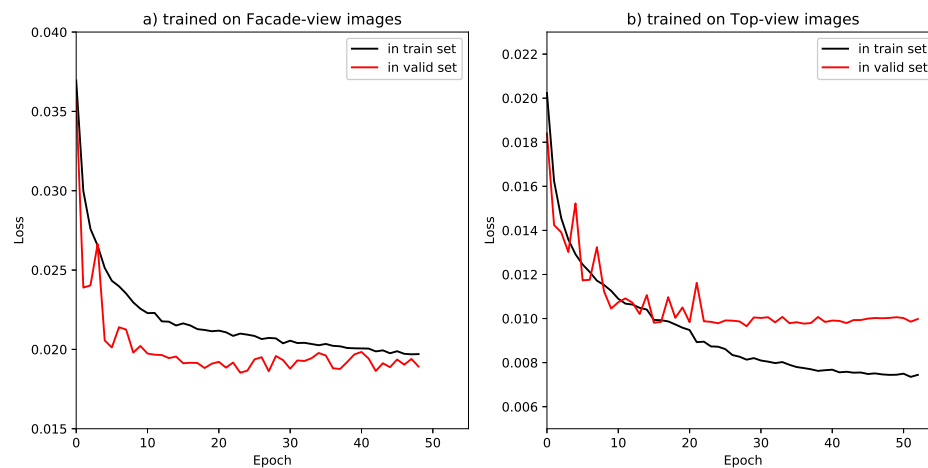


Figure 11.4: Loss function versus iteration number for the training and validation data. Here the training data are for the a) facade and b) top-view photos.

training and validation results show a crack prediction accuracy of 96% and 98%, respectfully. See equation 11.2 for the definition of accuracy.

The confusion matrices associated with the training and validation data for massifs PE and FL are shown in Table 11.2. The metric for estimating the error between a predicted and actual crack is defined in Table 11.1.

100 pixels/m		Actual	
		Crack	Non-Crack
Prediction	Crack	TP	FP
	Non-Crack	FN	TN

Table 11.1: Confusion matrix: TP=True Positive, TN=True Negative, FP=False Positive, and FN=False Negative. In Table 11.2, TN is removed and the human labels and machine label predictions are skeletonized to have a width of one pixel using the morphological processing program "skimage.skeletonize" ([https://scikit-image.org/docs/dev/auto\\_examples/edges/plot\\_skeleton.html](https://scikit-image.org/docs/dev/auto_examples/edges/plot_skeleton.html)). In this way we discount the prediction of the crack thickness and only examine its location accuracy. If the true and predicted labels are within 20 pixels of one another, then we consider this to be a true prediction. We then display the total length of the predicted cracks in units of meters. For example, the total length of correctly predicted cracks in Table 11.2 is 250 m. In the drone photos, we assumed that the pixel width is 1 cm.

### 11.2.3 Phase I Testing

After adequately training the two U-Nets, they were applied to the remaining photos to label the cracks. The results show good qualitative agreement between the cracks detected by a human observer and those delineated by the U-Net. As an example, Figure 11.5 shows the photos of massif

100 pixels/m		Top Crack	Top Non-Crack	Facade Crack	Facade Non-Crack
Prediction	Crack	250 m	540 m	480 m	583 m
	Non-Crack	53 m		75 m	

Table 11.2: Confusion matrices of the facade (3 photos) and top-view (6 photos) validation sets. In the above table, TP=250 m (480 m) of top-view (facade) cracks were correctly predicted, and 53 m (75 m) of top-view (facade) cracks were incorrectly predicted. Therefore,  $250/303 \cdot 100\% = 83\%$  ( $86\%$ ) is the crack-prediction accuracy where the number for TN is excluded to eliminate an overly optimistic number for accuracy. Here, the false-positive number, i.e. the number of false positives, is also ignored.

PE before and after crack detection. Comparison between the left and right photos mostly show good agreement between the inferred major cracks and the ones observed with the eye. These photos could provide accurate information for engineers to accurately estimate rock integrity related to large cracks.

#### 11.2.4 Phase II Training and Validation

Examination of the predicted cracks for the top-view photos showed an unacceptable accuracy. Therefore, 61+6=67 top-view photos of the the Flat facade were re-labeled with an aggressive definition of a crack<sup>2</sup>; 61 photos were used for training and 6 for validation. To improve accuracy, we used a more aggressive interpretation of what constitutes a crack, as can be seen in the original labeling and new labeling of a top-view photo in Figure 11.6. In this example, the crack equations were weighted, after empirical testing, by the multiplicative factor of 8.3 to account for the imbalance of crack and non-crack equations.

A total of 90 epochs were used for the U-Net training from the Phase II data, and the loss function for training and testing is plotted in Figure 11.7. To better compare the results from Phase I and II training sets, we define the Precision and Recall values in Table 11.3. A higher *Recall* value is desired because it means that there are more *True Positives* compared to the total number of *True Positives+False Negatives*. It is more important to identify too many cracks than to miss some important ones where FN is large.

Table 11.4 shows that the recall value of the Phase II data is  $R = 79\%$  compared to  $R = 65\%$  for the Phase I top-view data. This suggests that the aggressive relabeling of the top-view photos resulted in a more accurate prediction of top-view cracks. However, the Phase II retraining produced a Precision value of  $P = 44\%$ , which is lower than  $P = 55\%$  for the Phase I data. This means that the new network identifies more cracks in our new labeled photos, but it brings more noise (false positive) into our results.

A similar improvement in accuracy prediction was obtained when applying the two U-nets to 61 top-view photos from the Phase II data. Here, the U-Net trained on the Phase II data achieved  $R = 83\%$  compared to  $R = 62\%$  for the Phase I U-Net.

#### 11.2.5 Phase II Testing

Figure 11.8 shows a few test photos where the predicted cracks from the Phase II data show a higher density compared to those from the original Phase I data. These Phase II predictions mostly appear to be a reasonable interpretation of the cracks.

<sup>2</sup>Dark skinny elongated shadows in a photo were labeled as cracks with the new interpretation of a labeled crack. This resulted in a higher density of cracks in the photos.



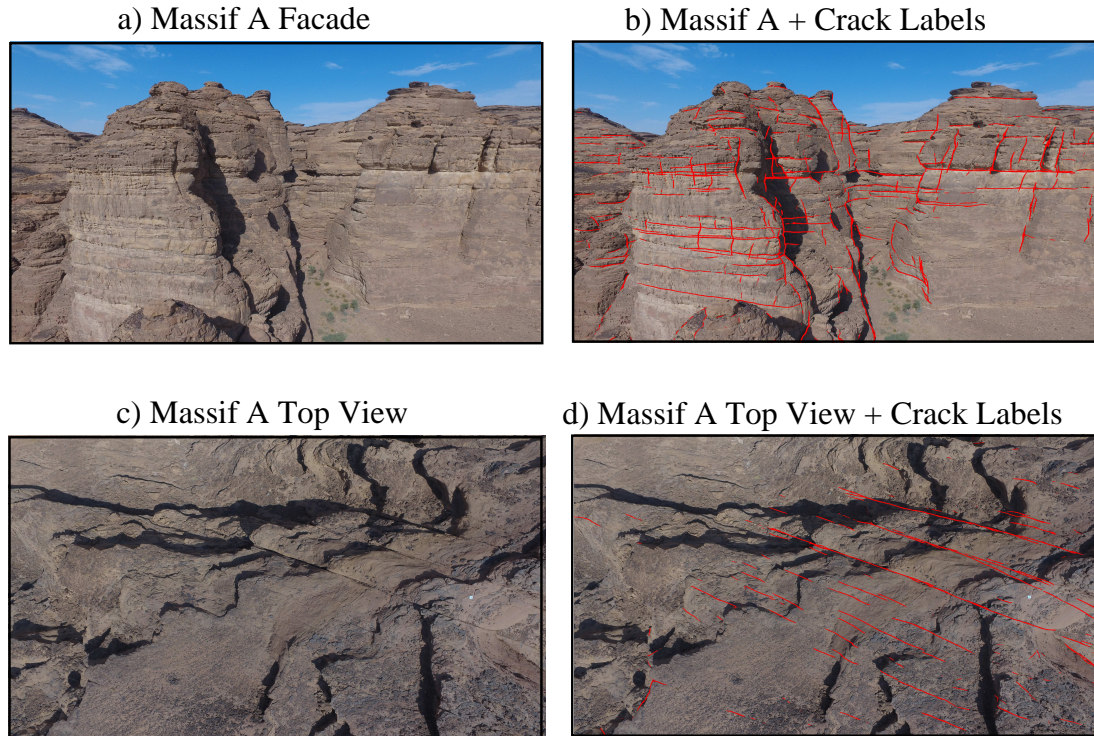


Figure 11.5: Photos of massif PE before and after labeling of the cracks by U-Net.

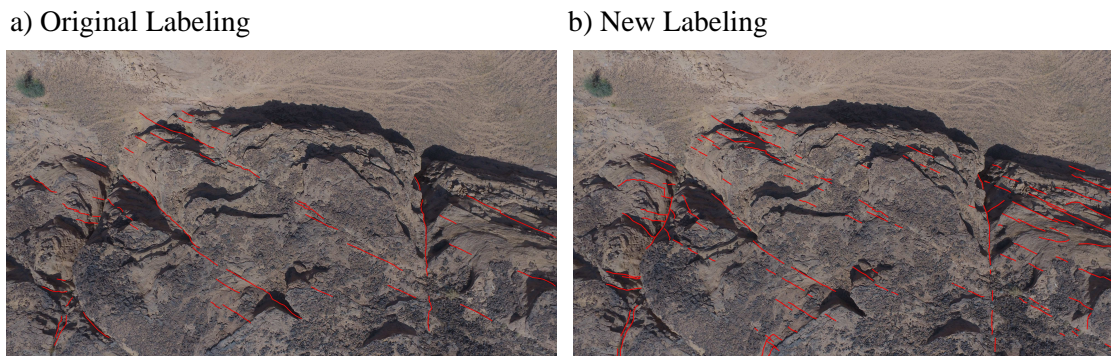


Figure 11.6: a) Original and b) new labeled top-view photos of The Flat. The new labeled photos were used for retraining the network to improve the crack prediction accuracy. The new labeling more aggressively labels a crack as indicated by the greater number of crack labels in b).

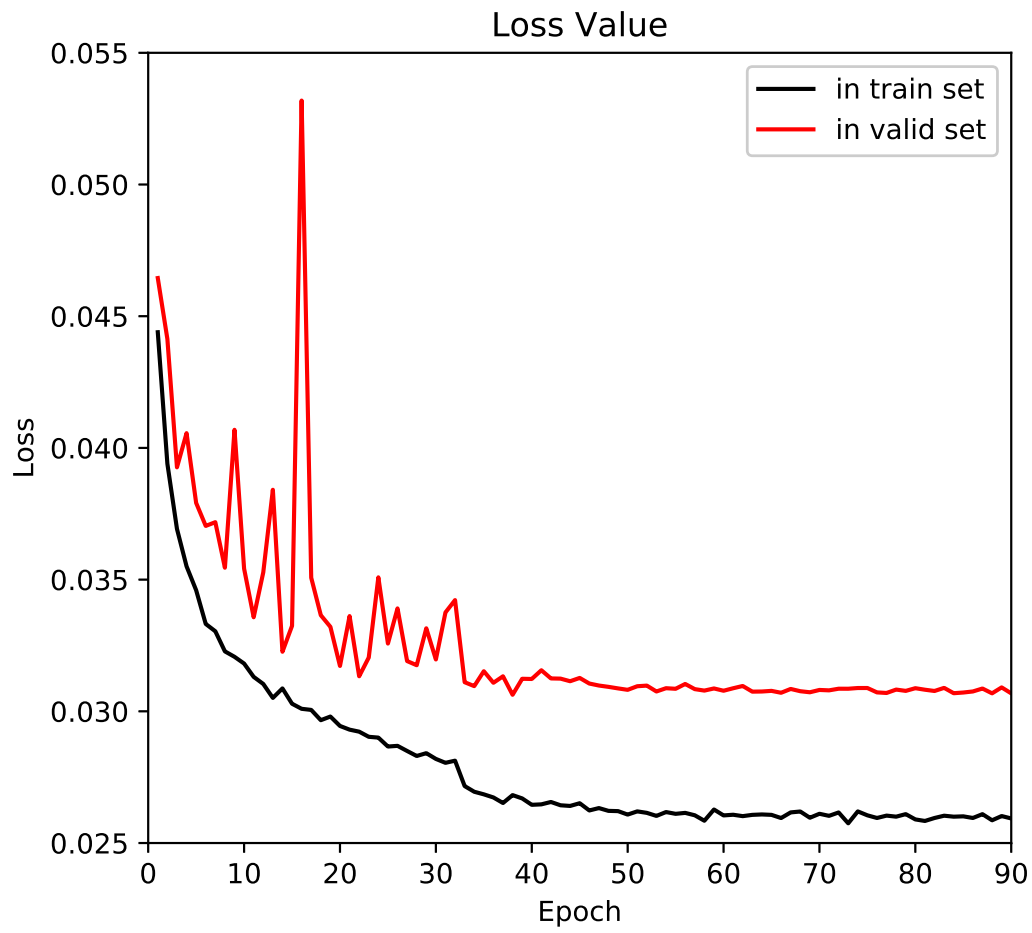


Figure 11.7: Loss function vs epoch number for the Phase II training.

a) Phase I Predictions

b) Phase II Predictions

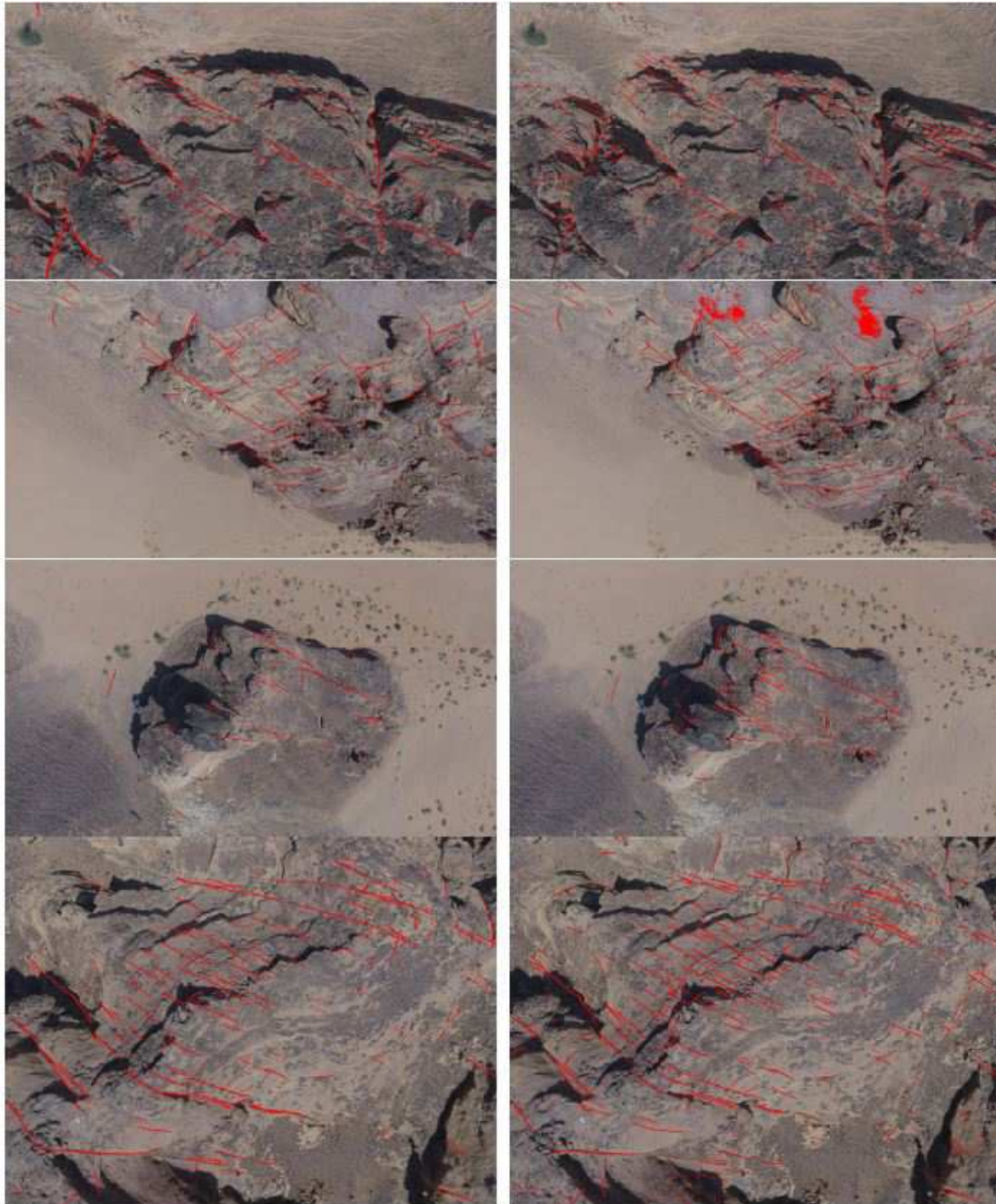


Figure 11.8: Photos of the top of The Flat predicted by the U-Net after being trained by the a) Phase I and b) Phase II training data. The large red blob on the photo in the second row is a correctable error introduced during training.



		Actual		
		Crack	Background	
Predict	Positive	TP	FP	$P = \frac{TP}{TP+FP}$
	Negative	FN	TN	
		$R = \frac{TP}{TP+FN}$		

Table 11.3: Confusion matrix and definitions of  $Recall = \frac{TP}{TP+FN}$  and  $Precision = \frac{TP}{TP+FP}$ .

Phase I Data	Crack	Background	
Positive	51 627	40 565	P=55%
Negative	26 909		
	R=65%		
Phase II Data	Crack	Background	
Positive	62 708	76 575	P=44%
Negative	15 828		
	R=79%		

Table 11.4: Confusion matrices for the validation set of 6 top-view photos. The U-Net trained on the Phase I data was used to compute element values in the top confusion matrix while the U-Net trained on the Phase II data was used to compute the element values in the bottom confusion matrix.

Phase I Data	Crack	Background	
Positive	436 440	429 439	P=50%
Negative	263 782		
	R=62%		
Phase II Data	Crack	Background	
Positive	583 447	786 560	P=42%
Negative	116 775		
	R=83%		

Table 11.5: Same as previous table except the U-Nets were applied to the 61 photos relabeled for the Phase II data.

### 11.3 Transfer Training

Transfer training is the reuse of the weights of a trained neural network model for labeling a completely different data set. As an example, the sandstone U-Net weights for detecting cracks in the sandstone massifs can be used to detect cracks in a volcanic section near a dam in Idaho, USA. Applying the sandstone U-Net model to the Figure 11.9a picture gives the detected cracks in the Figure 11.9b image.

For transfer training, we freeze the weights in blocks from 3 to 7 and allow four symmetric blocks (block 1,2 and 8,9 in Figure 11.3) to be trained on the newly labeled photos. This decreases the trainable parameters from 20,000,000 to 730,000. An Adam optimizer is used and we set the learning rate to be  $10^{-4}$  in order to fine-tune the trainable layers. The batch size is 5 and the total number of  $256 \times 256$  sub-photos is 120. The transfer training is halted at epoch 30 when the accuracy flattens out at 97%, which takes no more than 20 minutes of computation time on a workstation. We



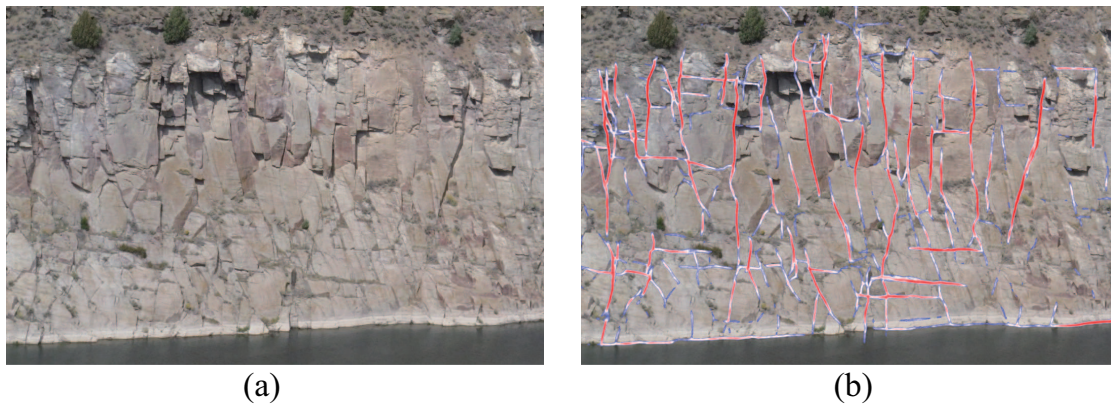


Figure 11.9: Photos of Idaho volcanic rocks a) before and b) after application of the sandstone U-Net model. The red markings correspond to a probability between 0.7 and 1.0 in the softmax output, while the white-bluish markings correspond to a probability between 0.3 and slightly less than 0.7.

denote this U-Net as the hybrid U-Net model because the original weights were trained on images of sandstones and adjusted to those for volcanic rocks.

After transfer learning, the sandstone U-Net and hybrid U-Net were applied to the top row of raw  $4000 \times 2000$  images in Figure 11.10a to give the labeled cracks in Figure 11.10b and Figure 11.10c, respectively. Figure 11.10c shows a much denser and more accurate collection of labeled cracks than the ones in Figure 11.10b. The important accomplishment is that it required no more than an hour of manual labeling to achieve this goal.

## 11.4 Summary

Semantic segmentation was used to detect and label large cracks in drone photos of rock massifs. The U-Net CNN was selected because of its excellent accuracy in labeling medical images. Results showed that the U-Net could be trained with the Phase I data of 127 photos so that it achieved an accuracy of up to 97% in the validation data. Inspection of the results for the open-side of the PE facade showed acceptable accuracy for engineering purposes.

However, the crack prediction of top-view photos in the Phase I data did not achieve an acceptable accuracy. To improve these results, 67 top-view photos were obtained for the Phase II data. These data were relabeled with an aggressive definition of a crack that resulted in relabeled photos with a higher density of labeled cracks. For 6 validation photos, results showed a  $R = 79\%$  *Recall* value compared to the  $R = 65\%$  for the Phase I data.

The one big improvement in detecting cracks is to increase the size and accuracy of the training data. Thus, we recommend that a Generative Adversarial Networks (GANs) or variational autoencoder (VAE) be used to generate a much larger set of labeled data.

## 11.5 Computational Labs

1. Go to *Crack Labeling* lab in *LAB1/Chapter.Book.GIMP/lab.html* and run the GIMP and MATLAB codes for labeling.
2. Go to *Crack U-Net training* lab in *LAB1/Chapter.Book.GIMP/lab1.html* and run the Keras codes for training a U-Net CNN.

a) Photos of the Dam Site



b) Processed with U-Net Trained with AIUla Weights



c) Processed with U-Net after Transfer Training with AIUla Weights



Figure 11.10: a) Raw photos from the Teton dam site, and crack labels computed by the b) AIUla sandstone U-Net and the c) hybrid U-Net models. The red labels have a softmax probability between 0.7 and 1.0.

3. Go to [https://colab.research.google.com/drive/1n7K1esyM\\_FmWb-6ZwT8eGWxylOk4xoOp](https://colab.research.google.com/drive/1n7K1esyM_FmWb-6ZwT8eGWxylOk4xoOp)  
This lab allows the user to train a U-Net Network for crack detection and transfer learning. We provide 57 photos with their labels to generate a training set for crack detection by U-Net. There are two more photos from the dam site for transfer learning. The default setting is to perform transfer learning.



## Chapter 12

# Recurrent Neural Networks

When we are talking to someone, the current word in our spoken sentence is strongly dependent on many of the previous words or phrases. For example, the phrase "Jill is ..." might be followed in the next instance by the word "healthy". In this case, the object "healthy" grammatically follows the noun-verb structure of the sentence, so it is highly correlated to the previous two words. In addition, Jill's state of health might have been discussed in the previous sentence, which is a clue for predicting the word "healthy". Therefore, writing or speech<sup>1</sup> can be considered as a highly correlated time series where each word at time  $t$  is correlated to a few or many of the previously spoken words/phrases. This is unlike the typical neural network used for image classification where the intensity values of pixels in an input image are not necessarily correlated to the next image in the training set. In this case, each image is similar to a very long word that has little correlation with the sequence of other input words, i.e., images.

To account for the correlation characteristics of, for example, speech, writing, film or music we have the recurrent neural network (RNN) which assumes that the current input strongly depends on some of the previous inputs. In practice, RNN has had tremendous success (Greff et al., 2015; Karpathy, 2019) in natural language processing (NLP) for language translation, speech recognition, natural language understanding, and natural language generation. Other areas of success are financial forecasting, and film and image captioning.

As an example, when we start typing words in an email the RNN often anticipates the next few words and provides its best guess at the next words that complete a sentence. It also can read someone's email and give us the option of clicking on a few suggested replies. The RNN is still considered as supervised learning in the sense that the training requires an output at each step - even if it's a label that you didn't need to obtain using some form of manual labeling. RNN is similar to that of image classification by CNN, where both are trained on thousands or millions of examples. It differs in that each word (of about 10,000 commonly used words) is a separate class compared to no more than a few hundred different classes that CNN assigns to objects in an image.

This chapter presents the theory of RNN, where the theory of easy-to-understand vanilla RNN (VRNN) is first presented and then followed by the theories of the more practical long short-term memory networks (LSTM) and gated recurrent units (GRUs). Practical examples of RNN applied to geoscience problems are then presented in the last section.

---

<sup>1</sup>For example, if the input sequence is a speech signal that corresponds to a spoken digit, then the final target output at the end of the sequence might be a label classifying the digit ([en.wikipedia.org/wiki/Recurrent\\_neural\\_network](https://en.wikipedia.org/wiki/Recurrent_neural_network)).

## 12.1 Theory of Recurrent Neural Networks

One of the goals of RNN in natural language processing (NLP) is to complete a sentence when only the first few words are known. An example is the auto-complete function when we type a message in our email. For word prediction, RNNs use anywhere from a few to a several dozen words of typed text to predict the next word or several words in the uncompleted sentence. For example, I type the text "Hope to see." and the computer email program anticipates that the next word (or phrase) is "you". How does it know that "you" is the correct word, why not "stars" or "Bill"? It chose "you" because it might have taken into account the context of your previously typed sentence and from what it learned in the huge training set; in our example I selected the most likely follow-on word as "you". Is this prediction correct? Not necessarily so, but RNN selects the most likely next word based on previously written sentences. How does the RNN work in detail? The next section presents the workflow and explanation of Vanilla RNN (VRNN) whose architecture is illustrated in Figure 12.1.

### 12.1.1 Vanilla RNN Workflow

The VRNN algorithm for natural language processing trains the network shown in Figure 12.1 in order to accurately predict the sequence of target words  $\hat{\mathbf{y}}_t$  from the input words  $\mathbf{x}_t$ . The vector  $\mathbf{x}_t$  is a unique  $D \times 1$  unit vector coded to a unique word in the  $D$ -word dictionary. The prediction is a feed-forward process where thresholding and matrix multiplication operations are applied to the word at  $\mathbf{x}_t$  to give the  $D \times 1$  output vector  $\mathbf{o}_t$ . The feed-forward process starts at an initial time  $t = 0$  and recursively applies the following equations in time to get the sequence of probability vectors  $(\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_T)$  for the predicted words.

$$\begin{array}{ll}
 \text{for } t = 1 : T & \\
 \quad H \times 1 \text{ hidden state vector :} & \mathbf{h}_t = \sigma(\mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{h}_{t-1} + \mathbf{b}_h), \\
 \quad D \times 1 \text{ output vector :} & \mathbf{o}_t = \mathbf{V}\mathbf{h}_t + \mathbf{b}_o, \\
 \quad D \times 1 \text{ probability vector of predicted words at time } t : & \mathbf{y}_t = \text{softmax}(\mathbf{o}_t), \\
 \quad D \times 1 \text{ unit vector of desired word at time } t : & \mathbf{d} = \hat{\mathbf{y}}_t, \\
 \quad \text{probability for predicting correct word at time } t : & y_t = \mathbf{y}_t \cdot \mathbf{d}, \\
 \quad \text{cross-entropy loss at time } t : & \epsilon_t = -\ln y_t, \\
 \text{end} & (12.1)
 \end{array}$$

where the matrices  $\mathbf{U}$ ,  $\mathbf{V}$ , and  $\mathbf{W}$  and the bias vectors  $\mathbf{b}_h$ ,  $\mathbf{b}_o$  are to be learned by training with a huge number of sentences. Here, the components of the  $D \times 1$  target vector  $\hat{\mathbf{y}}_t$  are all zero, except for the component associated with the correct word; this component takes on the value 1. The Greek symbol  $\sigma$  denotes a thresholding operation, which is tanh for the above pseudo-code. The memory of previously written words up to time  $t$  is contained in the  $H \times 1$  hidden-state vector  $\mathbf{h}_t$ . Since the target word  $\mathbf{d} = \hat{\mathbf{y}}_t$  at time  $t$  is a unit vector then  $y_t = \hat{\mathbf{y}}_t \cdot \mathbf{y}$  is the probability that the network predicted the correct output word. The  $i^{\text{th}}$  element of the  $D \times 1$  vector  $\mathbf{y}_t = \text{softmax}(\mathbf{o}_t)$  is the prediction probability for the word associated with the  $i^{\text{th}}$  element position. For example, if the unit-vector of the target word  $\hat{\mathbf{y}}_t$  has a 1 in the third element then  $y_t = \hat{\mathbf{y}}_t \cdot \mathbf{y}$  extracts the predicted probability of this target word.

The details of the VRNN algorithm are given below.

1. **Assignment of Input Words  $\mathbf{x}_t$  to One-hot Vectors.** We first define a dictionary with  $D$  of the most commonly used words. There might be  $D = 10,000$  commonly used words so each unique word  $\mathbf{x}_t$  in a sentence is assigned to a unique  $10,000 \times 1$  unit vector<sup>2</sup>. For

<sup>2</sup>If the word is not in our vocabulary, we create an unknown *UNK* tag and assign it to a unique unit

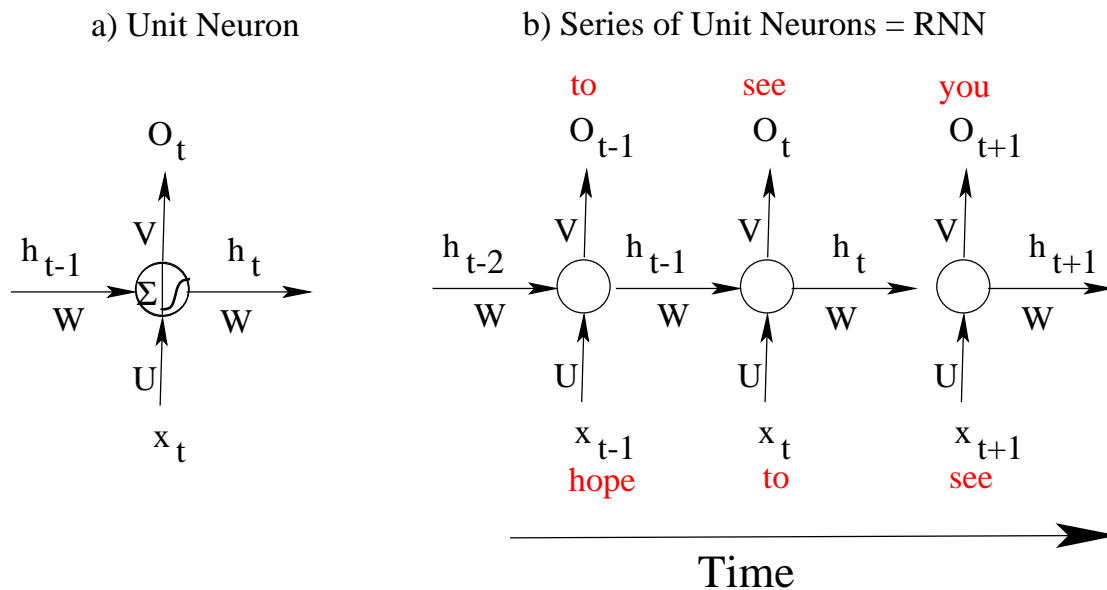


Figure 12.1: a) Unit neuron and b) series of three neurons that comprise the *Vanilla* RNN architecture. The total number of units is equal to the number of words in the training set; thus, the RNN for language processing is a wide neural network. The horizontal series of neurons constitutes one layer of the network. If a softmax function is applied to each output  $\mathbf{o}_t$  then the network in b) constitutes a 3-layer RNN ([en.wikipedia.org/wiki/Recurrent\\_neural\\_network](http://en.wikipedia.org/wiki/Recurrent_neural_network)).

example, the words in the sentence "Hope to see.." are assigned to the *one – hot* unit vectors

$$\begin{aligned}
 \text{hope} : \mathbf{x}_1 &= (0, 0, 1, 0, 0, \dots, 0); \\
 \text{to} : \mathbf{x}_2 &= (1, 0, 0, 0, 0, \dots, 0); \\
 \text{see} : \mathbf{x}_3 &= (0, 0, 0, 1, 0, \dots, 0); \\
 \text{you} : \mathbf{x}_4 &= (0, 1, 0, 0, 0, \dots, 0); \\
 &\vdots \\
 &\vdots \\
 &\vdots \\
 \text{UNK} : \mathbf{x}_D &= (0, 0, 0, 0, 0, \dots, 0, 1).
 \end{aligned} \tag{12.2}$$

The interpretation of each one-hot vector is that it assigns a unique class to each of the 10,000 words in our dictionary. For example, the word "hope" is indicated by the number 1 in the third element of the vector  $\mathbf{x}_t$  and zeros elsewhere. Any unknown words are replaced by "UNK" and is assigned to be the last word in our dictionary above. Numbers are assigned the word "N" at the second-to-last word in the dictionary. Punctuation symbols can also be assigned to unique elements of  $\mathbf{x}_t$ .

The subscript  $t$  in  $\mathbf{x}_t$  denotes the temporal ordering of the word in the sentence. For example,  $\mathbf{x}_t$  occurs just before  $\mathbf{x}_{t+1}$  in the sentence. The dimension of the dictionary is denoted as  $D$  so  $\mathbf{x}_t \in \mathbb{R}^{D \times 1}$ .

2. **Hidden-State Vector  $\mathbf{h}_t$ .** At each neuron in Figure 12.1b,  $\mathbf{h}_t$  is the hidden-state vector at time step  $t$ . Its role is to transmit information from previously used words that might be important for predicting the current output at time  $t$ . It is calculated as the thresholded weighted sum of the previous hidden-state vector  $\mathbf{h}_{t-1}$  and the current input  $\mathbf{x}_t$ :

$$\underbrace{\mathbf{h}_t}_{\text{current hidden state}} = \sigma(\mathbf{U}\mathbf{x}_t + \mathbf{W} \underbrace{\mathbf{h}_{t-1}}_{\text{previous hidden state}} + \mathbf{b}_h), \tag{12.3}$$

where  $\sigma(\cdot)$  is the activation function of choice, the tanh in this case, and the elements of the initial state  $\mathbf{h}_{t=0}$  can be zeroes. Here,  $\mathbf{h}_t, \mathbf{b}_t$  are  $H \times 1$  vectors so  $\mathbf{W} \in \mathbb{R}^{H \times H}$  and  $\mathbf{U} \in \mathbb{R}^{H \times D}$ . In practice,  $H$  can be chosen to be around 100.

3. **Probability Vector of Predicted Word:  $\mathbf{y}_t$ .** The  $D \times 1$  vector  $\mathbf{y}_t = \text{softmax}(\mathbf{o}_t)$  in Figure 12.1 is the probability vector of predicted words at time  $t$ :

$$\underbrace{\mathbf{y}_t}_{\text{probability vector of predicted words}} = \text{softmax}(\overbrace{\mathbf{V}\mathbf{h}_t + \mathbf{b}_h}^{\mathbf{o}_t}). \tag{12.4}$$

The  $i^{th}$  element value in the  $D \times 1$  vector  $\mathbf{y}_t$  is the probability for predicting the  $i^{th}$  word class.

For example, a perfect prediction at  $t = 1$  would be  $\mathbf{y}_{t=1} = (0, 0, 1, 0, \dots, 0)$  according to the dictionary of words in equation 12.2. But nothing is perfect so the actual output of "Hope" might be

$$\mathbf{y}_{t=1} = \text{softmax}(\mathbf{V}\mathbf{h}_t + \mathbf{b}_h)|_{t=1} = (0.5, 0.1, 0.4, 0, \dots, 0), \tag{12.5}$$

compared to the desired word given by

$$\hat{\mathbf{y}}_{t=1} = (0, 0, 1, 0, \dots, 0). \tag{12.6}$$

---

vector in our dictionary.



Here, the desired word "Hope" has the value 1 at the third element of  $\hat{\mathbf{y}}_{t=1}$  (see first row of equation 12.2) while the vector  $\mathbf{y}_{t=1}$  has the poor prediction probability of 0.4 at this third element<sup>3</sup>. Better training might have avoided this error.

We can extract this poor prediction value from  $\mathbf{y}_t$  by taking the dot product of it with the desired word vector  $\hat{\mathbf{y}}_t$  to get

$$y_t = \hat{\mathbf{y}}_t \cdot \mathbf{y}_t = 0.4, \quad (12.7)$$

where  $0 \leq y_t \leq 1$  is defined as the quality factor of our prediction. The highest quality prediction is when  $y_t = 1$  and the lowest quality is  $y_t = 0$ .

4. **Loss Function:**  $\epsilon$ . The quality factor  $y_t$  in equation 12.7 can be used to define the loss function  $\epsilon_t$  at time  $t$  as

$$\epsilon_t = -\ln y_t, \quad (12.8)$$

where  $\epsilon_i \geq 0$ . Therefore the average total loss function for all of the time steps is

$$\epsilon = -\frac{1}{T} \sum_{t=1}^T \ln y_t, \quad (12.9)$$

which is similar to the cross-entropy function<sup>4</sup>. A perfect prediction, i.e.  $y_t = 1$  at all times, yields the minimum value of  $\epsilon = 0$ .

---

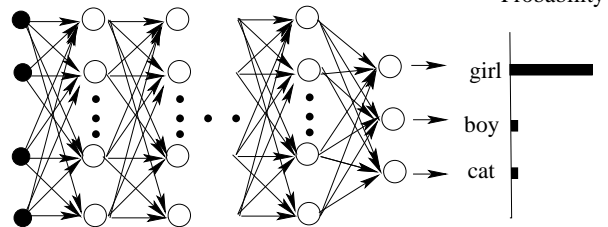
<sup>3</sup>The bottom image in Figure 12.2 shows an example of an output vector plotted as a histogram.

<sup>4</sup>The desired probability at  $t$  is  $\hat{\mathbf{y}}_t = 1$ , so we can harmlessly multiply  $\ln y_t$  by  $\hat{\mathbf{y}}_t$  to get  $\hat{\mathbf{y}}_t \ln y_t$ , which is the summand for the cross-entropy loss function.

## Image Classification vs NLP

### Image Classification

- 3 classes
- all images have the same size



### Language Modeling

- 10,000 classes
- sentences have different sizes: 1 to 100 words
- average sentence size is 15 words

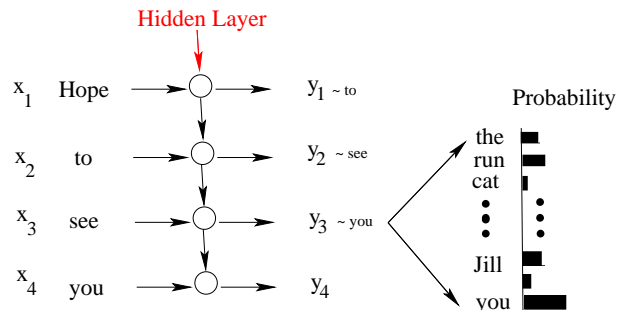


Figure 12.2: (top) Image classification by a standard neural network and (bottom) word prediction by a RNN, which is a natural language processing (NLP) method. The RNN feeds the words in a sentence one at a time into the neurons as they are written (or spoken) and the goal, in this example, is to predict the next word from the current and previous words. Illustration is a modification of a slide from Xavier Bresson's AI lecture notes.

**Box 12.1.1. Probabilistic Model for RNN Loss Function**

The RNN loss function in equation 12.9 was selected in an intuitive manner. However, we can justify this selection by analyzing the probability for predicting the sequence of words  $\mathbf{y} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T)$  by using the definition of the conditional probability  $p(\mathbf{x}|\mathbf{y}) = p(\mathbf{x}, \mathbf{y})/p(\mathbf{y})$ , or equivalently  $p(\mathbf{x}, \mathbf{y}) = p(\mathbf{x}|\mathbf{y})p(\mathbf{y})$ . The joint probability distribution  $p(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T)$  can be factorized as a concatenation of conditional probabilities:

$$\begin{aligned}
 p(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T) &= p(\mathbf{x}_1)p(\mathbf{x}_2, \mathbf{x}_3, \dots, \mathbf{x}_T|\mathbf{x}_1), \\
 &= p(\mathbf{x}_1)p(\mathbf{x}_2|\mathbf{x}_1)p(\mathbf{x}_3, \dots, \mathbf{x}_T|\mathbf{x}_1, \mathbf{x}_2), \\
 &\quad \cdot \\
 &\quad \cdot \\
 &\quad \cdot \\
 &= \prod_{t=1}^T p(\mathbf{x}_t|\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{t-1}), \\
 &= \prod_{t=1}^T p(\mathbf{x}_t|\mathbf{x}_{<t}), \tag{12.10}
 \end{aligned}$$

where the notation for the matrix coefficients and bias factors is silent. Here,  $p(\mathbf{x}_t|\mathbf{x}_{<t})$  is the probability distribution conditioned on the sentence up to the  $t-1$  word. This probability function implicitly depends on the matrix and bias coefficients, and the optimal coefficients are those that maximize equation 12.10. Equating the quality factor  $Q_t$  to  $p(\mathbf{x}_t|\mathbf{x}_{<t})$  and employing equation 12.10 probabilistically justifies the rationale for concatenating the quality factors in equation 12.8 and defining its negative logarithm as the loss function (Cho, 2015). For computational efficiency the probability function is approximated by the truncated Markov assumption

$$p(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T) \approx \prod_{t=T-n}^T p(\mathbf{x}_t|\mathbf{x}_{t-n}, \mathbf{x}_{t-n+1}, \dots, \mathbf{x}_{t-1}), \tag{12.11}$$

where we are only concerned about the previous  $n$  words that influence the current one.

5. **Iterative Gradient Estimate of Optimal  $\mathbf{U}, \mathbf{V}, \mathbf{W}, \mathbf{b}$ .** The iterative steepest descent formula for updating the unknown coefficients in  $\mathbf{b}$  and the  $\mathbf{U}, \mathbf{V}, \mathbf{W}$  matrices symbolized by  $\mathbf{Z}$  is given by

$$Z_{ij} = Z_{ij} - \alpha \frac{\partial \epsilon}{\partial Z_{ij}}, \tag{12.12}$$

where the gradient for  $W_{ij}$  is derived in Appendix 12.8 and the formulas for the other gradients are requested in Exercises 2-3. In the training phase, thousands of sentences are used for the training set and the RNN is trained until an acceptable prediction accuracy is achieved. Then the network is used to complete incomplete sentences or phrases, or perhaps translate sentences from one language to another. As will be noted later, a vanishing gradient limits the number of input words for any one RNN prediction.

In summary, the VRNN architecture is depicted in Figure 12.1. For natural language processing, its supervised training procedure is similar to that for classifying images by a standard neural network except a large comprehensive set of sentences are used for training the network. Unlike the standard neural network in Figure 12.2, the VRNN matrices  $\mathbf{U}, \mathbf{V}, \mathbf{W}$  are shared by all the neurons. Also the words in an input sentence are sequentially inserted into the network and a neuron is assigned to each input word. This compares to the simultaneous insertion of an entire image into all of the nodes in the first layer of a fully connected neural network. For VRNN, a vanishing gradient prevents remembering no more than a small number of previously used words (see Appendix 12.8) and so

ignores information that might be important for predicting the target words at time  $t$ . To remedy this problem, the next section presents the RNN known as *gated recurrent units* (GRU) networks. Its related predecessor, the long short-term memory (LSTM) network, is discussed in Appendix 12.9. Both of these improved networks consist of gates that prevent vanishing gradients so that they enjoy both long- and short-term memories.

### Comments.

- Many-to-Many RNN: Translating from one language to another<sup>5</sup>, requires the many-to-many RNN architecture illustrated in Figure 12.3a. However, current natural language translators (NLTs) are more complex than the VRNN, and one of the popular NLTs architectures is the RNN encoder-decoder structure (Silipo et al., 2019) consisting of two stacks of RNNs: one stack for the encoder of words from one language and the other for the decoding of the latent space code from the encoder. The practical RNN architecture also uses the long short-term memory (LSTM) cells to overcome the vanishing gradient problem.
- Many-to-One RNN: Sometimes the task of RNN is to assess the overall message of a sentence or a few sentences, e.g. the sentiment of a movie review (Maas et al., 2011). Is the movie good or bad? In this case, there are many input words but there is only one output needed at the end of review, as illustrated in Figure 12.3b.
- Many-to-Many RNN: In translating from written English to written French, there may be more words in the French translation than in the English phrase (Sutskever et al., 2014). Hence, this corresponds to an input with many words and an output with many more words, as illustrated in Figure 12.3c. Other RNN applications include text-to-speech (Graves et al., 2011).
- One-to-Many RNN: The input to the RNN consists of certain features obtained by applying CNN to an image, as shown in Figure 12.4. The goal of the RNN is to produce a caption that describes the image, so this is a one-to-many mapping as illustrated in Figures 12.3d and 12.4.
- RNN's can be used to train character-level language models, as illustrated in Figure 12.5. Similar to prediction of words by RNN, a character-based RNN can model the probability distribution of the next character given the sequence of characters before it. This is relevant to the application of RNN for predicting well logs from other well logs, where a log's resistivity, porosity and  $V_p$  values might be sequentially input with increasing depth to predict the  $V_s$  velocity profile at the same depth. Musical scores can also be created by RNN after being sufficiently trained ([soundcloud.com/optometrist-prime/recurrence-music-written-by-a-recurrent-neural-network](https://soundcloud.com/optometrist-prime/recurrence-music-written-by-a-recurrent-neural-network)). Time series can also be predicted by LSTMs, such as financial data ([//wiki.ubc.ca/Course : CPSC522/ Financial\\_Forecasting\\_using/ LSTM\\_Networks](https://wiki.ubc.ca/Course%3A%20CPSC522/Financial_Forecasting_using_LSTM_Networks)) or well logs (Zhang et al., 2018; Guo et al., 2019; Pham and Naeini, 2019).

### 12.1.2 Vanishing Gradient of VRNN

The VRNN is not used in most applications because it suffers from the vanishing gradient<sup>6</sup> problem (Bengio et al., 1994; Hochreiter et al., 2001). We can mathematically examine this problem

---

<sup>5</sup>RNN is also used for speech translation. Given an input sequence of acoustic signals, a RNN can predict a sequence of phonetic segments together with their probabilities.

<sup>6</sup>A persuasive video that demonstrates the disappearance of the gradient is in <https://www.youtube.com/watch?v=yCC09vCHzF8>.

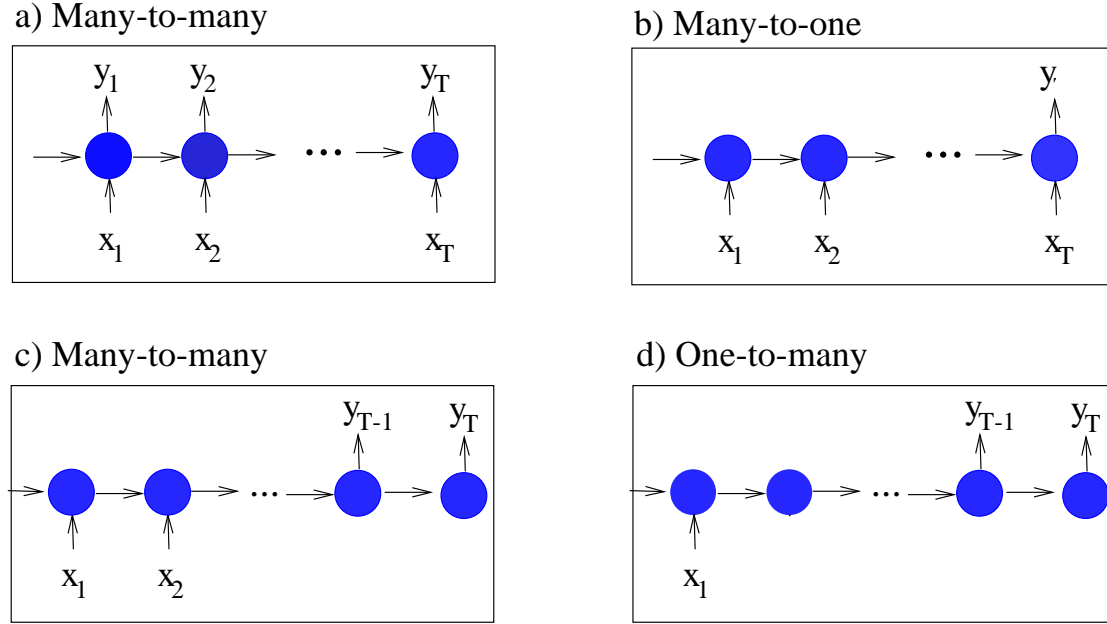


Figure 12.3: RNN architectures for a) many-to-many, b) many-to-one, c) many-to-many mappings and d) one-to-many mappings. The many-to-many diagram in a) differs from that in c) because it implicitly represents a combination of an encoder-decoder architecture with latent space variables (Silipo et al., 2019).

by taking the gradient of the RNN loss function in equation 12.9 to get

$$\frac{\partial \epsilon}{\partial W_{ij}} = \frac{\partial \epsilon}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_{t-1}} \frac{\partial \mathbf{h}_{t-1}}{\partial \mathbf{h}_{t-2}} \cdots \frac{\partial \mathbf{h}_2}{\partial \mathbf{h}_1} \frac{\partial \mathbf{h}_1}{\partial W_{ij}}. \quad (12.13)$$

For simplicity of exposition, consider  $\mathbf{h}_t$  and  $\mathbf{W}$  to be  $1 \times 1$  scalars for all values of  $t$ . In this case,  $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} = W\sigma'$  resembles a band-limited delta function. Thus, equation 12.13 for large values of  $t$  will be the product of many  $W\sigma'$ 's, which will be nearly zero everywhere except at the origin for reasonable values of  $|W| < 1$ . For  $|W\sigma'| < 1$ , the gradient will practically vanish at several neurons before the evaluation time  $t$ . Therefore, the residual at time  $t$  will be used to adjust the value of  $\mathbf{W}$  according to information only from the recent input words. It will forget about what was input at earlier times. This prevents the ancient historical context of a sentence or paragraph to assist in predicting the next word. Appendix 12.8 shows this statement to be true for higher dimensional vectors  $\mathbf{h}_t$  and matrices  $\mathbf{W}$ .

Therefore, the key to beating the vanishing gradient problem is to incorporate a *skip-like* connection into the VRNN architecture (see Figures 12.6a and 9.18) so one can remember words from the distant past (Srivastava et al., 2015). Recall that the skip connection in the residual neural network shown in Figure 12.6 takes a short-cut to jump over several layers ([en.wikipedia.org/wiki/Residual\\_neural\\_network](https://en.wikipedia.org/wiki/Residual_neural_network)). This short-cut unit in RNN that jumps over at least one unit is also known as a leaky unit (see Ali Ghodsi lectures in <https://www.youtube.com/watch?v=AvyhbrQptHk>). For the VRNN case, the skip connection can be incorporated by adding a weighted hidden-state vector to the output:

$$\mathbf{h}_t = \left(1 - \frac{1}{\tau_t}\right)\mathbf{h}_{t-1} + \frac{1}{\tau_t}\sigma(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}_h), \quad (12.14)$$

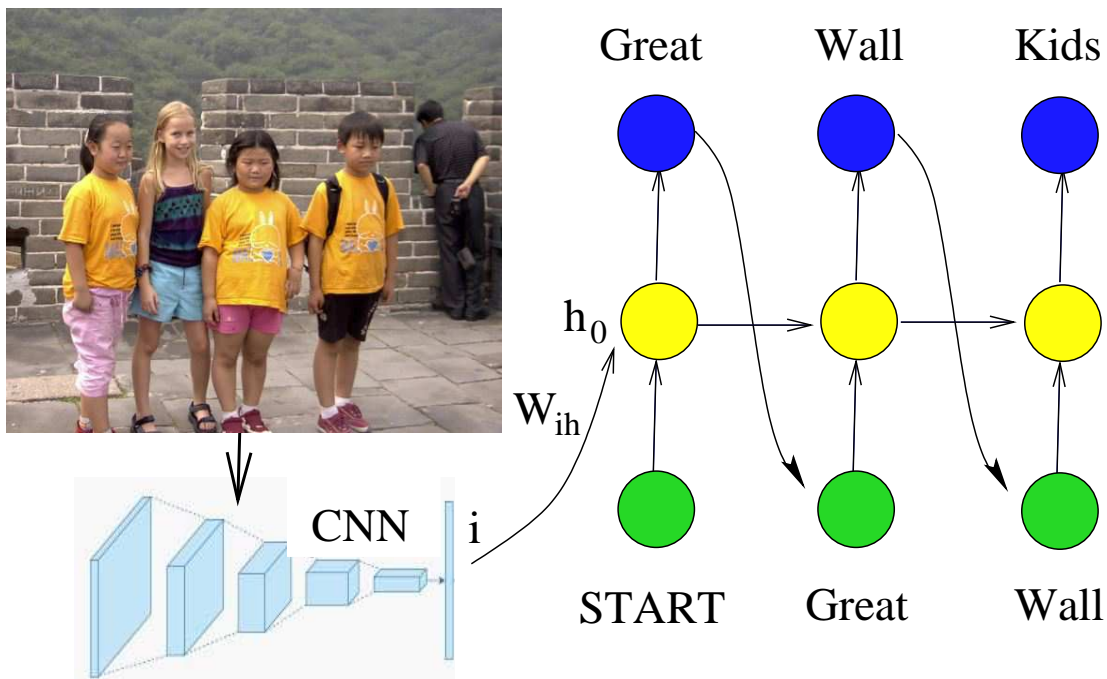


Figure 12.4: Image captioning by the combination of CNN and RNN. Here, the output vector  $\mathbf{i}$  from the last layer, without the softmax operation, of the CNN is used as part of the memory state for the first hidden-state neuron of the RNN. The activation function that gives  $\mathbf{h}_1$  in the first layer takes the form  $\mathbf{h}_1 = \tanh(\mathbf{W}_{xh}\mathbf{x}_0 + \mathbf{W}_{hh}\mathbf{h}_0 + \mathbf{W}_{ih}\mathbf{i})$ . Illustration inspired from the lecture by *Fei-Fei Li, Justin Johnson, Serena Yeung* lecture in CS231 in <https://www.youtube.com/watch?v=yCC09vCHzF8>.

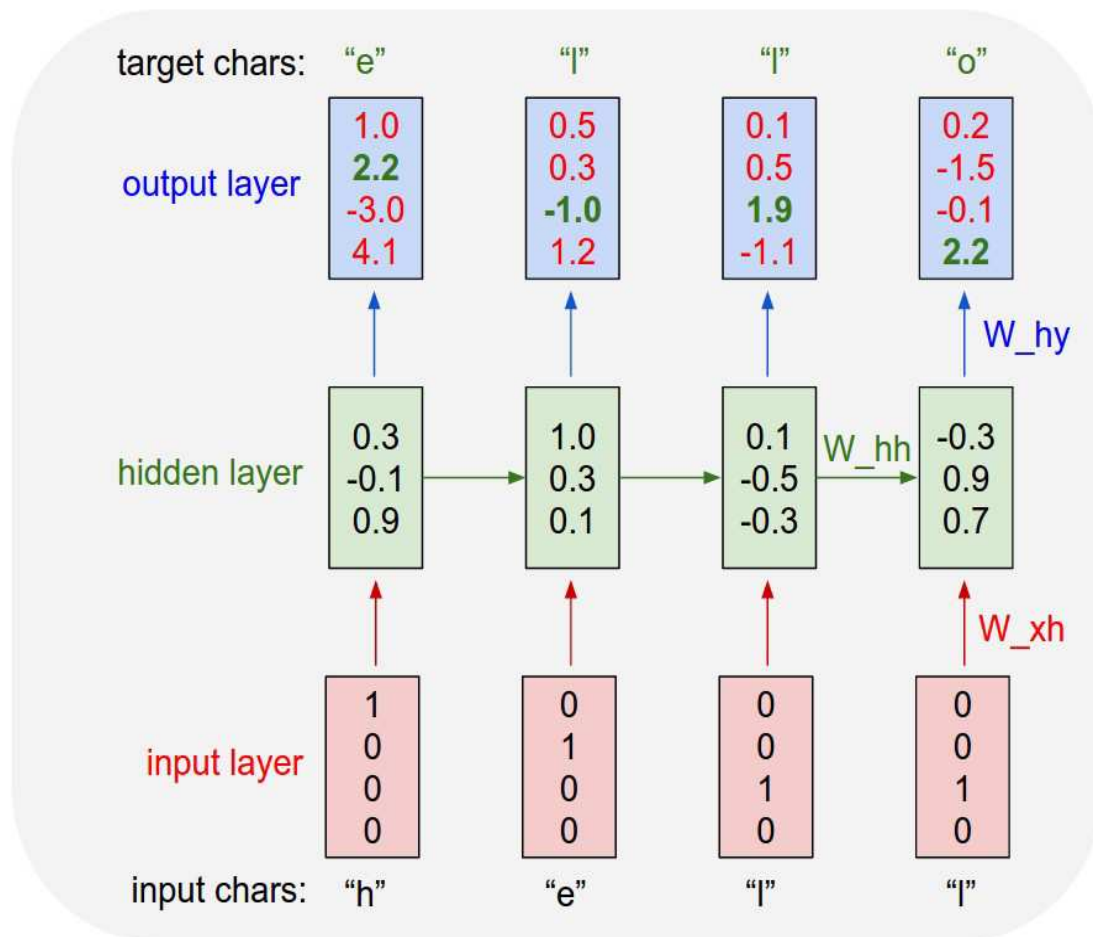


Figure 12.5: An example RNN with 4-dimensional input and output vectors, and 4 hidden 4 units (neurons). This diagram shows the activations in the forward pass when the RNN is fed the characters "hell" as input. The RNN computes the probability vector  $\mathbf{y}_t$  for predicting the next character (vocabulary is "h,e,l,o"); we want the green numbers to be high and red numbers to be low. A softmax operation can be applied to the output values to give a probability value to each element in the output vectors. Illustration and caption from [karpathy.github.io/2015/05/21/rnn-effectiveness/](https://karpathy.github.io/2015/05/21/rnn-effectiveness/) and Karpathy (2019).

where  $1 \leq \tau_t \leq \infty$ . If  $\tau_t = 1$  then this reduces to the VRNN. In this case,  $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \propto \sigma'$ , which leads to a vanishing gradient according to equation 12.13. However, if  $\tau_t \rightarrow \infty$  we simply *equate* the previous state  $\mathbf{h}_{t-1}$  to the new state  $\mathbf{h}_t$ ; in addition, the current input word  $\mathbf{x}_t$  is ignored. In this case,  $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} = 1$  for all the gradients in equation 12.13, until we encounter the time step  $t = t_i$  where  $\tau_t \approx 1$ . At that time step  $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \approx \sigma'$  so we can start using phrases or words around that time to adjust  $\mathbf{W}$  to reduce the residual.

In summary, we can choose the values of the leaky parameter  $\tau_t$  so that we can remember words input at much earlier times than the current time. This long-term memory is dictated by the choice of  $\tau_t$ . How are the values of  $\tau_t$  selected? They are learned from the training, and give rise to the *reset* gate in gated recurrent networks (GRUs) and the cell states in long short-term memory (LSTM) networks.

To generalize this memory capability we can choose  $\frac{1}{\tau}$  to be an  $M \times 1$  vector with either zeros or ones. Taking the element-by-element Hadamard product  $\frac{1}{\tau} \odot \mathbf{h}_{t-1}$  forces us to either remember the previous  $i^{th}$  hidden state vector  $\mathbf{h}_{t-1,i}$  ( $1/\tau_t = 1$ ) or forget its squashed version  $\sigma(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b})$  for  $1/\tau_t = 0$ . Thus the leaky unit in equation 12.14 allows one to overcome the vanishing gradient problem of VRNN by allowing the skip connection controlled by the gatekeeper  $\frac{1}{\tau_t}$  to allow previous words to enter the current memory. Rather than manually setting the  $1/\tau_t$  coefficients to be between one and zero at each neuron, we will let the network learn the best values by adding a gatekeeper in the network.

### Gatekeepers

Incorporating a sigmoid  $\sigma(\mathbf{z}_t)$  to an input vector  $\mathbf{z}_t$  squashes large negative elements to zero, and large positive elements to 1. In other words, it is a *gatekeeper* that only allows numbers larger than a small negative value to pass. As we will shown in the next section, replacing  $1/\tau_t$  in equation 12.14 by a term similar to  $\sigma(\mathbf{z}_t)$  will be the gatekeeper for long-term memory in GRUs. The  $\tanh(\mathbf{z}_t)$  is not considered to be a gatekeeper because it allows for the passage of both large positive and negative elements in  $\mathbf{z}_t$ .

## 12.2 Gated Recurrent Unit Network

We now discuss the gated recurrent unit (GRU) network that overcomes the vanishing gradient problem. An alternative to GRUs is the more complicated and sometimes less effective long short-term memory network discussed in Appendix 12.9.

The GRU was introduced by Cho et al. (2014), and its architecture is shown in Figure 12.7. It is simpler than standard LSTM models, and has been growing increasingly popular. The GRU equations are the following.

$$\text{reset gate: } \mathbf{g}_t^r = \sigma(\mathbf{W}_r(\mathbf{x}_t, \mathbf{h}_{t-1}) + \mathbf{b}_r), \quad (12.15a)$$

$$\text{update gate: } \mathbf{g}_t^u = \sigma(\mathbf{W}_u(\mathbf{x}_t, \mathbf{h}_{t-1}) + \mathbf{b}_u), \quad (12.15b)$$

$$\text{tanh fate: } \tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_h(\mathbf{x}_t, \mathbf{g}_t^r \odot \mathbf{h}_{t-1}) + \mathbf{b}_h), \quad (12.15c)$$

$$\text{skip gate: } \mathbf{g}_t^s = (1 - \mathbf{g}_t^u) \odot \mathbf{g}_{t-1}^s + \mathbf{g}_t^u \odot \tilde{\mathbf{h}}_t. \quad (12.15d)$$

There are many variations of LSTM and GRUs, too many to mention. Chung et al. (2014) performed a performance comparison between LSTMs and GRUs and the results are mixed. A recent study (Greff et al., 2017) compared the performance of eight LSTM variants and concluded that none of the variants *"can improve upon the standard LSTM architecture significantly, and demonstrate the forget gate and the output activation functions to be its most critical components. We further observe that the studied hyperparameters are virtually independent and derive guidelines for their efficient adjustment."*



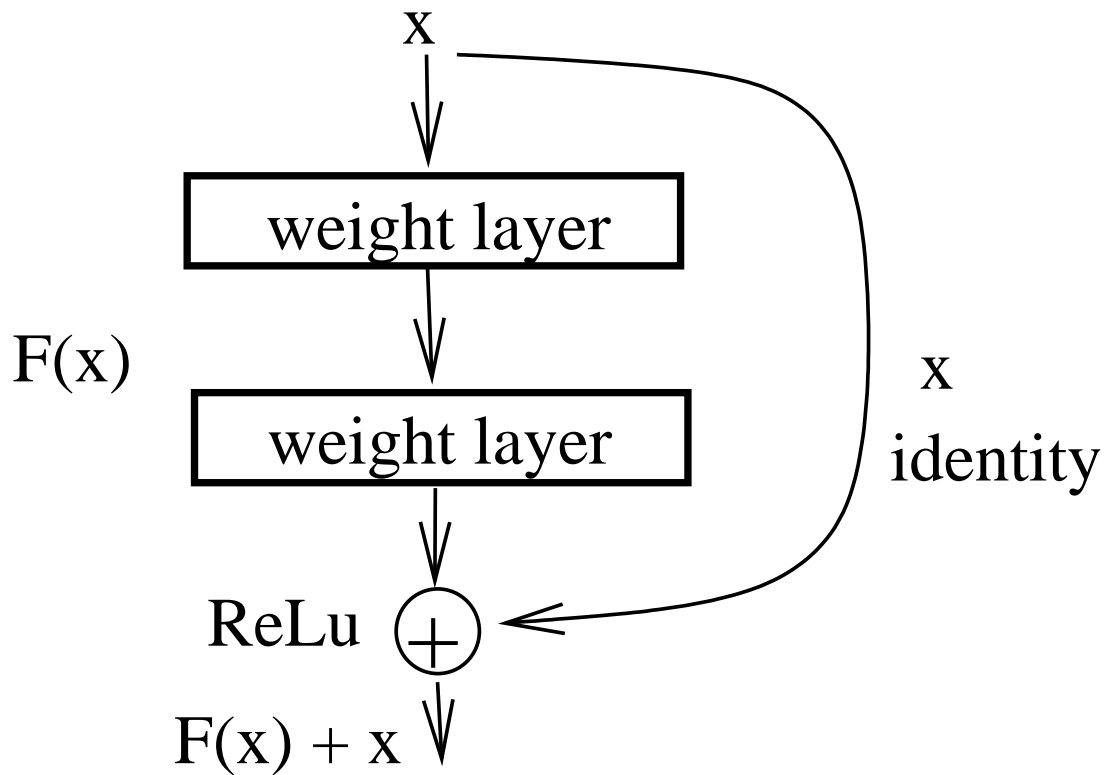


Figure 12.6: Skip connection in a residual CNN neural network where the input  $\mathbf{x}$  to the two weight layers is added to the output  $F(\mathbf{x})$  to give the new output  $F(\mathbf{x}) + \mathbf{x}$ . See [stats.stackexchange.com/questions/56950/neural-network-with-skip-layer-connections](https://stats.stackexchange.com/questions/56950/neural-network-with-skip-layer-connections).

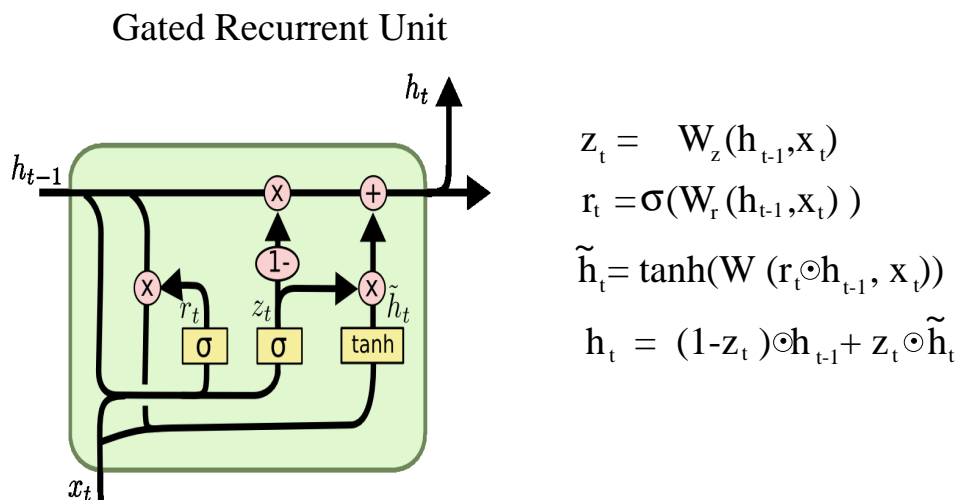


Figure 12.7: Tangled architecture of a Gated Recurrent Unit.

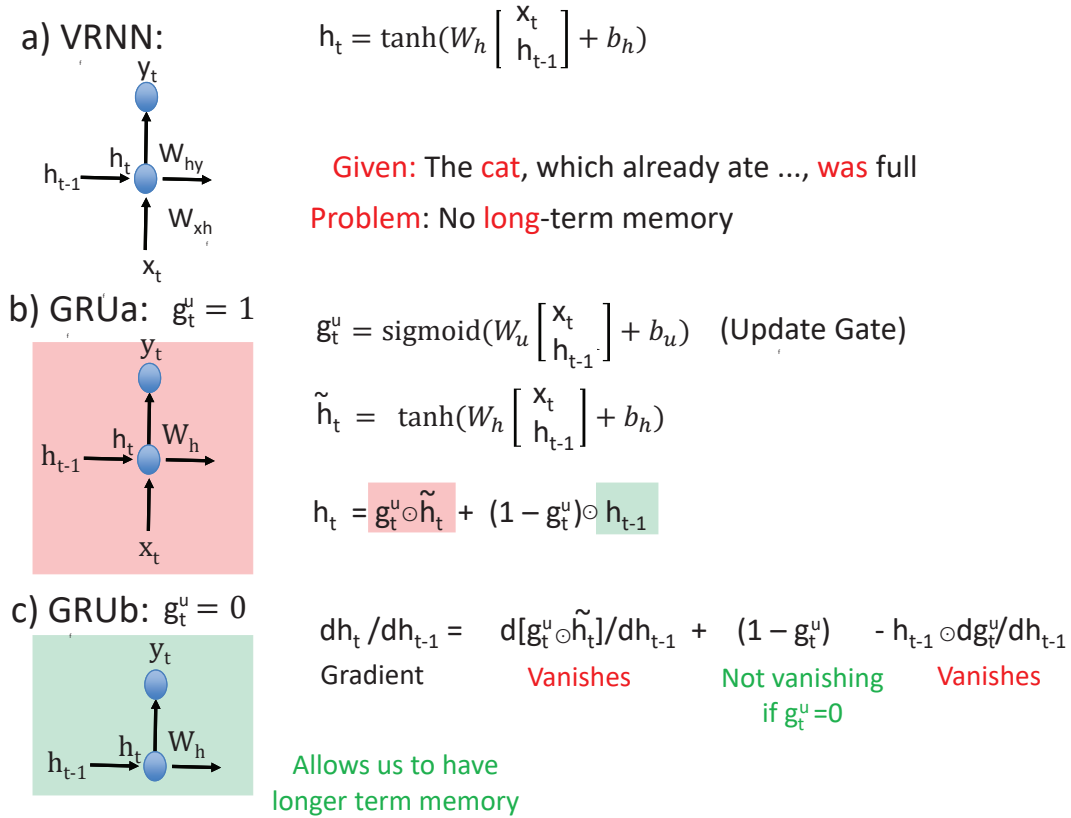


Figure 12.8: Neuron diagrams for a) VRNN, b) GRUa when  $g_u = 1$  and c) GRUb when  $g_u = 0$ . GRUa denotes the GRU without the reset gate. The circle with the dot in the middle is the Hadamard product, which is an component-by-component multiplication of the two  $M \times 1$  input vectors to give a  $M \times 1$  output vector.

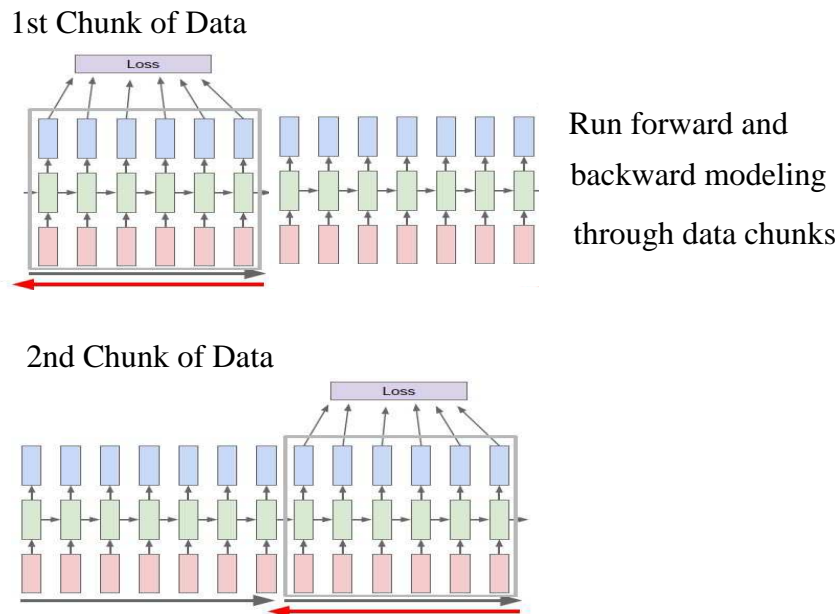


Figure 12.9: Truncated back-propagation through time: Forward and backward modeling should be run through chunks of data for updating the weights. Illustration adapted from *Fei – Fei Li, Justin Johnson, Serena Yeung lecture in CS231*.

## 12.3 Training RNNs

Training a RNN is similar to training a traditional Neural Network. We also use the back-propagation algorithm, but with a little twist. Because the parameters are shared by all time steps in the network, the gradient at each output depends not only on the calculations of the current time step, but also the previous time steps. For example, in order to calculate the gradient at  $t = 4$  we would need to back-propagate 3 steps and sum up the gradients. This is called Back-propagation Through Time (BPTT). In BPTT, we forward model the inputs throughout the entire sequence to compute loss, and then backward propagate the residual to compute the gradient.

### 12.3.1 Truncated Back-propagation

For a long sequence of input words, it is too expensive to back-propagate from the current time all the way backwards. Instead the forward and backward propagation is run through chunks of the data, as illustrated in Figure 12.9. In each case, the weights for the previous chunks are frozen while the weights are only updated in the current chunk. The loss function is also computed just for the current chunk (see equation 12.11). This is known as truncated back-propagation through time.

### 12.3.2 Bidirectional RNN

An extension of RNN is the bidirectional RNN (BRNN), which includes information from both the past and future to predict the current word. With standard RNN, the prediction of the current word uses the present and past input words, while the bidirectional RNN looks at both earlier and future words to predict the present. See Figure 12.10 for the BRNN architecture. BRNNs gives us a higher learning capacity (but we also need a lot of training data) compared to RNNs because

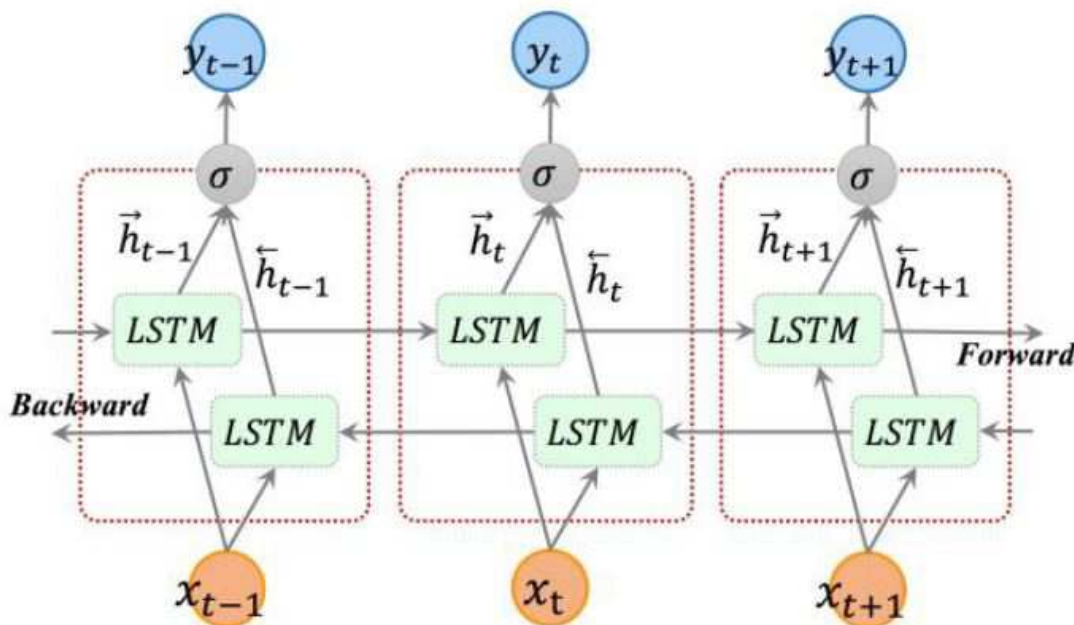


Figure 12.10: Bidirectional RNN with LSTM units, also denoted as a bidirectional LSTM. Figure taken from Cui et al. (2018).

more information is available for the current time prediction. Empirical studies show that bidirectional networks can be substantially better than unidirectional ones in many fields, like phoneme classification (Graves et al., 2005) and speech recognition (Graves et al., 2013).

To be more precise, the output state  $\mathbf{y}_t$  is given as a function of the hidden vectors  $\overleftarrow{\mathbf{h}}_t$  and  $\overrightarrow{\mathbf{h}}_t$ :

$$\mathbf{y}_t = \eta(\overleftarrow{\mathbf{h}}_t, \overrightarrow{\mathbf{h}}_t), \quad (12.16)$$

where  $\eta(\cdot)$  is a function that combines the two output sequences. It can be a concatenating function, a summation function, an average function or a multiplication function (Cui et al., 2018), and it might also include the softmax function.

### 12.3.3 Deep RNN

Deep learning assumes that a deep, hierarchical model can be exponentially more efficient at representing some functions than a shallow one (Bengio, 2009). For example, Delalleau and Bengio (2011) show that a deep sum-product network may require exponentially less units to represent the same function compared to a shallow sum-product network. Pascanu et al. (2014) tested to see if the same argument applies to recurrent neural networks. An example of a 4-layer RNN is shown in Figure 12.11, where the top output layer denotes the target values. Their empirical results showed that the proposed deep RNNs benefit from the depth and outperform the conventional, shallow RNNs.

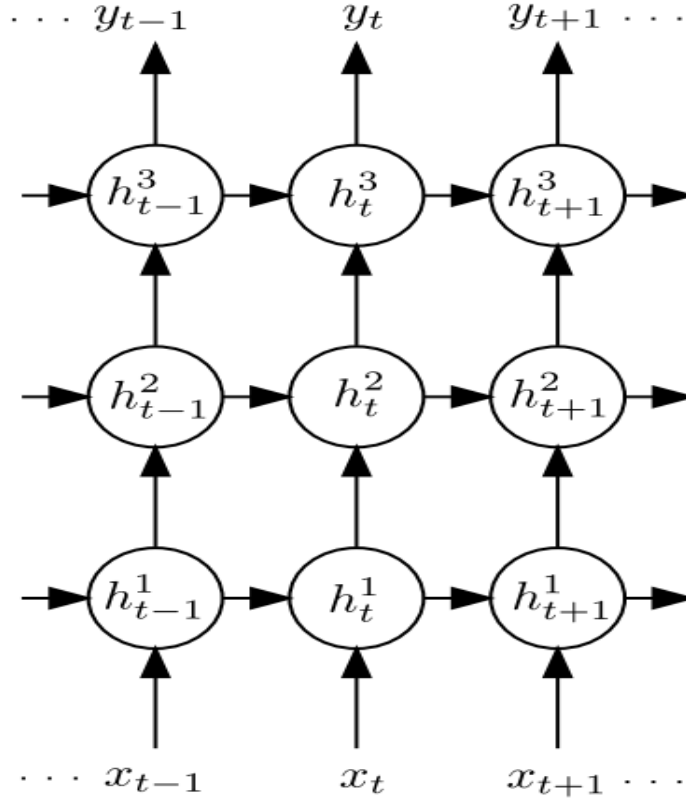


Figure 12.11: Four-layer RNN. Illustration from Graves et al. (2013). Here  $\mathbf{h}_t^{(1)} = \sigma(\mathbf{U}^{(1)}\mathbf{x}_t + \mathbf{W}^{(1)}\mathbf{h}_{t-1}^{(1)} + \mathbf{b}_h^{(1)})$  and  $\mathbf{h}_t^{(k)} = \sigma(\mathbf{U}^{(k)}\mathbf{h}_t^{(k-1)} + \mathbf{W}^{(k)}\mathbf{h}_{t-1}^{(k)} + \mathbf{b}_h^{(k)})$ , where the superscript refers to the layer number and each layer is associated with the  $k^{th}$ -layer matrices  $\mathbf{W}^{(k)}$ ,  $\mathbf{U}^{(k)}$  and bias vectors  $\mathbf{b}_h^{(k)}$ .

## 12.4 RNN Applications in Geoscience

There are many applications of RNN to geoscience problems, especially to those where the input is a correlated time series. We will present results from two such applications and for predicting well logs from seismic data.

### 12.4.1 Predicting Missing Well Logs by BRNN

Well logs record geophysical parameters such as  $V_p$ ,  $V_s$ , resistivity, rock density, saturation as a function of depth. The measurements at one depth are correlated with those at nearby depths because the geology is also correlated in depth. One of the problems with well logs is that they are expensive to record, and so in many cases only a small suite of different measurements are recorded per well. See Figure 12.12a for an example of five well logs from the same borehole.

Fortunately the measurements of several types of geophysical parameters are usually correlated to other types. So the idea is to train a neural network on wells that have almost a full suite of different types of measurements, and then use the learned weights to predict the logs that are missing at other wells. This is very similar to the goal of a many-to-many RNN architecture where one language is translated to another.

Phan and Naeini (2019) used a bidirectional RNN (BRNN) method to predict missing  $V_s$  well logs in the North Sea; the RNN is that of LSTM, except it is bidirectional. The training data consisted of 35 well logs from the North Sea, and these data each had 8 different types of logs. Three blind wells were selected and used to test the efficacy of their method. They used a bidirectional LSTM architecture for their training.

According to Pham and Naeini (2019), their architecture consist of *"..a BRNN followed by a FCNN. The BRNN has three forward LSTM layers and three backward LSTM stacks with 200, 300 and 400 neurons respectively.*

*For the well log application, there are four FCNN layers with 2048, 1024, 512, and 1 neurons respectively. The training dataset has 32 wells with gamma ray (GR), density (RHOB),  $V_p$ , resistivity (RT), volume of shale ( $V_{SH}$ ), water saturation ( $S_w$ ), neutron porosity (NPHI), depth (D), and the  $V_s$  log."* They calculate the minimum length of all wells in the training dataset and take samples from the beginning of each well to the maximum multiple of the minimum length. They then concatenate all types of logs except the  $V_s$  log along the last dimension. The training data is resampled as batches with "minimum-length" samples in each batch. This configuration avoids mixing samples of different depths from different wells in a batch. They then pre-process the data by removing the median and scaling according the interquartile range, which is robust to the outliers and removes the effects of erroneous spikes in the training data.

After 100 epochs of training, the predictions of  $V_s$  at one of the blind wells is shown in Figure 12.12a and a zoom view of the predicted  $V_s$  is in Figure 12.12b. The predicted curves in red closely match the actual  $V_s$  curves in black.

### 12.4.2 Predicting Well Logs from Seismic Reflection Data

Faraj and Regib (2018) used the 2-layer GRU architecture in Figure 12.13a to predict the density log from seismic data recorded over wells in the North Sea. The 2-layer GRU in Figure 12.13c is used to train on data, where the input consisted of a cube of 144 zero-offset reflection traces on the left of Figure 12.13c and the output is the well log on the right. Both the seismic data and well logs in the training were filtered with a low-pass filter to match the frequency content of training pairs. Each trace is manipulated to have a zero mean and a unit standard deviation. The traces are sequentially input at time  $t$  and the well log is predicted at time  $t$  as well. A mean square error is used to measure the misfit between the predicted and actual well log, and a gradient descent is used to update the parameters of the GRU. Early stopping is used to prevent overfitting, and there

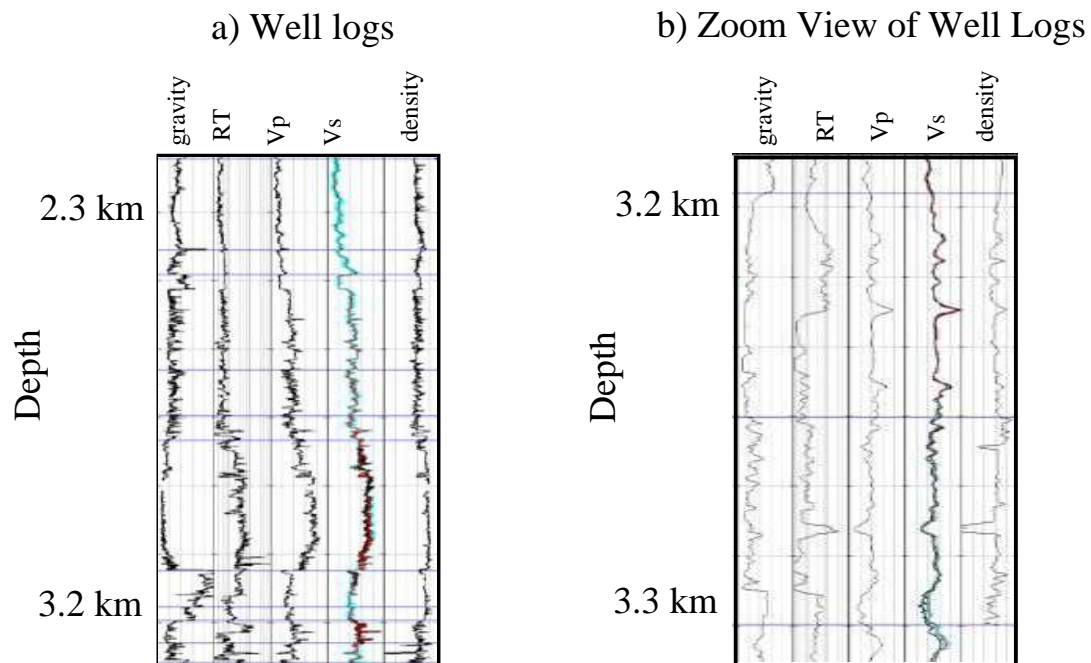


Figure 12.12: a) Predictions in red, the corresponding uncertainties in blue, and measured  $V_s$  in black. b) Zoom view of a blue predicted  $V_s$  log (images modified from Phan and Naeini, 2019).

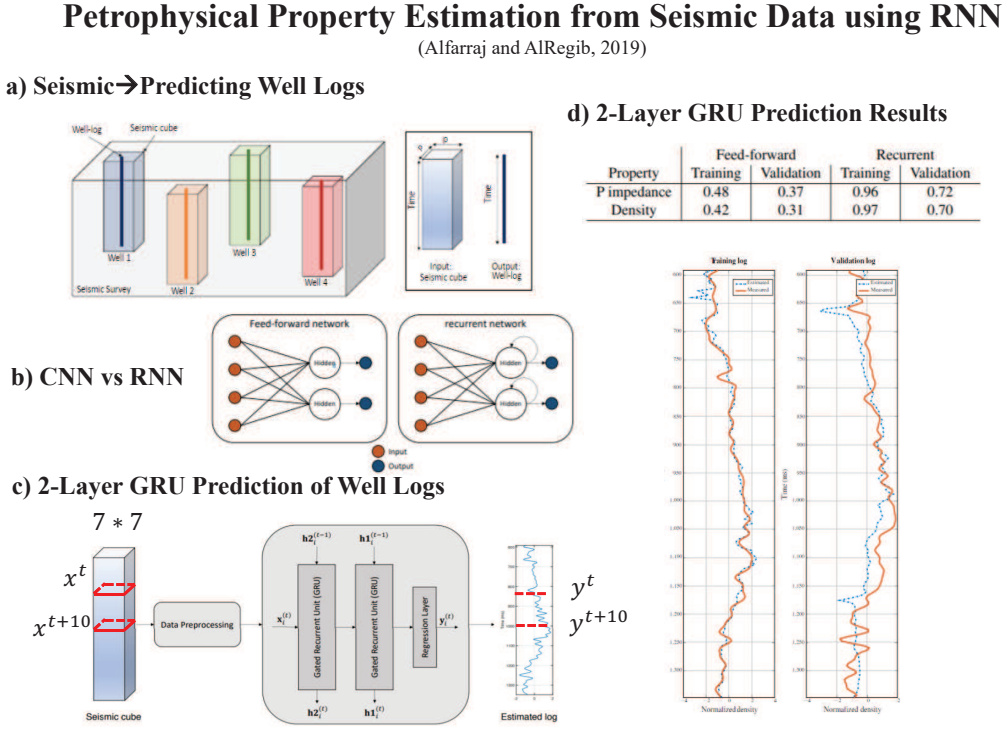


Figure 12.13: a) The GRU training pairs consist of the input cubes of seismic data and the output data are the well logs represented by the skinny colored tubes. b) Comparison between a simple NN architecture on the left and a RNN architecture on the right which recurs the output back into the network. c) Workflow of 2-layer GRU prediction of well log from seismic data and d) results from a test case for predicting density from seismic traces. These data pairs were recorded in the North Sea.

were 32 unknown parameters in the GRU. Three wells and the accompanying seismic data were used for training, and one well with seismic data were used for validation. The results are shown in Figure 12.13d, where the GRU was able to the density log from seismic data much more accurately than the feed-forward network. The feed-forward network was not able to properly train on such a small data set.

## 12.5 Summary

In summary, the RNN is a type of neural network that efficiently accounts for sequences of correlated atoms, where the atoms can be, for example, individual words, letters, pictures, well log readings, or sounds. The sequence of input vectors  $\mathbf{x}_t$  can be processed by applying the recurrence formula at every time step to get the new hidden-state vector  $\mathbf{h}_t$ :

$$\mathbf{h}_t = f_{\mathbf{w}, \mathbf{b}_h}(\mathbf{h}_{t-1}, \mathbf{x}_t), \quad (12.17)$$



where  $f_{\mathbf{W}, \mathbf{b}_h}(\cdot)$  represents a function with the parameters  $\mathbf{W}$  and  $\mathbf{b}_h$ . In general  $\mathbf{W}$  might represent the matrix coefficients for both the hidden-state, cell state, forget state, bias and input state tensors. For VRNN, the  $\mathbf{W}$  matrix is represented by  $[\mathbf{W}_{hh} \mathbf{W}_{xh}]$  and  $\mathbf{h}_t = \tanh([\mathbf{W}_{hh} \mathbf{W}_{xh}](\mathbf{x}_{t-1}, \mathbf{x}_t))$  and the output is  $\mathbf{y}_t = \mathbf{W}_{hy} \mathbf{h}_t$ . A softmax is usually applied to  $\mathbf{y}_t$  to get the component of the loss function.

The problem with VRNN is that a vanishing (or exploding) gradient restricts its short-term memory with no more than a few recently used input atoms. For sentence translation this means not more than a few recently used words can be remembered well. To lengthen the memory time we have the LSTM and GRU networks, and their variations. The key idea is to incorporate additive thresholding functions that prevent a vanishing gradient deep into the network. For example, the GRU derivative  $\partial \mathbf{h}_t / \partial \mathbf{h}_{t-1}$  in equation 12.15d gives the value 1 at  $\mathbf{z}_t = 1$ , which can be multiplied many times over and not vanish like the concatenated derivatives of a sigmoid function. The vanishing gradient is also not a problem with LSTMs because the output cell state is a weighted sum of the previous cell state that constructs a superhighway where information from the distant past can flow unimpeded. If an exploding gradient is a problem then clipping the gradient above a specified threshold is an ad hoc remedy (Goodfellow et al., 2016).

RNN can be used for many different types of mappings.

- One-to-many mapping of atoms to molecules, where the output molecule is the caption associated with the input atom of a picture.
- Many-to-many mapping where the input sentence from one language is translated to the output of another language. It could also be used for classifying videos, where the input is a sequence of frames and the output is a written description of the video's contents. In the case of the well log example, the readings from several different well logs are the input and the output is the well log of another type.
- Many-to-one mappings such as the input being a sequence of words in a movie or restaurant review and the output is an estimation of the sentiment of the review: good or bad.

Finally, we quote from Greff et al. (2014): *"Neural networks can be tricky to use for many practitioners compared to other methods whose properties are already well understood. This has remained a hurdle for newcomers to the field since a lot of practical choices are based on the intuitions of experts, as well as experiences gained over time."*

Recent improvements of RNN include Adaptive Computation Time, Neural Programmers, Attentional Interfaces, and Neural Turing Machines ([distill.pub/2016/augmented-rnns/](http://distill.pub/2016/augmented-rnns/)). They all rely on the same underlying trick known as attention. In attention, the RNNs focus on parts of the input that are most important for determining an accurate output. For example, in a movie review attention is focused on a pre-defined library of phrases such as *"worst movie ever"* or *"a real snoozer"* to decide if it's a good or bad movie. There is evidence that a 2D convolutional based neural network with causal convolution can outperform both RNN/LSTM and Attention-based models. Some even go as far to say *Drop your RNN and LSTM, they are no good!* (<http://towardsdatascience.com/the-fall-of-rnn-lstm-2d1594c74ce0> and <https://medium.com/swlh/attention-please-forget-about-recurrent-neural-networks-8d8c9047e117>).

The demise of standard RNNs was accelerated by the introduction of the Transformer architecture by Google researchers (Vaswani et al., 2017), who claimed that Transformers were faster and more accurate than LSTM and GRUs for many sequential processing tasks in NLP (Culurciello, 2018; <https://analyticsindiamag.com/why-transformers-are-increasingly-becoming-as-important-as-rnn-and-cnn/>). Within a few years of its introduction, the Transformer model has become the architecture of choice for many NLP problems (Wolf et al., 2020), replacing RNN models such as LSTMs. The Transformer model, the self-attention mechanism and their applications are discussed in Chapter 13.

## 12.6 Exercises

1. Derive the explicit formulas for  $\partial\epsilon_t/\partial\mathbf{h}_k$  and  $\partial^+\mathbf{h}_k/\partial W$  in equations 12.20. Compress these formulas into vector forms and express the formula for the gradient  $\partial\epsilon/\partial W$  in the form of matrix-vector multiplications, which includes the expression in equation 12.28.
2. For the VRNN architecture, derive the formula for the gradient  $\partial\epsilon_t/\partial V$ . Why doesn't it have the summation over the  $k$  indices as seen in equation 12.21?
3. For the VRNN architecture, derive the formula for the gradient  $\partial\epsilon_t/\partial U$ . In this case why does it have summation over indices as in equation 12.21? Derive the formulas for the gradients of the bias vectors.
4. Write a MATLAB code for a simple VRNN. In this case the input is a cosine function  $\sin t$  and the output is  $\cos t$ . Quantize the values of the input and output and describe your dictionary and one-hot coding.
5. Wisdom et al. (2016) show that the solution to the stacked RNN problem can be recast as the solution to an  $l_2$  minimization problem with  $l_1$  constraints. Their solution to minimizing the negative log-likelihood solves the following optimization problem:

$$\min_{\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_T} \sum_{t=1}^T \left( \frac{1}{2} \|\mathbf{x}_t - \mathbf{A}\mathbf{D}\mathbf{h}_t\|_2^2 + \lambda_1 \|\mathbf{H}_T\|_1 + \frac{\lambda_2}{2} \|\mathbf{D}\mathbf{h}_t - \mathbf{F}\mathbf{D}\mathbf{h}_{t-1}\|_2^2 \right), \quad (12.18)$$

where the regularization parameters are  $\lambda_1 = 2\tilde{\sigma}^2\nu_1$  and  $\lambda_2 = 2\tilde{\sigma}^2\nu_2$ . This problem can be numerically solved by a ISTA-like algorithm, except they name it sequential iterative soft-thresholding algorithm (SISTA). Relate the variables in equation 12.18 to the  $\mathbf{W}$ ,  $\mathbf{V}$ ,  $\mathbf{b}$  and  $\mathbf{U}$  in the VRNN formulation. Recast Algorithm 1 in Wisdom et al. (2016) in terms of these VRNN variables.

6. Repeat exercise 4 except use the SISTA algorithm in exercise 5.

## 12.7 Computational Labs

1. Go to the *Blind Deconvolution RNN* lab in *LAB1/Chapter.CNN.RNN/RNN\_lab.html* and run the MATLAB code.

## 12.8 Appendix: Vanishing Gradients

The gradient of the total loss function in equation 12.9 w/r to the component  $W$  of the  $\mathbf{W}$  matrix is

$$\frac{\partial\epsilon}{\partial W} = \frac{1}{T} \sum_{t=1}^T \frac{\partial\epsilon_t}{\partial W}, \quad (12.19)$$

where  $W$  represents an element of the  $\mathbf{W}$  matrix. For example, if this matrix element is  $W_{kl}$  then  $W \rightarrow W_{kl}$ .

Each loss  $\epsilon_t$  at time  $t$  in equation 12.9 is a composite function of the previous hidden-state vectors  $\mathbf{h}_{t-1}, \mathbf{h}_{t-2}, \dots$ , which in turn are each functions of  $\mathbf{W}$ , as illustrated in Figure 12.14. We can then express  $\epsilon_t = \epsilon(\mathbf{h}_t, \mathbf{h}_{t-1}, \mathbf{h}_{t-2}, \dots, \mathbf{h}_1)_t$  as a multivariable function, so that the multivariable

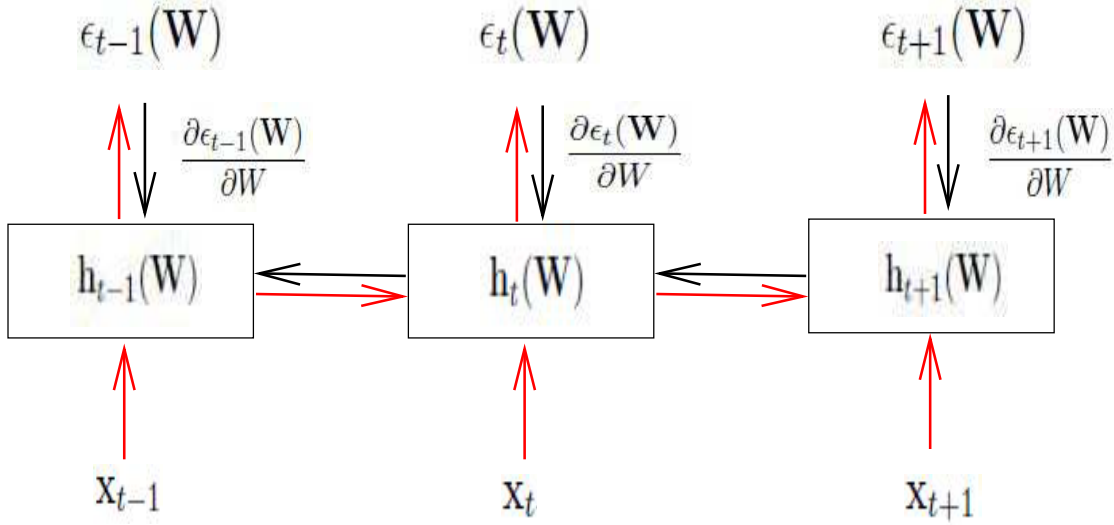


Figure 12.14: Forward (red arrows) and backward (black arrows) of an RNN.

chain rule<sup>7</sup> says that the partial derivative  $\partial \epsilon_t / \partial W$  in equation 12.19 is given by

$$\begin{aligned}
 \frac{\partial \epsilon_t(\mathbf{W})}{\partial W} &= \sum_{k=1}^t \frac{\partial \epsilon_t}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial W}, \\
 &= \sum_{k=1}^t \frac{\partial \epsilon_t}{\partial \mathbf{h}_t} \overbrace{\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k}}^{H \times H \text{ Jacobian}} \frac{\partial^+ \mathbf{h}_k}{\partial W},
 \end{aligned} \tag{12.20}$$

where

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} = \Pi_{t \geq j > k} \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}}, \tag{12.21}$$

and  $\partial^+$  in equation 12.20 indicates the immediate derivative notation<sup>8</sup> used in Pascanu et al. (2013). Equation 12.21 follows by the chain rule that says that the derivative of a composite function  $\mathbf{h}_t = f(\mathbf{h}_{t-1}(\mathbf{h}_{t-2}(\dots)))$  is a concatenated chain of derivatives, starting with the outermost composite function and ending with the one under differentiation.

The  $H \times H$  Jacobian matrix<sup>9</sup> is given by

$$\mathbf{J} = \partial \mathbf{h}_t / \partial \mathbf{h}_{t-1} = \frac{\partial(h_{t,1}, h_{t,2}, \dots, h_{t,H})}{\partial(h_{t-1,1}, h_{t-1,2}, \dots, h_{t-1,H})}, \tag{12.22}$$

<sup>7</sup>[en.wikipedia.org/wiki/Chain\\_rule](https://en.wikipedia.org/wiki/Chain_rule)

<sup>8</sup>Immediate derivative means that  $\partial^+ \mathbf{h}_k / \partial W$  only sees  $\mathbf{h}_k = \sigma(\mathbf{W}\mathbf{h}_{k-1} + \mathbf{U}\mathbf{x}_k + \mathbf{b}_k)$  as an *explicit* function of  $W$ , and ignores the fact that  $\mathbf{h}_{k-1}$  is implicitly a function of  $W$ .

<sup>9</sup>We are abiding by the definition of the Jacobian in [en.wikipedia.org/wiki/Jacobian\\_matrix\\_and\\_determinant](https://en.wikipedia.org/wiki/Jacobian_matrix_and_determinant). Some authors refer to the Jacobian as the transpose of  $\mathbf{J}$  in equation 12.23.

in equation 12.21 takes the explicit form

$$\begin{aligned} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} &= \left[ \frac{\partial \mathbf{h}_t}{\partial h_{t-1,1}} \quad \frac{\partial \mathbf{h}_t}{\partial h_{t-1,2}} \cdots \frac{\partial \mathbf{h}_t}{\partial h_{t-1,H}} \right], \\ &= \begin{bmatrix} \frac{\partial h_{t,1}}{\partial h_{t-1,1}} & \frac{\partial h_{t,1}}{\partial h_{t-1,2}} & \cdots & \frac{\partial h_{t,1}}{\partial h_{t-1,H}} \\ \frac{\partial h_{t,2}}{\partial h_{t-1,1}} & \frac{\partial h_{t,2}}{\partial h_{t-1,2}} & \cdots & \frac{\partial h_{t,2}}{\partial h_{t-1,H}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial h_{t,H}}{\partial h_{t-1,1}} & \frac{\partial h_{t,H}}{\partial h_{t-1,2}} & \cdots & \frac{\partial h_{t,H}}{\partial h_{t-1,H}} \end{bmatrix}, \end{aligned} \quad (12.23)$$

where  $[\mathbf{J}]_{ij} = \frac{\partial h_{t,i}}{\partial h_{t-1,j}}$ .

From equation 12.3 the  $i^{th}$  component of  $\mathbf{h}_t$  is

$$h_{t,i} = \sigma \left( \sum_{j=1}^H \overbrace{W_{ij} h_{t-1,j}}^{arg_i} + (\mathbf{U}\mathbf{x}_t)_i + b_i \right), \quad (12.24)$$

so that differentiating  $arg_i$  (defined by the overbrace in equation 12.24) w/r to  $h_{t-1,k}$  gives

$$\frac{\partial \sigma(arg_i)}{\partial h_{t-1,k}} = W_{ik}. \quad (12.25)$$

Defining  $\sigma(arg_i)' = \partial \sigma(arg_i) / \partial arg_i$  allows the Jacobian components in equation 12.23 to be expressed as

$$\begin{aligned} J_{ik} = \frac{\partial h_{t,i}}{\partial h_{t-1,k}} &= \frac{\partial \sigma(arg_i)}{\partial arg_i} \overbrace{\frac{\partial arg_i}{\partial h_{t-1,k}}}^{eq. 12.25: W_{ik}}, \\ &= \overbrace{\sigma(arg_i)'}^{D_{t,i}} W_{ik}, \end{aligned} \quad (12.26)$$

which can be expressed in matrix form as

$$\mathbf{J} = \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} = \mathbf{D}_t \mathbf{W}. \quad (12.27)$$

Here  $\mathbf{D}_t$  is a  $H \times H$  diagonal matrix such that  $[\mathbf{D}_t]_{ip} = \delta_{ip} \sigma'((\mathbf{W}\mathbf{h}_{t-1})_i + (\mathbf{U}\mathbf{x}_t)_i + b_i)'$ .

Now we can put it all together to show that we have a vanishing gradient problem. Plugging equation 12.27 into equation 12.21 gives

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} = \Pi_{t \geq j > k} \mathbf{D}_j \mathbf{W}. \quad (12.28)$$

The  $\sigma'$  is the derivative of a tanh function where its absolute value never exceeds 1. Therefore, the norm of equation 12.28 is bounded above by  $\|\mathbf{W}\|$  raised to the  $t^{th}$  power. If  $\|\mathbf{W}\| < 1$  is less than one then the  $H \times H$  Jacobian  $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k}$  and the gradient in equation 12.19 vanish for large values of  $t - k$ . This is consistent with the empirical evidence that VRNN is not a practical natural language

translation method if a memory significantly larger than a few words is required. On the other hand, if the  $\|\mathbf{W}\| \gg 1$  then the gradient can explode.

## 12.9 Appendix: Long Short-Term Memory Architecture

The LSTMGRUs architecture is shown in Figure 12.15b and consists of three gates, each indicated by the Hadamard product  $\odot$  symbols associated with the element-by-element multiplication of a gate filter and input vector. We will now present the feed-forward propagation formulas for each gate and examine the role of each gate in regulating what information passes through the network (Graves and Schmidhuber, 2005). Don't worry if you can't understand why these gates work, but somehow the coefficients can be learned so LSTM produces some amazing results.

### Forget Gate

The first step in the LSTM unit is to decide which parts of the  $H \times 1$  cell-state vector  $\mathbf{c}_{t-1}$  get used. This decision is made by the  $H \times 1$  forget vector  $\mathbf{f}_t$  in Figure 12.16b. If  $\mathbf{f}_t$  has a zero in its  $i^{th}$  element then the Hadamard product  $\mathbf{f}_t \odot \mathbf{c}_{t-1}$  will also have a zero in its  $i^{th}$  element, and therefore forget the cell-state value, i.e. word, at this element. If  $\mathbf{f}_t$  has a one in its  $i^{th}$  element then the product  $\mathbf{f}_t \odot \mathbf{c}_{t-1}$  remembers the  $i^{th}$  element of  $\mathbf{c}_{t-1}$  completely. Any fractional value  $\mathbf{f}_t$  between 0 and 1 downgrades the value at the  $i^{th}$  element. The element values of  $\mathbf{f}_t$  are determined by the forget gate formula

$$\mathbf{f}_t = \sigma(\mathbf{W}_f(\mathbf{h}_{t-1}, \mathbf{x}_t) + \mathbf{b}_f), \quad (12.29)$$

where  $\mathbf{b}_f$  is the  $H \times 1$  bias vector and  $0 \leq c_{t-1,i} \leq 1$ . Here,  $\sigma$  is the sigmoid function,  $\mathbf{W}_f = [\mathbf{W} \ \mathbf{U}]$  is the  $H \times 2H$  matrix formed by concatenating the matrices  $\mathbf{W}$  and  $\mathbf{U}$  along the row direction; and  $(\mathbf{h}_{t-1}, \mathbf{x}_t)$  is the vector formed by concatenating the vectors  $\mathbf{h}_{t-1}$  and  $\mathbf{x}_t$  along the column direction. Gers et al. (1999) introduced the first forget gate as a modification of the LSTM network, which allows the LSTM to reset its own state.

Roughly speaking, a *skip* connection is the information superhighway (Srivastava et al., 2015) in the RNN architecture, as denoted by the horizontal arrow<sup>10</sup> in Figure 12.16a. The  $\mathbf{c}_t$  is denoted as the cell state in Figure 12.16a. An example is now taken from *colah.github.io/posts/2015-08-Understanding-LSTMs/*: "a language model trying to predict the next word based on all the previous ones. In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.". How did it know that the previous word needed to be forgotten? Well, supposedly the network has been sufficiently trained to get the correct learned weights for the matrices and bias vector.

### Input Gate

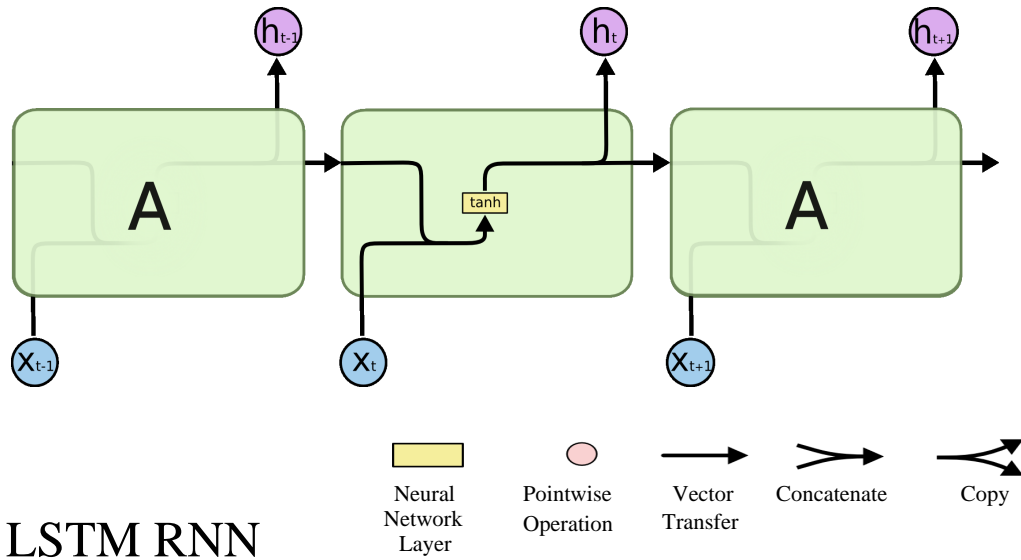
The next step determines how to additively update the old cell state  $\mathbf{c}_{t-1}$  with the new cell adjustment  $\tilde{\mathbf{c}}_t$ :

$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c(\mathbf{h}_{t-1}, \mathbf{x}_t) + \mathbf{b}_c). \quad (12.30)$$

This consists of two parts as illustrated in Figure 12.17. The first step is to decide how much to

<sup>10</sup>Notice that there is no explicit thresholding function applied to the input data  $\mathbf{c}_{t-1}$  along this horizontal highway. Therefore the gradient of the loss function for information entering this superhighway will not be explicitly multiplied by the coefficients of the unknown matrices. This means that the gradient of this information will not be damped by powers of the matrix coefficients.

## a) Vanilla RNN



## b) LSTM RNN

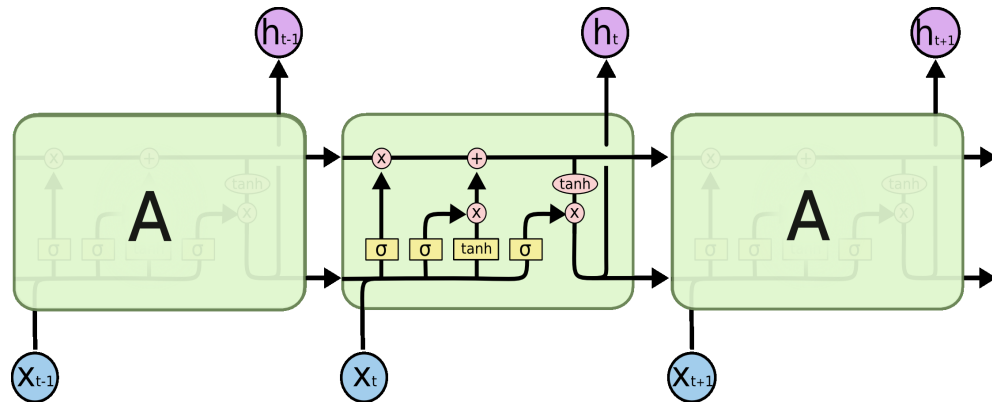


Figure 12.15: Comparison between a) standard VRNN and b) LSTM architectures. The key improvement with the LSTM is that there is an information superhighway (horizontal arrow in b) where the gradient does not vanish and the gates (denoted by the Illustration adapted from [colah.github.io/posts/2015-08-Understanding-LSTMs/](https://colah.github.io/posts/2015-08-Understanding-LSTMs/)).

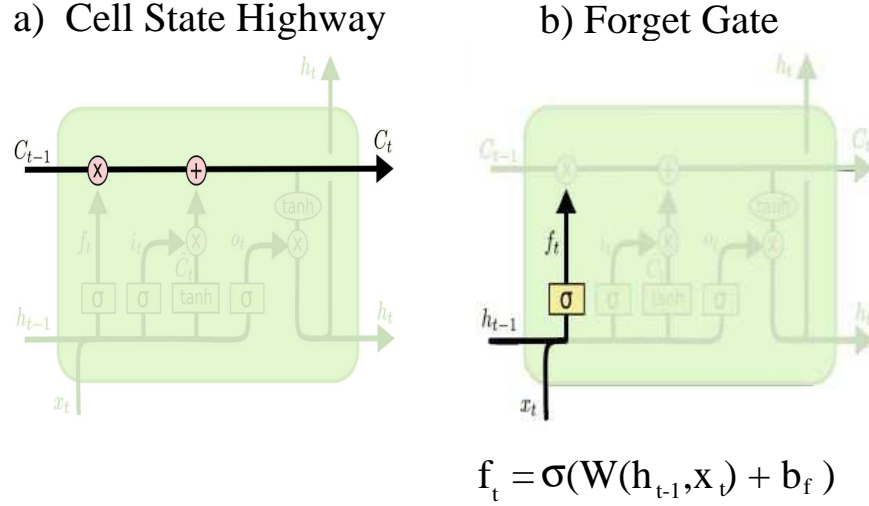


Figure 12.16: a) Cell state where LSTM unit cell is outlined in green and the *cell* highway denoted by the black arrow. b) Forget gate. Illustration adapted from [colah.github.io/posts/2015-08-Understanding-LSTMs/](https://colah.github.io/posts/2015-08-Understanding-LSTMs/).

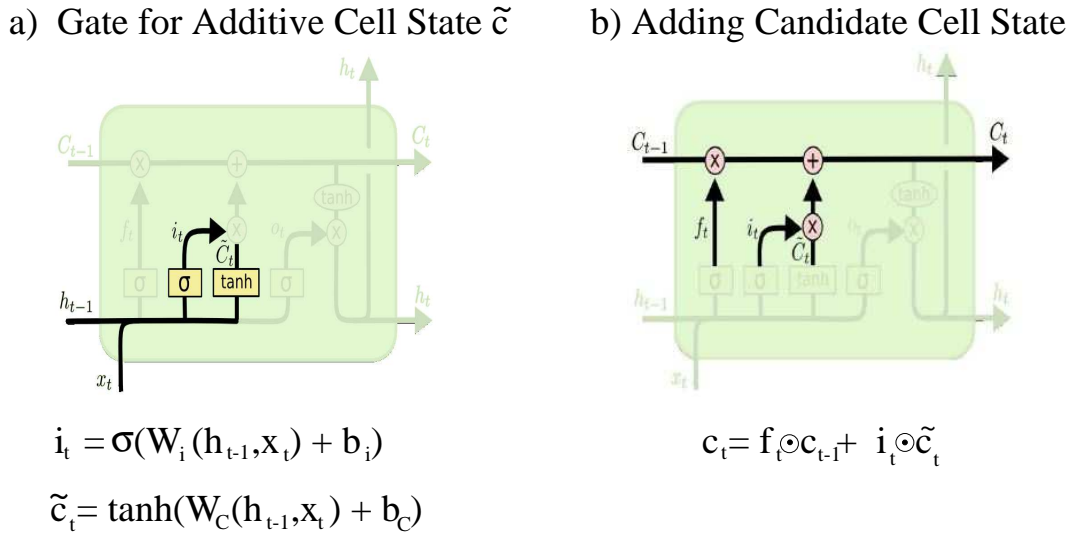


Figure 12.17: a) Gate for the additive cell state and b) adding the old cell state to the candidate one. Illustration adapted from [colah.github.io/posts/2015-08-Understanding-LSTMs/](https://colah.github.io/posts/2015-08-Understanding-LSTMs/).

admit into the additive cell state with the input gate

$$\mathbf{i}_t = \sigma(\mathbf{W}_i(\mathbf{h}_{t-1}, \mathbf{x}_t) + \mathbf{b}_i), \quad (12.31)$$

where  $\mathbf{W}_i$  is the  $H \times H$  input matrix and  $\mathbf{b}_i$  is the  $H \times 1$  input bias vector. Here,  $\sigma$  is the sigmoid function that makes the decision of what to keep and forget with the element-by-element Hadamard product  $\mathbf{i}_t \odot \tilde{\mathbf{c}}_t$ . The new cell state is a weighted summation of the old cell state  $\mathbf{c}_{t-1}$  and the additive cell state  $\tilde{\mathbf{c}}_t$ :

$$\mathbf{c}_t = \overbrace{\mathbf{f}_t \odot \mathbf{c}_{t-1}}^{\text{skip-like long-term memory}} + \overbrace{\mathbf{i}_t \odot \tilde{\mathbf{c}}_t}^{\text{short-term memory}}. \quad (12.32)$$

Here  $\mathbf{W}_c$  is the  $H \times H$  additive cell matrix and  $\mathbf{b}_c$  is the  $H \times 1$  additive cell bias vector. Unlike terms whose gradients vanish, the new cell state is the weighted addition of the old cell state and new additive cell state. There is no explicit activation operation applied to  $\mathbf{c}_{t-1}$  so its derivative with respect to  $\mathbf{c}_{t-n}$  will be free of the matrix coefficients. For example, assume  $\mathbf{c}_{t-1} = \mathbf{c}_{t-30}$  and the forget vectors have allowed this condition. In this case, the estimate of the gradient w/r to the coefficients of  $\mathbf{W}_c$  will not vanish for  $\frac{\partial \mathbf{c}_{t-1}}{\partial \mathbf{c}_{t-30}}$ . This superhighway of information for the cell state adds pre-existing information to the adjusted cell state without applying the activation function.

### Output Gate

The output gate is displayed in Figure 12.18 which decides what is contained in the new hidden-state vector  $\mathbf{h}_t$  given by

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t). \quad (12.33)$$

where the output vector  $\mathbf{o}_t$  is

$$\mathbf{o}_t = \sigma(\mathbf{W}_o(\mathbf{h}_{t-1}, \mathbf{x}_t) + \mathbf{b}_o). \quad (12.34)$$

Here,  $\mathbf{W}_o$  is the  $H \times H$  output matrix,  $\mathbf{b}_o$  is the  $H \times 1$  output bias vector.

The compact form of the LSTM equations is the following.

$$\begin{aligned} \begin{bmatrix} \mathbf{f}_t \\ \mathbf{i}_t \\ \tilde{\mathbf{c}}_t \\ \mathbf{o}_t \end{bmatrix} &= \begin{bmatrix} \sigma \\ \sigma \\ \tanh \\ \sigma \end{bmatrix} \odot \begin{bmatrix} \mathbf{W}_f & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{W}_i & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{W}_c & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{W}_o \end{bmatrix} \begin{bmatrix} (\mathbf{h}_t, \mathbf{x}_t) \\ (\mathbf{h}_t, \mathbf{x}_t) \\ (\mathbf{h}_t, \mathbf{x}_t) \\ (\mathbf{h}_t, \mathbf{x}_t) \end{bmatrix} + \begin{bmatrix} \mathbf{b}_f \\ \mathbf{b}_i \\ \mathbf{b}_c \\ \mathbf{b}_o \end{bmatrix} \\ \mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t, \\ \mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t), \end{aligned} \quad (12.35)$$

where  $\mathbf{W}$  is a  $H \times H$  matrix. In summary, the roles of the input, learned parameters and gates in equation 12.35 are the following.

- $\mathbf{x}_t \in \mathbb{R}^D$  Input vector to the LSTM.
- $\mathbf{W} \in \mathbb{R}^{H \times D}$ ,  $\mathbf{U} \in \mathbb{R}^{H \times D}$ ,  $\mathbf{b} \in \mathbb{R}^H$  Weight matrices and bias vector which are learned during training.
- $\mathbf{f}_t \in \mathbb{R}^H$  Forget gate: How much should we erase  $\mathbf{c}_{t-1}$ .
- $\mathbf{i}_t \in \mathbb{R}^H$  Input gate: activation vector for input/output gate.
- $\mathbf{o}_t \in \mathbb{R}^H$  Output gate: activation vector for output gate.



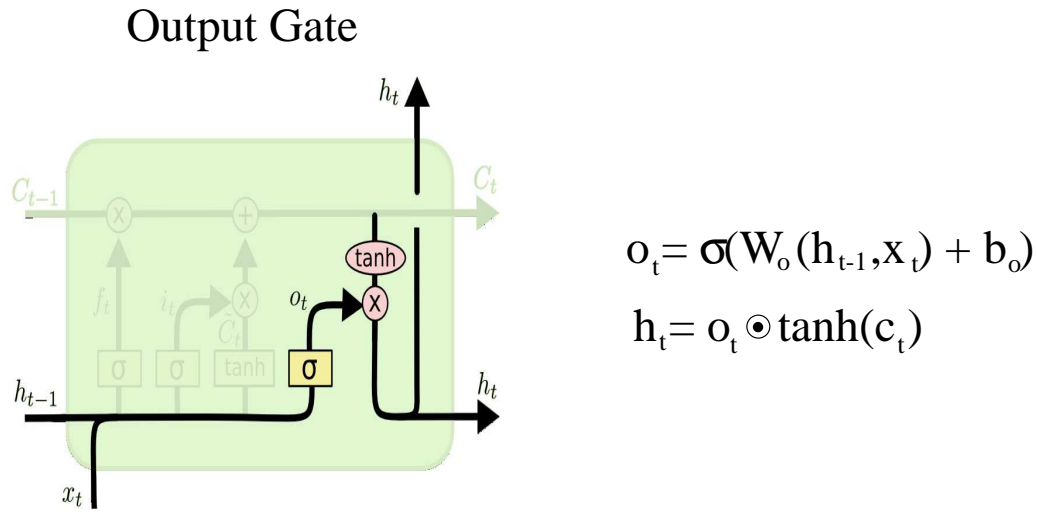


Figure 12.18: a) Gate for the additive cell state and b) adding the old cell state to the candidate one. Illustration adapted from [colah.github.io/posts/2015-08-Understanding-LSTMs/](https://colah.github.io/posts/2015-08-Understanding-LSTMs/).

- $\mathbf{h}_t \in \mathbb{R}^H$  Hidden state vector.
- $\mathbf{c}_t \in \mathbb{R}^H$  Cell state gate.
- $\tilde{\mathbf{c}}_t \in \mathbb{R}^H$  Cell input activation vector.



## Chapter 13

# Self-attention and Transformers

In 2017, Google Brain researchers (Vaswani et al., 2017) introduced the attention architecture as a faster and more accurate replacement of LSTM and GRUs for many sequential processing tasks in NLP (Culurciello, 2018; <https://analyticsindiamag.com//why-transformers-are-increasingly-becoming-as-important-as-rnn-and-cnn/>). The Transformer architecture extended the limited LSTM and GRU recall of a few-hundred words to that of the entire input data set with thousands of words. Instead of processing the input data sequentially, a Transformer can process the entire sequence all at once and so its computations, unlike LSTM or GRU, can be easily parallelized on distributed memory computers. This can result in a processing speedup that can be an order-of-magnitude faster than processing sequential data by a RNN.

The most important component of the Transformer is that it uses the self-attention mechanism ([https://en.wikipedia.org/wiki/Transformer\\_\(machine\\_learning\\_model\)](https://en.wikipedia.org/wiki/Transformer_(machine_learning_model))) to upweight pairs of words with similar meaning and downweight dissimilar pairs of words (Vaswani et al., 2017). This is important for language translation because related words in a sentence can be located far from one another in one language, and be close to one another in the translation. Identifying the similarity and common meaning of word pairs provide the context of words in a sentence. For example, the sentence *Bank of the river* says that the words *bank* and *river* are related to one another even though they are not adjacent to one another in the sentence. The self-attention mechanism will assign a large weight  $k_{14}$  that indicates that the first and fourth words are related to one another, and therefore provide a proper meaning and context for these words.

Within a few years of its introduction, the Transformer model has become the architecture of choice for some NLP problems (Wolf et al., 2020), replacing RNN models such as LSTMs. We will now explain the attention concept, and then describe its use in the Transformer architecture.

### 13.1 Attention

The self-attention, aka attention, mechanism for language translation focuses on related words in a sentence or in a sequence of sentences. It does this by assigning large weights to pairs of words that are related to one another and downweighting unrelated pairs of words. For example, each word vector  $\mathbf{x}_i \forall i \in \{1, 2, \dots, W\}$  in a  $W$ -word sentence can be assigned an  $N \times 1$  embedding vector  $\mathbf{v}_i$ , where each element of  $\mathbf{v}_i$  is a number related to a *characteristic or meaning* of that word.

For example, the meaning of the word *Bank* in the sentence at the top of Figure 13.1a is ambiguous unless other words in the sentence are taken into account. Bank can be associated with the characteristics of, for example, money, the lean angle of a racing track or the sedimentary deposits along the arcuate side of a river. In the context of the sentence *Bank of the river*, *bank* is associated with the sedimentary deposits that border the side of a river. But how does a computer judge which words in the sentence have similar characteristics to the word *bank*?

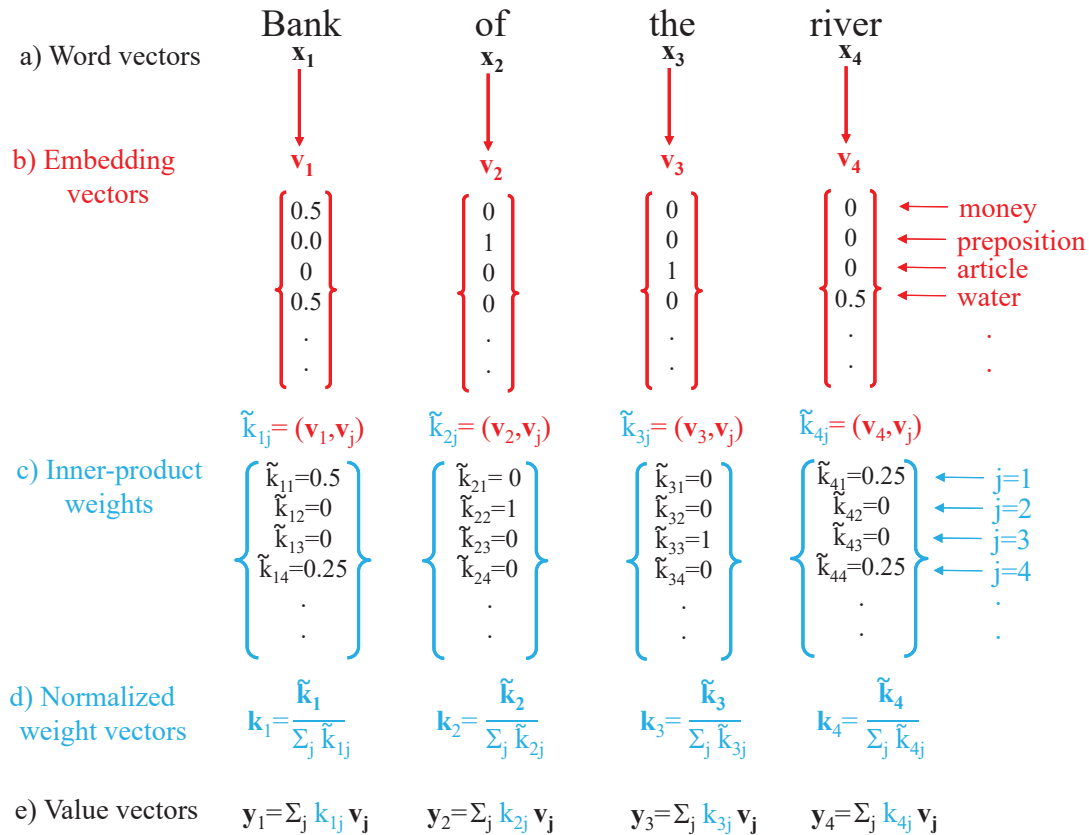


Figure 13.1: Untrained self-attention workflow with the aim of getting the Value vectors  $\mathbf{y}_i$  that indicate the meaning of the words  $\mathbf{x}_i$  in the input sentence. Example adopted from the Youtube video at <https://learning.rasa.com/transformers/self-attention/>.

Word similarities can be deduced from the word-embedding vector

$$\mathbf{v} = (v_1, v_2, v_3, v_4, \dots) = (\text{money}, \text{preposition}, \text{article}, \text{water content}, \dots), \quad (13.1)$$

where the embedding elements, in the example of Figure 13.1b, correspond to the characteristics of the word's association with money, role as a preposition, assignment as an article, relationship to water content, and so on. A positive number can be assigned to each characteristic, where the maximum value of 1 indicates that the word might have a high correlation to that characteristic and 0 indicates no relationship. The word-embedding vector  $\mathbf{v}$  might have a dimension greater than a few hundred and can be drawn from a pre-existing embedding library for each unique word  $\mathbf{x}$  in the input data set.

The elements of the embedding vector for the first word  $\mathbf{x}_1 \rightarrow \text{bank}$  are given by the embedding vector  $\mathbf{v}_1$

$$\text{bank} \rightarrow \mathbf{v}_1 = (0.5, 0, 0, 0.5, \dots), \quad (13.2)$$

in Figure 13.1b where  $(\mathbf{v}_1)_1 = 0.5$  indicates that a commercial bank is associated with money or, if it refers to a river bank, is strongly associated with water so that  $(\mathbf{v}_1)_4 = 0.5$ . The prepositional and article characteristics  $(\mathbf{v}_1)_2 = (\mathbf{v}_1)_3 = 0$  in equation 13.1 are obviously assigned values of 0 for the noun *bank*. In contrast, the corresponding elements of the embedding vector  $\mathbf{v}_4$  for the word *river* are

$$\text{river} \rightarrow \mathbf{v}_4 = (0, 0, 0, 0.5, \dots), \quad (13.3)$$

which only has a large value of  $(\mathbf{v}_4)_4 = 0.5$  for the water-content characteristic<sup>1</sup>. However, the context of the sentence suggests that its subject matter is related to the bank of a river, so there is a contextual similarity between the words *water* and *bank*. This similarity can be quantified by taking the inner product of the first and fourth word-embedding vectors in Figure 13.1b to get

$$\begin{aligned} \tilde{k}_{14} &= (\mathbf{v}_1, \mathbf{v}_4), \\ &= (0.5, 0, 0, 0.5) \cdot (0, 0, 0, 0.5)^T = 0.25, \end{aligned} \quad (13.4)$$

in Figure 13.1c. The relatively large value of the inner product  $\tilde{k}_{14} = 0.25$  suggests a strong similarity in contextual meaning for the first and fourth words. Recognizing this similarity is essential for correctly translating this sentence from one language to another.

The weights  $\tilde{k}_{ij} = (\mathbf{v}_i, \mathbf{v}_j)$  are normalized in Figure 13.1d to give the normalized weight  $k_{ij}$ , where

$$k_{ij} = \frac{\tilde{k}_{ij}}{\sum_{j=1}^W \tilde{k}_{ij}}; \quad i, j \in \{1, 2, \dots, W\}. \quad (13.5)$$

Normalization allows us to recognize that relatively values of  $1 > k_{ij} >> 0$  close to unity indicate that the  $i^{\text{th}}$  input word is contextually similar to the  $j^{\text{th}}$  word. Correspondingly, small values  $1 >> k_{ij} > 0$  indicate that the  $i^{\text{th}}$  and  $j^{\text{th}}$  words are weakly related to one another.

The weighted sum of  $W$  word-embedding vectors in a sentence

$$\mathbf{y}_i = \sum_{j=1}^W k_{ij} \mathbf{v}_j, \quad (13.6)$$

gives the Value vector  $\mathbf{y}_i$  in Figure 13.1e, which provides a more contextual meaning to the  $i^{\text{th}}$  word

---

<sup>1</sup>For convenience, we will assume  $(\mathbf{v}_i)_k = 0$   $k \in \{5, 6, \dots\}$ .

compared to the embedding vector. For example, the embedding vector  $\mathbf{v}_4 = (0, 0, 0, 0.5)$  for the fourth word *river* in equation 13.3 suggests that it could only refer to a flowing river, but the Value vector for the fourth word is

$$\begin{aligned} \mathbf{y}_4 &= k_{41}\mathbf{v}_1 + k_{44}\mathbf{v}_4, \\ &= \alpha 0.25\mathbf{v}_1 + \beta 0.25\mathbf{v}_4, \end{aligned} \quad (13.7)$$

which is a weighted sum of the embedding vectors for the first word *bank* and the fourth word *river*. Thus, the value vector  $\mathbf{y}_4$  recognizes that the word *bank* might be related to *river* as in a *river bank*, and is not necessarily just a place to deposit money. Here,  $k_{12} = k_{13} = 0$  and the normalization terms are  $\alpha = 1/0.75$  and  $\beta = 1/0.5$ .

In summary, the untrained self-attention mechanism illustrated in Figure 13.1 computes a weighted sum of embedding vectors for each input word, where the weights are inner-products of embedding vectors. It is untrained in the sense that the weights are computed by simple inner products of embedding vectors from the input sequence and do not require a supervised learning procedure.

The problem with this untrained approach is that a word in a complicated sentence, such as "The bank of the river is where the First National bank will be built", might have contextual relationships with many other words in the sentence. The first word *bank* has the meaning of a river bank, while the second use of this word is that of a money bank. Moreover, the word *built* refers to the money bank, not the river bank. To extract the contextual meanings of these different combinations of words requires something more powerful than the untrained-attention mechanism. That is, we need an attention mechanism that can learn multiple contextual meanings of a single word. Such a capability exists by training a neural network from a large training library of different sentences.

The first step for transforming Figure 13.1 into a neural network is to replace the normalization operation with the softmax function in Figure 13.2, which allows for a probabilistic interpretation of the output. The second step is to incorporate learnable weights into the attention mechanism, as described in the next section. This will eventually include a fully connected neural network as described in section 13.2.1 that describes the multi-head attention mechanism.

## 13.2 Trained Attention

The untrained-attention architecture in Figure 13.2 can incorporate learnable weights by introducing the  $N \times N$  Query  $\mathbf{M}^Q$ , Value  $\mathbf{M}^V$  and Key  $\mathbf{M}^K$  matrices in Figure 13.3. The top row of embedding vectors  $\mathbf{v}(i)$  in Figure 13.3a are multiplied by the  $N \times N$  Query matrix  $\mathbf{M}^Q$  to give the  $N \times 1$  Query vector

$$\boldsymbol{\omega}_i = \mathbf{M}^Q \mathbf{v}_i \quad i \in \{1, 2, \dots, W\}, \quad (13.8)$$

where  $W$  is the number of words in the sentence. The weights in  $\mathbf{M}^Q$  are unknown but, as later sections show, these weights can be found by a gradient descent method applied to a large training set with sentences translated from one language to another. Similarly, the  $N \times 1$   $\mathbf{v}_i$  vectors along the leftside column of Figure 13.3b are multiplied by the Key matrix  $\mathbf{M}^K$  to give the  $N \times 1$  Key vector  $\tilde{\boldsymbol{\omega}}_i$ :

$$\tilde{\boldsymbol{\omega}}_i = \mathbf{M}^K \mathbf{v}_i \quad i \in \{1, 2, \dots, W\}. \quad (13.9)$$

The inner product of the Key and Query vectors gives the un-normalized weight

$$\tilde{k}_{ij} = (\tilde{\boldsymbol{\omega}}_i, \boldsymbol{\omega}_j) \quad i, j \in \{1, 2, \dots, W\}, \quad (13.10)$$

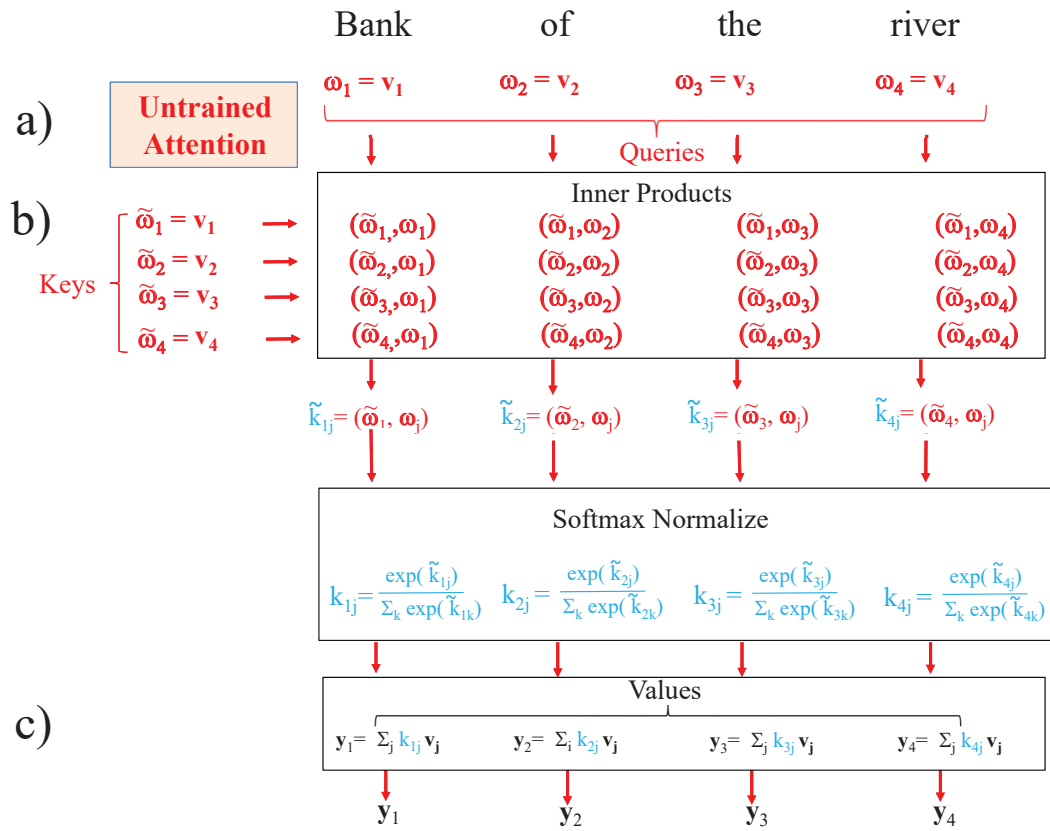


Figure 13.2: Untrained attention blocks obtained from Figure 13.1. Example inspired by Rasa's Youtube video at <https://youtu.be/wj3ZYbKKUHI>.

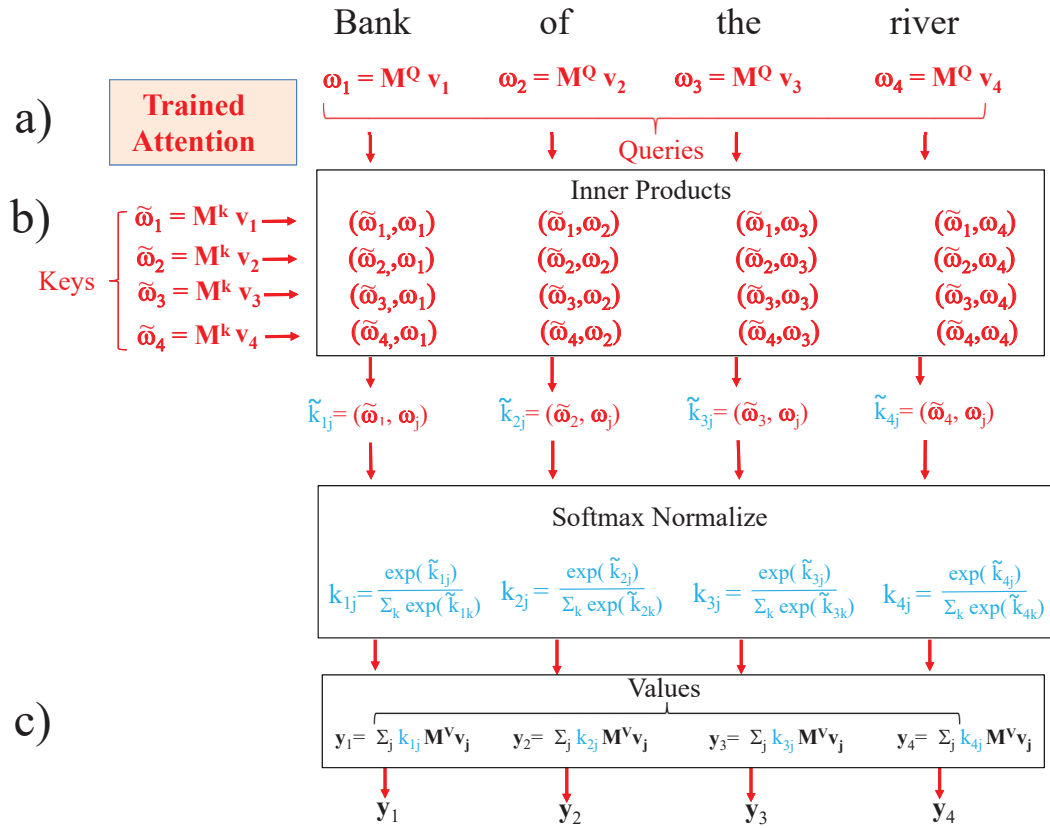


Figure 13.3: Trained attention blocks obtained from Figure 13.2 by multiplying  $v_i$  by the Query  $M^Q$ , Value  $M^V$  and Key  $M^K$  matrices. Example inspired by the Youtube video at <https://www.youtube.com/watch?v=23XUv0T9L5c&list=RDCMUCJ0V6493mLvqdiVwOKWBODQ&index=3>.



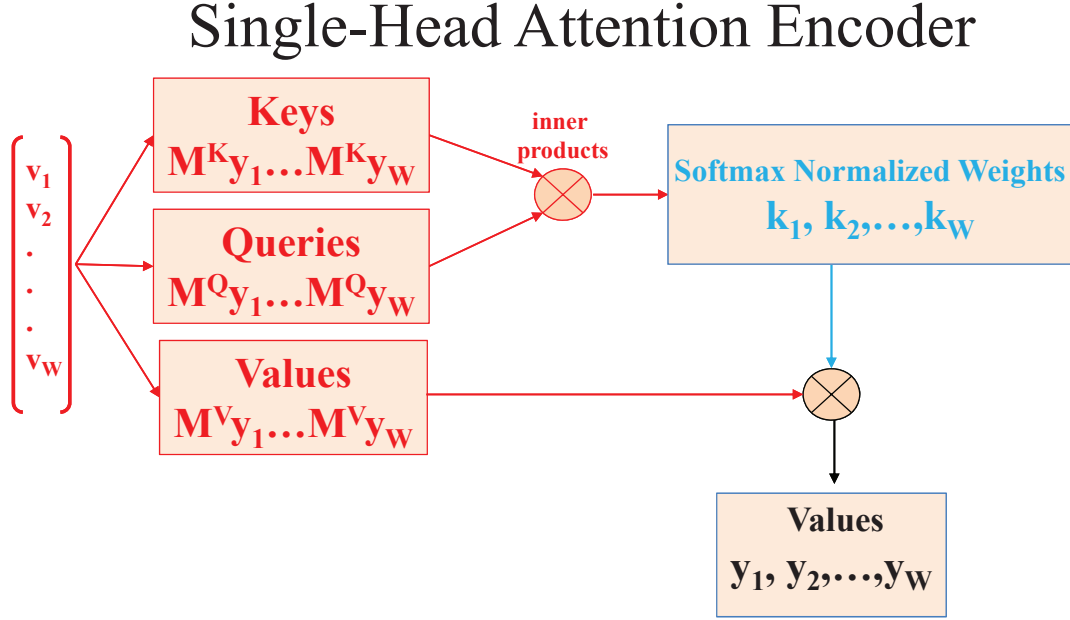


Figure 13.4: Single-head attention architecture, where  $k_{ij}$  is given in equation 13.11. Example adopted from Rasa’s Youtube video at <https://youtu.be/wj3ZYbKKUHL>.

in Figure 13.3b. To achieve a more robust convergence (Vaswani et al., 2017), the inner products of the embedding vectors that are input into the Key and Query matrices are divided by the square root of the dimension  $N$  of the embedding vectors. For example, if  $N = 64$  then the  $(\tilde{\omega}_i, \omega_j)$  is divided by 8. For this example, we assumed that the Key, Query and Value matrices are  $N \times N$  square matrices. In practice, these matrices can be rectangular with dimension  $N' \times N$ , where  $N' < N$  (Vaswani et al., 2017; <http://jalammr.github.io/illustrated-transformer/>).

The weights in equation 13.10 should be normalized, which can be done by the softmax operation

$$k_{ij} = \text{normalize}(\tilde{k}_{ij}) = \frac{\exp(\tilde{k}_{ij})}{\sum_j \exp(\tilde{k}_{ij})} \quad i, j \in \{1, 2, \dots, W\}, \quad (13.11)$$

where  $k_{ij}$  can be interpreted as the probability that the  $j^{\text{th}}$  word is similar to the  $i^{\text{th}}$  word. These normalized weights are then used to obtain the  $N \times 1$  Value vectors

$$\mathbf{y}_i = \sum_{j=1}^W k_{ij} \mathbf{M}^V \mathbf{v}_j, \quad i \in \{1, 2, \dots, W\}, \quad (13.12)$$

in Figure 13.3c. These Value vectors can provide more accurate contextual meanings to the words in the sentence than the embedding vectors  $\mathbf{v}_i$ .

The procedure described in Figure 13.3 is depicted as a single-head attention mechanism in the block diagrams of Figure 13.4. The three tan-colored blocks, the inner product and the softmax operations are classified as a single head of the attention mechanism. A boost in processing power can be attained by increasing the number of heads to get the multi-head attention architecture shown in Figure 13.5. The  $k^{\text{th}}$  head for the *Query* computes the matrix-vector multiplication  $\mathbf{M}^{Q^k} \mathbf{v}_i = \boldsymbol{\omega}_i^k$  to get the output vector  $\boldsymbol{\omega}_i^k$  for the  $k^{\text{th}}$  head. Here we assume  $h$  heads.

## Multi-Head Attention Encoder

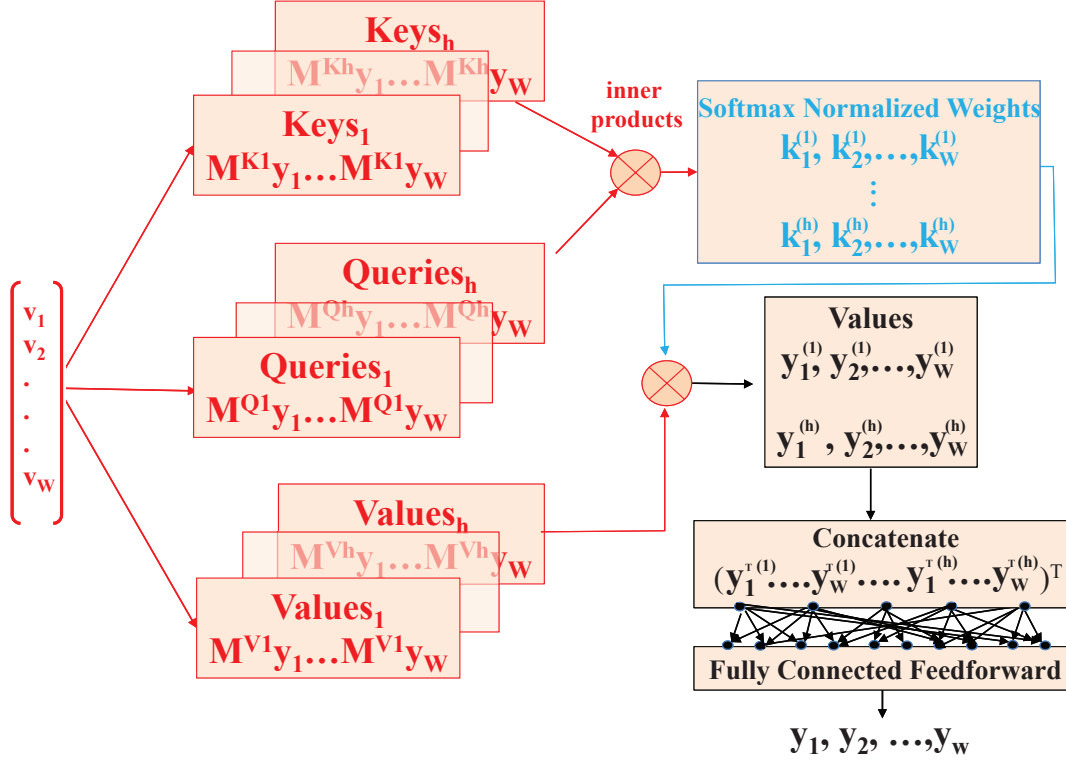


Figure 13.5: Multi-head attention architecture, where the  $h^{th}$ , e.g. Key  $\mathbf{M}^{Kh}$ , matrix has the symbol  $h$  appended to the superscript. The result of  $\mathbf{M}^{Kh}\mathbf{v}_i = \tilde{\omega}_i^{(h)}$ .

The inner products for the outputs from the Key and Query blocks are computed, which are then used as inputs into the softmax operation to get the vectors  $\mathbf{y}_i^{(k)}$   $k \in \{1, 2, \dots, h\}$ ,  $i \in \{1, 2, \dots, W\}$ . These outputs are concatenated together as the tall vector at the bottom right of Figure 13.5, and fed into a fully connected feed-forward operation to get a shorter sequence of  $N \times 1$  Value vectors  $\mathbf{y}_i$  for  $i \in \{1, 2, \dots, W\}$ .

### 13.2.1 Single- and Multi-head Attention Architectures

The single- and multi-head attention architectures in Figures 13.4-13.5 are depicted in the Vaswani et al. (2017) paper as the block diagrams in Figure 13.6.

- **Masking.** The masking operation is optional in the encoder. If it is used, then self-attention modules in the decoder can use masking to ensure the prediction of future words only uses inputs from previous words. See Figure 13.7c for an illustration of this mask.
- **Inner Products and Scaling.** The inner products of the Key and Query vectors are obtained. The *Scale* block in the single-head rectangle scales the inner product of the Key and Query vectors by  $1/\sqrt{d}$ , where  $d = N$  is the dimension of the word-embedding vector  $\mathbf{v}^{(i)}$ . This scaling is needed, otherwise large values of  $N$  might result in huge inner products in the argument of the softmax, which leads to small gradients.

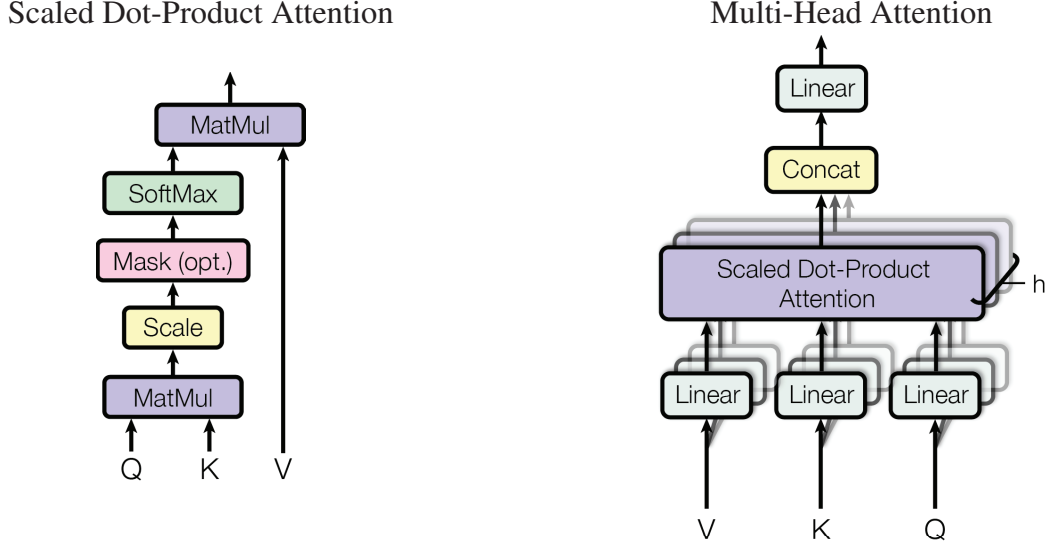


Figure 13.6: Block diagrams for single- and multi-head attention architectures in the Vaswani et al. (2017) paper. These are related to the block diagrams in Figure 13.4-13.5.

- **Softmax.** After scaling, the inner products of the scaled Query and Key vectors gives the un-normalized  $\tilde{k}_{ij}$ , which are fed into the softmax operation to give the normalized weights  $k_{ij}$ . These weights  $k_{ij}$  multiply the vector  $\mathbf{M}^V \mathbf{v}_j$ , and the product is summed over the  $j$  index to give the Value vector  $\mathbf{y}_i$  for a single head.
- **Symbolic Representation of Attention.** These operations are symbolically expressed in Vaswani et al. (2017) as

$$Attention(\mathbf{M}^Q, \mathbf{M}^K, \mathbf{M}^V) = softmax\left(\frac{\mathbf{M}^Q (\mathbf{M}^K)^T}{\sqrt{d}} \mathbf{M}^V\right), \quad (13.13)$$

which assumes that the embedding vectors are in row format. In our notation we assume the embedding vectors are in column format.

- **Block Diagram of Multi-head Attention.** The block diagram of the multi-head attention mechanism is shown on the right of Figure 13.6. It can also be expressed in a compact manner as

$$\begin{aligned} MultiHead(\mathbf{M}^Q, \mathbf{M}^K, \mathbf{M}^V) &= Concat(head_1, head_2, \dots, head_h) \mathbf{M}^O, \\ Attention(\mathbf{M}^{Q_i}, \mathbf{M}^{K_i}, \mathbf{M}^{V_i}) & \end{aligned} \quad (13.14)$$

where the  $\mathbf{M}^{Q_i}$ ,  $\mathbf{M}^{K_i}$ , and  $\mathbf{M}^{V_i}$  are projection matrices for the  $i^{th}$  head and the matrix coefficients are learnable. The output vectors of the heads after a scaled inner product and softmax operation are concatenated into a long sequence of vectors, which is then multiplied by the matrix  $\mathbf{M}^O$  to shape the result into a specified number of value vectors with a desired dimension. Similar to the other matrices, the coefficients of  $\mathbf{M}^O$  are learnable by supervised training. The dimension of the output vector after multiplication of the embedding vector

by the Key or Query matrices is  $d_K$ , but the dimension  $d_V$  of  $\mathbf{M}^V \mathbf{v}_i$  is not necessarily the same as  $d_K$ . The number of heads in the Vaswani et al. (2017) paper is selected to be 8,  $d_K = d_V = 64$ , and the dimension of the embedding vector is  $N = 512$ . Reducing the output dimension of each matrix-vector multiplication to  $d_K = d_V = 64$  makes the multi-head computational cost to be the same as that for a single-head for  $d_K = d_V = N = 512$ .

- **Positional Encoding Vector  $\mathbf{p}_i$ .** Information about the order of words in a sentence must be injected into the attention mechanism, as indicated by the round *pos* symbols in Figure 13.7. At first glance, we might just assign the number 1 to the first word, the number 2 to the second and so on. The problem with this approach is that the numbers can get very large for long sequences of words. Also, the maximum number of words in the training set might be less than that of the test sequences. Moreover, the training set might not have a certain number of words in a sequence that characterizes sentences in the test set. Consequently the training is incomplete, which might degrade the accuracy of translations. These problems can be avoided by introducing sinusoidal position vectors, as shown in the next paragraph.

To introduce word order into the training, a  $N \times 1$  *positional encoding* vector  $\mathbf{p}_i$  is added to the  $N \times 1$  embedding vectors  $\mathbf{v}_i$ :

$$\mathbf{v}'_i = \mathbf{v}_i + \mathbf{p}_i, \quad (13.15)$$

where  $\mathbf{p}_i$  is used to equip each word with information about its position in a sentence. There are many choices of positional encodings, both learned and fixed as described in Gehring et al. (2017). Vaswani et al. (2017) used sine and cosine functions to give the following  $N \times 1$  positional vector  $\mathbf{p}_i$ :

$$\mathbf{p}_i = \begin{bmatrix} \sin(i\omega_1) \\ \cos(i\omega_1) \\ \sin(i\omega_2) \\ \cos(i\omega_2) \\ \vdots \\ \vdots \\ \vdots \\ \sin(i\omega_N) \\ \cos(i\omega_N) \end{bmatrix}, \quad (13.16)$$

where  $i$  is the position of the word in the sentence and the angular frequency is

$$\omega_k = \frac{1}{10000^{2k/N}}. \quad (13.17)$$

A plot of  $\mathbf{p}_i$  for  $N = 128$  and a 50-word sentence from Kazemnejad (2019) is shown in Figure 13.8. Each row has a unique color pattern that represents the elements of a unique positional vector  $\mathbf{p}_i$  with a different word position  $i \in \{1, 2, \dots, 50\}$ . Each word position in the sentence is assigned a unique color pattern that resembles that of a binary number assigned to each word position.

As an illustrative example, assume a simple binary encoding scheme such that the first word is added to the positional binary vector  $\mathbf{p}_1 = (1, 0, 0, \dots)$ , the second word vector added to  $\mathbf{p}_2 = (0, 1, 0, 0, \dots)$ , the third word vector added to  $\mathbf{p}_3 = (1, 1, 0, 0, \dots)$ , the fourth word vector added to  $\mathbf{p}_4 = (0, 0, 1, 0, \dots)$  and so on. Similar to the color patterns in Figure 13.8, the corresponding binary color patterns of the binary positional vectors increase the presence of non-zero numbers, i.e. colors, to the right with increasing word position.

In summary, the positional encoding vector is added to the embedding vector to indicate the

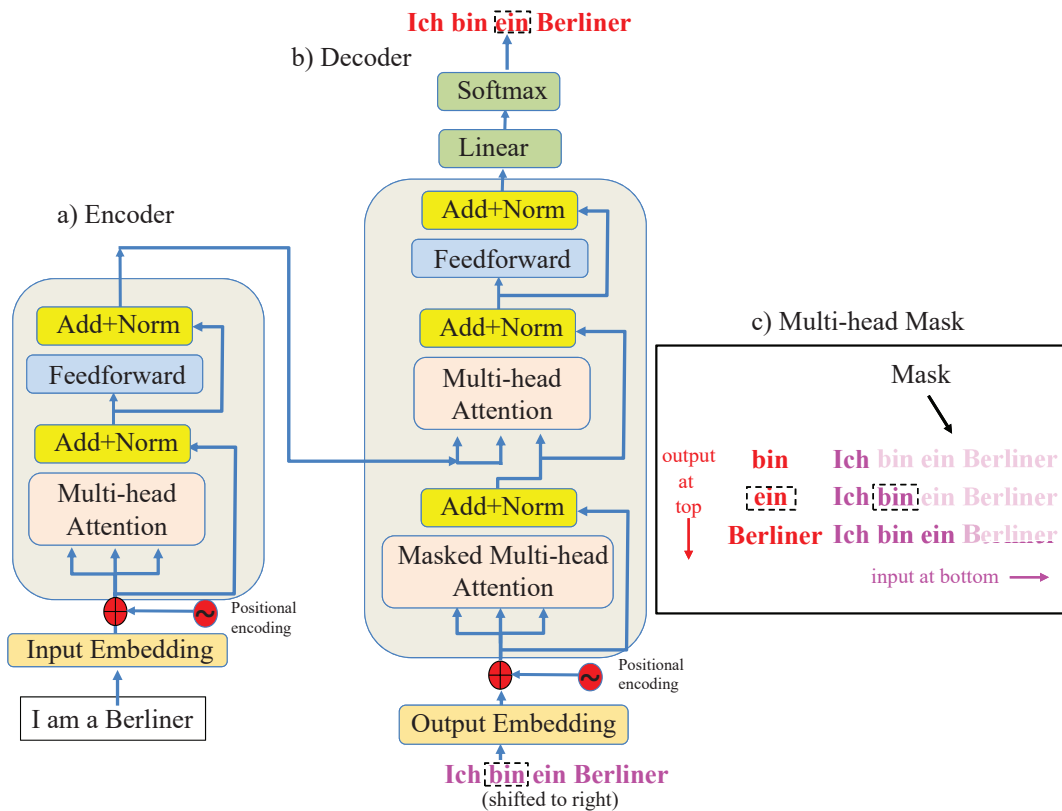


Figure 13.7: Transformer architecture with the a) encoder, b) decoder and c) mask blocks. The attention blocks from the encoder identify the related words in the English sentence *I am a Berliner*, and the lower-attention block in b) indicates the related words in the translated German sentence. These self-attention indicators are combined in the upper-attention block in b) to establish the relevant relationships between the words in the English and German sentences. The mask in c) ensures that the current output word in German only uses the previous input words in German. Figure adapted from Vaswani et al. (2017).

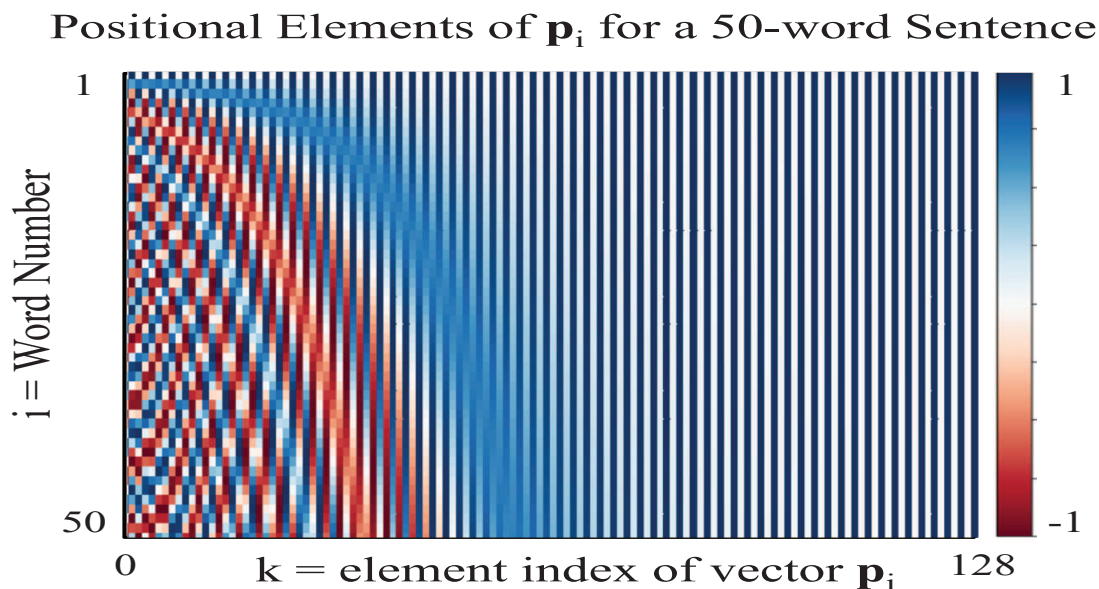


Figure 13.8: Positional encoding vectors  $\mathbf{p}_i$  plotted for  $N = 128$  and a 50-word sentence. Each row is a different  $1 \times 128$  positional vector  $\mathbf{p}_i$  whose element values are color coded according to the color bar and equation 13.17. Example adopted from Kazemnejad (2019).

position of each word.

- **Feed-forward Block.**

The softmax operations in the multi-head attention architecture of Figure 13.8 produce a sequence of  $h$  vectors, which are concatenated together into a long vector. This long vector is then multiplied by a dense matrix of learnable elements to form a sequence of  $W$  vectors, each of which can be of dimension  $N \times 1$ .

For the decoder in Figure 13.7, the number of output vectors in the topmost linear block can be the same number of usable words in the German language. In this case, the word with the highest softmax probability is selected to be the output word.

There are two linear transformations in each feed-forward network with a ReLU activation in between. While the linear transformations are the same across different positions, they use different parameters from layer to layer.

### 13.3 Transformer Architecture and Training

The multi-head attention architecture in Figure 13.7b largely comprises the encoder part the Transformer architecture (Vaswani et al., 2017). The decoder part takes the Value vectors  $\mathbf{v}_i$  from the encoder and decodes them into the output sentence of the translated language. Both the encoder and decoder use block architecture diagrams similar to that for the multi-head attention block diagram in Figure 13.6. A detailed description of the Transformer architecture is in <http://jalammr.github.io/illustrated-transformer/>. The following description of the encoder and decoder workflows is largely taken from Vaswani et al. (2017).

- **Encoder.** Each encoder consists of two parts: a self-attention mechanism and feed-forward neural network. Only one encoder layer is shown in Figure 13.7a, but any number of these

layers can be stacked on top of one another where the output of a layer is the input into the next layer. For the example in Vaswani et al. (2017), they assumed a stack of  $N_x = 6$  identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward neural network. A residual connection, aka a skip connection, is used around each of the two sub-layers, followed by layer normalization (Ba et al., 2016). That is, the output of each sub-layer is given by  $\text{LayerNorm}(\mathbf{x} + \text{Sublayer}(\mathbf{x}))$ , where  $\text{Sublayer}(\mathbf{x})$  is the function implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce output vectors with the same dimension.

- **Decoder.** The decoder in Figure 13.7b is also composed of a stack of  $N_x = 6$  identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, residual connections are employed around each of the sub-layers to avoid a vanishing gradient, which is then followed by layer normalization (Ba et al., 2016). The self-attention sub-layer is also modified in the decoder stack with a masking operation illustrated in Figure 13.7c. Masking, combined with the fact that the output embeddings are offset by one position, ensures that the word predictions for position  $i$  at the top of Figure 13.7b can depend only on the known inputs at positions less than  $i$  at the bottom of Figure 13.7b.
- **Training.** The examples in Vaswani et al. (2017) translated English to French and English to German sentences. Each training batch consisted of a set of sentence pairs containing approximately 25000 source words and 25000 target words. A dropout of about 10% was applied to the output of each sub-layer, before it is added to the sub-layer input and normalized. Dropout was also applied to the sums of the embeddings and the positional encodings in both the encoder and decoder stacks. For training, the English sentences were used as input into the encoder, and the German sentences were used as the input and output of the decoder. Each embedding vector had a dimension of  $512 \times 1$ . The German embedding words in Figure 13.7b are fed in one at a time, while the output words at the top are advanced by a one-word sample. The output of the softmax operation at the top of Figure 13.7b might be a, for example,  $10000 \times 1$  vector where each element corresponds to the probability of the output belonging to one of the 10 000 most-frequently-used German words. The element with the highest probability is the one selected as the output word during the testing phase.

## 13.4 Advances in Transformer Models

Within a few years of its introduction, Transformer models such as BERT and GPT-3 (Goled, 2021) have become the architecture of choice for many NLP problems (Wolf et al., 2020), replacing RNN models such as LSTMs. It has found great success in other fields such as computer vision (Carion et al., 2020) and audio processing (Gulati et al., 2020; Lin et al., 2021) and promises to expand in other areas of engineering and science such as application programming interfaces (Yang et al., 2022), chemistry (Schwaller et al., 2019) and life sciences (Rives et al., 2021).

For example, application programming interfaces (API) are software modules that can be linked together and used to solve many types of engineering and science problems. The MATLAB software toolbox and the Java Software Development Kit come with hundreds of modules that can be reused over and over again to solve different problems. The result of using APIs is a great reduction in development time and effort. Pretrained Transformer modules can also be used in an API environment where programmers frequently search for reviews of APIs using mainstream software question and answer (Q and A) platforms. Traditionally, the software for Q and A platforms is based on traditional ML algorithms. However, Yang et al. (2022) found that six pretrained Transformer modules for *aspect-based API review classification* outperformed the standard one based on a traditional ML algorithm.

A survey of dozens of Transformer models is in Lin et al. (2021). They introduce the vanilla Transformer, and then classify a wide variety of Transformer-like models. Variants such as the Transformer XL model effectively allows for the processing of long-range dependencies. This is important when related words in a long sequence are significantly separated from one another.

### 13.4.1 Transformer Models and Event Picking in Seismograms

Geophysical applications of Transformer models are starting to emerge, but one promising application is the detection of P-wave and S-wave arrivals in earthquake records (Mousavi et al., 2020; Hu et al., 2021; Stepanov et al., 2021). In contrast, Ba et al. (2022) avoided the Transformer model and only used a fully connected and LSTM network with self-attention modules to successfully pick arrivals in earthquake records. The self-attention mechanism is also used with CNN U-Net models for successfully picking earthquake records (Wei et al., 2021). All of the above papers compared their picking performance to that of standard phase pickers and claimed the superiority of their network models. An essential component for success with these different network architectures is the self-attention mechanism.

As a Transformer example, Mousavi et al. (2020) applied a complicated network model to 5 weeks of continuous earthquake data recorded in Japan. The network consisted of one very-deep encoder and three separate decoders composed of 1D convolutions, bi-directional and unidirectional LSTMs, residual connections, feed-forward layers, a Transformer, and self-attention layers. They were able to detect and locate more than double the number of earthquakes using data recorded by less than 1/3 of the seismic stations. The network picks of P and S phases were as accurate as those picked by human analysts.

Instead of training a Transform model directly from time-domain traces, Stepanov et al. (2021) trained their model on the spectrograms computed from the time records. They used a transformer architecture with a self-attention mechanism and without any convolution blocks, and it only required the learning of 57 000 parameters. Their pre-trained Transformer provided the best classification and computation performance compared to other pickers when tested on local earthquake data. The model code is available online (see Exercise 13.6.1) and can perform real-time arrival picking of seismic traces.

The simple architecture of a Transformer's encoder was used by Hu et al. (2021) for picking P-arrivals in hundreds of aftershock records. They transformed each earthquake seismogram into a pseudo-word format by assigning each word to contiguous windows of time samples in a trace. These windows have a length of  $T$  time samples in the left part of Figure 13.9a. Here, the colored blocks depict the E-W (green), N-S (yellow), and vertical (red) waveform components in a hypothetical three-component recording, where the time axis is along the vertical. Transposing these columns of seismic amplitudes gives the row vectors in the middle of Figure 13.9a, where the time axis is along the horizontal.

Taking the transpose of columns in the leftmost diagram gives the block diagram in the middle image of Figure 13.9a. Every  $T = 2$  columns of blocks in the middle image can be converted into a tall  $3T \times 1$  vector of blocks on the right side of a). This reshaped set of blocks now resembles the word vectors associated with a sentence, where there are  $k$  words in the sentence and each word has  $3T$  letters. This reformatting of the 3-component seismograms is now in the format for input into a Transformer. The goal is to train a Transformer model that can detect P arrivals in earthquake seismograms. In this case, the Transformer classifies each of the  $k$  windows in a three-component seismogram as containing a P arrival or not. Once this model is trained, then it can be used to identify P arrivals in new seismic data.

This block of reshaped data in Figure 13.9a is fed into the fully connected layer in Figure 13.9b to give the pseudo-word embedding vectors  $\mathbf{v}_i$  for  $i \in \{1, 2, \dots, k\}$ . Each embedding vector has dimension  $3T \times 1$  and there are  $k$  such vectors in a pseudo-sentence. The  $3T \times 1$  positional coding vectors  $\mathbf{p}_i$  for  $i \in \{1, 2, \dots, k\}$  are created on the right side of Figure 13.9b, and are added to  $\mathbf{v}_i$ .



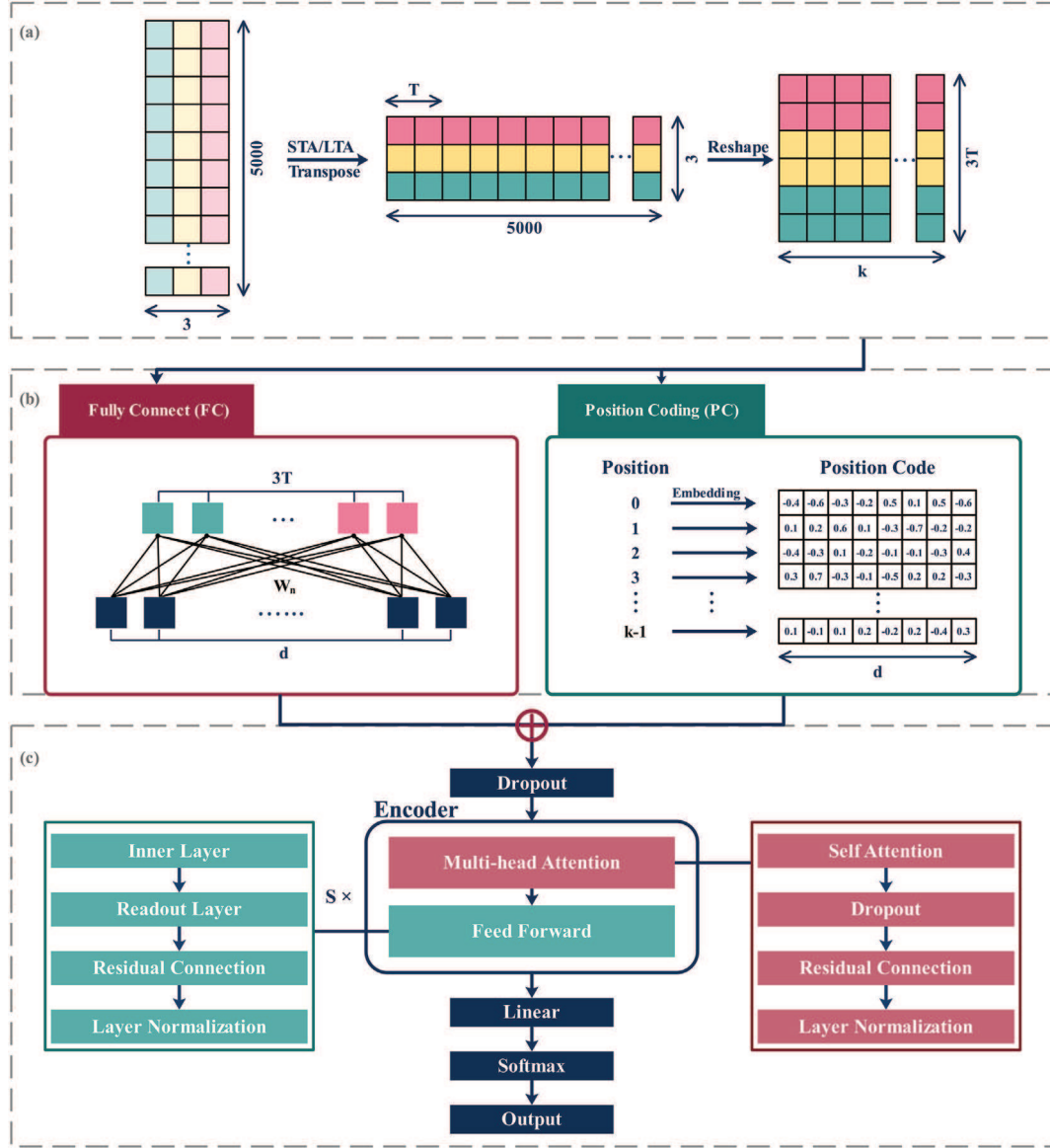


Figure 13.9: a) Digitized three-component traces on the left reshaped into pseudo-word format on the right and b) the fully-connected network reshaping a  $3T \times 1$  pseudo-word into a  $d \times 1$  embedding vector and adding a  $d \times 1$  position vector. The encoding architecture is shown in c). Hu et al. (2021) employed the STA/LTA algorithm to create input traces from the three-component data. Figure from Hu et al. (2021).

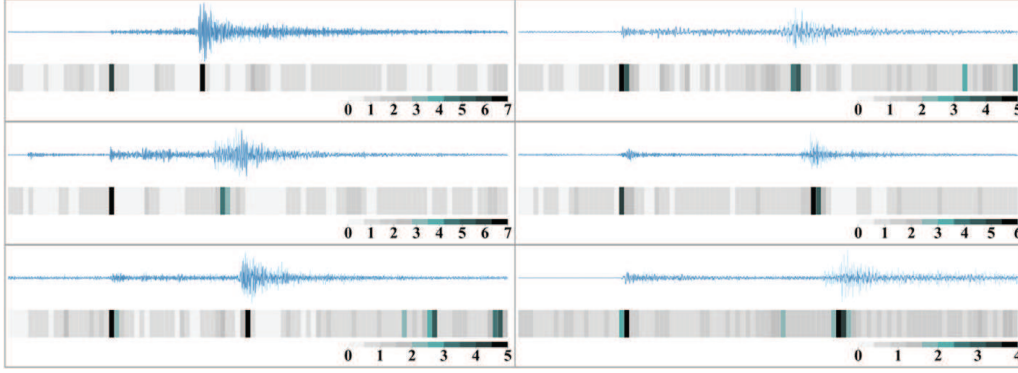


Figure 13.10: Seismograms and attention scores averaged over 8 heads in the multi-attention architecture. The strongest attention is mainly focussed at the P- and S-arrival times as indicated by the black bands. Figure from Hu et al. (2021).

The results  $\mathbf{v}_i + \mathbf{p}_i$  for  $i \in \{1, 2, \dots, k\}$  are fed into the encoder in Figure 13.9c. A 10% dropout operation is used in the fully-connected layer in b) and the self-attention module in c) to regularize the algorithm.

Hu et al. (2021) only use the encoder part of a Transformer architecture for picking P arrivals. The architecture of the encoder in Figure 13.9c probabilistically classifies the input pseudo-words as to whether each window contains a seismic P-wave event or not. A gradient descent method is applied to a regularized cross entropy loss function for training on a labeled data set. Once the encoder model is trained on labeled data, the encoder is used to pick P-wave arrivals in new data.

The above picking scheme, denoted as TransQuake, is tested on aftershock data recorded by 14 stations after a 7.9  $M_W$  earthquake in China. The data consist of 109 719 noise traces and 9891 traces with P arrivals. 5/6 of the data set was used for training, 1/6 for validation (1/12) and 1/6 for testing (1/12). There are 8 attention heads, the window length for some tests is  $T = 50$  samples (i.e. a 0.5 s window), and each 3-component record is reshaped from  $3 \times 5000$  into the shape  $k \times 3T$ , where  $k = (3 \times 5000)/3T$  is the number of window positions. That is, each pseudo-word corresponds to a 0.5 s time window of amplitudes in the trace. Attention scores, similar to  $\frac{1}{k} \sum_{j=1}^k w_{ij}$ , are averaged over the 8 heads in the multi-head attention architecture depicted in Figure 13.9. Figure 13.10 depicts 6 different seismograms and the corresponding attention plots averaged over eight attention heads. These plots indicate that the most attention (black bands) is at the P and S arrivals in the seismograms. This can be a useful diagnostic tool for identifying other phases as well.

The accuracy in picking P arrivals by TransQuake is compared to that of eight other picking methods listed in Figure 13.11. Here, the acronyms are defined as

$$\begin{aligned}
 Accuracy &= \frac{TP + TN}{TP + TN + FN + FP}, \\
 Precision &= \frac{TP}{TP + FP}, \\
 Recall &= \frac{TP}{TP + FN}, \\
 F1 &= \frac{1}{1/Precision + 1/Recall},
 \end{aligned} \tag{13.18}$$

where *Accuracy* indicates the percent of correctly-predicted samples, *Precision* the percent of true

Method	Accuracy	Precision	Recall	F1
Logistic Regression	0.499	0.073	0.504	0.127
Support Vector Machine	0.697	0.079	0.300	0.126
RNN (LSTM)	0.857	0.163	0.234	0.192
Random Forest	0.879	0.259	0.352	0.298
CNN (CPIC)	0.932	0.521	0.739	0.611
CNN (ConvNetquake)	0.946	0.649	0.581	0.613
MSDNN	0.952	0.678	0.638	0.658
MSDNN + Multi-task Learning	0.954	0.683	0.667	0.675
<b>TransQuake</b>	<b>0.956</b>	<b>0.712</b>	<b>0.667</b>	<b>0.689</b>

Figure 13.11: Comparison of accuracy scores for 9 picking methods for identifying P arrivals in the aftershock data set. Table from Hu et al. (2021).

samples we predict compared to the total number of true-positive and false-positive predictions, and *Recall* detects the percent of true-positive samples compared to the number of true and false-negative predicted samples. The metric *F1* indicates a balance between Precision and Recall, where  $F1 = 1$  indicates a perfect prediction capability. The Figure 13.11 results suggest that TransQuake typically provides a better performance than the other methods. Here, the MSDNN picker combines the advantage of both CNN and LSTM.

There is not a significant difference between some of the scores for MSDNN, CNN and TransQuake, and so further testing on other data sets is necessary before drawing general conclusions. There are efforts at hyperparameter optimization for TransQuake, but extensive optimization is not reported for the other methods.

## 13.5 Summary

LSTMs and GRUs typically require large computational resources to train from huge sequential-data sets, are not easy to parallelize and have a word recall capability limited to only a few hundred words. To mitigate these problems for translation systems, Vaswani et al. (2017) introduced the Transformer architecture which allows for much faster and more accurate processing of NLP tasks. Unlike an RNN, the entire sequence of a sentence is simultaneously input into the Transformer and the data in each head in a multi-head attention block can be independently computed by a separate node in a parallel computer. In addition, the attention mechanism highlights important word pairs so that the computations can analyze words separated by thousands of other words without a vanishing gradient.

The architecture of the vanilla Transformer consists of the encoder and decoder blocks shown in Figure 13.7. Each of these blocks consist of a stack of  $N_x$  identical blocks. The encoder block is mainly a sequence of multi-head self-attention, position-wise, and feed-forward network (FFN) operations. To build a deeper model that mitigates the vanishing gradient problem, residual connections are incorporated into each module, followed by a layer normalization module. Decoder blocks, unlike the self-attention blocks in the encoders, incorporate cross-attention modules between the multi-head self-attention modules and the position-wise FFNs. Furthermore, the self-attention modules in the decoder can use masking to prevent each position from using subsequent positions. There are dozens of Transformer models, where many of them are listed in Figure 3 in Lin et al.

(2021).

The recent invention of the Transformer mechanism has naturally limited its widespread applications to geoscience. However, there are a number of successful tests of its utility with the detection of earthquake events and arrival-time picking. At least three groups report its superior performance compared to standard picking methods. Several other groups also report superior picking performance when the self-attention mechanism is combined with a neural network architecture, which can include LSTM. The Fuzzy clustering method in Chapter 17 also claims superior performance when compared to standard picking methods.

So, which picking method is the best? We can answer this question by following the example of the computer science community: create a public data base with seismic simulations and field recordings of earthquakes, vibroseis, microseisms and controlled sources. These data should also be accompanied by the groundtruth traveltimes of P-arrivals, and S-arrivals if relevant. There should also be a data base of noise traces as well that can be added to the clean data for testing. Such noise can be passive recordings of, for example, wind noise, urban noise such as traffic, and glitches. The noise can be added by the user at various SNR levels. Thus, the geoscience community will have a standard by which to judge the efficacy of different picking methods.

## 13.6 Exercises

1. Test the sensitivity of hyperparameter values for the Stepanov et al. (2021) algorithm in picking phases in earthquake records. The Stepanov et al. (2021) data, examples, and pre-trained model are online at GitHub: <https://github.com/jamm1985/seismo-performer> (accessed on 18 September 2021). Datasets for Sakhalin and Dagestan are available online in HD5 format at Google Drive (the links are also present on the GitHub page): [https://drive.google.com/file/d/1dH2\\_JF9TQmyB6GpIB\\_dY1jiWAI5uqp6ED](https://drive.google.com/file/d/1dH2_JF9TQmyB6GpIB_dY1jiWAI5uqp6ED) (accessed on 18 September 2021), <https://drive.google.com/file/d/156w3I9QVnhkCo0u7wjh-c6xekE9f6B3G> (accessed on 18 September 2021).
2. Compare the traveltimes picking method by Fuzzy Clustering in section 17.5.1 with that from Stepanov et al. (2021). Use synthetic data with various noise levels as the input data. List the relative advantages and liabilities of each method.

**Part IV**

**Unsupervised Learning**



## Chapter 14

# Autoencoders

Convolutional neural networks are often used in the context of supervised learning where the input data must be classified before it is used for training. To avoid this expense autoencoders are used where the goal is to determine the weights that produce an output that is identical to the input, as shown in Figure 14.1. In this case no classification of inputs is required so autoencoders are often classified as an unsupervised learning method. However, some refer to an autoencoder as a semi-supervised learning method because a target output is required (Dai and Le, 2015).

The autoencoder architecture in Figure 14.1 is symmetrical about the middle layer, and the output image is deemed to be the same as the input image. The left-half architecture of the autoencoder is known as an encoder because it encodes the high-dimension input image into a low-dimensional code, known as the latent variables  $\mathbf{z}$ . Thus, the autoencoder is said to be undercomplete because of the thin information bottleneck in the middle layer. The latent variables serve as a means for data reduction without losing valuable information about the input, and is one of the most practical uses of autoencoders for data reduction.

The right-half architecture is known as the decoder because it decodes the latent variables into an approximation of the input image. The decoder can be thought of as a generative modeler in the sense that small perturbations of the latent variables associated with a single input image will lead to an output image that was never seen before. However, this output image might be unrealistic.

There are a number of practical uses of autoencoders.

1. **Data compression.** A practical use of the autoencoder is to reduce the size of the original input data to that of the middle bottleneck layer. In this case the latent-space information only needs to be transmitted or stored, which can reduce transmission and storage costs. Of course, the weights associated with the decoding layers must be eventually used in order to decode the latent space variables.
2. **High-frequency random noise reduction.** Another practical use of the autoencoder is to filter unwanted high-frequency random noise, which is automatically achieved by using a low-dimensional latent space (Vincent et al., 2008) that does not overfit the data. Therefore it tends to ignore high-frequency random noise and only learn weights that capture the lower-frequency information associated with the coherent signals. An example is demonstrated with least squares migration in Chapter 24 and seismic noise filtering in Chapter 15. Sometimes white noise is added to the data prior to training and the autoencoder is trained to produce an uncontaminated image. This forces the network to ignore random noise that might be in the images. It also is a form of regularization so that the autoencoder does not overfit the data. Results from adding high-frequency noise to MNIST images and the output of the trained autoencoder is shown in Figure 14.2.
3. **Robustness to partial input data destruction.** The input data can be corrupted by

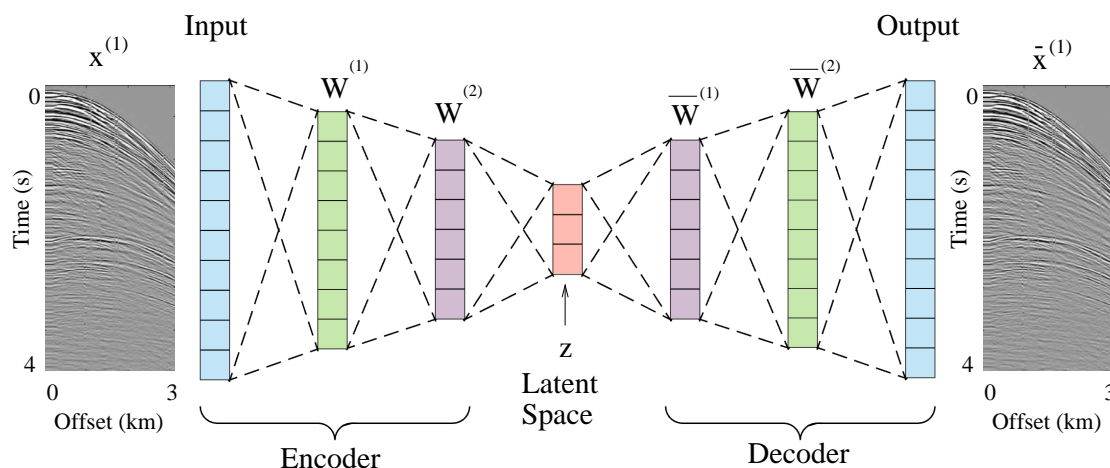


Figure 14.1: Autoencoder architecture with a three-dimensional latent space. Here the input image is a seismic shot gather and the desired output is the same as the input image. Many different shot gathers are used to train the AE, whose weights can be those for a fully connected NN or a CNN.

removing large portions of the data, and forcing the autoencoder to infill the missing portions to produce uncorrupted data. An example is that of missing seismic data in a common shot gather. Well sampled traces data can be deliberately corrupted by removing, for example, the near-offset traces. The output will be the uncorrupted shot gather. This is called inpainting (Xie et al., 2012; Pathak et al., 2016) by the computer vision community and more generally is referred to as interpolation of data. An example is shown in Figure 14.3 where the output of an autoencoder is iteratively improved by inserting it again into another trained autoencoder.

4. **Pretraining for supervised learning algorithms.** Recurrent neural networks (RNN) are often used to predict what comes next in a sentence, which is one of the goals of natural language processing. Dai and Le (2015) used an autoencoder to read the input sequence of words and then predict the input sequence again. The learned weights of the autoencoder are then used in supervised recurrent network models to accelerate convergence. In their experiments they found that the pretrained recurrent neural networks (RNN) showed improved stability and generalization properties than a standard RNN. They labeled this type of pretrained RNN as *semi-supervised sequence learning*.
5. **Coloring grayscale pictures.** Grayscale images can be colored by training an autoencoder with a large set of colored images deliberately corrupted to be grayscale, and the output is the original colored images. Once the network is sufficiently trained then it can be used to colorize grayscale images. The example in Figure 14.4 is generated by the Keras code in <https://www.kaggle.com/valkling/image-colorization-using-autoencoders-and-resnet>. The autoencoder (AE) can also be used to transform colored images into grayscale images, as illustrated in Figure 14.5.

The autoencoder characterized by the loss function in equation 14.1 is also known as a deterministic autoencoder (Ghosh et al., 2019), which does not enforce an a prior distribution on the latent space. In comparison, the variational autoencoder (VAE) in section 14.3.5 is a regularized autoencoder that learns the statistical parameters associated with the data (Kingma and Welling,



## Denoising Autoencoder

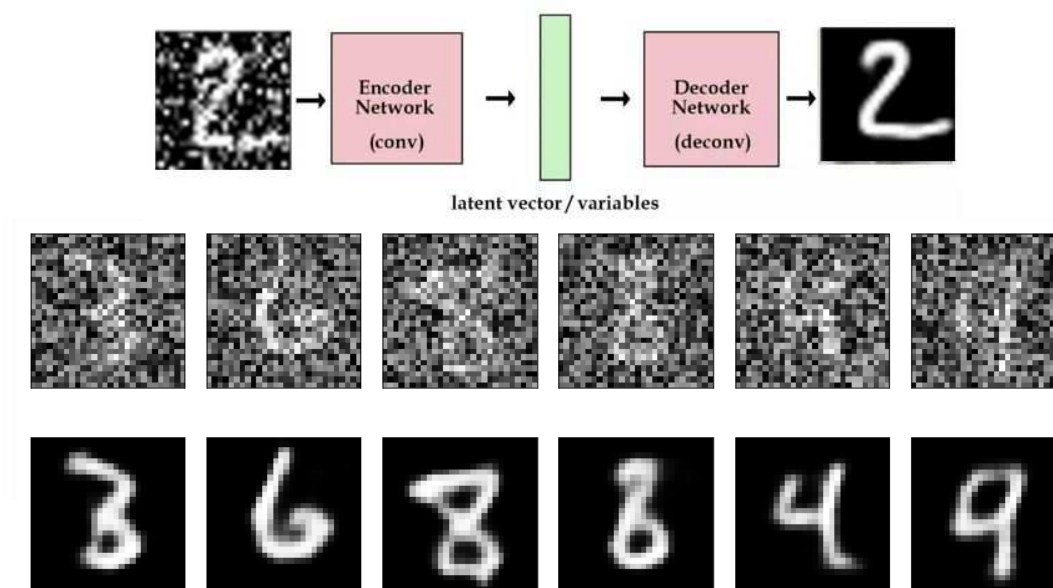


Figure 14.2: (Top row) Autoencoder with noisy MNIST images, (middle row) some noisy input images, and (bottom row) output images from the trained autoencoder (images from <https://www.andreaamico.eu/unsup-learning/2019/06/24/autoencoder.html>).

2014; Doersch, 2016). They encode inputs as distributions instead of points and whose latent space is regularized by constraining distributions returned by the encoder to be close to a standard Gaussian.

The advantage of the VAE is that the latent space vectors can be interpolated to give new output images that are more realistic than interpolated images from autoencoders. This can be useful for training CNN architectures without the need for a large set of classified data. In comparison, a deterministic autoencoder more often produces unrealistic output images when the latent-space vectors are interpolated (Foster, 2019).

As alternatives to AE- or VAE-based interpolation, regularized autoencoders and adversarial autoencoders have been proposed as effective interpolators of latent spaces generated by autoencoders. Ghosh et al. (2019) proposed that a deterministic autoencoder with explicit and implicit regularization applied to the latent space vector can produce output images comparable to or better than VAEs. This claim is supported by some empirical examples, but there is no mathematical proof that it generalizes better than a VAE. In contrast, Berthelot et al. (2018) proposed a combination of a critic network and a regularization procedure which encourages interpolated outputs to appear more realistic by fooling a critic network. Empirical tests showed that the adversarial autoencoder often produced more realistic output images than a VAE or autoencoder for interpolated latent space vectors. Later, Oring et al. (2020) proposed regularization of the latent-space variables for improving the accuracy of interpolating data with an AE.

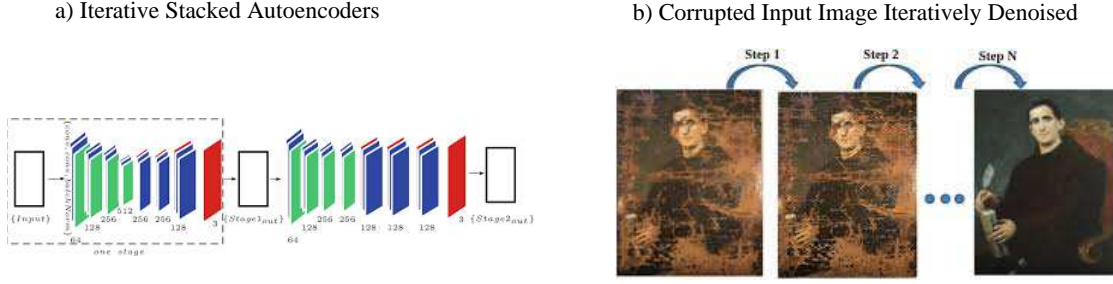


Figure 14.3: a) Series of stacked autoencoders where there are multiple layers per decoder and encoder, and b) restored photo corrupted by time. Images from Quan and Kim (2019).

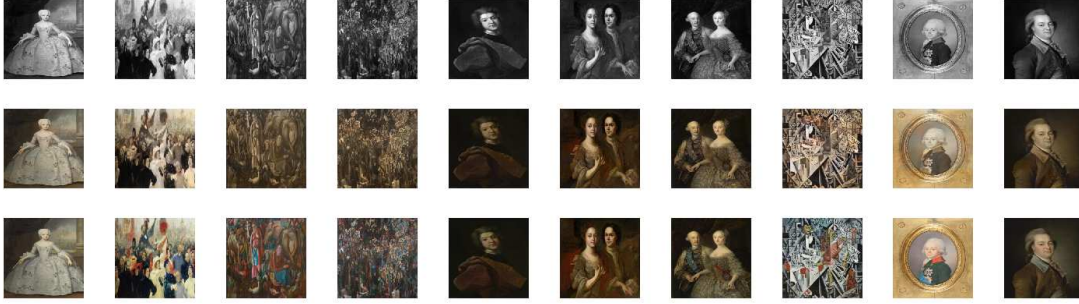


Figure 14.4: (Top row) Grayscale images and (middle and bottom rows) color images colorized by an autoencoder network (<https://www.kaggle.com/valking/image-colorization-using-autoencoders-and-resnet>).

## 14.1 Theory of Autoencoders

An autoencoder is a semi-supervised neural network in which the predicted output is the same as the input data, as illustrated in Figure 14.1. An autoencoder is trained to learn the extremely low-dimensional representation of the input data, also denoted as the skeletonized representation, in an unsupervised manner. The parameters of the autoencoder neural network are determined by finding the values of  $\mathbf{W}^{[n]}$  and  $\mathbf{b}^{[n]}$  for the special two-layer case  $n = 1, 2$  that minimize the following objective function:

$$\begin{aligned} \epsilon(\mathbf{W}^{[1]}, \mathbf{b}^{[1]}, \mathbf{W}^{[2]}, \mathbf{b}^{[2]}) &= \sum_{m=1}^M (\tilde{\mathbf{x}}^{(m)} - \mathbf{x}^{(m)})^2, \\ &= \sum_{m=1}^M (g(\mathbf{W}^{[2]} \overbrace{[f(\mathbf{W}^{[1]} \mathbf{x}^{(m)} + \mathbf{b}^{[1]})]}^{\mathbf{z}=\text{latent-variable}} + \mathbf{b}^{[2]}) - \mathbf{x}^{(m)})^2, \end{aligned} \quad (14.1)$$

where  $M$  is the number of input examples,  $\tilde{\mathbf{x}}^{(m)}$  ( $\mathbf{x}^{(m)}$ ) denotes the  $m^{\text{th}}$  predicted (given) data vector, and  $\mathbf{W}^{[n]}$  and  $\mathbf{b}^{[n]}$  are the weights and bias terms in the  $n = 1, 2$  layers. Here,  $f(\cdot)$  and  $g(\cdot)$  are non-linear thresholding functions such as a ReLU. The architecture for the autoencoder can be a convolutional neural network as long as there is a series of layers that form an encoder, a lower-dimensional latent space in the middle, and a decoder network that is often a mirror image of the encoder.

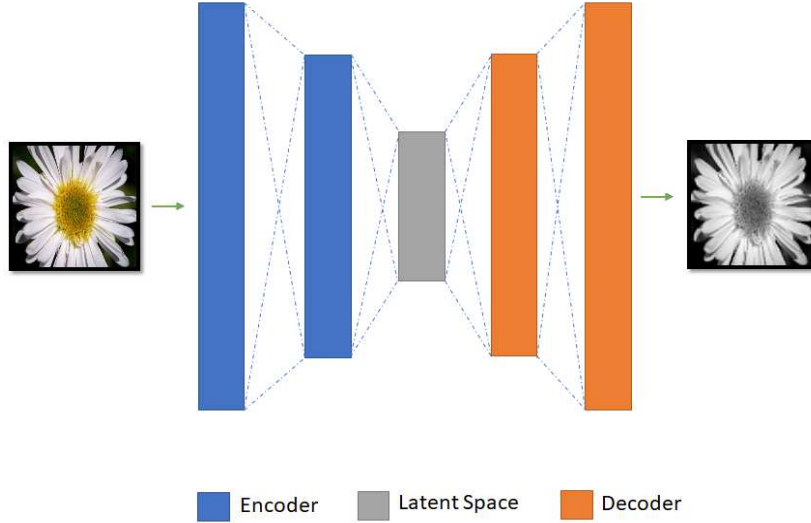


Figure 14.5: Autoencoder network for transforming colored images to grayscale images (<https://towardsdatascience.com/autoencoder-for-converting-an-rbg-image-to-a-gray-scale-image-3c19a11031c9>).

The typical autoencoder architecture includes three parts: the encoder, the low-dimensional latent space, and the decoder.

- **Encoder:** Semisupervised learning by an autoencoder uses a set of training data consisting of  $M$  training samples  $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(M)}\}$ , where  $\mathbf{x}^{(m)}$  is the  $m^{th}$  input vector with dimension  $D \times 1$ . For the seismic trace example,  $D$  is the number of time samples along the time axis<sup>1</sup>. The encoder neural network in Figure 14.1 encodes the high dimension input data  $\mathbf{x}_i$  into a low-dimension latent space with dimension  $C \times 1$  using a series of neural layers with a decreasing number of neurons; here  $C$  is smaller than  $D$ . For two hidden layers  $n = 1, 2$ , the encoding operation can be mathematically described as  $\mathbf{z}^{[n]} = f(\mathbf{W}^{[n]}\mathbf{z}^{[n-1]} + \mathbf{b}^{[n]})$ , where  $\mathbf{z}^{[n-1]}$  is the input vector for the  $n^{th}$  layer and the superscript  $(i)$  for the data sample index in  $\mathbf{z}^{[n-1]}$  is silent. Also,  $\mathbf{b}^{[n]}$  is the bias term for that layer,  $\mathbf{W}^{[n]}$  is the weight matrix for the  $n^{th}$  layer, and  $f(\cdot)$  is the activation function for that layer. Here, the layers for the encoder are  $n = [1, 2]$ , and the activation functions can be, for example, sigmoid, ReLU, or Tanh.
- **Latent Space:** The compressed data  $\mathbf{z}^{[n]}$  with dimension  $C \times 1$  in the latent space layer (see middle layer in Figure 14.1) is the lowest dimension  $C = 3$  space in which the input data are reduced and the key information is preserved. The latent space is an effective low-dimensional representation of the high-dimensional input data. These low-dimensional attributes can be used by the decoder to reconstruct the original input. This is similar to principal component analysis (PCA), which is generally used to represent input data with a smaller-dimensional space (Hotelling, 1933). However, PCA is a linear operation that finds the optimal rotation of the original data axes that decorrelates the data. In comparison, the autoencoder with a sufficient number of layers can find a non-linear sparse mapping between the latent-space

<sup>1</sup>A seismic shot gather is usually stored as an  $nt \times ng$  matrix. If you want to input it into a 1-D autoencoder, then you need to unroll it to be a vector with length  $nt \cdot ng$ . In this case  $D$  equals  $nt \cdot ng$ .

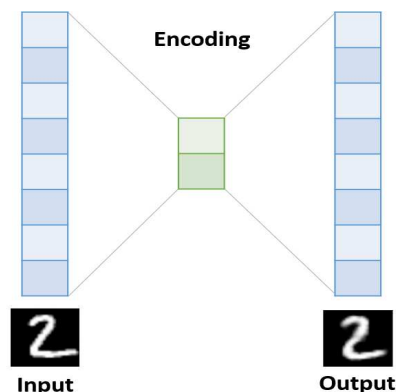
variables and the output image. In fact, for the simple case of a 1-layer encoder with no activation function and mean squared error loss, the *linear* autoencoder behaves like PCA, learning to project the input in the span of the first  $K$  principle components of the data (Bengio, 2009; Jacques et al., 2017).

- **Decoder:** The decoder portion of the neural network (see Figure 14.1) is represented by a series of neural network layers with an increasing number of units per layer. The reconstructed data  $g(\mathbf{W}^{[n]}\mathbf{z}^{[n-1]} + \mathbf{b}^{[n]})$ , where the index for the decoder layer is  $n = 2$  for this example. If the matrix  $\mathbf{W}$  for each layer in the decoder is constrained to be the transpose of the matrix for the encoder then this is said to have tied weights (Vincent et al., 2008).

In the above description of the autoencoder, the encoder and decoder are deterministic functions that map  $\mathbf{x}$  to  $f(\mathbf{x}) = \mathbf{z}$  and  $\mathbf{z}$  to  $g(\mathbf{z}) = \tilde{\mathbf{x}}$ . Therefore, the only stochastic component is the probability distribution that governs the input data. Since the mapping between the latent space vectors is highly non-linear, then a linear interpolation of the two latent space vectors does not necessarily lead to a realistic output image. In other words, the mapping between latent space vectors and images is not necessarily continuous. For example, if the input data consist of pictures of different dogs and cats, the weighted sum of a latent-space vector belonging to a German shepherd and one for a Siamese cat will likely lead to a strange animal that is neither cat or dog. Even the interpolation of the latent vectors for a german shepherd and a bulldog will not necessarily produce a picture of a realistic hybrid dog.

A more concrete example of this non-linear mapping is illustrated in Figures 14.6. Here, the

a) Autoencoder Diagram for MNIST Data



b) Latent Space Points for MNIST Data

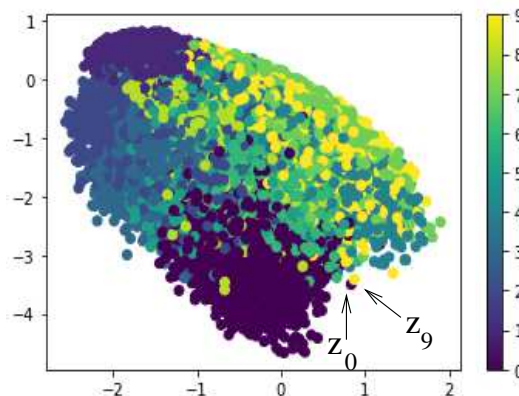


Figure 14.6: a) Input and output pictures of the digit "2" from the autoencoder trained on the MNIST data set. b) Latent-space points for each of the input examples from the MNIST data set. The variables for the horizontal and vertical axes are  $z_1$  and  $z_2$  where  $\mathbf{z} = (z_1, z_2)$ . The color bar indicates the digit number for each input sample. Images from Ali Masri's blog at <https://towardsdatascience.com/autoencoders-in-keras-273389677c20>.

Keras code in Appendix 14.7 is that of an autoencoder with the  $28 \times 28$  MNIST images as the input and their approximations are the output. An example of the input and reconstructed output of the digit "2" is shown in Figure 14.6.

Plotting the  $2 \times 1$  latent space vector  $\mathbf{z} = (z_1, z_2)$  for a batch of input MNIST samples gives the graph shown in Figure 14.6b. The different colors correspond to the different labels for digits associated with a latent-space point, where the colder colors indicate smaller values of the digits. It is noticed that two neighboring filled circles, for example the black  $z_0$  and yellow  $z_9$  circles near the

bottom right, correspond to the digits "0" and "9". The interpolated latent space variable

$$\bar{\mathbf{z}} = 0.5\mathbf{z}_0 + 0.5\mathbf{z}_9, \quad (14.2)$$

and its decoded number  $g(\bar{\mathbf{z}})$  will likely result in an uninterpretable symbol of a digit. In other words,  $g(\mathbf{z})$  is a nearly discontinuous function of  $\mathbf{z}$ , where a small perturbation  $\delta\mathbf{z}$  leads to an enormous change in  $g(\mathbf{z} + \delta\mathbf{z}) - g(\mathbf{z})$ . Therefore, an autoencoder might not be a reliable means for augmenting a labeled data set by decoding interpolated points in the latent space. This problem is addressed by regularizing the autoencoder (see section 14.3).

### 14.1.1 Linear Autoencoder and PCA

The linear autoencoder is related to PCA, as demonstrated in <https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73>. Figure 14.7 supports this claim by showing the PCA components and linear AE components for the input MNIST data set. The image on the left depicts the PCA components (color coded according to the digits 0 – 9) associated with the two strongest eigenvectors. For comparison, the output points from the linear autoencoder are on the right, where there is no non-linear activation function in the network. It can be seen that a rotation of one image by about 90 degrees and a reflection about the vertical axis brings the two images into rough agreement.

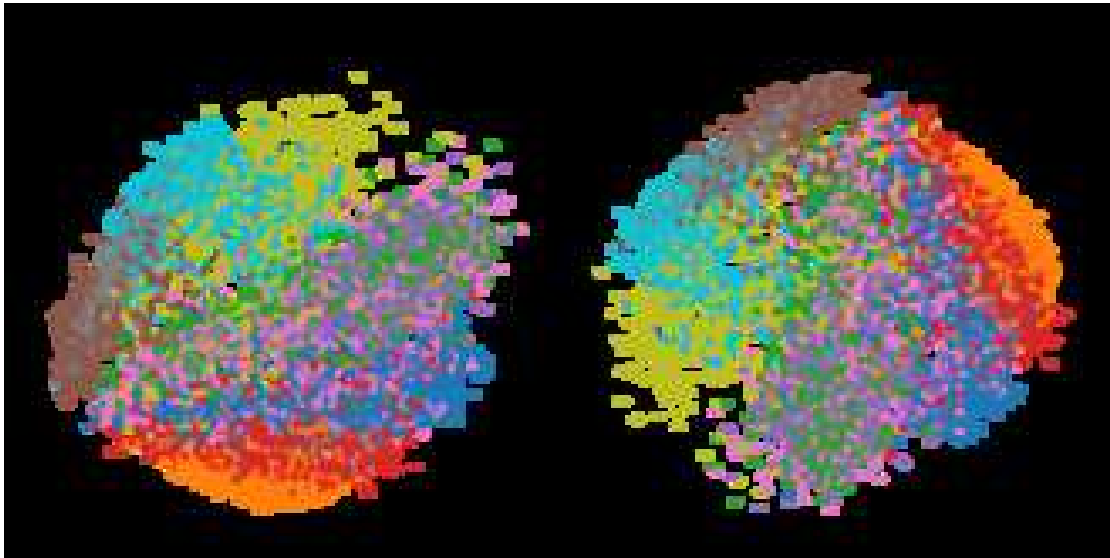


Figure 14.7: Plots of (left) PCA components and b) linear AE latent-space variables. Images from [https://en.wikipedia.org/wiki/Autoencoder#cite\\_note-:10-28](https://en.wikipedia.org/wiki/Autoencoder#cite_note-:10-28).

The meaning of each latent variable can be obtained by a trial-and-error procedure, where the image associated with each latent variable is computed by applying the decoder to that variable. However, Ladjal et al. (2019) adds a covariance regularization term to the autoencoder's objective function. This loss term encourages the latent variables to be statistically independent of one another, similar to that of a PCA. Iteratively increasing the number of latent variables with this objective function has the effect of ordering the latent-space variables in decreasing order of importance in representing that data. Thus, the latent-space variables can be interpreted in terms of physically meaningful properties in the data.

## 14.2 Problems with Autoencoders

Ideally, we would like the decoder  $g(\mathbf{z}, \mathbf{W}')$  to be a generative modeler so that small perturbations of the latent vector  $\mathbf{z}$  associated with a single input image will lead to a realistic output image that was never seen before. However, this output image is often unrealistic (Makhzani et al., 2016; Berthelot et al., 2018) because the AE can discontinuously map the input images to the latent-space domain, even if the input images are similar to one another. Smoothing this mapping to be more *continuous* has led to the development of regularized autoencoders (Hinton, 2013; Berthelot et al., 2018). Such regularized autoencoders can be used to generate realistic unseen images by regularized interpolation of the latent space variables, a generative procedure.

As an example, the middle row of Figure 14.8 shows an unrealistic output of interpolated points in the  $\mathbf{z}$  domain. The top row consists of rotated-line images, and the middle row consists of the output of an AE. Ideally, the goal is to interpolate two latent-space vectors and decode the interpolated result to give the *new* output image. In this case, the encoder transforms the vertical-line image  $\mathbf{x}_1 = \mathbf{x}_{left}$  and horizontal-line image  $\mathbf{x}_2 = \mathbf{x}_{right}$  in the top row of Figure 14.8 into the latent-space vectors  $f(\mathbf{x}_1) = \mathbf{z}_1$  and  $f(\mathbf{x}_2) = \mathbf{z}_2$ . To augment this data set by interpolation, the latent-space variables are interpolated by

$$\tilde{\mathbf{z}} = \alpha \mathbf{z}_1 + (1 - \alpha) \mathbf{z}_2, \quad 0 < \alpha < 1, \quad (14.3)$$

so that the decoded interpolated output is

$$\begin{aligned} \hat{\mathbf{x}} &= g\{\tilde{\mathbf{z}}\} = g\{\alpha \mathbf{z}_1 + (1 - \alpha) \mathbf{z}_2\}, \\ &= g\{\alpha f(\mathbf{x}_1) + (1 - \alpha) f(\mathbf{x}_2)\}. \end{aligned} \quad (14.4)$$

Hopefully, the interpolated image  $\hat{\mathbf{x}}$  will be a realistic-looking image not seen in the training set. Unfortunately this is not generally the case as illustrated by the interpolated output in the middle row of Figure 14.8. The interpolated AE images above the two-sided arrow are not realistic renderings of rotated lines.

Why does the autoencoder fail to produce believable interpolated images and so fail to be a reliable generator of new input images  $\mathbf{x}$ ? A key problem is that the autoencoder does not always produce a somewhat continuous mapping between the input points  $\mathbf{x}$  and output points  $\mathbf{z}$ . Equivalently, there is no constraint on the mean and variance of the  $\mathbf{z}$  points for different classes of input data  $\mathbf{x}$ . This failure was recognized early by Hinton (2013) and described as "fracturing the manifold" of  $\mathbf{z}$  into many separated domains for similar input images (Makhzani et al., 2016). An example of this fractured manifold is shown in Figure 14.9 where the input images are mapped to scalar values of  $z$  along the real line. In the latent space there is a large gap between the full-moon and half-moon representations in the latent space. Thus, decoding the value of, e.g.  $z = 0$ , in the fracture zone gives an image that will not look similar to any of the images in the training set.

A more familiar example is in Figure 14.10, where the  $2 \times 1$  latent space vectors  $\mathbf{z}$  are computed for different input images. Each colored dot is the result of inputting a written digit from the MNIST data set into the autoencoder. There is a large fracture or gap between the purple (the digit 7) and the yellow (the digit 1) points. Consequently, the decoded digit in the upper right is an unrealistic rendering of a written number. Also, note the intermingling of purple and yellow points. The associated numerical letters are likely to be quite different, which indicates a discontinuous mapping between nearby input points  $\mathbf{x}$  and their latent space points  $\mathbf{z}$ . In other words, the Fréchet derivative  $\partial \mathbf{x} / \partial \mathbf{z}$  of the encoder is unstable or discontinuous.

What promotes large fractures in the latent space? If there is a large  $l^2$  difference  $\|\mathbf{x}_{1/2 \text{ moon}} - \mathbf{x}_{full-moon}\|$  between half-moon  $\mathbf{x}_{1/2 \text{ moon}}$  and full-moon  $\mathbf{x}_{full-moon}$  images, then it is possible that there is a relatively large difference  $\|\mathbf{z}_{1/2 \text{ moon}} - \mathbf{z}_{full-moon}\|$  between their latent-space variables. The ReLU operation in the encoder will tend to preserve large  $l^2$  differences in the  $\mathbf{z}$ -domain if there

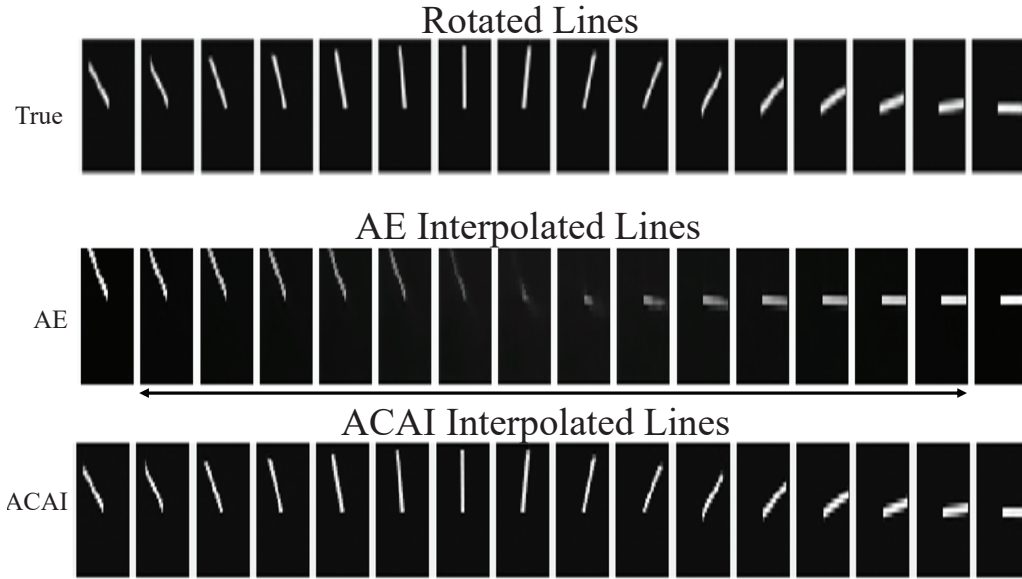


Figure 14.8: Top row corresponds to images of rotated lines, middle row of images between the two-sided arrow corresponds to the output of the autoencoder after interpolation of the latent-space variables  $\mathbf{z}_1$  and  $\mathbf{z}_2$  using equation 14.4. The bottom row of images between the two-sided arrow is computed using the adversarially constrained autoencoder interpolation (ACAI) network (Berthelot et al., 2018). Here,  $\mathbf{x}_1$  is the image on the far left and  $\mathbf{x}_2$  is the image on the far right. Image adapted from Berthelot et al. (2018). For this example, the autoencoder was properly trained with images of many training lines rotated at different angles.

are large distances between input images with different classes. This problem will be amplified if the  $\mathbf{W}$  matrices have large eigenvalues. Fracturing is less likely if the different classes consists of objects that are somewhat similar to a neighboring class, such as a more complete training set consisting of  $1/4 - \text{moon}$ ,  $1/3 - \text{moon}$ ,  $2/3 - \text{moon}$  and  $3/4 - \text{moon}$  pictures. Other partial remedies are available such as stochastic neighbor embedding, which is similar to putting springs on the  $\mathbf{z}$  variables with different classes, so they aren't too far apart or too close together (Hinton, 2013).

## 14.3 Regularization of Autoencoders

The previous section describes three problems with autoencoders: overfitting, fracturing of the latent space with large gaps between different classes of points, and intermingling of points in  $\mathbf{z}$ -space with different classes. We now show that different regularization methods can mitigate some of these problems to enable generalization of an AE.



## Autoencoder with Fractured Latent Space

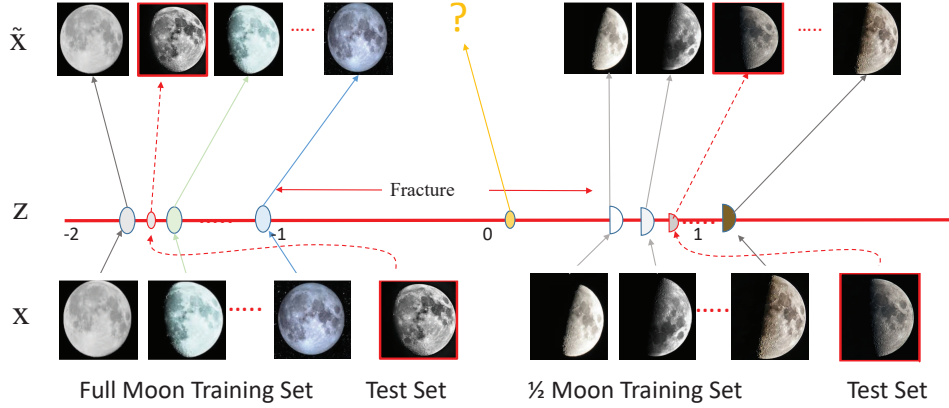


Figure 14.9: AE input  $\mathbf{x}$ , latent-space vector  $\mathbf{z}$  and output  $\tilde{\mathbf{x}}$  for the training set consisting of full- and half-moon images. The various full-moon images should be clustered together in  $z$ -space and somewhat separated from the half-moon  $z$ -values. If this separation is too wide then it is denoted as a fracture (Hinton, 2013) of the  $z$ -space into widely separated clusters, where the point, e.g.  $z = 0$ , in a fracture zone decodes into an unrealistic image  $\tilde{\mathbf{x}}$ . The  $l^2$  norm difference  $\min_{\text{samples}} \|\mathbf{x}_{\text{full-moon}} - \mathbf{x}_{1/2\text{-moon}}\|$  is large, so the width  $\min_{\text{samples}} \|\mathbf{z}_{\text{full-moon}} - \mathbf{z}_{1/2\text{-moon}}\|$  of the fracture zone can be large as well.

To mitigate the *fracture* problem we need a regularizer “that will make it costly to fracture the manifold” (see Figure 14.11b) in the latent space (Hinton, 2013). The regularizer can also be used to prevent collapse of the input images to the same latent variable (see Figure 14.11c), encourage the network to resist noisy perturbations to the input image as shown in Figure 14.11d, and prevent overfitting. These benefits are enabled by finding the weights that minimize the regularized objective function:

$$\epsilon(\mathbf{W}, \mathbf{W}') = \sum_{m=1}^M \left[ \overbrace{\|\tilde{\mathbf{x}}^{(m)} - \mathbf{x}^{(m)}\|^2}^{\text{data misfit}} + \lambda \overbrace{R(\mathbf{x}^{(m)}, \tilde{\mathbf{x}}^{(m)})}^{\text{regularizer}} \right], \quad (14.5)$$

where  $R(\mathbf{x}^{(m)}, \tilde{\mathbf{x}}^{(m)})$  is the regularization term and the summation is over the  $M$  training examples. The value of the damping term  $\lambda > 0$  determines which objective is more important, reducing the data misfit or the regularizing penalty term. Figure 14.12 illustrates how increasing the complexity of the neural network to the right of the critical point (red asterisk) leads to overfitting where the test-data accuracy starts to decrease. The tendency to overfit can be reduced by using a larger value of the regularization parameter  $\lambda > 0$ . The next subsections present some regularization methods for AEs.



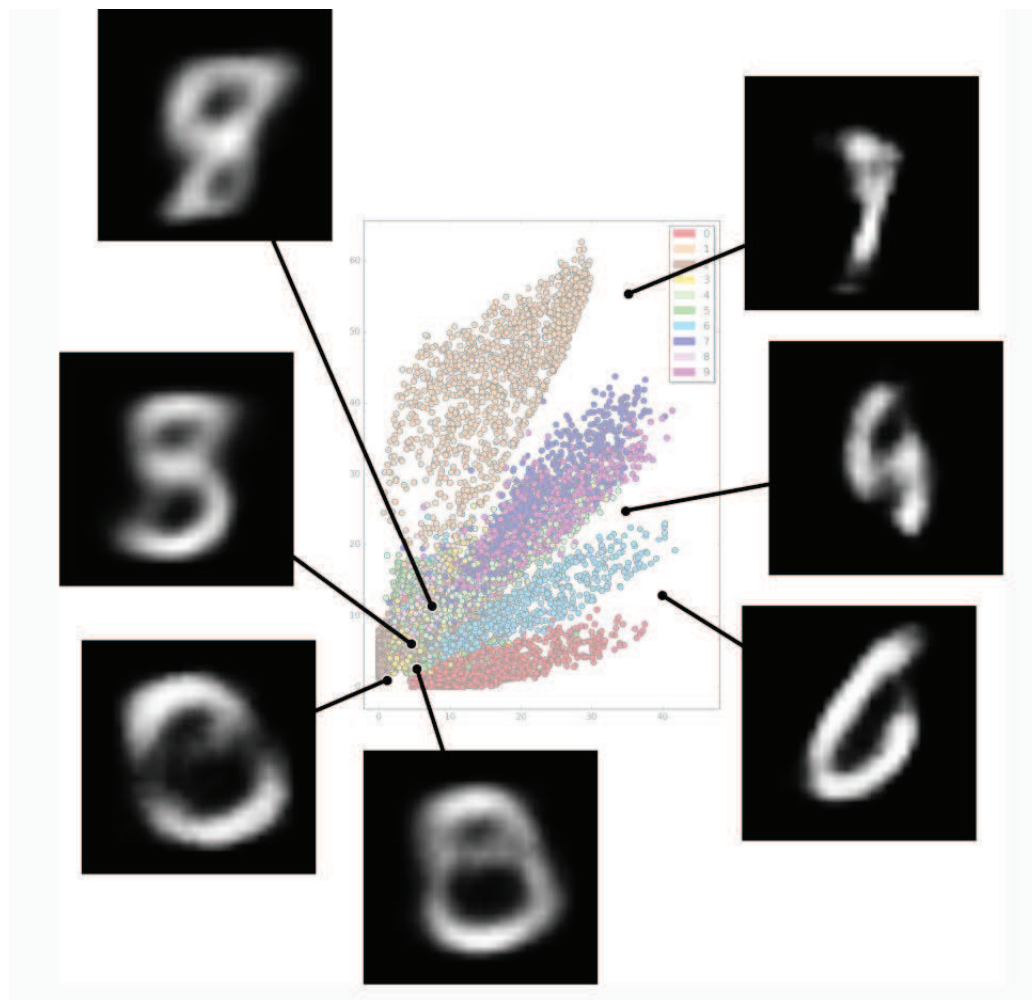


Figure 14.10: Two-dimensional latent-space points  $\mathbf{z}$  computed by an autoencoder, where the input consists of the MNIST data of written digits. Note the large gap, or fracture, between the gold  $\rightarrow$  *digit – four* and purple  $\rightarrow$  *digit – seven* points and the blurry rendering of what is supposed to be an interpolated digit. Illustration from Stewart (2019).

### 14.3.1 Dropout

Wager et al. (2013) and Srivastava et. al. (2014) showed that randomly eliminating nodes in a layer is equivalent to regularizing the objective function (see Appendix 14.8). Reducing the number of nodes is equivalent to reducing the complexity of the network and the tendency to overfit in Figure 14.12.

### 14.3.2 Denoising Autoencoders

The input images are noisy or corrupted versions of the clean output, as illustrated in Figure 14.11d. It can be shown that training with noisy images is equivalent to Tikhonov regularization (Bishop, 1995; Rifai et al., 2011), which prevents memorization of the input (Vincent et al., 2008) and overfitting by the network. The mapping of input images to a small region in  $\mathbf{z}$ -space is illus-

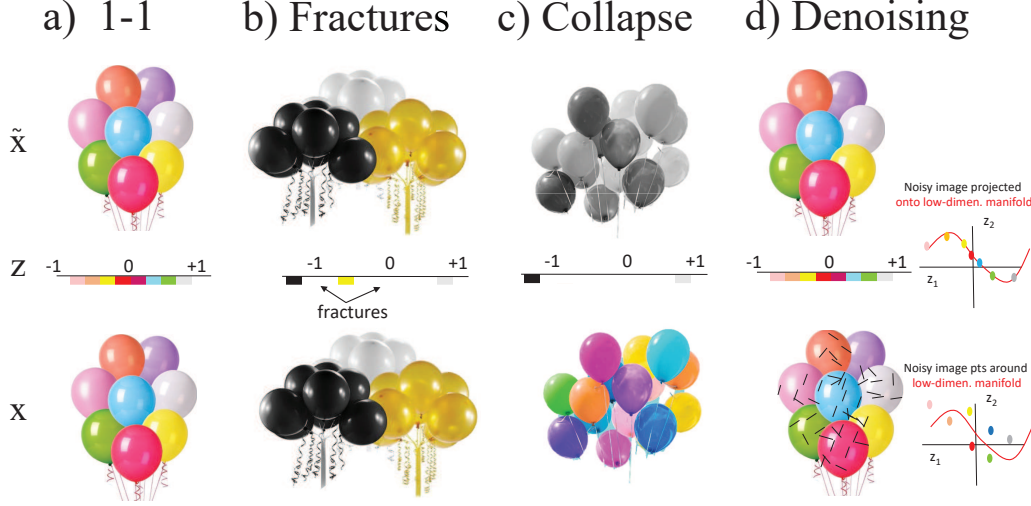


Figure 14.11: AE mappings for a) a well-behaved AE that does not *memorize* the input to give an accurate rendering of the new test image, b) an AE with a fractured latent space where the  $l^2$  differences between the black, white and yellow balloon images are also large, c) an AE with too small of a bottleneck so many colorful input images *collapse* to the same latent-space value  $z$ , and d) a denoising AE where the  $z$ -space codes resist noisy variations of the same balloon image to give a clean output image. The noisy image points surrounding the line on the bottom right in d) are projected by the encoder to be on the red 1D manifold line in the leftmost middle graph (Vincent et al., 2008).

trated in Figure 14.13a. Here, flatter slopes  $\partial z / \partial x \approx 0$  of the red solid lines means that a wide variety of noisy full-moon images map to nearly the same latent code  $\mathbf{z}$  of a "clean" full-moon image. Unfortunately, this diagram also suggests that flatter slopes of the red line lead to undesirably wide-fracture zone compared to the thin-fracture zone in b). In contrast, a thin fracture zone is associated with steeper slopes in the red lines.

### 14.3.3 Contractive Autoencoders

Assume a small  $l^2$  difference  $\|\mathbf{x}'_{full-moon} - \mathbf{x}_{full-moon}\| \approx 0$  between similar full-moon images in Figure 14.9. For a smooth mapping between  $\mathbf{x}$  and  $\mathbf{z}$  then  $\mathbf{z}'$  and  $\mathbf{z}$  should also have a small  $l^2$  difference  $\|\mathbf{z}' - \mathbf{z}\| \approx 0$ . To encourage this we impose the constraint that the Fréchet derivative of the output of the encoder's activation function should be small,  $\|\frac{\partial f(\mathbf{W}, \mathbf{x})}{\partial \mathbf{x}}\| \rightarrow \text{small}$ . Thus, many different full-moon images will map to a small region in  $z$ -space.

Contractive mapping is illustrated in Figure 14.13a, where the flatter solid-red lines are associated with smaller regions in the  $z$ -space. Here,  $\partial z / \partial x \approx 0$  is equivalent to  $\partial f(\mathbf{W}, \mathbf{x}) / \partial \mathbf{x} \approx 0$ .

The objective function with contractive regularization is

$$\epsilon(\mathbf{W}, \mathbf{W}') = \sum_{m=1}^M \overbrace{\|\tilde{\mathbf{x}}^{(m)} - \mathbf{x}^{(m)}\|^2}^{\text{data misfit}} + \lambda \sum_{m=1}^M \|\nabla_x f_i^{[m]}\|^2, \quad (14.6)$$

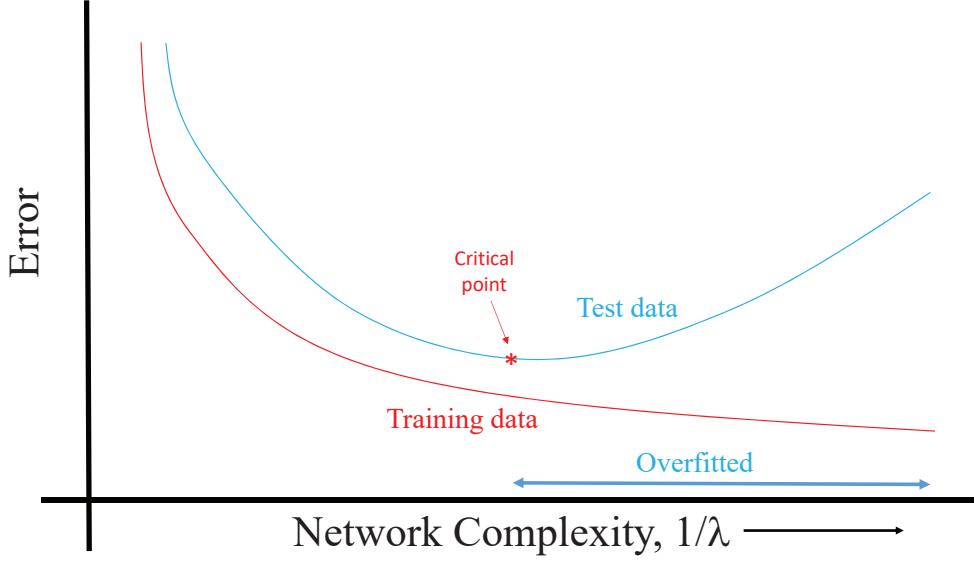


Figure 14.12: Prediction accuracy of the training and test data versus complexity/ $1/\lambda$  of the AE. If the network is too complex, starting at the red critical point, then it starts to memorize both noise and signal in the input images. Increasing the value of the regularization parameter  $\lambda$  tends to eliminate memorization.

where the summation is over the training examples and  $f_i^{[m]}$  is the  $i^{th}$  component of the encoder's activation function for the  $m^{th}$  example.

The gradient of equation 14.6 is related to that of a Fréchet derivative (Schuster, 2017) in seismic inversion, i.e.  $\nabla_k f_i^{[m]} = \partial f_i^{[m]} / \partial x_k$  where  $x_k$  is the  $k^{th}$  component of the data vector input into the encoder's only hidden layer. It is a measure of the sensitivity of the encoder's activation output to the change in the  $k^{th}$  component of input data. The goal of contractive autoencoders is to keep this sensitivity small so as to compress the cloud of input data points  $\mathbf{x}^{[m]}$  with the same class into a small region in  $\mathbf{z}$  space (see Figure 14.13a). It is assumed that batch normalization is used to stabilize and enhance the convergence rate (Ioffe and Szegedy, 2015).

#### 14.3.4 Sparse Autoencoders

The sparse regularizer encourages the activation of a small number of neurons per layer. This is a cure for overfitting if there are a large number of hidden units in the network; e.g., there are more hidden units than inputs. Reducing the number of units reduces the tendency to memorize the input images, but not too much of a reduction so it still is able to extract important features of the input data (Asperti, 2018). Such a regularized network will be denoted as a sparse autoencoder (Ng, 2013; Zhou, 2018).

An example of a sparsity regularizer is the  $L_1$  norm of the  $I$  components of the activation functions  $f_i^{[m]}$   $i \in [1, 2, \dots, I]$  that can be added to the encoder's objective function:

$$\epsilon(\mathbf{W}, \mathbf{W}') = \sum_{m=1}^M \overbrace{\|\tilde{\mathbf{x}}^{(m)} - \mathbf{x}^{(m)}\|^2}^{\text{data misfit}} + \lambda \sum_{m=1}^M \overbrace{\sum_{i=1}^I \|f_i^{[m]}\|}^{\text{regularizer}}, \quad (14.7)$$

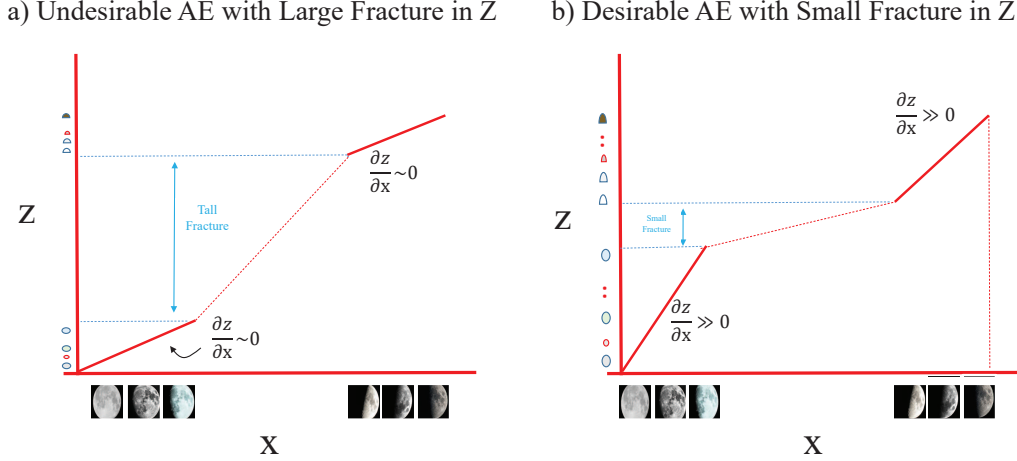


Figure 14.13: Mapping of input images of the moon to the associated 1D  $\mathbf{z}$ -domain points along the slanted red lines. As illustrated in a), a contractive mapping encourages flatter solid lines that map to a smaller region size in the  $\mathbf{z}$ -domain. This is similar to denoising AEs which resist small perturbations in the input image so they map to the same  $\mathbf{z}$ . On the other hand, an adversarial AE tends to encourage steeper slopes that reduce the size of the fracture zones in b).

where the summation in  $m$  is over the  $M$  training examples. After training, the smallest number of nodes in the hidden layer is employed by the network to compute  $\mathbf{z}$ , which avoids the overfitting tendency of a complex network with many nodes. Fewer nodes means less complexity and encourages the trained network to be at or just to the left of the critical red point in Figure 14.12. The sparsest number of nodes that yields accurate predictions forces the model to focus on the really important features, highly reducing the risk of overfitting. An alternative sparse regularizer is to incorporate the Kullback-Leibler penalty function in the objective function (see Appendix 14.9). Also, the manifold of the latent space variables can be shaped with smoothness regularizers (Oring et al., 2020).

A numerical example that illustrates the performance of sparsity is shown in Figure 14.14. Here the MNIST data set is used to train the standard AE and the AE with the  $L_1$  regularizer. The output AE images are nearly identical to the input images for both the standard and regularized AEs. However, the mean square error for the regularized MSE is about 5% less than that for the standard AE.

Figure 14.15 shows that there are far fewer activated neurons than generated in the standard AE. This will help mitigate problems with overfitting. However, it theoretically does not overcome the problem of fracturing illustrated in Figure 14.13.

### 14.3.5 Variational Autoencoder

The autoencoder can be trained to denoise input images and to reduce the dimension of the input image into a smaller-dimensional latent vector  $\mathbf{z}$ , as illustrated by the AE architecture in Figure 14.1. However, interpolating between two points  $\bar{\mathbf{z}} = (\mathbf{z}_1 + \mathbf{z}_2)/2$  and feeding the interpolated vector  $\bar{\mathbf{z}}$  into the decoder  $D(\bar{\mathbf{z}})$  often fails to generate a new image that is realistic, as seen in the

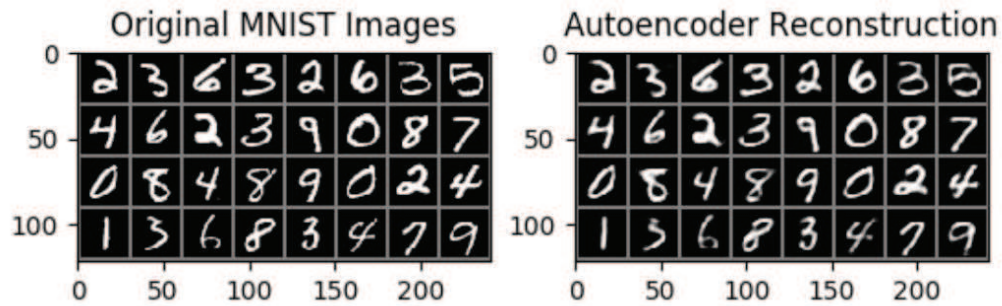


Figure 14.14: MNIST images as (left) input and (right) output of a  $L_1$  regularized AE (from Zhou, 2018).

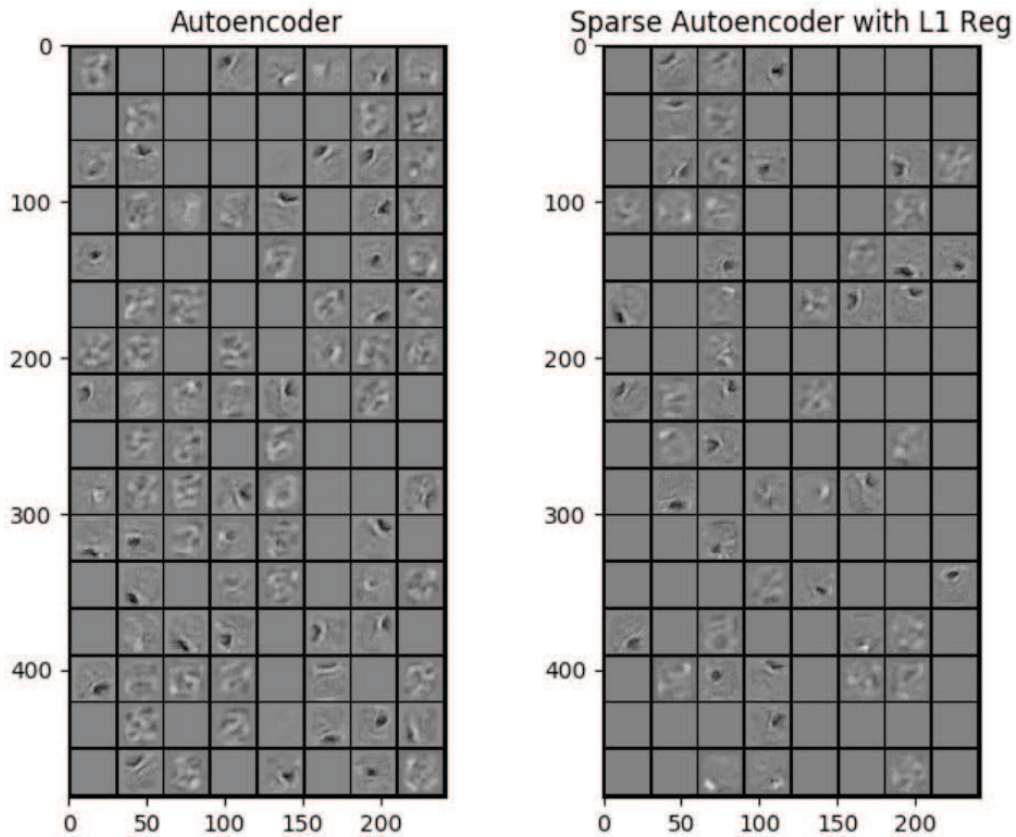


Figure 14.15: Activated nodes from the (left) standard and (right)  $L_1$  regularized AE for the MNIST data depicted in Figure 14.14 (from Zhou, 2018). There are many fewer activated nodes for the regularized AE.

middle row of images in Figure 14.2. The problem is that the mapping between latent space and the decoder's image space is highly non-linear, so there is a wide fracture in the  $\mathbf{z}$  domain as illustrated

## Variational Autoencoder

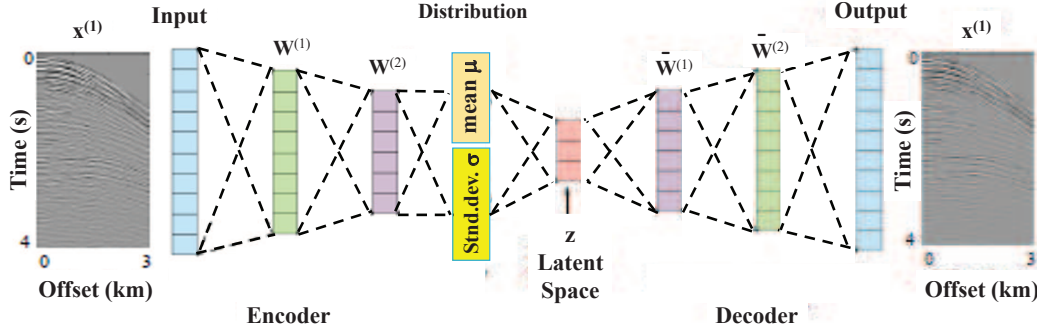


Figure 14.16: Architecture of the variational autoencoder (VAE) where a large set of training examples is used to compute the learnable parameters  $(\mathbf{W}^{(i)}, \bar{\mathbf{W}}^{(i)}, \mu, \sigma)$  where  $(\mu, \sigma)$  are the mean and standard deviation vectors for a multivariate Gaussian. Once the VAE is trained, then a new output  $\bar{x}$  is produced by sampling the distribution to get a new latent vector  $\bar{z}$ , and then inserting  $\bar{z}$  into the decoder to get  $\bar{x}$ .

in Figure 14.9. Thus, a small change in the values of the latent variables can lead to an enormous change in the output image. This is similar to standing on the edge of a steep cliff, where a small change in the horizontal  $(x, y)$  coordinates of your feet can lead to an enormous change in the output of your elevation  $z$ .

A partial remedy to the AE fracturing problem (Ghosh et al., 2019) is the variational autoencoder (Kingma and Welling, 2014; Doersch, 2016). The variational autoencoder (VAE) is a *regularized autoencoder* that learns both the NN weights and the statistical parameters  $(\mu, \sigma)$  associated with the training data. Its architecture is almost identical to that of the AE except it includes a *distribution* layer where the output is drawn from a statistical distribution as illustrated in Figure 14.16. The VAE encodes the input training data  $\mathbf{x}$  as a probability distribution instead of points; and its latent space variables  $\mathbf{z}$  are regularized by constraining these points to be drawn from a standard Gaussian. This pdf is not steeply sided, so a small change in the component  $z_i$  of the latent variable will lead to an output that has similar probability and, therefore, resemblance to the input.

The mean  $\mu$  and standard deviation  $\sigma$  vectors in the Gaussian distribution describe those learned from the training data. Thus, interpolating between points in this distribution should provide new outputs that realistically resemble those from the training data. In comparison, a deterministic autoencoder often produces unrealistic output images when the latent-space vectors are interpolated (Foster, 2019).

The following steps describe the workflow for the VAE in Figure 14.16.

1. Use a large data set of input images  $\mathbf{x}^{(i)}$  to train the VAE by a gradient descent method. For forward modeling, the input  $\mathbf{x}^{(i)}$  is fed into the encoder's input with weights  $\mathbf{W}^{[n]}$ , which can be those for a fully connected neural network or a CNN. These layers compute the input into the layer with a multivariate Gaussian probability distribution. This pdf is sampled (see Chapter 19) to produce the output vector denoted as the latent vector  $\mathbf{z}$ . The latent vector  $\mathbf{z}$  is then fed into the decoder to compute the output  $\approx \mathbf{x}^{(i)}$ .
2. Once the network is trained, then a new unseen output  $\bar{x}$  is computed by sampling from the pdf, or interpolating neighboring points in the  $\mathbf{z}$  domain, to generate the latent vector  $\bar{z}$ . The



## Reparameterized Variational Autoencoder

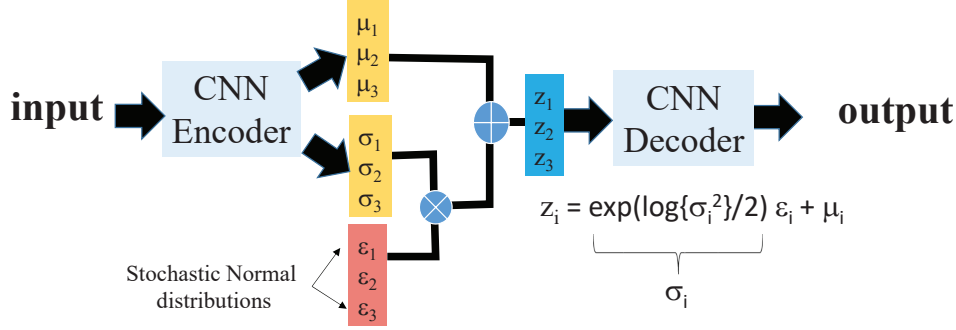


Figure 14.17: Architecture of the reparameterized variational autoencoder for a 3-dimensional latent space with  $\mathbf{z} = (z_1, z_2, z_3)$ .

latent vector  $\bar{\mathbf{z}}$  is then fed into the trained decoder to get  $\bar{\mathbf{x}}$ .

The problem with the above approach is not with the forward modeling, but in the back-propagation by a gradient descent method. Gradient descent works well with the deterministic layers because their operations are represented by deterministic functions such as sigmoids or operations such as convolutions. In these cases, the formulas for the Fréchet derivatives can easily be obtained. However, the computational operation associated with the distribution layer in Figure 14.16 is the non-deterministic sampling of a probability function, whose formula for the Fréchet derivative cannot be easily obtained. To overcome this problem, the  $i^{th}$  component  $z_i$  of the latent vector  $\mathbf{z}$  is re-parameterized by the following formula:

$$z_i = \mu_i + \sigma_i \epsilon_i, \quad (14.8)$$

where  $\epsilon_i \propto \exp(-\frac{1}{2}z_i^2)$  is the standard normal Gaussian distribution with zero mean and unity variance, and  $\sigma_i$  is re-expressed<sup>2</sup> as  $\sigma_i = \exp(\log\{\sigma_i^2\}/2)$ . Now, the Fréchet derivative of  $z_i$  w/r to the learnable parameters  $\sigma_i$  or  $\mu_i$  can easily be derived and computed. This reparameterization trick allows for the mean and log-variance to remain learnable parameters and still retain the stochasticity of the sampling.

Figure 14.17 depicts the VAE with a  $3 \times 1$  latent vector  $\mathbf{z}$ . Since the distributions  $\epsilon_i$  in the red rectangle are not functions of the learnable mean and variance parameters, then there is no need for a Fréchet derivative of  $\epsilon_i$ .

A TensorFlow code for the sampling operation is taken from Paul (2020):

```
def sampling(args)
    z_mean, z_log_var = args
    epsilon = K.random_normal(shape=(batch_size, latent_dim), mean=0.)
    return z_mean + K.exp(z_log_var / 2) * epsilon
```

<sup>2</sup>Here,  $\log\{\sigma_i^2\}$  is known as the log-variance and is used as the output to the decoder prior to sampling because it forces the encoder to have the output range of the real numbers rather than just the positive values of variances. This allows for smoother representations for the latent space (Paul, 2020).

Experience suggests that a KL regularizer should be added to the mean square error (MSE) loss function:

$$\epsilon = \frac{1}{2} \|F(\mathbf{x}) - \mathbf{x}\|^2 + \lambda KL(P||P_0), \quad (14.9)$$

where  $F(\mathbf{x})$  represents the operations of the VAE,  $\mathbf{x}$  is the input,  $\lambda > 0$  is the damping coefficient and  $KL(P||P_0)$  is the KL regularizer that penalizes the VAE distribution  $P(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\sigma})$  if it departs from the desired one  $P_0 = P(\mathbf{x}|\boldsymbol{\mu}_0, \boldsymbol{\sigma}_0)$ . See Appendices 14.9 and 20.11 for further information about the KL regularizer.

In summary, the variational autoencoder (VAE) is a regularized autoencoder that learns the statistical parameters associated with the data (Kingma and Welling, 2014; Doersch, 2016). The VAE encodes inputs as distributions instead of points and whose latent space is regularized by constraining distributions returned by the encoder to be close to a standard Gaussian. Without this regularization, there are neighboring areas in latent space which don't represent any of our observed data. The advantage of the variational autoencoder (VAE) is that the latent space vectors can be interpolated to give new output images that are more realistic than interpolated images from autoencoders (Foster, 2019). This can be useful for training CNN architectures without the need for a large set of classified data.

The VAE can also be considered as a sparse AE because it encourages the internal coding of the encoder to become sparse (Dai et al., 2018; Asperti, 2018). Regularization is explicitly incorporated into the VAE by using the Kullback-Leibler component in the objective function (Asperti, 2018). Regularization can be manually improved by eliminating the poorly activated nodes in the VAE as training progresses. If the number of active units is too few then this phenomenon is referred to as over-pruning (Young et al., 2018). However, Berthelot et al. (2018) shows examples where the VAE significantly underperforms compared to ACAI.

### 14.3.6 Adversarially Constrained Autoencoder Interpolation

The ACAI (Berthelot et al., 2018) uses equation 14.3 to interpolate two latent-space vectors  $\mathbf{z}_1$  and  $\mathbf{z}_2$ , and then decodes the interpolated vector  $\tilde{\mathbf{z}}$  with equation 14.4 to get the interpolated output image  $\hat{\mathbf{x}}$ . If the output image appears to be a new but realistic input image, then the ACAI could be an efficient means for data augmentation.

To promote the construction of a *realistic interpolated output*  $\tilde{\mathbf{x}}$  that appears to be part of the input training set  $\mathbf{x}$ , Berthelot et al. (2018) regularizes the AE to include a critic network, as illustrated in Figure 14.18. This critic network, denoted by  $d(\tilde{\mathbf{x}}, \gamma)$ , predicts the actual interpolation coefficient  $\alpha$  and is incorporated in the regularized objective function  $\epsilon^{RAE}$ :

$$\epsilon^{RAE} = \sum_{m=1}^M \|\overbrace{\hat{\mathbf{x}}(\mathbf{W}, \mathbf{W}')^{(m)}}^{\text{input image}} - \mathbf{x}\|^2 + \lambda \sum_{m=1}^M \|d(\overbrace{\hat{\mathbf{x}}^{(m)}}^{\text{interpolated image}}, \gamma)\|^2, \quad (14.10)$$

where  $\lambda$  is the regularization parameter and  $\gamma$  represents the coefficients of the critic network that, in this case, are functions of the coefficients of  $\mathbf{W}$  and  $\mathbf{W}'$ . The components of  $\mathbf{W}$ ,  $\mathbf{W}'$  are computed by stochastic gradient descent (SGD) to minimize  $\epsilon^{RAE}$ , and then the critic's coefficients in  $\gamma$  are computed that minimize the critic's objective function

$$\epsilon^{critic} = \lambda \sum_{m=1}^M \|d(\hat{\mathbf{x}}^{(m)}, \gamma) - \alpha\|^2. \quad (14.11)$$

This alternating procedure of training for  $\mathbf{W}$ ,  $\mathbf{W}'$  in equation 14.10 and then for  $\gamma$  in equation 14.11 is repeated until suitable convergence.



The goals of minimizing  $\epsilon^{RAE}$  and  $\epsilon^{critic}$  are contradictory: in equation 14.10 we find  $\mathbf{W}, \mathbf{W}'$  that minimizes both the data misfit and the predicted interpolation coefficient  $d(\hat{\mathbf{x}}^{(m)}, \gamma) \rightarrow 0$ , while in equation 14.11 we find the components of  $\gamma$  that correctly predict the actual interpolation coefficient  $\alpha > 0$ . If the predicted interpolation coefficient  $d(\hat{\mathbf{x}}^{(m)}, \gamma)$  is nearly zero, then this suggests that  $\hat{\mathbf{x}}$  is of high quality because it is mostly interpolated from a true input  $\mathbf{x} = g((1 - \alpha)\mathbf{x}_1 + \alpha\mathbf{x}_2) \approx \tilde{\mathbf{x}}_1$  for  $\alpha \approx 0$ . However, a high quality interpolation is also possible if the weights  $\mathbf{W}, \mathbf{W}'$  are computed to mostly eliminate fracturing zones in, e.g. Figure 14.10, and intermingling of different classes<sup>3</sup>. This is the role of training equation 14.10 by SGD: find the weights that try to eliminate fracture zones and intermingling of points from different classes.

If the critic successfully predicts the interpolation coefficient  $\alpha \neq 0$  used by the IAE then it has succeeded in identifying  $\hat{\mathbf{x}}$  as an interpolated image. However, if the interpolation point is in a fractured zone (see blurry digit reconstruction in upper right of Figure 14.10) then  $\alpha \gg 0$  indicates an interpolated latent point which is decoded into a poor-quality decoded image  $\hat{\mathbf{x}}$ . Nevertheless, the hope is that the image quality is high because the AE has eliminated the fracture zone problem so that  $d(\hat{\mathbf{x}}, \gamma) \approx 0$ . If the value of  $\lambda$  is too large then  $\gamma$  is undesirably prioritized to correctly predict the actual interpolation coefficient  $0.5 > \alpha \gg 0$ . If  $\lambda$  is small enough then training  $\mathbf{W}, \mathbf{W}'$  is prioritized to mitigate fracture zone problems and allow for high quality interpolations with  $0.5 > \alpha \gg 0$  where the critic is fooled  $\|d(\hat{\mathbf{x}}^{(m)}, \gamma)\| \rightarrow 0$  into choosing the coefficients in  $\gamma$  that suggest the interpolated image  $\hat{\mathbf{x}}$  is realistic and selected from the original training set. This tension between minimizing  $\epsilon^{RAE}$  and  $\epsilon^{critic}$  is similar to that between the goals of the generator and discriminator in a generative adversarial network (GAN) discussed in Chapter 18. It is related to the Nash equilibrium point discussed in section 18.3.

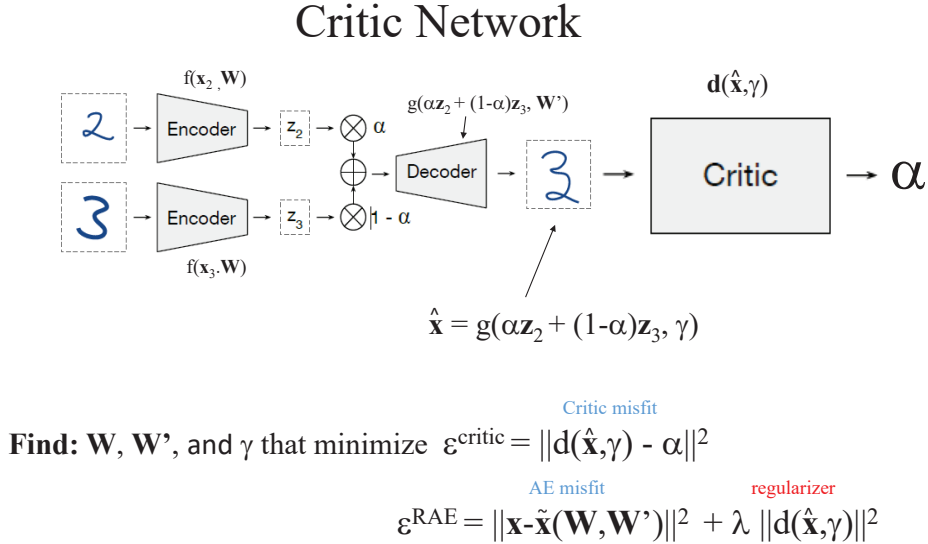


Figure 14.18: Critic network and the formulas for the  $\epsilon^{critic}$  and  $\epsilon^{RAE}$  objective functions at the bottom.

As pointed out by Berthelot et al. (2018), the regularization term  $\sum_{m=1}^M \|d(\hat{\mathbf{x}}^{(m)}, \gamma)\|^2$  is *effectively trying to make the critic output 0 regardless of the value of  $\alpha$ , thereby "fooling" the critic into thinking that an interpolated input is non-interpolated (i.e., having  $\alpha = 0$ )*. The parameters  $\mathbf{W}$  and

<sup>3</sup>There are many non-unique AEs that accurately code and decode the same input, but we are seeking the one with desired properties such as the sparse AE on the rightside of Figure 14.15.

$\mathbf{W}'$  are optimized with respect to  $\epsilon^{RAE}$  (which gives the autoencoder access to the critic's gradients) and  $\gamma$  is optimized with respect to  $\epsilon^{critic}$ . Berthelot et al. (2018) refer to interpolating  $z$  – domain codes with equations 14.10 and 14.18 as ACAI.

As an example, the bottom row of Figure 14.8 shows the result of using an ACAI network to interpolate points in the  $z$ -space. The ACAI interpolated lines just below the two-sided arrow are almost identical to the ground-truth images in the top row. Berthelot et al. (2018) show that ACAI is superior to many of the other regularized AE's (Verma et al., 2018) such as the VAE (Kingma and Welling, 2014; Dai et al., 2018; Asperti, 2018), vector quantized VAEs (Yadov, 2019), and adversarial AEs (Makhzani et al., 2016). Another regularized AE includes the generative AE (Mescheder et al., 2017; Roy et al., 2018; Mao et al., 2018) which has an architecture that is similar to that of the ACAI. Orling et al. (2020) proposed shaping the latent space manifold using smoothness regularizers.

## 14.4 Numerical Examples

The application of autoencoders is now presented for seismic data. First, autoencoders are used to interpolate seismic data, and then they are used to invert seismic data for the velocity model. Finally, an example is given for using autoencoders to interpolate time-lapse data for  $CO_2$  monitoring.

### 14.4.1 Seismic Interpolation Example

Seismic traces in exploration surveys are often spatially undersampled, where the sampling interval of the geophones is larger than the smallest horizontal wavelength (Yilmaz, 2001). For example, surface waves recorded in land surveys are severely aliased because their slow velocities lead to very short wavelengths at higher frequencies. In addition, there are large gaps in seismic surveys because of, for example, large obstructions in the survey area, localities are off-limits due to regulations, or the near-offset traces are missing in marine surveys due to the difficulties in deploying hydrophones next to the boat and the airgun sources.

To fill in the trace gaps, many schemes have been promoted such as model-based interpolation (Stolt, 2002; Fomel, 2003) predictive deconvolution (Spitz, 1991; Abma and Claerbout, 1995), normal-moveout and FK filtering (Yilmaz, 2001), and compressive sensing (Herrmann and Hennenfent, 2008; Gan et al., 2015; Wang et al., 2014). All of these methods have their strengths and limitations, and no method is perfect under severe aliasing conditions.

One of the candidates for interpolating aliased seismic data is the denoising autoencoder used for inpainting (Xie et al., 2012; Pathak et al., 2012) by the computer vision community. For inpainting seismic data, Mandelli et al. (2018) proposed an autoencoder network with the U-Net architecture (Ronneberger et al., 2015) illustrated in Figure 14.19. The latent layer is at the bottom of the diagram where the latent-space vector has the dimension of  $512 \times 1$ .

The following is a summary of the seismic interpolation work performed by Mandelli et al. (2018). Seismic data from a marine survey are used to give 1001  $1024 \times 128$  shot gathers in the  $t - x$  domain, one of which is shown in Figure 14.20a. These shot gathers are decimated by factors of 10%, 30% and 50% in Figures 14.20b-14.20d. Each shot gather is segmented into  $128 \times 128$  patches, these patches are demeaned and normalized, and then more than 28,000 patches are used for the process of training, validation and testing. In the following examples, 57%, 18% and 25% of the original data set are, respectively, used for training, validation, and testing.

The input layer consists of the concatenation of three decimated  $128 \times 128$  images: the original shot gather patch  $\mathbf{P}$  and the  $128 \times 128$  spatial gradients  $\partial \mathbf{P} / \partial x$  and  $\partial \mathbf{P} / \partial y$  of this patch in the  $x$ – and  $y$ – directions. Empirical tests suggest that the gradient components are important for this problem (Mandelli et al., 2018).

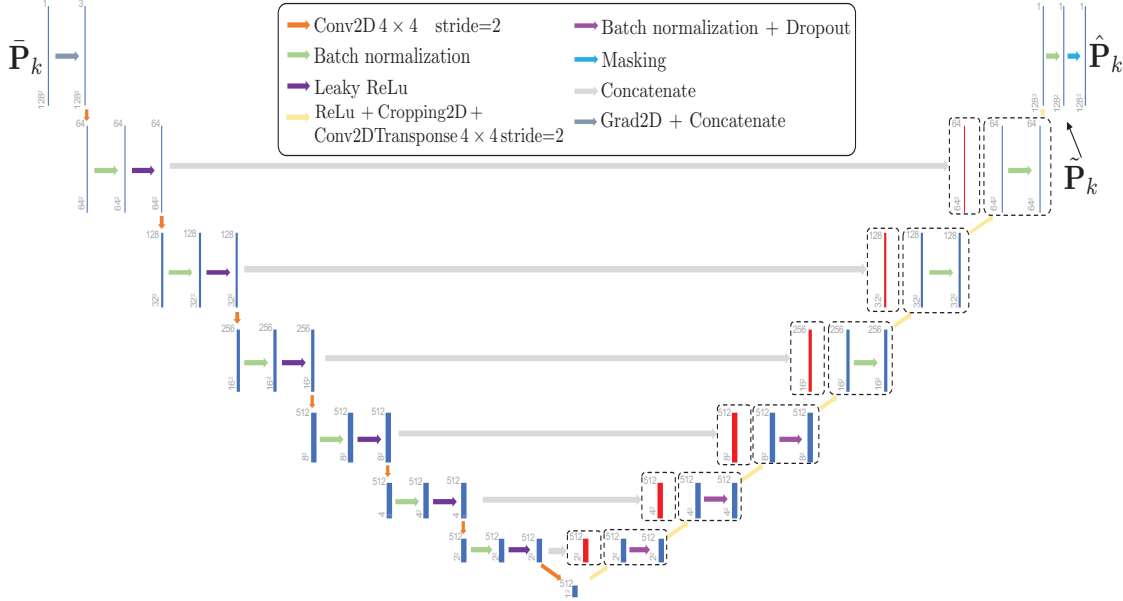


Figure 14.19: U-Net architecture used for the autoencoder (Mandelli et al., 2018) to *inpaint* the corrupted traces.

The training data are inserted into the U-Net architecture in Figure 14.19, where each hidden layer in the encoding layers, i.e. left side, of the U-Net consists of three operations: 2D convolution by  $4 \times 4$  filters, batch normalization, and a leaky ReLU operation. The output image is the uncorrupted version of the decimated input image. To reduce the size of the output feature maps from one layer to the next, a stride of two is used for the 2D convolutions and padding is employed for each convolution so the output image is the same size as the input image. The first hidden layer uses  $64 \ 4 \times 4$  filters, and the next layer doubles the number of filters to  $128 \ 4 \times 4$  filters while halving the size of the feature maps by a stride of two. This doubling the number of filters and halving the size of the feature maps at each layer continues until the last layer at the bottom of the U-Net. In this last layer the latent-space vector has a dimension of  $512 \times 1$ .

The decoding, i.e. right side, of the U-Net is similar to that of the coding portion except 1) the encoding output at the neighboring layer to the left of the decoding layer is concatenated onto the decoder's input, and the transpose operation (<https://medium.com/apache-mxnet/transposed-convolutions-explained-with-ms-excel-52d13030c7e8>) is used for the 2D convolution to give an output feature map twice the size of the input feature map.

An Adam optimization method (Kingma and Ba, 2014) is used for batches of 128 images, and an initial learning rate of 0.01 is used. The learning rate was halved when the residual reduction stalled, and no more 20 epochs were required to get acceptable interpolations, as shown in Figures 14.21b-14.21d. These results were noticeably more accurate than those using the Multichannel Singular Spectrum Analysis (Oropeza and Sacchi, 2011) method.

#### 14.4.2 Seismic Tomography

We now present an autoencoder example for inverting skeletonized seismic data. The mathematical details and FWI examples are presented in Chapter 25. For now, we will test this methodology on seismic data recorded by the crosswell survey shown on the left side of Figure 14.22. Here, there are 77 sources (red stars) along the left well and 156 receivers (blue quadrilateral) along the right

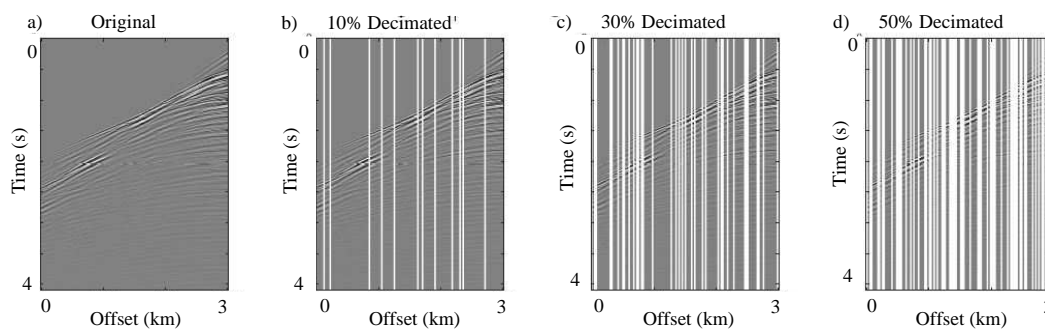


Figure 14.20: a) Original shot gather and shot gathers with b) 10%, c) 30% and d) 50% decimation of traces. Images from Mandelli et al. (2018).

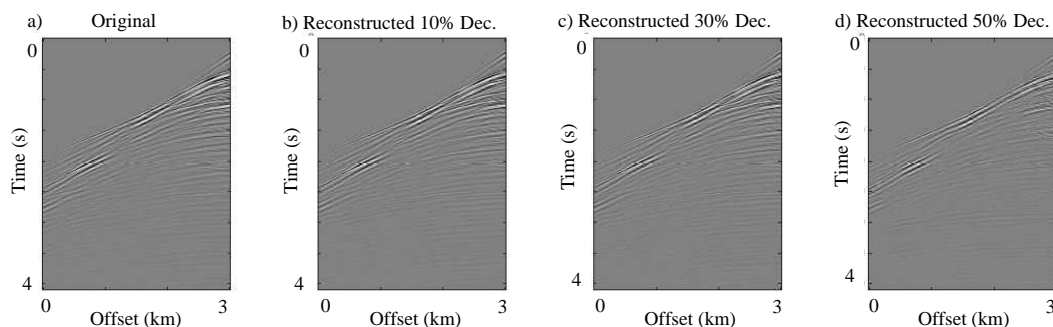


Figure 14.21: a) Original shot gather and shot gathers reconstructed from b) 10%, c) 30% and d) 50% missing traces in Figure 14.20. Images from Mandelli et al. (2018).

well, and the source and receiver sampling intervals are 20 m and 10 m, respectively. We mainly focus on the inversion of the transmitted arrivals by windowing the input data around the early arrivals.

The inversion strategy, as outlined in Figure 14.23, is to employ an autoencoder to skeletonize the input traces as low-dimensional latent-space variables. Then, use the wave-equation method similar to that of Luo and Schuster (1991a,b) to invert for the associated velocity model. Instead of picking the transmission traveltimes the trained autoencoder automatically determines the latent variables for each trace. In our examples, we will train for just one latent variable for each trace, and compare this to inversion of several latent variables per trace (Chen and Schuster, 2019).

The following workflow is used to train the autoencoder network.

1. Build the training set from just one of every five shot gathers. For the Gaussian-anomaly data, the training data consist of 2496 seismic traces. Each shot gather is computed by a finite-difference solution to the 2D acoustic wave equation.
2. Data processing. The input trace is Hilbert transformed to get its envelope, then the resulting traces are normalized by the variance of that trace. Figures 14.24a and 14.24b show a seismic trace before and after processing, respectively.
3. Training the autoencoder. We feed the processed training set into an autoencoder where the dimension of its latent space is equal to 1. In other words, each training trace with a

## Gaussian Anomaly & Crosswell Data

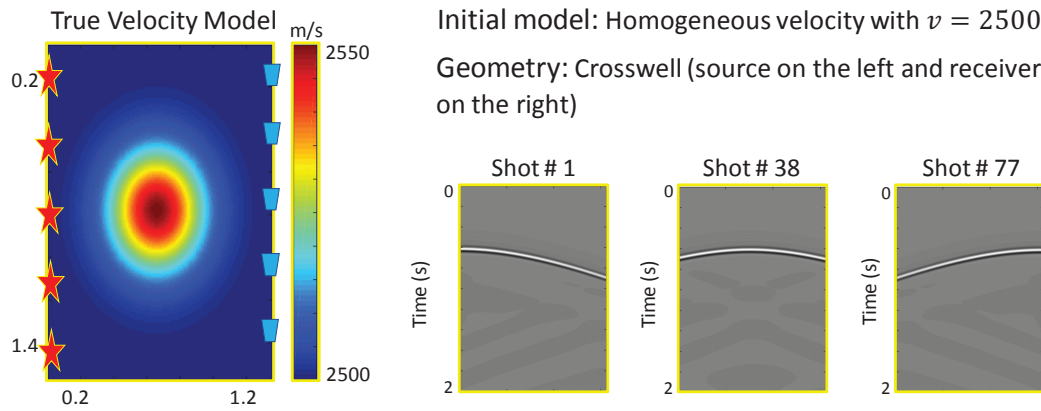


Figure 14.22: Gaussian anomaly model, In this example there are 77 sources (red stars) along the left well and 156 receivers (blue quadrilaterals) along the right well. Figure adapted from Chen and Schuster (2020).

### Machine Learning + Wave Equation Inversion of Skeletonized Data

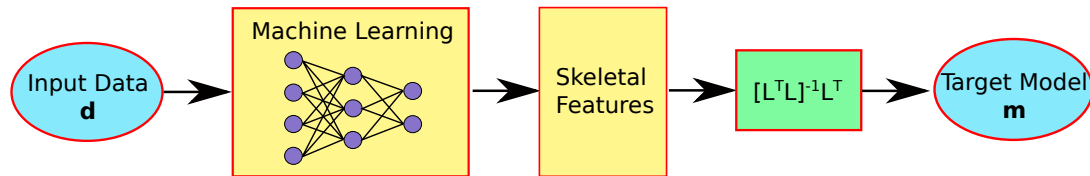


Figure 14.23: The inversion strategy is to generate the skeletal features, i.e. latent variables, by an autoencoder, and invert these features by an iterative wave-equation inversion method. Figure from Chen and Schuster (2020).

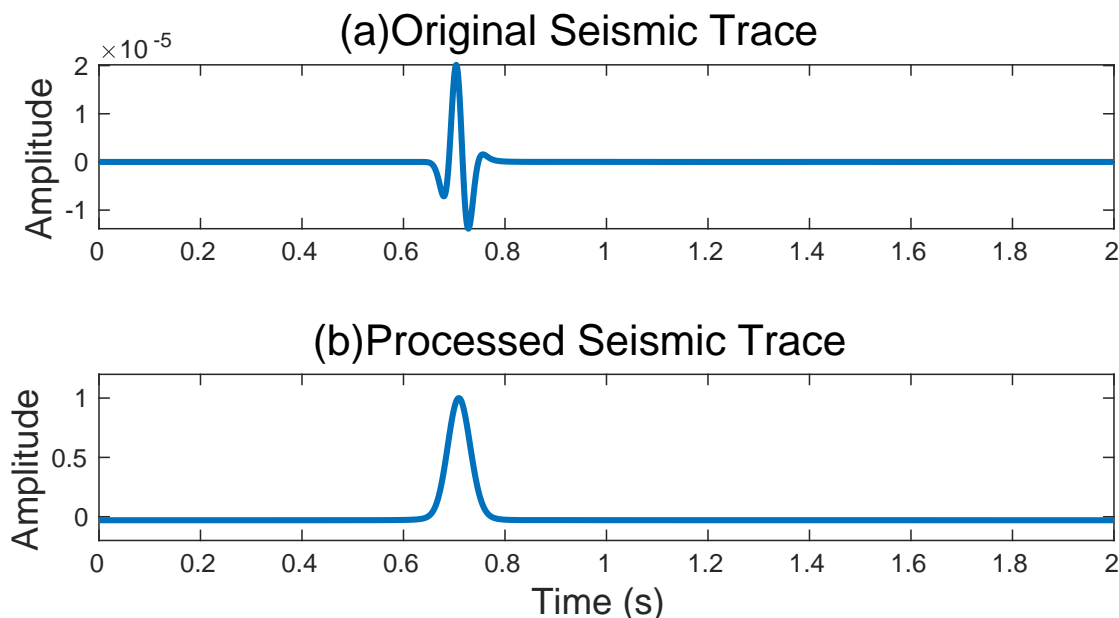


Figure 14.24: The a) original seismic trace and the b) processed seismic trace. Figure from Chen and Schuster (2020).

dimension of  $nt \times 1$  will be encoded as one latent variable by the encoder. The autoencoder parameters are updated by iteratively minimizing equation 14.1 using the architecture shown in Figure 14.25. Figures 14.26a and 14.26b show, respectively, an input training example and its corresponding reconstructed signal by the autoencoder.

Figures 14.27a, 14.27b and 14.27c show three observed shot gathers which are not included in the training set, and their encoded values are shown in Figures 14.27d, 14.27e and 14.27f which are the skeletonized representations of the input seismic traces. The encoded values do not have any units and can be considered as skeletonized attributes of the data, even though they appear to have about the same moveout patterns as the traveltimes of the first arrivals. The autoencoder believes that these encoded values are the best low-dimensional representations of the original input in the least-square sense.

We compare the traveltimes and the encoded values for the observed (black curves) and synthetic (red curves) data in Figure 14.28. Figures 14.28b shows a larger traveltime difference than Figures 14.28a and 14.28c as its propagating waves are affected more by the Gaussian anomaly than the other two shots. Similar to the traveltime values, the encoded values are also sensitive to the velocity changes.

Finally, the far rightmost tomogram Figure 14.29 results from the inversion of two latent variables per trace. It can be seen that there is a slight accuracy improvement with the two-latent variable result compared to the one-latent tomogram. In comparison, the wave-equation traveltime inversion (WT) tomogram (Luo and Schuster, 1991a, b) is shown on the left of Figure 14.29, which provides a modestly more accurate estimate of the true model. The WT method uses solutions to the wave equation to invert the first-arrival traveltimes and so avoids the high-frequency approximation of ray tracing. However, the WT method requires picking of the traveltimes while the autoencoder automatically estimates the latent values as the skeletonized data. This can, in principle, result in a great amount of time savings when millions of traveltimes need to be picked for 3D data. This claim should be tested with data recorded in large field surveys.

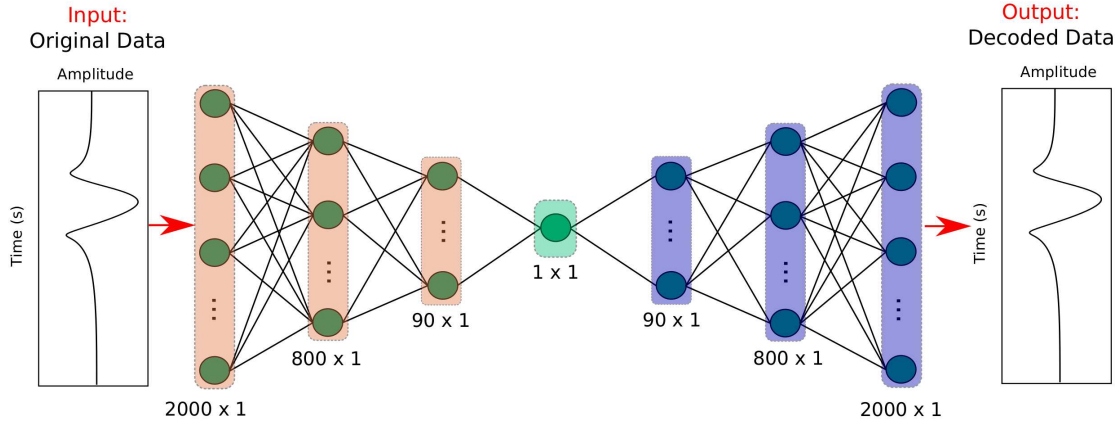


Figure 14.25: The architecture of the autoencoder neural network used for inverting the Gaussian-anomaly data. Figure from Chen and Schuster (2020).

The problem with the above result is that the decoder  $g(\mathbf{z})$  is not necessarily a continuous function of  $\mathbf{z}$  for all values of  $\mathbf{z}$ . Empirical experiments with latent-space vectors with dimensions larger than 1 show significant problems in convergence. This problem is partly addressed with the constraints introduced in Chen and Schuster (2020). One possibility is to include regularization of the latent-space vectors in the AE objective function (Oring et al., 2020).

### 14.4.3 4D Seismic Image Monitoring and Forecasting

The autoencoder can be used for interpolating data in 4D seismic monitoring of CO<sub>2</sub> sequestration projects. The frequency of seismic data acquisition in the CO<sub>2</sub> sequestration project is limited by high cost, availability of equipment, exclusion zones around surface structures, etc (Koster et al., 2000). To improve the security and applicability of the CO<sub>2</sub> sequestration project, the missing data between the infrequent acquisitions are required for real-time monitoring. An interpolation network can be used to “connect the dots” between the seismic data in years and its structure is given in Figure 14.30. Two encoders and the three decoders share their weights with other encoders and decoders. A compressed representation of the seismic data is generated in the latent space by the encoder-decoder network, then a linear regularization is imposed on the latent space. The latent space is interpolated based on the total CO<sub>2</sub> injected volume:

$$L_{n+i} = L_n + \frac{m-i}{m-n}(L_{n+m} - L_n), \quad (14.12)$$

where  $d_n$  and  $d_{n+m}$  are the observed seismic data with total CO<sub>2</sub> injected volume  $n$  and  $n + m$ ,  $L_n$  and  $L_{n+m}$  are their latent space representations. The interpolated latent space  $L_{n+i}$  with total CO<sub>2</sub> injected volume  $n + i$  is fed into the decoders to reconstruct the interpolate seismic data.

The interpolation network is tested with the Sleipner data and an example is shown in Figure 14.31. When no real seismic data are acquired, the data gap between the years can be filled by the high fidelity synthetic seismic data being generated by the interpolation network. In comparison, the linear interpolation in the data domain can only generate the cross-dissolve transition in Figure 14.31b.

The autoencoder can be also used for forecasting the seismic data in the CO<sub>2</sub> sequestration projects. To better learn the complex temporal dynamics of CO<sub>2</sub> plume migration, a long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997) layer is incorporated with the autoencoder for

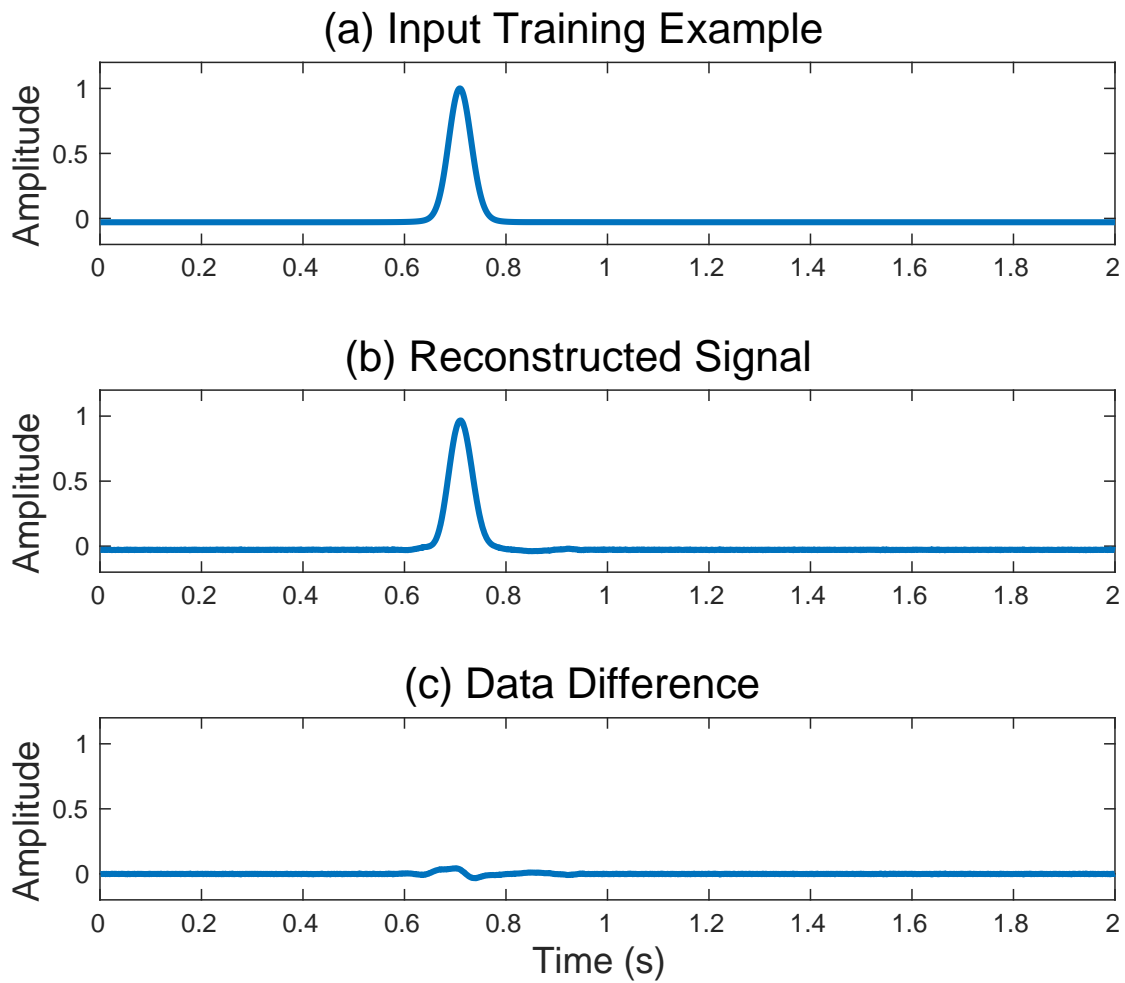


Figure 14.26: The a) input training example, b) reconstructed signal by autoencoder and c) their difference. Figure from Chen and Schuster (2020).



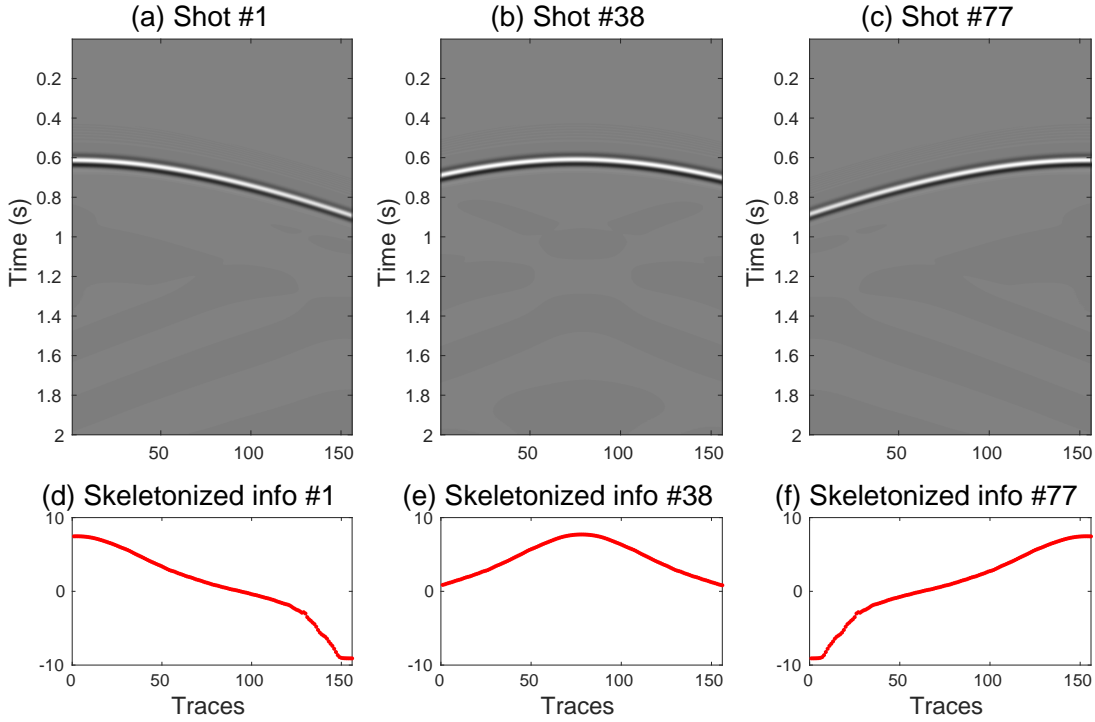


Figure 14.27: Three shot gathers with their corresponding encoded data. Figure from Chen and Schuster (2020).

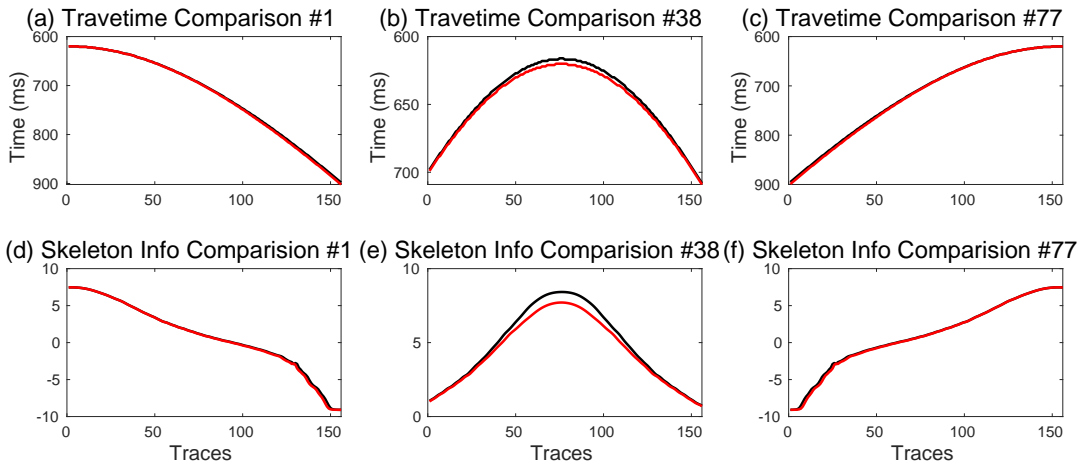


Figure 14.28: Comparison of (top row) traveltimes and (bottom row) skeletal-data latent values for different shot gathers: black and red curves represent the observed and synthetic data, respectively. The latent variables in the bottom row are computed from the traces generated from either the tomographic model or the Gaussian-velocity model. Figure from Chen and Schuster (2020).

## Numerical Results

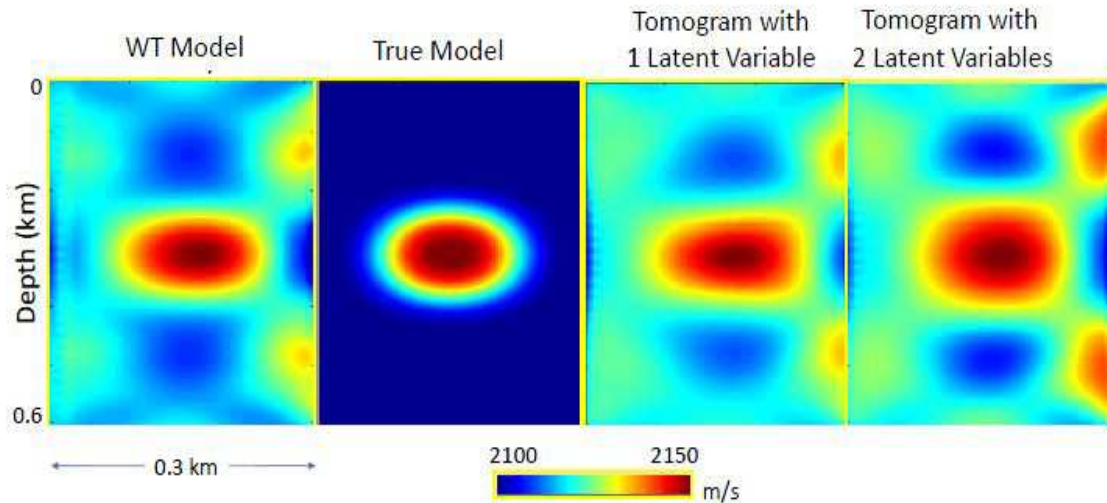


Figure 14.29: Left to right: WT, true, 1-latent-variable, and 2-latent-variable velocity models.

the extrapolation of the seismic data in Figure 14.32. Instead of using seismic data as input, the extrapolation network has better performance using seismic data change as the input. LSTM takes the historical latent representations of the seismic data change as input and predicting the future states. In addition, an optical flow (Horn and Schunck, 1981) regularization is applied to constrain the movement direction of the reflection events over time.

The extrapolation network is also tested on the Sleipner data. Figure 14.31 shows the data change in the CO<sub>2</sub> plume area from 9.6 to 12.0 Mt. The predicted data change from linear interpolation is far away from the real data. With the regularization of the optical flow, the predicted data change from the extrapolation network is more consistent with the real data than that from the extrapolation network without optical flow.

## 14.5 Summary

The autoencoder architecture is defined as a symmetrical set of layers where the number of neurons per layer progressively diminish towards the center of the architecture, where the central layer has the fewest number of units. If the number of units is too few in the central layer then it will be impossible for the autoencoder to produce a nearly identical copy of the input image at the output layer. Thus, a trial-and-error method is used to determine the smallest dimension of the bottleneck latent space that can be decoded and still reproduce an acceptably accurate rendering of the input image. The benefit is that the latent space has many fewer dimensions than the input space so that memory and computation costs can be significantly reduced. Small memory also results in small transmission costs for WiFi or across copper wires or optical cables.

Autoencoders can also be used for denoising, especially for restoring corrupted images, videos and musical recordings. The latent space can be fractured, so interpolating latent-space points  $\mathbf{z}$  can often produce unrealistic output images. The partial cure is to regularize the autoencoder, such as accomplished by dropout of the weights, adding noise to the system, constraining the size of the

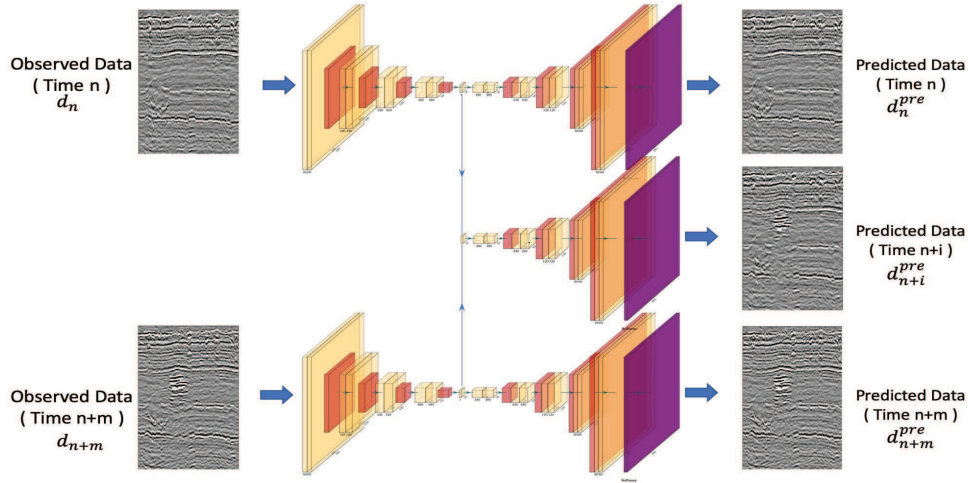


Figure 14.30: A schematic illustration of the interpolation network which contains two encoder networks and three decoder networks (Feng et al, 2021b).

Fréchet derivatives, sparsifying the weights of the AE, incorporating damping terms for the latent space, implementing a VAE that forces the latent space points to be fitted to a Gaussian, or using an adversarial network such as ACAI.

For seismic interpolation, the autoencoder can be an accurate interpolator for marine surveys and seismic monitoring of time-lapse data. Further testing is needed to define its limitations and benefits for a comprehensive set of corrupted data.

In the case of traveltimes tomography, a trained autoencoder does not necessarily require travel-time picking of first arrivals, and an enormous savings can be gained in eliminating manual picking of traveltimes. However, the decoder  $g(\mathbf{z})$  is not necessarily a continuous function of  $\mathbf{z}$  so this presents problems in convergence for a latent space dimension larger than one. Further work is needed to increase the robustness of this method.

## 14.6 Exercises

1. Derive the AE formulas for the objective function and the gradient where the regularization term is  $\|\mathbf{z}\|^2$ . Discuss the benefits of using this type of regularization if the data are overfitted. Compare this regularizer to those presented in Oring et al. (2020).
2. Derive the AE formulas for the objective function and the gradient where the regularization term is  $\sum_i (z_i - z_{i-1})^2$ . What is the physical meaning of this regularization term in terms of smoothness?
3. Derive the formula for the gradient of the AE objective function using a total variation regularization (Estrela et al., 2016) of the latent space vectors. When would this regularization be better than the penalty term  $\|\mathbf{z}\|^2$ ?
4. Discuss the relationship between an AE and an AE with skip connections in the form of a U-Net. Can the AE with skip connections be used to form a U-Net with skip connections?

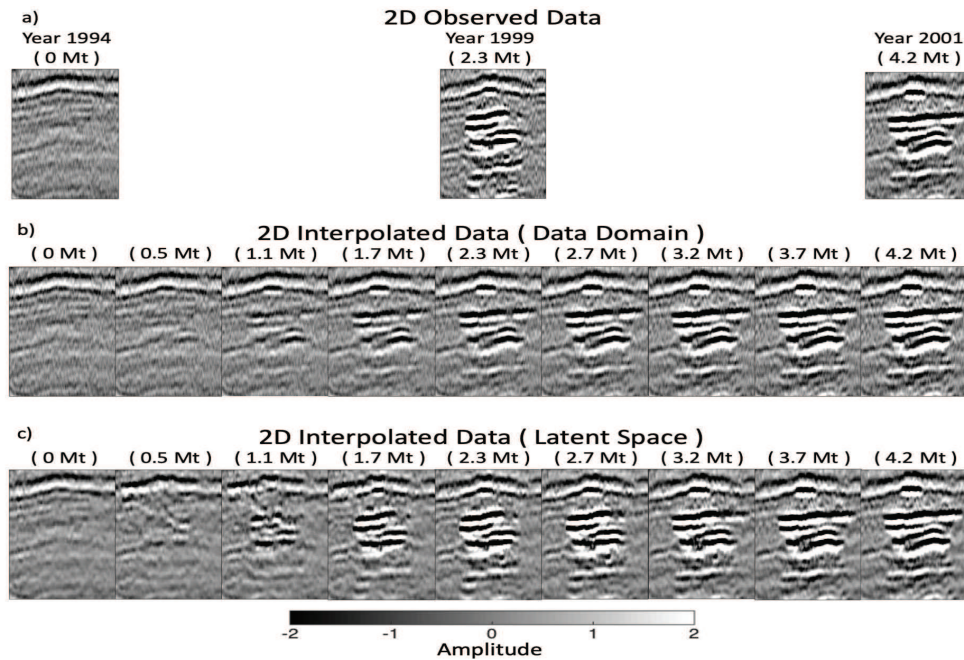


Figure 14.31: a) 2D observed data, interpolated data in the b) data domain and c) latent space with total CO2 injection volume from 0 to 4.2 Million tons (Feng et al, 2021b).

## 14.7 Appendix: Autoencoder in Keras

We now create an autoencoder code in Keras, where the logical flow and Keras code are following that of Ali Masri at [towardsdatascience.com](https://towardsdatascience.com).

We will create a Keras code for the following autoencoder architecture:

1. An input layer with  $28 * 28 = 784$  neurons.
2. A Hidden layer with 2 neurons.
3. An output layer with  $28 * 28 = 784$  neurons (same as the input layer).

The input data set will be the  $28 \times 28$  MNIST images (see images in Figure 14.10) and the goal will be to autoencode compress them to the encoded dimension of  $2 \times 1$ .

The Keras code is below.

```
% First, we import the dataset:

from keras.datasets import mnist
(data, labels), (_, _) = mnist.load_data()

% Need to reshape and rescale:

data = data.reshape(-1, 28*28) / 255.

% Time to define the network. We need three layers:
%   An input layer with size 28*28
```

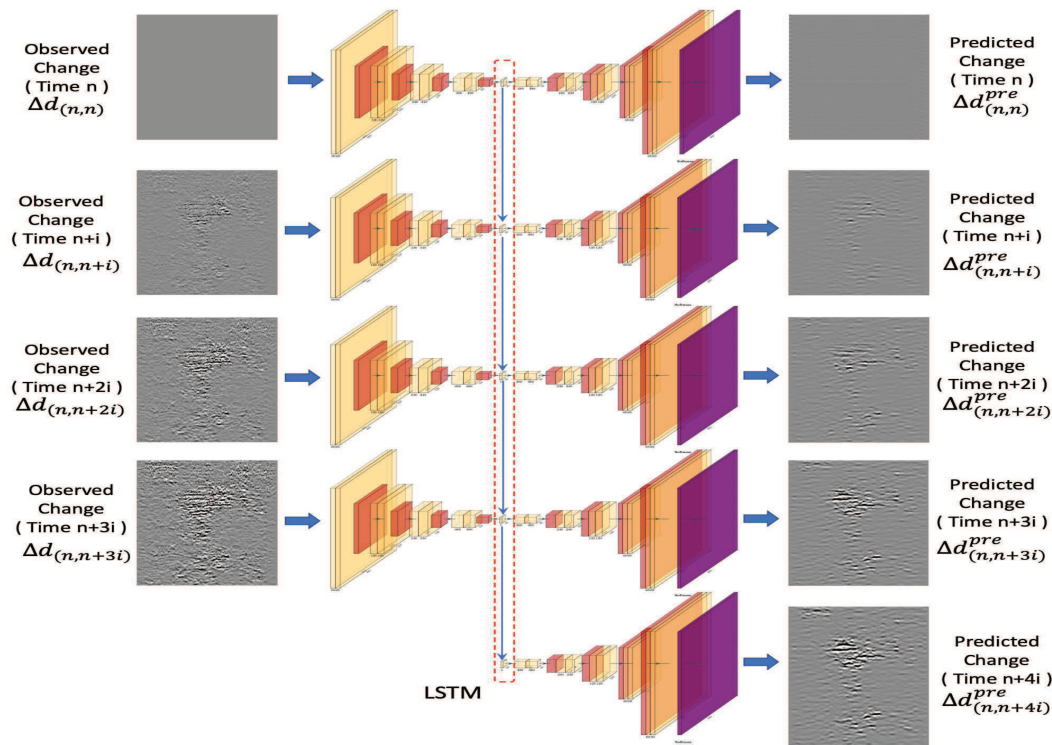


Figure 14.32: A schematic illustration of the extrapolation network which contains four encoder networks and five decoder networks (Feng et al, 2021b).

```
% A hidden layer with size 2
% An output layer with size 28*28

% from keras import models, layers

input_layer = layers.Input(shape=(28*28,))
encoding_layer = layers.Dense(2)(input_layer)
decoding_layer = layers.Dense(28*28)(encoding_layer)
autoencoder = models.Model(input_layer, decoding_layer)

% Compile and Train: Fit the model using a binary cross entropy loss
% between the pixel values:

autoencoder.compile('adam', loss='binary_crossentropy')
autoencoder.fit(x = data, y = data, epochs=5)
Did you notice the trick? X = data and y = data as well.
```

The results of the decoded picture are on the right side of Figure 14.6. After fitting the model, the network is supposed to learn how to calculate the hidden encodings. But we still have to extract the layer responsible for this. In the following, we define a new model where we remove the final layer since we do not need it anymore:



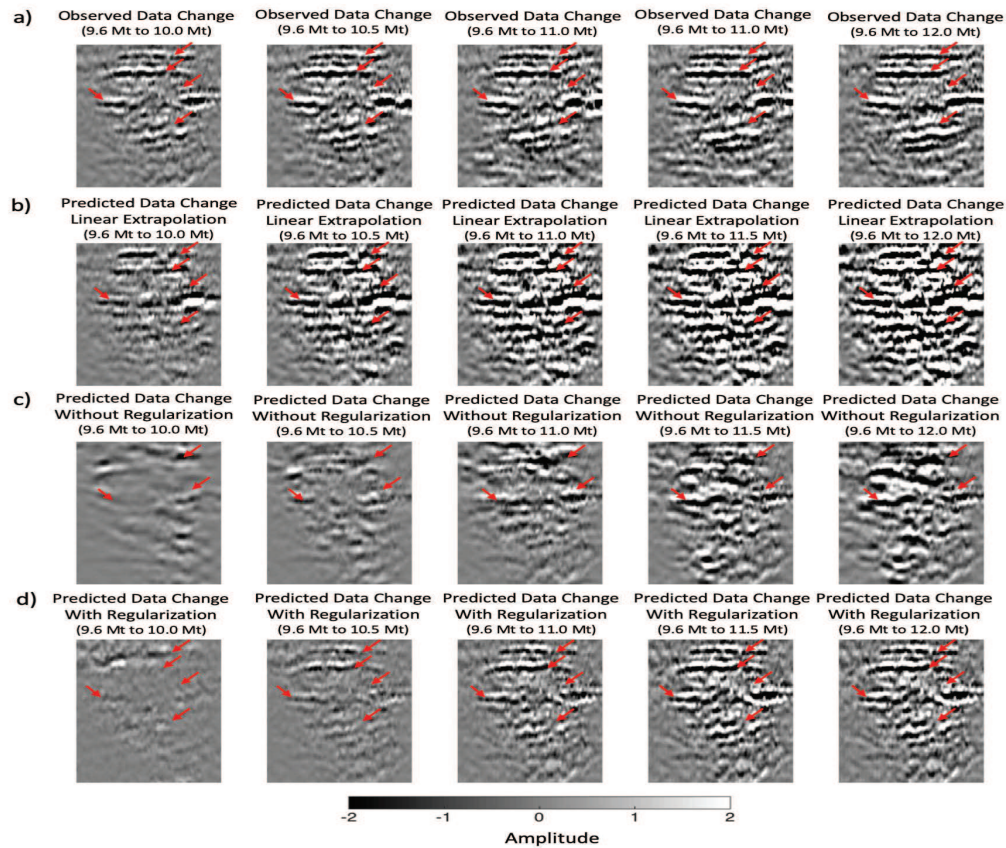


Figure 14.33: a) Observed data change, predicted data change b) by linear extrapolation, by extrapolation network c) without regularization and d) with regularization from 9.6 to 12.0 Mt (Feng et al, 2021b).

```
encoder = models.Model(input_layer, encoding_layer)

% Now instead of predicting the final output, we are predicting
% only the hidden representation using the following command:

encodings = encoder.predict(data)
```

The encoded variables are represented by an  $(n, m)$  array where  $n$  is the number of examples and  $m$  is the number of dimensions. The first column is the first feature and the second column is the second one. But what are those features? Actually, we do not know. We only know that they are well representatives of each input value. Figure 14.6 plots the encoded values for all of the examples, where they cluster into about 6 or 7 distinct colors. Interesting, since there are 10 numbers and a few of them are visually similar. Clearly it learned the different characteristics for each digit and how they are distributed in a 2D space.

## 14.8 Appendix: Dropout=Regularization

Consider a single-layer neural network as shown in Figure 14.34 with output a single scalar value denoted by the target variable  $t$ . For simplicity, we assume a linear activation function, with the understanding that the theory can be extended to a non-linear activation function. The input is the  $M \times 1$  vector  $\mathbf{x}$  and the weights are denoted by the  $M \times 1$  vector  $\mathbf{w}$ . Following Ranjan (2019c), the goal is to find the weights that optimally predict the given target scalar  $t$  in the least squares sense.

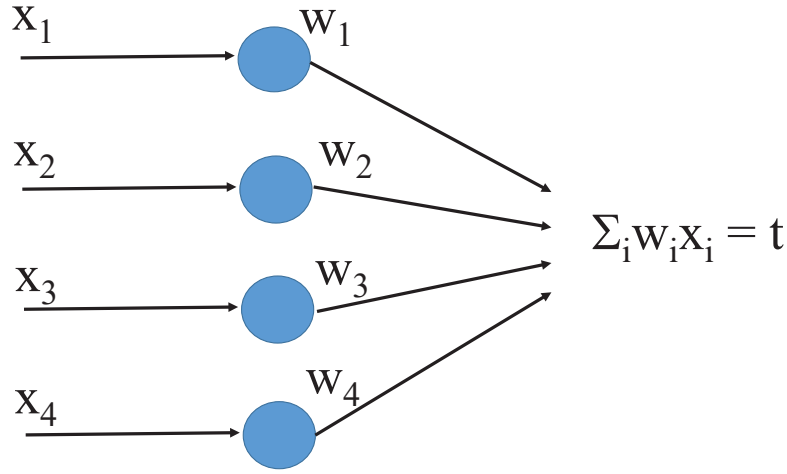


Figure 14.34: Single-layer linear network (Ranjan, 2019c).

For the regularized least squares problem the objective function is

$$\epsilon = \frac{1}{2} \left( \sum_{i=1}^M w_i x_i - t \right)^2 + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w}, \quad (14.13)$$

where  $\lambda > 0$  is the damping parameter. An extra summation could be added to the objective function to account for many input samples of  $\mathbf{x}$ , but the results we get for the single example in equation 14.13 are applicable to the case of a large training set.

The  $k^{th}$  gradient of the regularized objective function 14.13 is

$$\begin{aligned} \frac{\partial \epsilon}{\partial w_k} &= x_k \overbrace{\left( \sum_{i=1}^M w_i x_i - t \right)}^{\text{residual}} + \lambda w_k, \\ &= \sum_{i=1}^M (x_k x_i + \lambda \delta_{ik}) w_i - x_k t, \end{aligned} \quad (14.14)$$

where the residual  $\sum_{i=1}^M w_i x_i - t$  is the difference between the predicted  $\sum_{i=1}^M w_i x_i$  and actual  $t$  target values.

### 14.8.1 Least Squares Minimization with Dropout

We now define a random binary dropout variable  $f$  such that  $f = 1$  has the probability  $p$  and  $f = 0$  with probability  $(1 - p)$ . The composite discrete probability function is  $p^{(f)}(1 - p)^{(1-f)}$ . For the  $i^{th}$  data example we will assign a separate random variable as  $f_i$ . Therefore if we exclude the regularization term in equation 14.13 but now multiply each  $w_i$  by the dropout variable  $f_i$  we have

$$\epsilon = \frac{1}{2} \left( \sum_{i=1}^M f_i w_i x_i - t \right)^2. \quad (14.15)$$

The average of the  $k^{th}$  gradient of this regularized objective function is

$$\begin{aligned} \left\langle \frac{\partial \epsilon}{\partial w_k} \right\rangle &= \left\langle f_k x_k \left( \sum_{i=1}^M f_i w_i x_i - t \right) \right\rangle, \\ &= x_k \sum_{i=1}^M \left\langle f_k f_i \right\rangle w_i x_i - x_k t \overbrace{\left\langle f_k \right\rangle}^{\bar{f}}, \\ &= \left\langle f_k^2 \right\rangle x_k^2 w_k + \left[ \sum_{i=1, i \neq k}^M \left\langle f_k f_i \right\rangle x_k x_i w_i \right] - x_k t \bar{f}, \end{aligned} \quad (14.16)$$

where  $\langle \rangle$  indicates averaging over the jointly distributed dropout variables  $f_1, f_2, \dots, f_M$  and  $\bar{f} = \langle f_k \rangle$ . Recall that if  $f_k$  and  $f_i$  are independent variables then

$$\langle f_i f_k \rangle = \langle f_i \rangle \langle f_k \rangle = \bar{f}^2. \quad (14.17)$$

It is also known that the variance  $\sigma^2$  of the random variable  $f_k$  is given by

$$\sigma^2 = \langle (f_k - \bar{f})^2 \rangle = \langle f_k^2 \rangle - \bar{f}^2 \rightarrow \langle f_k^2 \rangle = \sigma^2 + \bar{f}^2. \quad (14.18)$$

Substituting equations 14.17 and 14.18 into equation 14.16 gives

$$\begin{aligned} \left\langle \frac{\partial \epsilon}{\partial w_k} \right\rangle &= (\bar{f}^2 + \sigma^2) x_k^2 w_k + \bar{f}^2 \left[ \sum_{i=1, i \neq k}^M x_k x_i w_i \right] - x_k t \bar{f}, \\ &= \sigma^2 x_k^2 w_k + \bar{f}^2 \sum_{i=1}^M x_k x_i w_i - x_k t \bar{f}, \\ &= \bar{f}^2 \sum_{i=1}^M (x_k x_i + \lambda \delta_{ik}) w_i - x_k t \bar{f}, \end{aligned} \quad (14.19)$$

where  $\lambda = \sigma^2 / \bar{f}^2$ . Thus, this averaged dropout gradient is the same as the regularized objective function

$$\epsilon' = \frac{1}{2} \left( \sum_{i=1}^M x_i \bar{f} w_i - t \right)^2 + \frac{\lambda}{2} \bar{f}^2 \mathbf{w}^T \mathbf{w}, \quad (14.20)$$

seen in equation 14.13, except the  $\mathbf{w} \rightarrow \bar{f} \mathbf{w}$ .



## 14.9 Appendix: KL Regularization

Mapping similar input images to a small region of z-space should encourage a small average value  $\hat{\rho}_j$  of the encoder's  $j^{th}$  activation function:

$$\hat{\rho}_j = \frac{1}{M} \sum_{m=1}^M f_j^{(m)}, \quad (14.21)$$

where  $M$  is the number of input examples. We would like this average value to be a reasonably small number such as  $\rho = 0.05$ , so we could incorporate a penalty term (<http://deeplearning.stanford.edu/tutorial/>) such as  $\sum_{j=1}^J (\hat{\rho}_j - \rho)^2$ . There are other choices of penalty functions, but one choice favored by some statisticians is

$$\sum_{j=1}^J \text{KL}(\rho || \hat{\rho}_j) = \sum_{j=1}^J \rho \log \frac{\rho}{\hat{\rho}_j} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_j}, \quad (14.22)$$

which is the Kullback-Leibler (KL) divergence between a Bernoulli random variable with mean  $\rho$  and a Bernoulli random variable with mean  $\hat{\rho}_j$ . This is a standard metric for measuring the difference between two probability distributions. As illustrated in Figure 14.35, the minimum is reached when the network is trained so that  $\hat{\rho}_j = \rho = 0.2$ . See Appendix 20.11 for more details about KL regularization.

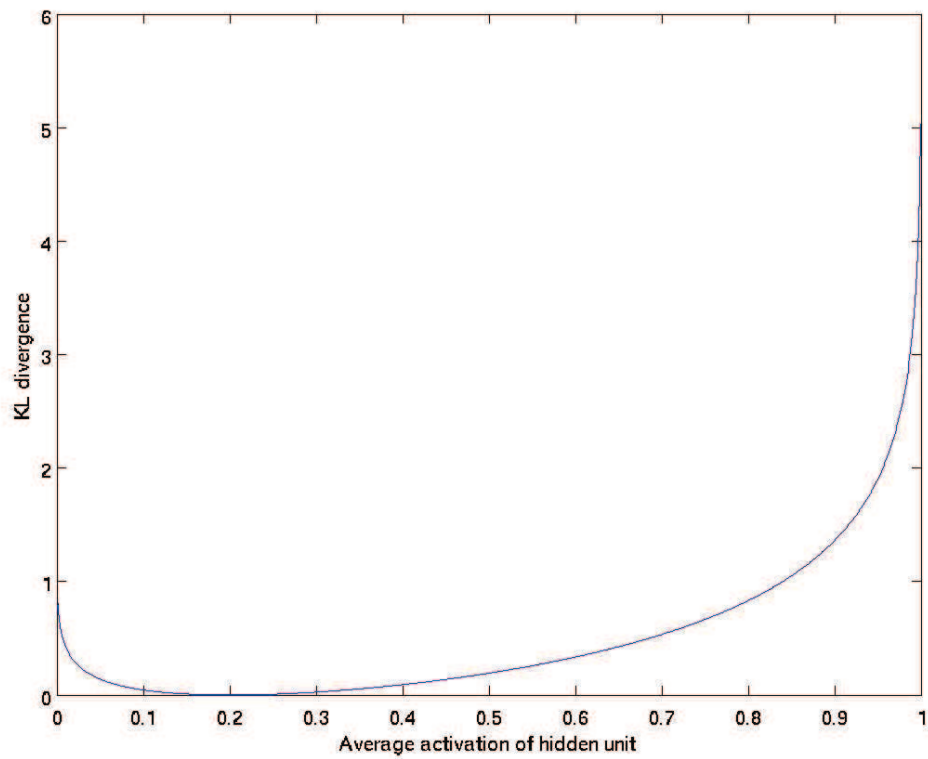


Figure 14.35: KL penalty function plotted against  $\hat{\rho}$  for  $\rho = 0.2$ . Illustration from <http://deeplearning.stanford.edu/tutorial/>.

## Chapter 15

# Convolutional Sparse Coding

According to Papayan et al. (2017b), the NN operation of the feature map convolved with a layer's filter can be interpreted as a weighted summation of basis functions. Here, the basis functions are the filters and the values in the feature maps are interpreted as the coefficients. Therefore, a signal polluted by noise, both coherent and random, can be filtered by decomposing the input into desirable and undesirable basis functions. The undesirable ones can be eliminated by muting and the input can be approximated by the weighted sum of desirable basis functions, which is the signal. This process, denoted as denoising by convolutional sparse-coding (CSC), is now described.

### 15.1 Introduction

Seismic data may suffer from different sources of noise during field acquisition, and this noise generally falls into two categories: (1) coherent noise such as ground roll and multiples; (2) incoherent (random) noise generated by activities in the environment such as the acquisition truck, drill rig, seismic vessels, wind and animal movements. Seismic noise attenuation is one of the critical components of seismic data processing. This type of noise, either coherent or random, can be attenuated by a convolutional type of denoising method which we denoted as CSC denoising.

Traditional seismic denoising methods exploit the different characteristics of the noise and the seismic data in a transform domain. The transform domain is selected so that the signal and noise have different characteristics. For example, the band-pass,  $f - k$ , and  $k_x - k_y$  filtering method will separate the undesirable noise from the signal in the Fourier domain (Yilmaz, 2001). Some other examples include the following.

- Chanerley and Alexander (2002) use the stationary wavelet transform as an alternative to band-pass filtering for denoising.
- The curvelet transform is used by (Hennenfent and Herrmann, 2006) to attenuate both random and coherent noise in seismic data.
- Ibrahim and Sacchi (2014) use the hyperbolic Radon transform for deblending seismic data obtained from a survey with blended sources.
- The seislet transform is used for seismic denoising by Chen (2016).
- Other seismic random noise attenuation methods include the nonlocal means method (Bonar and Sacchi, 2012) and the sparsity and low-rank regularization method (Li et al., 2020).
- Coherent noise attenuation methods include prediction filters (Guitton, 2002), empirical mode decomposition (Bekara and Van der Baan, 2009), and nonstationary polynomial fitting (Liu et al., 2011).

The transform methods mentioned above are typically model-driven processes based on a specified mathematical model of the data. The performance may not be satisfactory due to the inability to adapt to changing data structures (Zhu et al., 2015). Alternatively, the sparse dictionary learning method is a data-driven process, which learns its dictionary so that complicated data can be represented by a sparse set of weighted basis functions. Kaplan et al. (2009) use the sparse-coding algorithm to attenuate both coherent and incoherent noise. Beckouche and Ma (2014) evaluate a dictionary learning method for seismic-data denoising. Zhu et al. (2015) develop a seismic data denoising method based on a parametric dictionary learning scheme. Chen et al. (2016) combine learning based dictionaries and fixed-basis transforms and propose a double-sparsity dictionary to better handle the special features of seismic data. Li et al. (2018) proposed a novel method based on unsupervised learning to recover the weak residual signal from seismic data. Their synthetic and real data examples indicate that this method is capable of recovering signal even if it is orders of magnitude weaker than the surrounding noise.

Most of the sparse coding (SC) denoising methods partition the seismic data into overlapping patches, and process each patch separately. These methods, however, ignore the consistency of structures in overlapping patches. That is, the learned features often contain shifted versions of the same features so that latent structures of the underlying signal may be lost when dividing it into small patches (Bristow et al., 2013; Heide et al., 2015). Liu et al. (2020) propose a convolutional sparse coding (CSC) method based on the seismic denoising method to address the consistency issue. As opposed to SC, CSC operates on whole seismic images, thereby seamlessly capturing the correlation of features in nearby neighborhoods.

The CSC method can learn a set of translation invariant dictionaries for a large variety of different training signals (Xiong et al., 2019). It has been demonstrated effective for solving problems in neural and brain information processing (Jas et al., 2017; Peter et al., 2017), and in different image processing tasks, such as image inpainting (Heide et al., 2015), super-resolution (Gu et al., 2015), high dynamic range imaging (Serrano et al., 2016), and high-dimensional signal reconstructions (Choudhury et al., 2017). CSC is shown to be closely connected to convolutional neural networks (CNN) by Pappayan et al. (2017a,b) and has been applied to the seismic least-square migration problem (Liu and Schuster, 2020; Liu et al., 2020). Al-Madani et al. (2019) use CSC for seismic random noise attenuation and interpolation. They didn't discuss how to use CSC to attenuate the coherent noise, and the efficacy of this method is validated with synthetic data. However, further studies are needed to assess its capabilities.

The CSC problem can be solved by using the Alternating Direction Method of Multipliers (ADMM) (Boyd et al., 2011) in the Fourier domain (Wohlberg, 2016; Heide et al., 2015) or in the space domain (Pappayan et al., 2017a,b; Zisselman et al., 2019) which alternate between finding the dictionary (dictionary learning problem) and then finding the sparse coefficients (sparse pursuit problem).

The next sections will apply the fast and flexible CSC method proposed by Heide et al. (2015) to seismic denoising. The theory of the fast and flexible CSC method is proposed along with its workflow for noise attenuation by CSC. Then the CSC denoising method is applied to synthetic data and field records. Numerical results indicate that the proposed method achieves good denoising performance.

## 15.2 Theory

The convolutional sparse coding problem can be defined as finding the optimal  $\mathbf{d}$  and  $\mathbf{z}$  that minimize the following objective function (Heide et al., 2015):

$$\arg \min_{\mathbf{d}, \mathbf{z}} \frac{1}{2} \|\mathbf{x} - \mathbf{M} \sum_{k=1}^K \mathbf{d}_k * \mathbf{z}_k\|_2^2 + \beta \sum_{k=1}^K \|\mathbf{z}_k\|_1 + \sum_{k=1}^K \text{ind}_C(\mathbf{d}_k), \quad (15.1)$$

where  $\mathbf{x}$  is an  $m \times n$  image in vector form,  $\mathbf{d}_k$  refers to the  $k$ -th  $d \times d$  filter in vector form,  $\mathbf{z}_k$  is a vector of sparse coefficients with size  $(m + d - 1) \times (n + d - 1)$ ,  $\beta$  controls the  $l_1$  penalty, and  $*$  denotes the 2D convolution operator.  $\mathbf{M}$  is a binary diagonal matrix that masks out the boundaries of the padded estimation  $\sum_{k=1}^K \mathbf{d}_k * \mathbf{z}_k$ . The term  $\text{ind}_C(\cdot)$  is an indicator function:

$$\text{ind}_C(\mathbf{d}) = \begin{cases} 0 & \mathbf{d} \in C \\ 1 & \mathbf{d} \notin C, \end{cases} \quad (15.2)$$

which is defined on the convex set of the constraints  $C = \{\mathbf{d} \mid \|\mathbf{d}\|_2^2 \leq 1\}$ . Equation 15.1 can be expressed as the following sum of functions

$$\arg \min_{\mathbf{d}, \mathbf{z}} f_1(\mathbf{D}\mathbf{z}) + \sum_{k=1}^K (f_2(\mathbf{z}_k) + f_3(\mathbf{d}_k)), \quad (15.3)$$

where,

$$f_1(\mathbf{v}) = \frac{1}{2} \|\mathbf{x} - \mathbf{M}\mathbf{v}\|_2^2, \quad f_2(\mathbf{v}) = \beta \|\mathbf{z}_k\|_1, \quad f_3(\mathbf{v}) = \text{ind}_C(\mathbf{v}). \quad (15.4)$$

Here,  $\mathbf{z} = [z_1^T, \dots, z_K^T]^T$  is the coefficient matrix and  $\mathbf{D} = [D_1, \dots, D_K]$  is a concatenation of Toeplitz matrices, each one representing a convolution with respect to the filter  $\mathbf{d}_k$ . Equation 15.3 is a sum of functions  $f_i$ , which are simple to optimize individually. However, computing their sum is challenging. Following Heide et al. (2015), equation 15.3 is a bi-convex problem for  $\mathbf{z}$  (or  $\mathbf{d}$ ) when  $\mathbf{d}$  (or  $\mathbf{z}$ ) is fixed. So, we can use the alternating coordinate descent method to solve it. First, compute the filter update:

$$\arg \min_{\mathbf{d}} f_1(\mathbf{D}\mathbf{z}) + \sum_{k=1}^K f_3(\mathbf{d}_k). \quad (15.5)$$

Then compute the coefficient update:

$$\arg \min_{\mathbf{z}} f_1(\mathbf{D}\mathbf{z}) + \sum_{k=1}^K f_2(\mathbf{z}_k). \quad (15.6)$$

Repeat the above two steps until there is no more progress in both directions.

### 15.2.1 Generalization of the Objective Function

The above two subproblems have the same format and so we can define their generalized form as

$$f(\mathbf{K}\mathbf{u}) = \sum_{i=1}^I f_i(\mathbf{K}_i\mathbf{u}), \quad (15.7)$$

where  $\mathbf{K} = [\mathbf{K}_1, \mathbf{K}_2, \dots, \mathbf{K}_I]^T$ .  $\mathbf{K}_i$  are arbitrary matrices, and  $I$  is the number of functions. Here,  $\mathbf{u}$  can be  $\mathbf{d}$  or  $\mathbf{z}$ . For example, if  $\mathbf{K}_1 = \mathbf{D}$ ,  $\mathbf{K}_2\mathbf{u} = \mathbf{z}_1 \dots$ , the problem for the coefficient update can be written as:

$$\arg \min_{\mathbf{u}} f(\mathbf{K}\mathbf{u}) = \arg \min_{\mathbf{z}} f_1(\mathbf{D}\mathbf{z}) + \sum_{k=1}^K f_2(\mathbf{z}_k). \quad (15.8)$$

Heide et al. (2015) use the alternating direction method of multipliers (ADMM) (Boyd et al., 2011) to solve for  $\mathbf{u}$  in equation 15.8. We will see that the resulting minimization by ADMM becomes separable for all the  $f_i$ . For example, the following problem is the same as the problem in equation 15.8:

$$\arg \min_{\mathbf{u}} h(\mathbf{u}) + f(\mathbf{y}) \quad \text{subject to} \quad \mathbf{K}\mathbf{u} = \mathbf{y}, \quad (15.9)$$

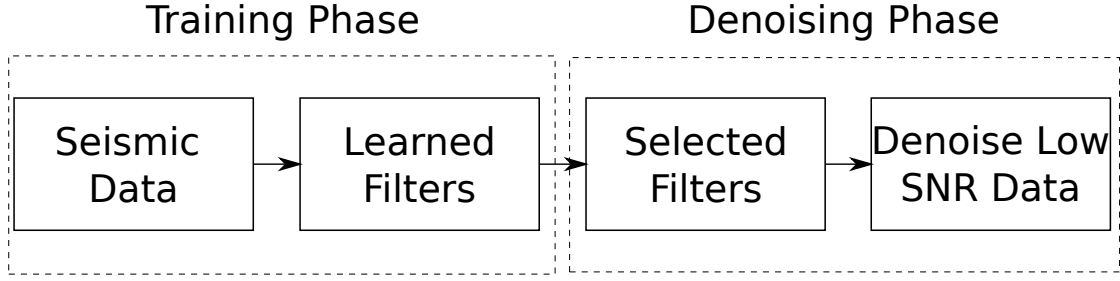


Figure 15.1: Workflow of CSC for noise attenuation.

where  $h(\mathbf{u}) = 0$ . Its augmented Lagrangian can be written as:

$$L_\rho(\mathbf{u}, \mathbf{y}, \boldsymbol{\alpha}) = h(\mathbf{u}) + f(\mathbf{y}) + \boldsymbol{\alpha}^T (\mathbf{K}\mathbf{u} - \mathbf{y}) + (\rho/2) \|\mathbf{K}\mathbf{u} - \mathbf{y}\|_2^2.$$

The scaled form of the augmented Lagrangian is:

$$L_\rho(\mathbf{u}, \mathbf{y}, \boldsymbol{\alpha}) = h(\mathbf{u}) + f(\mathbf{y}) + \frac{\rho}{2} \|\mathbf{K}\mathbf{u} - \mathbf{y} + \boldsymbol{\lambda}\|_2^2, \quad (15.10)$$

where  $\boldsymbol{\lambda} = \frac{1}{\rho} \boldsymbol{\alpha}$ . The ADMM of equation 15.10 is described by the following three steps.

- **u update:**

$$\begin{aligned} \mathbf{u}^{k+1} &= \arg \min_{\mathbf{u}} L_\rho(\mathbf{u}^k, \mathbf{y}^k, \boldsymbol{\lambda}^k), \\ &= \arg \min_{\mathbf{u}} \|\mathbf{K}\mathbf{u}^k - \mathbf{y}^k + \boldsymbol{\lambda}^k\|_2^2. \end{aligned} \quad (15.11)$$

- **y update:**

$$\begin{aligned} \mathbf{y}^{k+1} &= \arg \min_{\mathbf{y}} L_\rho(\mathbf{u}^{k+1}, \mathbf{y}^k, \boldsymbol{\lambda}^k), \\ &= \arg \min_{\mathbf{y}} \rho \left\{ f(\mathbf{y}^k) / \rho + \frac{1}{2} \|\mathbf{K}\mathbf{u}^{k+1} - \mathbf{y}^k + \boldsymbol{\lambda}^k\|_2^2 \right\}. \end{aligned} \quad (15.12)$$

- **error update:**

$$\begin{aligned} \boldsymbol{\lambda}^{k+1} &= \arg \max_{\boldsymbol{\lambda}} L_\rho(\mathbf{u}^{k+1}, \mathbf{y}^{k+1}, \boldsymbol{\lambda}^k), \\ &= \boldsymbol{\lambda}^k + (\mathbf{K}\mathbf{u}^{k+1} - \mathbf{y}^{k+1}). \end{aligned} \quad (15.13)$$

Because  $f$  is a sum of  $f_i$ , equation 15.12 can be separated as:

$$\begin{aligned} \mathbf{y}_i^{k+1} &= \arg \min_{\mathbf{y}_i} f_i(\mathbf{y}_i^k) / \rho + \frac{1}{2} \|\mathbf{K}_i \mathbf{u}_i^{k+1} - \mathbf{y}_i^k + \boldsymbol{\lambda}_i^k\|_2^2, \\ &\text{for all } i \in \{1, \dots, I\}. \end{aligned} \quad (15.14)$$

Appendix 15.7 shows how to solve the problems defined by equations 15.11 and 15.14.

### 15.2.2 Workflow

The workflow of CSC for noise attenuation is shown in Figure 15.1. It includes the following two steps:

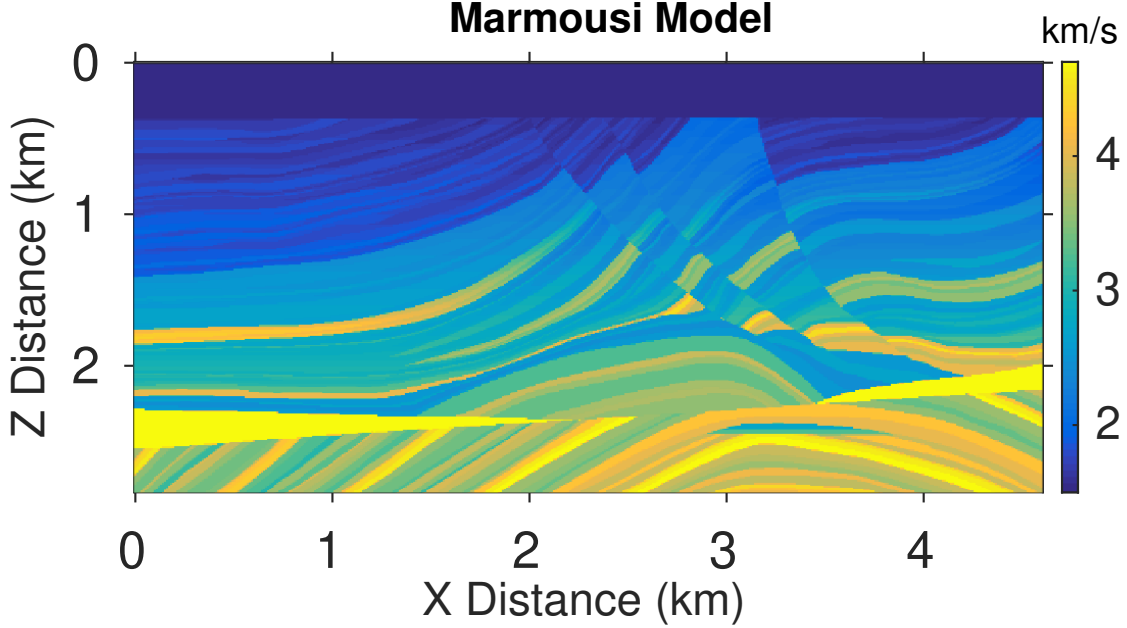


Figure 15.2: Marmousi velocity model used for computing the synthetic data.

- Training phase: solve for the filter  $\mathbf{d}_k$  and the coefficient vector  $\mathbf{z}_k$ . The training data are chosen with a relatively high signal-to-noise ratio (SNR). We should remark that training on high-SNR data is not necessary because the CSC process has a noise rejection capability like the traditional SC (Elad and Aharon, 2006). However, training on high-SNR data will have better denoising performance than training on the noisy data because it will learn cleaner filters. The filter size should be larger than one wavelength.
- Denoising phase: solve for the coefficient vector  $\mathbf{z}_k$  with the knowledge of the learned vector  $\mathbf{d}_k$ . Change the value of the coefficient  $\beta$  for the  $l_1$  penalty function to control the degree of noise removal. For coherent noise, remove some of the learned filters that are more indicative of noise than signal, and the remaining filters are used for denoising.

## 15.3 Numerical Examples

The performance of CSC for denoising seismic data is demonstrated with two examples: (1) synthetic data from the Marmousi model and (2) field data from Saudi Arabia. The synthetic data example tests the performance of CSC in removing random noise in seismic data, and the field data example demonstrates the removal of coherent noise. The SNR used in our synthetic test is defined as follows:

$$\text{SNR} = 10 \log_{10} \left( \frac{\|X_{\text{signal}}\|_2^2}{\|X_{\text{noise}}\|_2^2} \right) \quad (15.15)$$

where  $X_{\text{signal}}$  and  $X_{\text{noise}}$  denote the signal and noise, respectively.

### 15.3.1 Synthetic Test

The first example is a 2-D synthetic dataset calculated from the Marmousi model, as shown in Figure 15.2. There are 116 sources with a 40 m interval and 231 receivers with a 20 m interval

located along the free surface at the top of the model. The source wavelet is a Ricker wavelet with the center frequency of 15 Hz. The recording time is 4s with a time sampling of 1 ms. For each common shot gather (CSG), we extract part of data for testing, where the time interval is from 1.6s to 2.5s, and the maximum source-receiver offset is 2 km. For example, the extracted data from the second CSG are displayed in Figure 15.3a, and the dimension of each input panel is  $181 \times 100$ .

We first compare the denoising performance of the CSC method versus the traditional SC method based on the K-SVD algorithm proposed by Elad and Aharon (2006). We use the noisy data with a SNR of 4.27 (Figure 15.3b) to train the dictionaries. We use 32  $11 \times 11$  filters for CSC, which are initialized with zero-mean random numbers between -1 and 1. For the patch-based traditional SC method, we first divide the noisy data set into overlapping patches each with size  $11 \times 11$  and the stride size is 1. The total number of the patches is  $171 \times 90$ , and we randomly select 10,000 among them as a training set. The number of the atoms (filters) is set to the size of the filters, which make a dictionary of 121 atoms.

A factor of redundancy of two is also a popular choice (Beckouche and Ma, 2014), but here we set the factor of redundancy as one. The atoms are initialized randomly, and the atom sparsity level is set to 10.

Figure 15.4a and 15.4b show the learned filters by the convolutional and traditional SC methods. If we compare the filters marked by the red rectangles in Figure 15.4, we see that the traditional SC method learns shifted versions of the same features. The filters are learned from the red area of the data shown in Figure 15.3a. The denoised results by the convolutional and traditional SC methods are shown in Figure 15.3c and 15.3d, respectively. The SNRs of the denoised data by the convolutional and traditional SC methods have increased to 14.13 and 13.95, respectively. From the comparison, we see that CSC denoised results may have better SNR than provided by the traditional sparse coding method.

We then test the performance of CSC denoising by training the filters from a clean data set. Five CSGs with a high SNR are selected for training. There are 50 learned filters (see Figure 15.6) and the filter size is  $55 \times 55$ . Assume that the second CSG is contaminated with some noise (see Figure 15.5b) and its SNR is -1.76. The denoised CSG is shown in Figure 15.5c and its SNR increases to 14.5. We can change the coefficient  $\beta$  of the  $l_1$  penalty function to control the noise level. Figures 15.7a and 15.7b show the denoised results by setting  $\beta$  to 0.2 and 5, respectively. We observe that more noise is removed with larger values of  $\beta$ . But it may also hurt some useful signals, which can be seen in the comparison of the residuals in Figures 15.7c and 15.7d.

### 15.3.2 Field Data Test

Seismic data are recorded over the Qadimah fault system (Li et al., 2019; Liu and Lu, 2018), approximately 30 km north of the KAUST campus, Saudi Arabia. There are 228 sources with a source spacing of 10 m. Each CSG has 456 receivers with a receiver spacing of 5 m. The dataset is first filtered by a bandpass filter with a frequency range from 10 Hz to 60 Hz. Ten CSGs are chosen for training, and one of the CSGs is shown in Figure 15.8a. Its offset range is from 0.2 km to 1.3 km. The surface waves are muted out in the training set. There are 30  $21 \times 21$  learned filters displayed in Figure 15.8b. Some features of the coherent noise are learned from the training data set, as indicated by the red boxes in Figure 15.8b.

We select the first CSG (see Figure 15.9a) for denoising, where the offset range is from 0.8 km to 1.9 km.  $\beta$  is set to 0.8 during the denoising phase. Using all the learned filters in Figure 15.8b for denoising gives the denoised results displayed in Figure 15.9b. Its corresponding residual is shown in Figure 15.10a. There is still coherent noise in the area indicated by the red box in Figure 15.9b. The reason is that the learned filters include features with coherent noise.

Next we exclude the filters with noise features indicated by the red boxes in Figure 15.8b. Then we apply the remaining filters for denoising. The denoised result and its residuals are shown in Figures 15.9c and 15.10b, respectively. We see that the noise level within the red box of Figure 15.9b



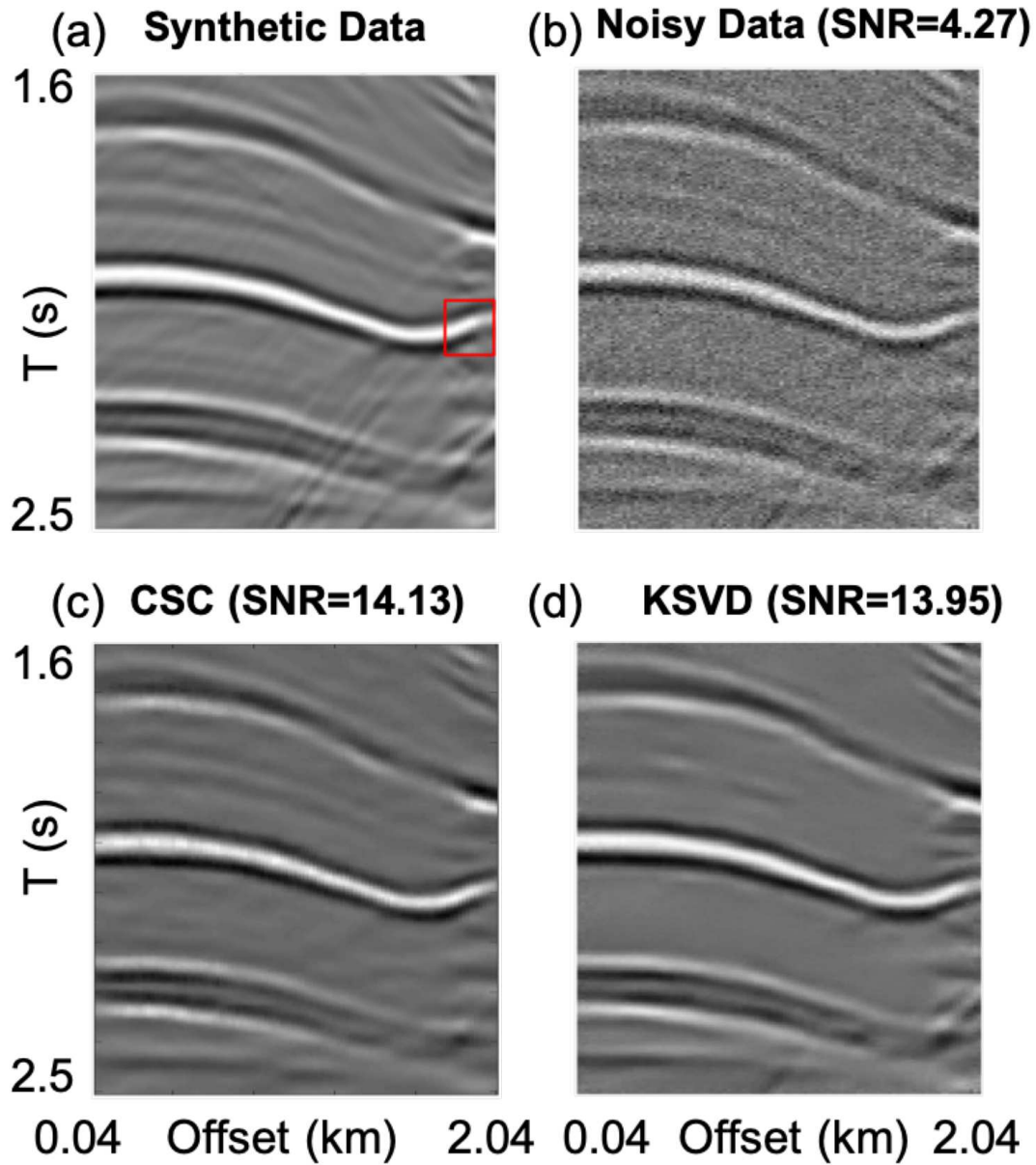


Figure 15.3: a) Synthetic data; b) noisy data; the denoised results by the c) CSC and d) traditional SC methods based on the K-SVD algorithm. The red rectangle in a) is selected for comparison in Figure 15.4.

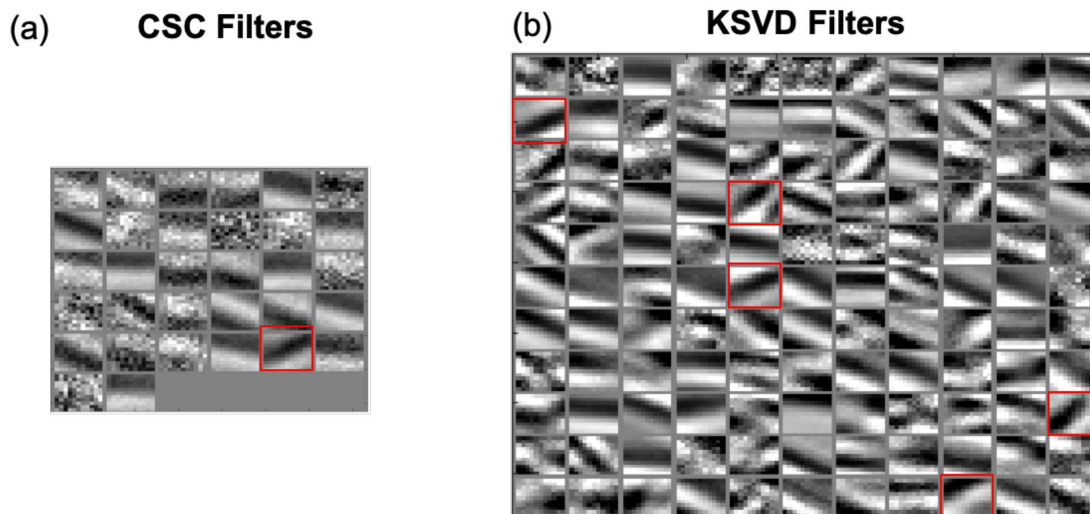


Figure 15.4: Comparison of the learned filters from the noisy data by a) CSC and (b) traditional SC based on the K-SVD method. Here, the red rectangles show the filters with similar features.

is reduced by only using the selected filters. We continue to exclude more filters as indicated by the yellow boxes in Figure 15.8b during the denoising phase. The denoised result is shown in Figure 15.9d and its residual is shown in Figure 15.10c. From the residuals in Figure 15.10, we see that more coherent noise is removed.

Next, we choose the near-offset traces with offsets ranging from 0.05 km to 0.35 km (see Figure 15.11a) that are contaminated with surface waves to perform the test. We use all the filters shown in Figure 15.8b for denoising. The denoised data and the residuals are shown in Figures 15.11b and 15.11c, respectively. The learned filters do not include the features from the surface waves, so the surface waves are removed.

## 15.4 Discussion

In the synthetic data example, for simplicity, we test our denoising algorithm on only a part of the CSG. The application to all of the CSGs is straightforward. From the synthetic results, we see that for CSC denoising there are two training options: using the noisy data set itself, or training on a similar but noiseless data set. By the comparison of the learned filters from the convolutional and traditional SC methods, we find that the filters from the traditional SC method are highly redundant because they have to capture all shifted copies of the filters.

The computational cost of CSC is discussed in Papyan et al. (2017). Papyan et al. (2017) show that the dictionary learning step in CSC can be solved in the spatial domain by K-SVD instead of ADMM in the Fourier domain proposed by Heide et al. (2015), the method used in this paper. They conclude that the K-SVD-based method scales linearly with the global dimension, while the method used in this paper grows as  $N \log(N)$  where  $N$  is the dimension of the signal  $\mathbf{x}$ . So, when the dimension of data is larger, the KSVD-based method should be used with the CSC algorithm (Papyan et al., 2017).

The CSC-based denoising method is an unsupervised learning process that differs from supervised learning processes such as CNN-based denoising suggested by Yu et al. (2019). It does not heavily depend on the availability of a huge amount of training data. However, it may need human

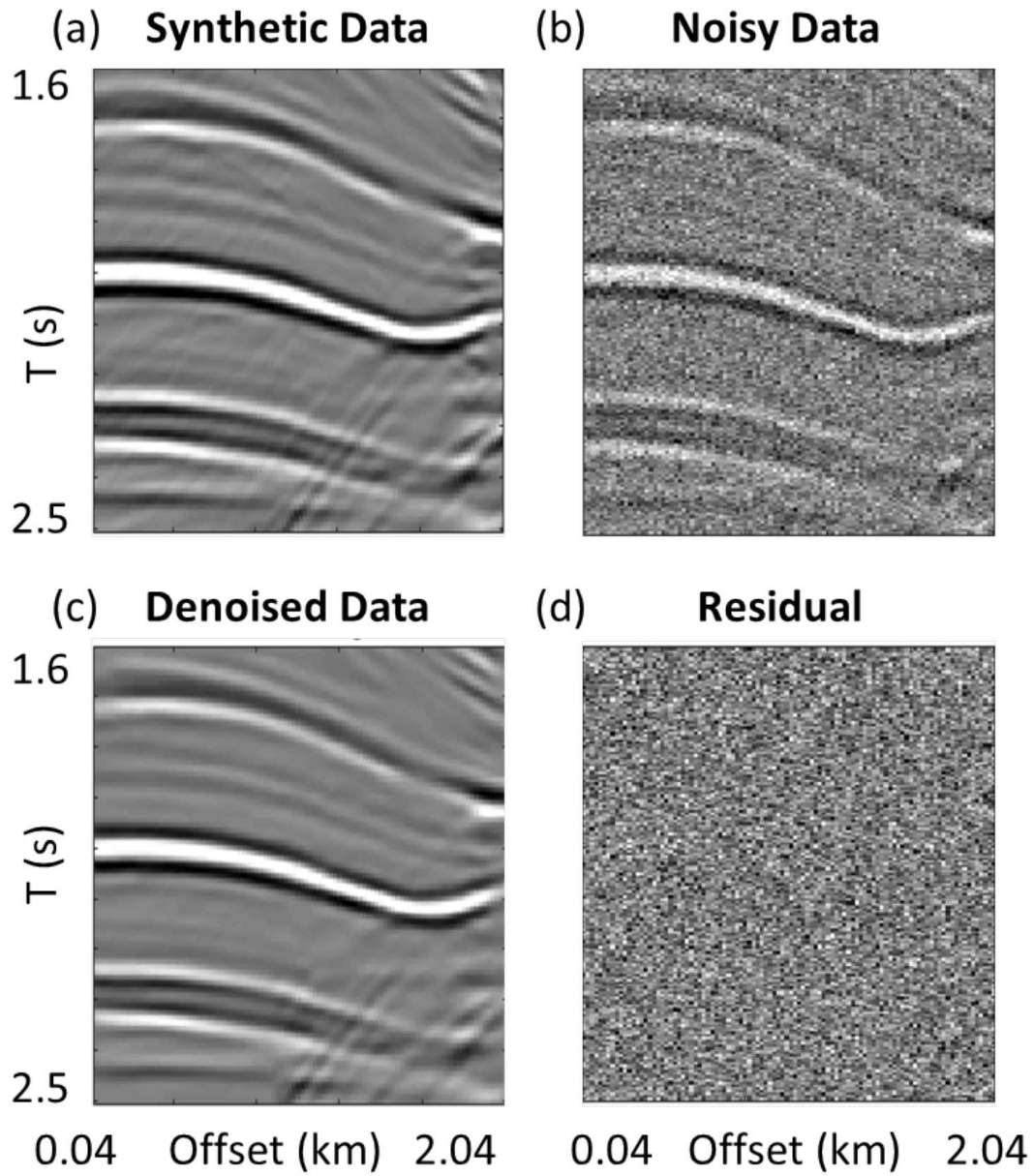


Figure 15.5: a) Synthetic data from Marmousi model and b) noisy data with the SNR of -1.76; c) the denoised data with the SNR of 14.5 and d) the residual between b) and c).

## Learned Features

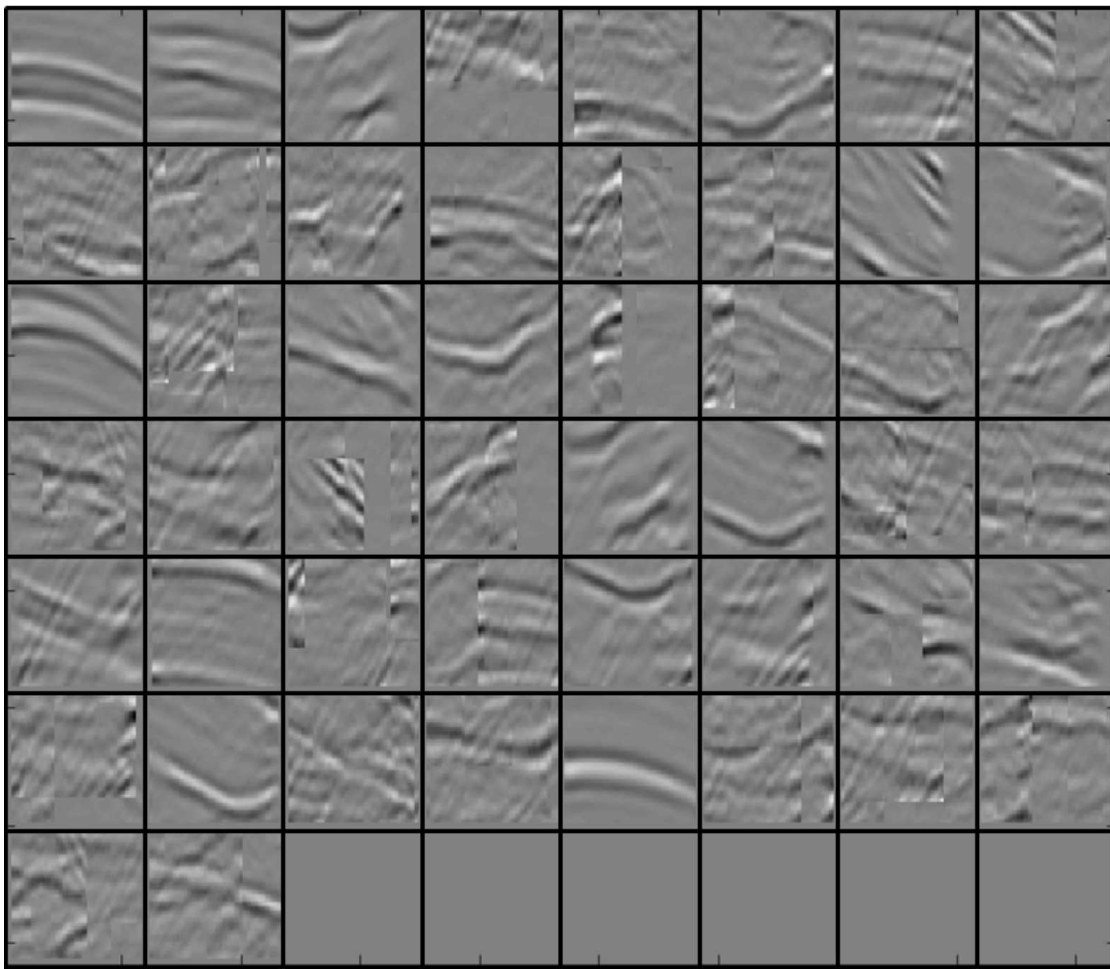


Figure 15.6: Fifty  $55 \times 55$  filters learned by CSC from five CSGs with a high SNR.

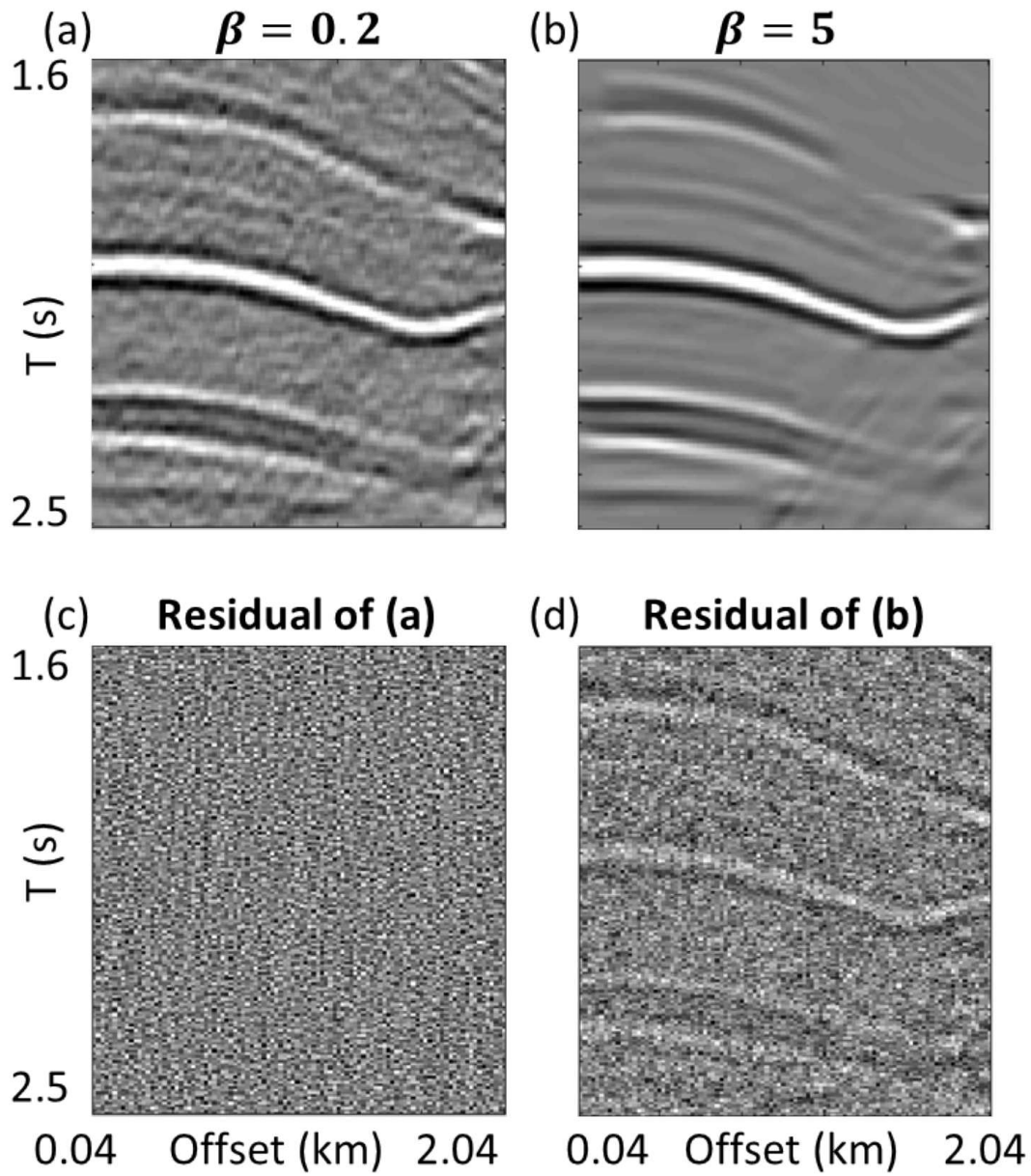


Figure 15.7: Denoised data by setting  $\beta =$  a) 0.2 and b) 5 and their corresponding residuals in c) and d), respectively.

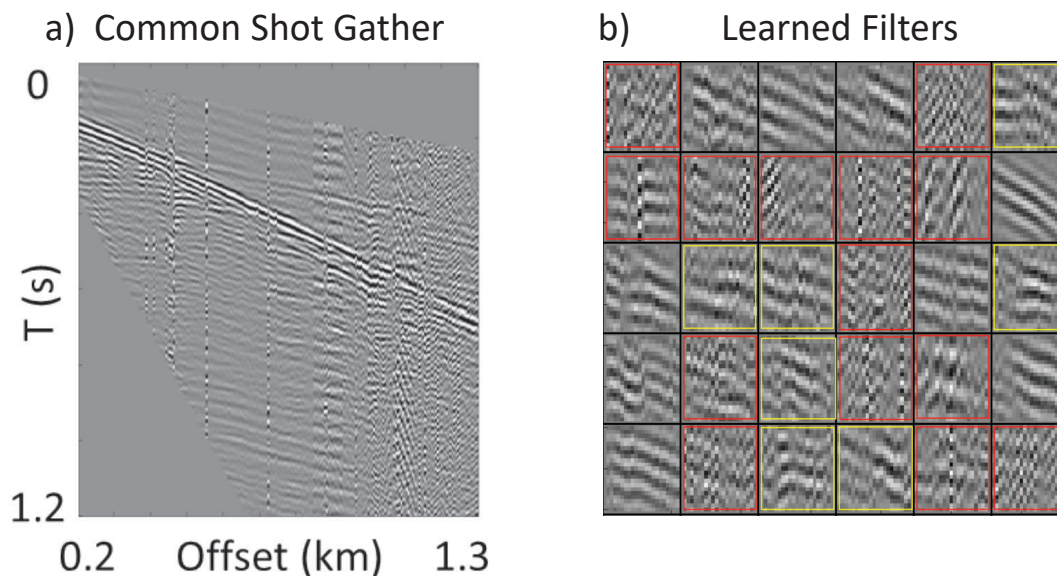


Figure 15.8: a) A CSG from the training data recorded over the Qademah fault with the offset range from 0.2 km to 1.3 km; the ground roll has been muted. b) 30  $21 \times 21$  filters learned by CSC.

intervention for inspecting the quality of the filters, and we may need to visually separate the noisy filters from the clean filters.

## 15.5 Summary

The convolutional sparse coding (CSC) method can be used to eliminate both coherent and random noise in seismic data. It mitigates the consistency issue suffered by sparse coding (SC). The seismic data with a relatively high signal-to-noise ratio are chosen for training to get the learned basis functions, and then we use all (or a subset) of the basis functions to attenuate the random or coherent noise in the seismic data. For denoising random noise, the value of the coefficient of the  $L^1$  penalty function is selected by a trial-and-error procedure. For attenuation of coherent noise, the learned basis functions must exclude coherent noise features, such as ground roll.

The CSC can learn a set of shifted invariant filters, which can reduce the redundancy of learned filters in the traditional SC denoising method. The synthetic numerical tests indicate that CSC can achieve better performance when training on similar but noiseless data compared to training on a noisy data set. Even training with noisy data, CSC still has a good denoising performance. Numerical results from the Qademah data indicate that CSC can effectively suppress seismic noise. By excluding filters with features of coherent noise, the proposed method can further attenuate coherent noise and separate ground roll. When attenuating the coherent noise, it requires human intervention to inspect the quality of the filters. This limitation can be relaxed by labeling the basis functions according to the physical attributes that differentiates signal from noise.



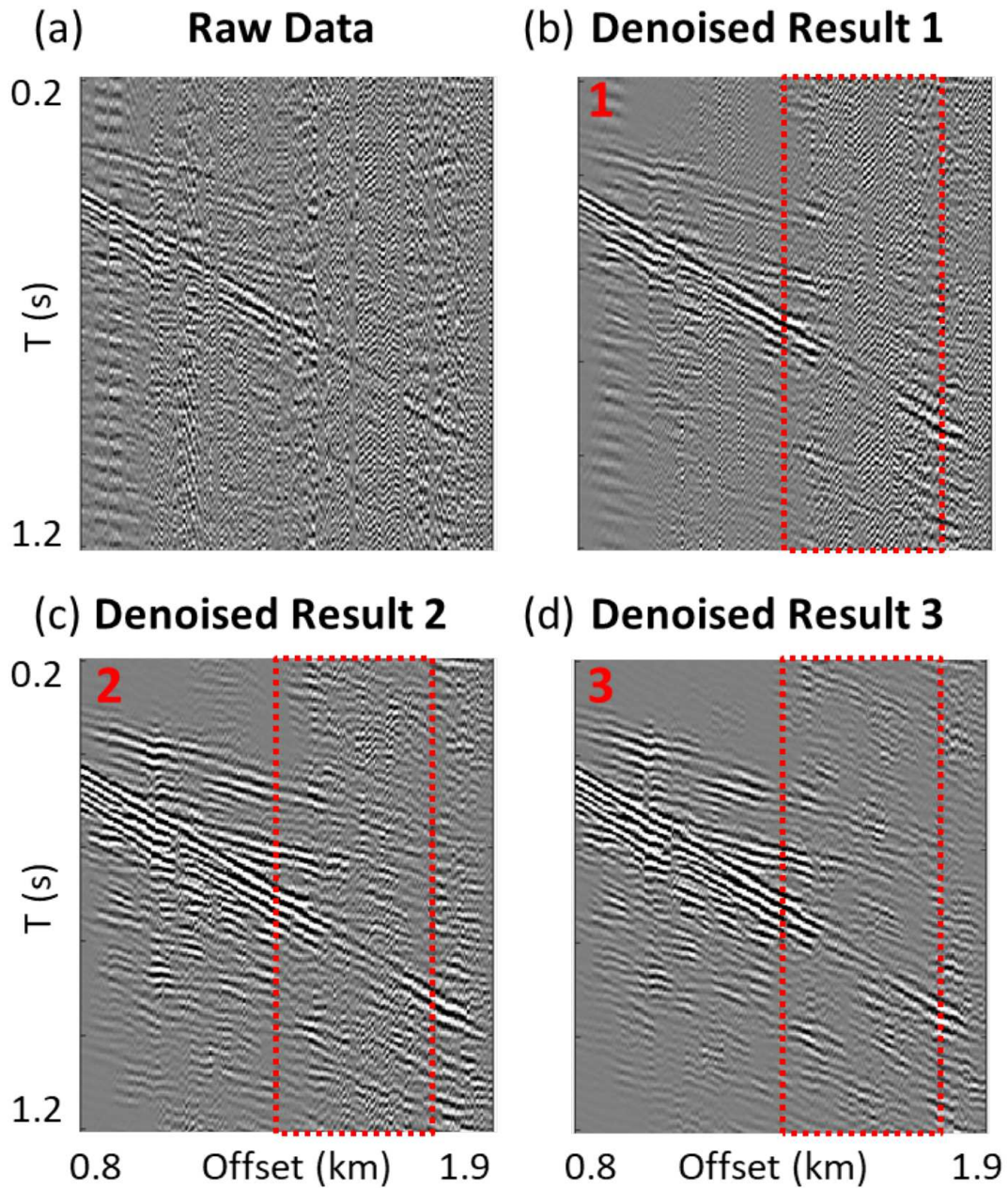


Figure 15.9: a) Noisy data from the far offset of the Qademah data with an offset range from 0.8 km to 1.9 km; the denoised data by a) all filters, and selected filters excluding those indicated by b) red and c) red and yellow boxes in Figure 15.8b.

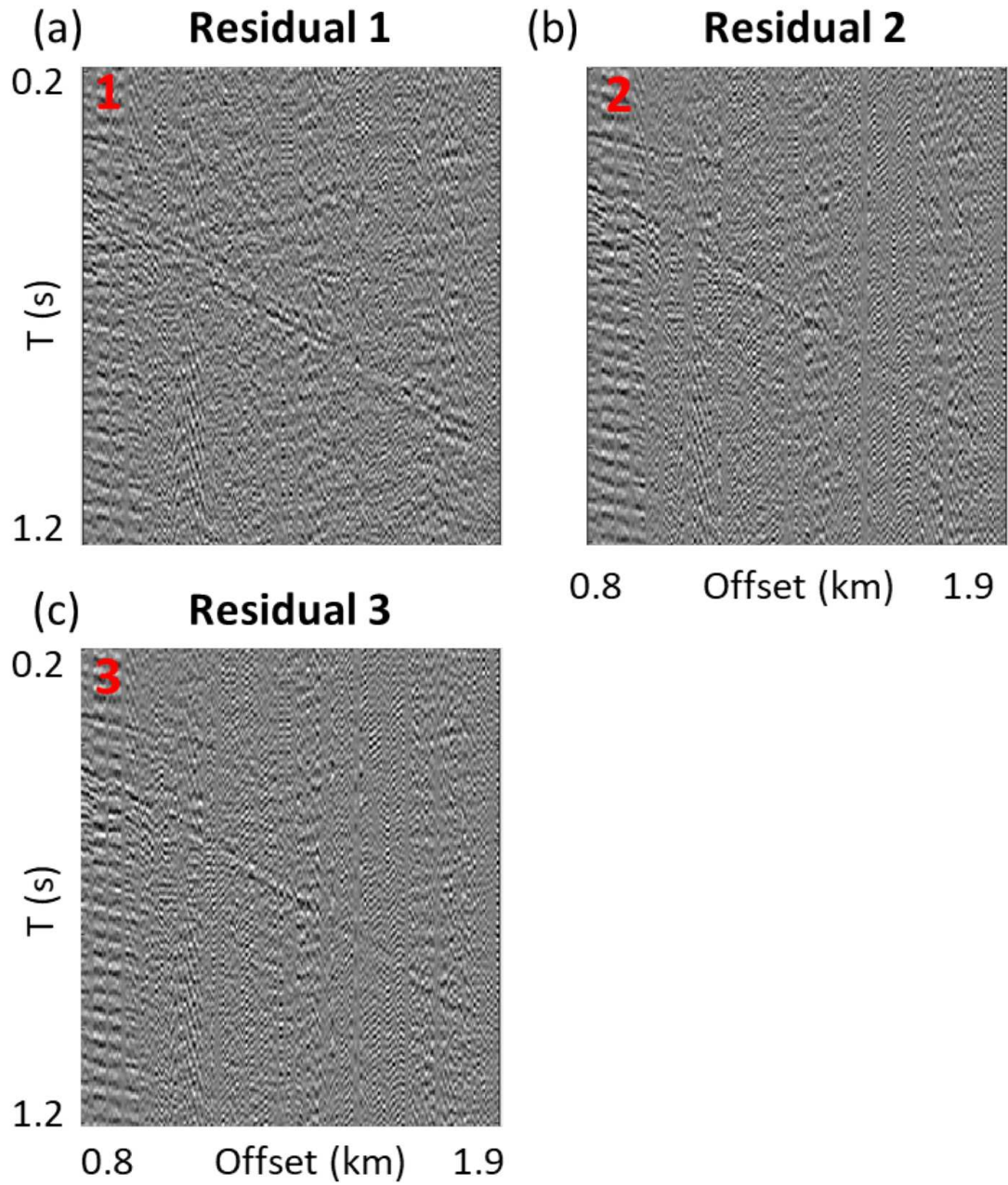


Figure 15.10: Residuals of the denoised data shown in Figure 15.9 b-d.



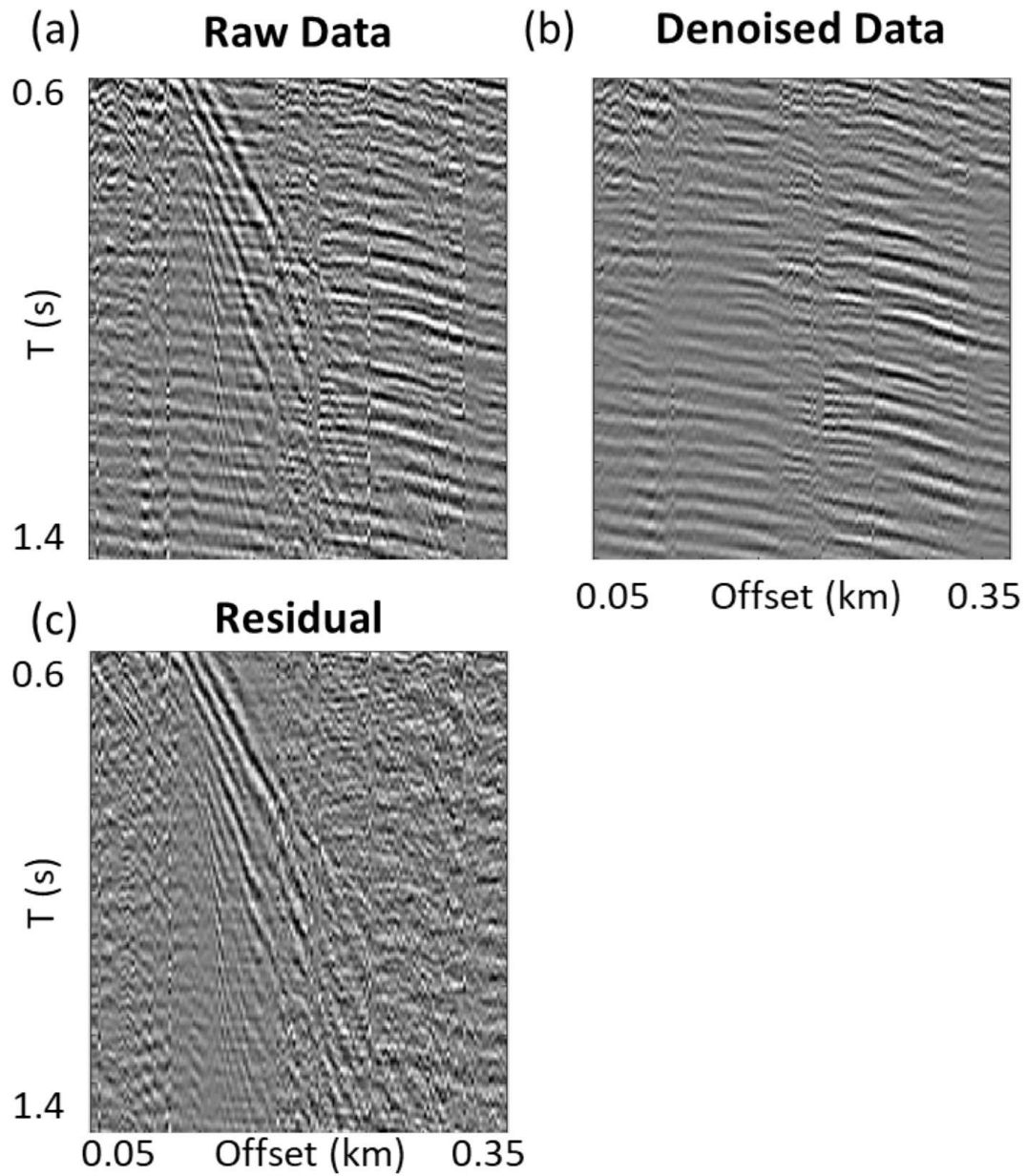


Figure 15.11: a) Near-offset data with surface waves, b) denoised data, and c) its residual.

## 15.6 Computational Labs

1. Go to the sparse coding lab at [LAB1/Chapter.Book.Sparse/Chapter.Sparse2/lab.html](http://LAB1/Chapter.Book.Sparse/Chapter.Sparse2/lab.html) and perform the exercises.

## 15.7 Appendix: ADMM

We will show how to use the ADMM procedure to solve the problems defined by equations 15.11 and 15.14. For the quadratic problem in equation 15.11, its solution is

$$\mathbf{u}_{opt} = \arg \min_{\mathbf{u}} \|\mathbf{K}\mathbf{u} - \boldsymbol{\tau}\|_2^2 = (\mathbf{K}^T \mathbf{K})^{-1} (\mathbf{K}^T \boldsymbol{\tau}), \quad (15.16)$$

where  $\boldsymbol{\tau} = \mathbf{y} - \boldsymbol{\lambda}$ .

For the problem defined in equation 15.14, it can be divided into the following three problems.

1. When  $f_i = f_1(\mathbf{v}) = \frac{1}{2} \|\mathbf{x} - \mathbf{M}\mathbf{v}\|_2^2$ , the problem becomes

$$\mathbf{y}_{1opt} = \arg \min_{\mathbf{y}_1} \frac{1}{2\rho} \|\mathbf{x} - \mathbf{M}\mathbf{y}_1\|_2^2 + \frac{1}{2} \|\mathbf{K}_1 \mathbf{u}_1 - \mathbf{y}_1 + \boldsymbol{\lambda}_1\|_2^2, \quad (15.17)$$

and its solution is

$$\mathbf{y}_{1opt} = (\mathbf{I} + \frac{1}{\rho} \mathbf{M}^T \mathbf{M})^{-1} (\frac{1}{\rho} \mathbf{M}^T \mathbf{x} + \mathbf{K}_1 \mathbf{u}_1 + \boldsymbol{\lambda}_1). \quad (15.18)$$

2. When  $f_i = f_2(\mathbf{v}) = \beta \|\mathbf{v}\|_1$ , the problem becomes:

$$\mathbf{y}_{2opt} = \arg \min_{\mathbf{y}_2} \frac{\beta}{\rho} \|\mathbf{y}_2\|_1 + \frac{1}{2} \|\mathbf{K}_2 \mathbf{u}_2 - \mathbf{y}_2 + \boldsymbol{\lambda}_2\|_2^2. \quad (15.19)$$

This is a soft threshold problem. Assume that  $\mathbf{v} = \mathbf{K}_2 \mathbf{u}_2 + \boldsymbol{\lambda}_2 = (v_1, v_2, \dots)^T$ ,  $\mathbf{y}_2 = (y^1, y^2, \dots)^T$ , and its solution is:

$$\mathbf{y}_{2opt}^i = \begin{cases} 0 & \frac{\beta}{\rho} \leq v_i \leq \frac{\beta}{\rho}, \\ v_i - \frac{\beta}{\rho} & v_i > \frac{\beta}{\rho}, \\ v_i + \frac{\beta}{\rho} & v_i < \frac{\beta}{\rho}, \end{cases} \quad (15.20)$$

The above solution can be expressed in a vector form:

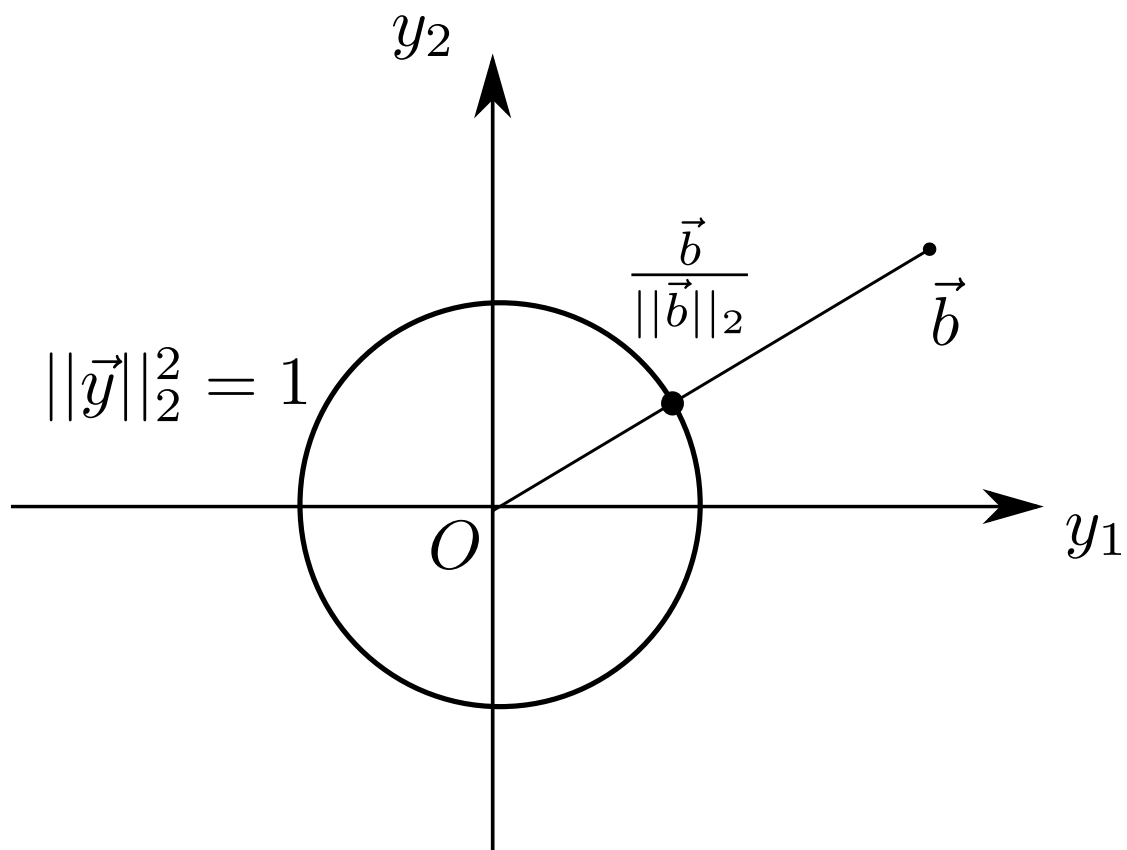
$$\mathbf{y}_{2opt} = \max(1 - \frac{\beta}{\rho} \frac{1}{|\mathbf{v}|}, \mathbf{0}) \circ \mathbf{v},$$

where  $\circ$  is the element-wise product operator,  $1/|\mathbf{v}|$  is a vector where each of its elements is the inverse of the absolute value of the corresponding element in vector  $\mathbf{v}$ .

3. When  $f_i = f_3(v) = \text{ind}_C(\mathbf{v})$  where the convex set  $C$  is  $\{\mathbf{d} \mid \|\mathbf{d}\|_2^2 \leq 1\}$ , assume  $\mathbf{b} = \mathbf{K}_3 \mathbf{u}_3 + \boldsymbol{\lambda}_3$  and the problem becomes:

$$\mathbf{y}_{3opt} = \arg \min_{\mathbf{y}_3 \in C} \frac{1}{2} \|\mathbf{y}_3 - \mathbf{b}\|_2^2, \quad (15.21)$$

if  $\mathbf{b} \in C$ ,  $\mathbf{y}_{3opt} = \mathbf{b}$ . if  $\mathbf{b} \notin C$ ,  $\mathbf{y}_{3opt}$  will have the minimum distance to  $\mathbf{b}$ . So,  $\mathbf{y}_{3opt} = \frac{\mathbf{b}}{\|\mathbf{b}\|_2}$ , as shown in Figure 15.12.

Figure 15.12: Projection of  $\mathbf{b}$  on the convex set  $\|\mathbf{y}\|_2^2 \leq 1$ .

If we define the proximal operator as:

$$\text{prox}_f(v) = \arg \min_x \left( f(x) + \frac{1}{2} \|x - v\|_2^2 \right), \quad (15.22)$$

the above three solutions can be expressed as:

$$\begin{aligned} \text{prox}_{\frac{f_1}{\rho}}(\mathbf{K}_1 \mathbf{u}_1 + \boldsymbol{\lambda}_1) &= (\mathbf{I} + \frac{1}{\rho} \mathbf{M}^T \mathbf{M})^{-1} (\frac{1}{\rho} \mathbf{M}^T \mathbf{x} + \mathbf{K}_1 \mathbf{u}_1 + \boldsymbol{\lambda}_1), \\ \text{prox}_{\frac{f_2}{\rho}}(\mathbf{K}_2 \mathbf{u}_2 + \boldsymbol{\lambda}_2) &= \max(1 - \frac{\beta}{\rho \|\mathbf{K}_2 \mathbf{u}_2 + \boldsymbol{\lambda}_2\|}, 0) \circ (\mathbf{K}_2 \mathbf{u}_2 + \boldsymbol{\lambda}_2), \\ \text{prox}_{\frac{f_3}{\rho}}(\mathbf{K}_3 \mathbf{u}_3 + \boldsymbol{\lambda}_3) &= \begin{cases} \mathbf{K}_3 \mathbf{u}_3 + \boldsymbol{\lambda}_3 & : \|\mathbf{K}_3 \mathbf{u}_3 + \boldsymbol{\lambda}_3\|_2^2 \leq 1, \\ \frac{\mathbf{K}_3 \mathbf{u}_3 + \boldsymbol{\lambda}_3}{\|\mathbf{K}_3 \mathbf{u}_3 + \boldsymbol{\lambda}_3\|_2} & : \text{else.} \end{cases} \end{aligned} \quad (15.23)$$

## Chapter 16

# Principal Component Analysis

Large multidimensional data sets are increasingly being recorded, and therefore becoming more difficult for the geoscientist to interpret. To mitigate this problem, principal component analysis (PCA) is an unsupervised machine learning method that compresses data to a smaller dimension with minimal information loss. This assumes that there are significant linear correlations between the components of the data vectors. The data compression is accomplished by finding an orthogonal coordinate system that maximizes the variability, i.e. variance, of the components along the principal component axis. The components along the principal axis are decorrelated from the ones along the other orthogonal axes. The components with higher-variance explain more information and help reveal hidden patterns in the data.

For strong linear correlations in the data components, dimensional reduction is achieved by approximating the data along a few eigenvector axes with the largest eigenvalues. This also allows for the discovery of hidden, i.e. correlated, patterns in the data.

### 16.1 Introduction

Principal component analysis (PCA) is an unsupervised machine learning method that finds an orthogonal coordinate system which maximizes the variability of the components along different axes (Jolliffe, 2002; Jolliffe and Cadima, 2016; Bishop, 2007). It implicitly assumes that there is a *linear* correlation between different components of the data vector. For example, the normally distributed points  $(x_1, x_2)$  are linearly correlated along the  $45^\circ$  line in Figure 16.1 and are plotted over the range  $-1 < x < 1$ . Their  $x_1$ -components and  $x_2$ -components vary over a range of 2 units along each axis. If the coordinate system is rotated by  $45^\circ$  then the variation of the components  $x'_1$  along the slanted line, denoted as the principal axis, has the maximum range of  $2\sqrt{2} > 2$  units. In contrast, there is a much smaller variation of the point projections along the orthogonal axis tilted at  $135^\circ$ . If these variations of the data along the  $135^\circ$  axis are much smaller than those along the  $45^\circ$  axis then the important portions of the original data can be approximated by their projections  $\mathbf{x} \approx x'_1 \hat{\mathbf{x}}'_1$ , where  $\hat{\mathbf{x}}'_1$  is the unit vector along the principal-component axis. This approximation reduces the dimension of the original data vector by one dimension.

In general, the high dimension of the original data points can be significantly reduced by projecting them onto the hyperplane which maximize their variance. For example, the points  $\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, x_3^{(i)})$  mainly plot along the blue hyperplane in Figure 16.2 which is spanned by the unit vectors  $\hat{\mathbf{a}}_1$  and  $\hat{\mathbf{a}}_2$ . Principal component analysis (PCA) is the procedure for finding a rotated coordinate system which maximizes variance of the data along the principal axes.

Another example is the PCA analysis of images of the 10 digits between 0 and 9. Figure 16.3a depicts an  $8 \times 8$  image which is a hand-written version of the digit 4. The image plots as a point

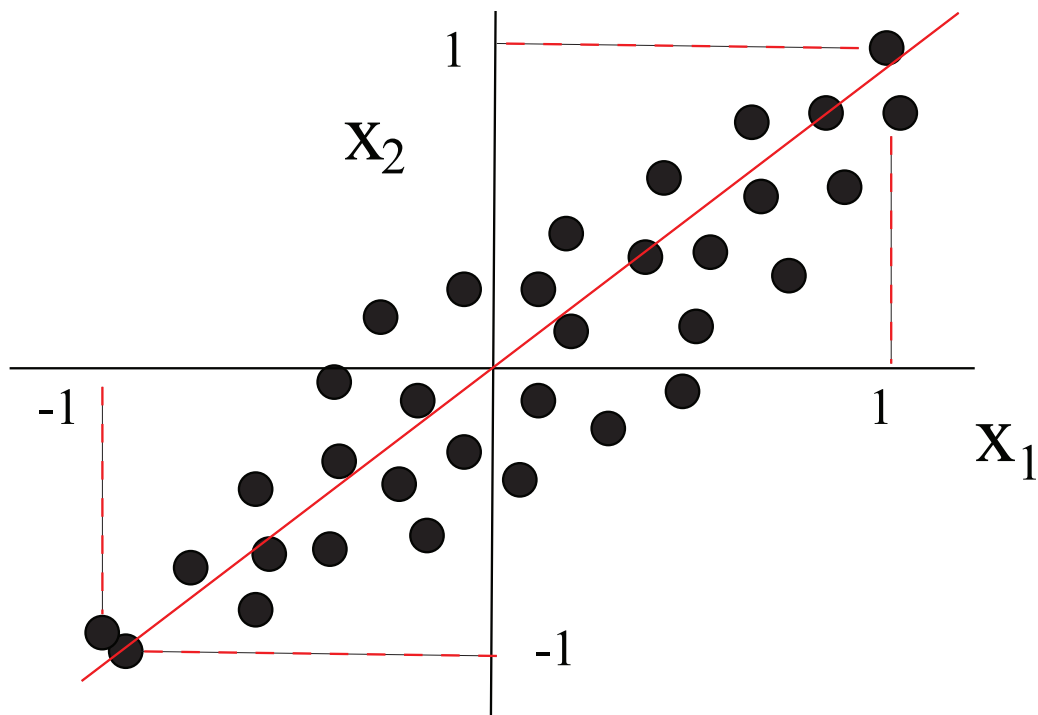


Figure 16.1: Elliptical clouds of points where the projections on the horizontal and vertical axis vary over the range  $-1 \leq x, y \leq 1$ . The data points are mostly correlated along the red axis.

Most of data are in  $x_1 - x_3$  plane

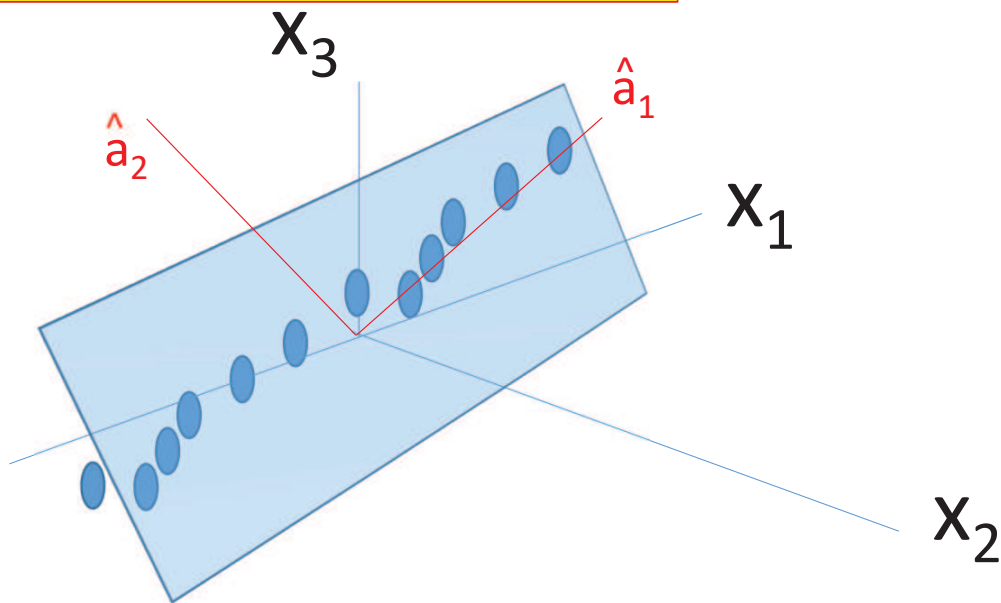


Figure 16.2: The blue points  $\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, x_3^{(i)})$  can be accurately approximated by only using two components  $\mathbf{x}^{(i)} \approx y_1^{(i)} \hat{\mathbf{a}}_1 + y_2^{(i)} \hat{\mathbf{a}}_2$  in the rotated coordinate system defined by the unit vectors  $\hat{\mathbf{a}}_1$  and  $\hat{\mathbf{a}}_2$ . The components in the rotated coordinate system are given by  $y_2^{(i)} = (\mathbf{x}^{(i)}, \hat{\mathbf{a}}_2)$  and  $y_1^{(i)} = (\mathbf{x}^{(i)}, \hat{\mathbf{a}}_1)$ .

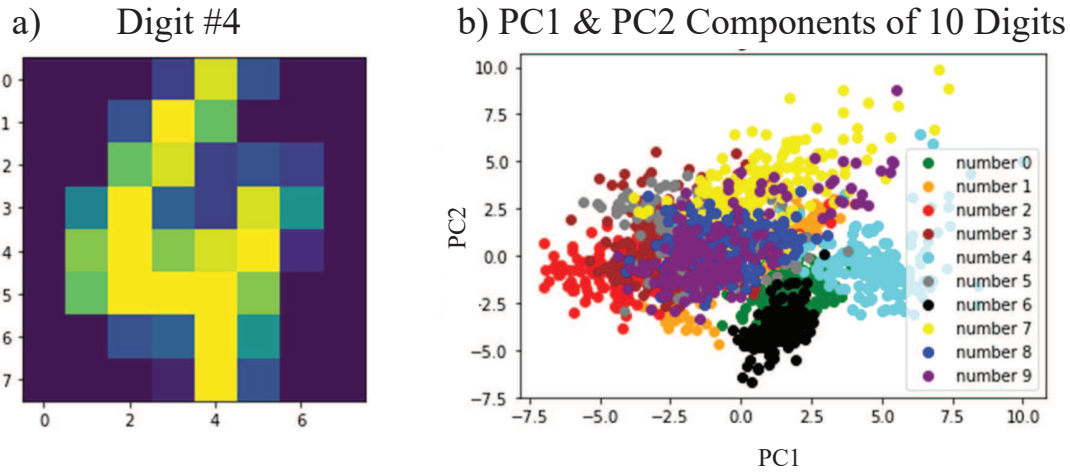


Figure 16.3: a) An  $8 \times 8$  image of the hand-written digit 4 and b) the points associated with handwritten digits between 0 and 9 plotted on the PC1 and PC2 axes.

in the 64-dimensional space spanned by the unit vectors  $\hat{\mathbf{u}}_1 = \{1, 0, 0, \dots, 0\}$ ,  $\hat{\mathbf{u}}_2 = \{0, 1, 0, \dots, 0\}$ ,  $\dots$ ,  $\hat{\mathbf{u}}_{64} = \{0, 0, 0, \dots, 0, 1\}$ . This high-dimensional space cannot be easily visualized in order to determine the proximity of one digit point from another. However, projecting these points to the principal-component axes  $PC1$  and  $PC2$  gives the 2D plot shown in Figure 16.3b. Here, the purple dots (representing the digit 9) are in close proximity to the dark-blue dots (representing the digit 8), which is not surprising given their similar shapes.

In summary, PCA finds a rotated coordinate system which maximizes the variability of the components of the points along the *principal component axes*. It can be used both for dimensional reduction of the data and enhances the ability to visualize the data in the reduced dimensional space. Similar to a NN, it employs the data to form the best basis functions for a sparse expansion. In this reduced dimensional space, correlated points data can be more easily recognized and filtered from the background noise.

### 16.1.1 Historical Background of PCA

PCA was introduced by Pearson (1901) and independently developed by Hotelling (1936). For different fields of science and engineering, the PCA goes by different names such as the Karhunen-Loève transform (KLT) in signal processing, the Hotelling transform in multivariate quality control, and the singular value decomposition that is used in many different fields (Golub and Van Loan, 1996; [https://en.wikipedia.org/wiki/Karhunen-Loeve\\_theorem](https://en.wikipedia.org/wiki/Karhunen-Loeve_theorem)). The machine learning community uses PCA as an unsupervised learning method that, combined with a clustering algorithm, can classify unlabeled data into different classes if the data have distinct correlations in the different classes. Many variants of PCA have been developed (Jolliffe, 2002) and have been adapted to special types of data (Diamantaras and Kung, 1996). Recent overviews of PCA are in Jolliffe and Cadima (2016), Wu et al. (2018) and [https://en.wikipedia.org/wiki/Principal\\_component\\_analysis](https://en.wikipedia.org/wiki/Principal_component_analysis).

### Some PCA Geophysical Applications

PCA has been used in many areas of geoscience. A recent paper (Hussein et al., 2020) shows that PCA and self-organized maps can be used to detect subtle faults in seismic sections without



supervision. The PCA was used to compactly integrate the geologic information contained in many different attributes computed from 3D seismic data. Another example of data integration by PCA is in a geothermal field in France (Ars et al., 2020). Here, the geological models were computed from log data, joint inversion of gravity recordings, ambient seismic noise and resistivity recordings. The log data included resistivity, density and shear-velocity models, so that PC1 reveals geophysical anomalies related to the geothermal system. The second component (PC2) represents 22% of the total variance and is strongly correlated with the resistivity. The third component seems to be related to the 3D geometry of the geological units.

Predicting the flood frequency of catchment basins is important information for relief agencies, farmers and city planners. In this regard, Rahman and Rahman (2020) used 8 characteristics of catchment basins as the input  $8 \times 1$  vector from 88 different basins in Australia. They used PCA to remove the linear correlation between components, and combined the PC components with a clustering technique. This technique provided some useful predictions for smaller-sized catchment basins.

PCA has been extensively used for condensing and interpreting the many different components of geophysical attributes (Zhang et al., 2009; Liu et al., 2011; Chopra and Marfurt, 2014) and measurements recorded in well logs. For example, Niculescu and Andrei (2016) applied PCA to a geophysical logging data set and conjectured that the first principal components respond to major lithological changes or shale/clay content variations. The higher-order principal components most likely reflect fluid-related data variability, such as fluids type and/or volume. They concluded that PCA successfully complements the standard log interpretation and formation evaluation methods. This is one of the examples where PCA has benefited well log analysis, including identification and characterization of pressure seals and low permeability intervals (Moline et al., 1992), delineation of lithostratigraphic units, identification of aquifer formations and distinction between hydraulic flow units (Kassenaar, 1991; Barrash and Morin, 1997; Goncalves, 1998), establishing interdependency and correlation between some hydraulic properties and geophysical and petrophysical parameters (Morin, 2006), distinguishing anhydrite, siliciclastics, and carbonates in 3D well surveys (Hu et al., 2017) and well-to-well correlation (Lim et al., 1998).

PCA is also used to denoise seismic signals. Ma and Zhai (2018) used PCA to denoise 2D panels of seismic traces. regularization of traces was employed by adding random noise. Results with both synthetic and field data show a significant reduction of noise. Huang et al. (2016) assume that the random noise and irregularly missing data are additive and uncorrelated with the signal, and utilize the PCA method to simultaneously reconstruct and de-noise seismic data. They derive an operator that acts on the extracted principal components to make the reconstructed data closer to the signal. Their results on synthetic data and recorded traces show a marked improvement compared to conventional PCA filtering of seismic noise. A similar improvement in denoising seismic data was achieved by applying the non-local Robust PCA (RPCA) to seismic data (Duarte et al., 2012; Sha et al., 2019), which is classified as a sparse transform method. Sha et al. (2019) improved the efficiency of this sparsity algorithm by sequentially upgrading the signal and downgrading the noise. Liu et al. (2020a,c) also use RPCA for denoising seismic data. Liu et al. (2020a) created windowed patches of the seismic section and grouped them based on the nonlocal similarity. For each group, they established a similar block matrix and set up the objective function of the RPCA. The weights for the PCA were determined by a minimization procedure for each block, and all of the results for different blocks are aggregated together to get the denoised seismic data. Their results appeared to surpass standard denoising methods, where standard PCA is ineffective in the presence of strong additive noise.

The following sections describe the theory of PCA and demonstrate some of its applications to geoscience problems. Finally we show how PCA can be used with a clustering algorithm for automatic classification of points. The PCA lab is at the [colab.research.google.com](https://colab.research.google.com) site<sup>1</sup>.

<sup>1</sup><https://colab.research.google.com/drive/1ghJRvIdKkgEQy-ABoU5A7NNZNjubHU90#scrollTo=py1FLHEWiGsT>

## 16.2 Theory of Principal Component Analysis

Assume  $\mathbf{x}$  is an  $N \times 1$  column vector where each component  $x_n$  is a random variable governed by a Gaussian probability distribution with mean value  $\mu_n = 0$ . In our analysis we will assume that the data  $\mathbf{x}$  has been demeaned before applying PCA, so that the data covariance matrix  $\langle \mathbf{x}\mathbf{x}^T \rangle$  for zero-mean data is defined as

$$\begin{aligned} \langle \mathbf{x}\mathbf{x}^T \rangle &= \left\langle \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix} \begin{pmatrix} x_1 & x_2 & \dots & x_N \end{pmatrix} \right\rangle \\ &= \begin{bmatrix} \langle x_1 x_1 \rangle & \langle x_1 x_2 \rangle & \dots & \langle x_1 x_N \rangle \\ \langle x_2 x_1 \rangle & \langle x_2 x_2 \rangle & \dots & \langle x_2 x_N \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle x_N x_1 \rangle & \langle x_N x_2 \rangle & \dots & \langle x_N x_N \rangle \end{bmatrix}, \end{aligned} \quad (16.1)$$

$$\approx \frac{1}{I-1} \sum_{i=1}^I \begin{bmatrix} x_1^{(i)} x_1^{(i)} & x_1^{(i)} x_2^{(i)} & \dots & x_1^{(i)} x_N^{(i)} \\ x_2^{(i)} x_1^{(i)} & x_2^{(i)} x_2^{(i)} & \dots & x_2^{(i)} x_N^{(i)} \\ \vdots & \vdots & \ddots & \vdots \\ x_N^{(i)} x_1^{(i)} & x_N^{(i)} x_2^{(i)} & \dots & x_N^{(i)} x_N^{(i)} \end{bmatrix}, \quad (16.2)$$

where  $\langle \rangle$  indicates averaging of the random variable  $x_i$  over the infinite set of experimental outcomes, which we call an ensemble. We approximate this averaging by the weighted summation over the product  $x_k^{(i)} x_l^{(i)}$  of  $I$  random variables, where  $x_k^{(i)}$  is the outcome of  $i^{th}$  experiment for the  $x_k$  random variable. The components of the vector  $\mathbf{x}$  are normalized by their standard deviations in order to avoid imbalanced component values when the components have different units (Jolliffe and Cadima, 2016). In this case the normalized covariance matrix is referred to as the correlation matrix.

As an example, consider the two prestack migration images *Mig1* and *Mig2* in Figure 16.4a where the reflectivity amplitude is non-zero along the yellow reflector. Assume each image has been demeaned<sup>2</sup>. The non-zero amplitudes in Figure 16.4a are correlated along the yellow reflector. Each migration image can be discretized into a  $2 \times 2$  grid of cells<sup>3</sup>. The discretized amplitude at the center of each cell is taken to be the average amplitude value in that cell's original image. Flattening each image into a  $4 \times 1$  vector gives the feature vectors  $\mathbf{x}^{(1)}$  and  $\mathbf{x}^{(2)}$  in Figure 16.4a, and their outer products  $\mathbf{x}^{(1)}\mathbf{x}^{(1)T}$  and  $\mathbf{x}^{(2)}\mathbf{x}^{(2)T}$  in Figure 16.4b can be added together to give the product  $\mathbf{X}\mathbf{X}^T$  of the data matrices  $\mathbf{X}$  and  $\mathbf{X}^T$ . Each row of  $\mathbf{X}^T$  is the feature vector of a migration image; equivalently each column of  $\mathbf{X}$  is a feature vector. The product  $\mathbf{X}\mathbf{X}^T$  in Figure 16.4d gives the summation of the two  $4 \times 4$  matrices in Figure 16.4b, which approximates the data-covariance matrix.

The goal of PCA is to find the  $N \times 1$  principal axis vector unit  $\hat{\mathbf{a}}$  that maximizes the variability of the projection of the  $N \times 1$  vector  $\mathbf{x}$  onto  $\hat{\mathbf{a}}$ . This projection  $\mathbf{x}^T \hat{\mathbf{a}}$  is interpreted as the length between the origin and the projected point along the unit vector  $\hat{\mathbf{a}}$ , denoted as  $x_1$  in Figure 16.1 for  $\hat{\mathbf{a}}$  along the horizontal axis. Mathematically this principal axis vector is found by finding the unit

<sup>2</sup>The mean value of the  $x_n$  random variable in the  $n^{th}$  pixel can be approximated as  $\mu_n \approx \frac{1}{I} \sum_{i=1}^I x_n^{(i)}$ . In contrast, the variance in equation 16.2 is divided by  $I-1$ , rather than  $I$ , because the mean is already computed to give one equation of constraint with  $I$  RVs. Thus, there are now only  $I-1$  independent RVs when this constraint equation is taken into account. See Exercise 16.7.2.

<sup>3</sup>In reality, the number of pixels in each migration image is on the order of  $O(10^5)$  or more. For simplicity of exposition we assume a  $2 \times 2$  image.

### Data Covariance Matrix for a Small Window in Migration Image

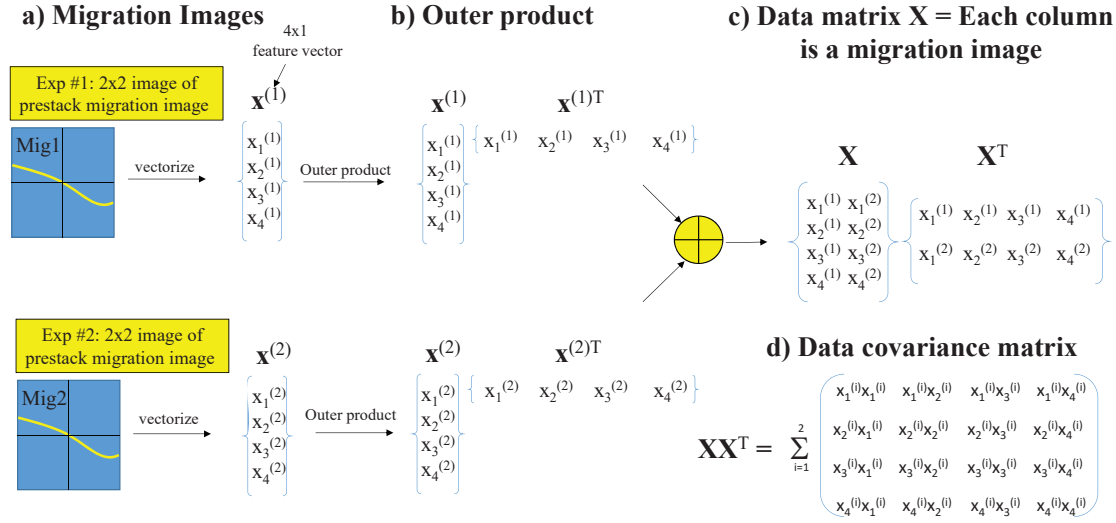


Figure 16.4: a) Two  $2 \times 2$  migration images, denoted as Mig1 and Mig2, are used to b) form the outer-product matrices  $\mathbf{x}^{(i)}\mathbf{x}^{(i)T}$  for  $i = 1, 2$ , which can be added together to get c), which approximates d) the data covariance matrix  $\mathbf{XX}^T$ .

vector  $\hat{\mathbf{a}}$  that maximizes the sum of the normalized squared lengths  $(\mathbf{x}^{(i)T}\hat{\mathbf{a}})^2$ :

$$\begin{aligned} \epsilon &= \frac{1}{I-1} \times \text{sum of squared lengths of } \mathbf{x}^{(i)} \text{ projected onto } \hat{\mathbf{a}} \\ &= \frac{1}{I-1} \sum_{i=1}^I (\hat{\mathbf{a}}^T \mathbf{x}^{(i)}) (\mathbf{x}^{(i)T} \hat{\mathbf{a}}) + \lambda (1 - \hat{\mathbf{a}}^T \hat{\mathbf{a}}), \\ &= \hat{\mathbf{a}}^T \left( \frac{1}{I-1} \sum_{i=1}^I [\mathbf{x}^{(i)} \mathbf{x}^{(i)T}] \right) \hat{\mathbf{a}} + \lambda (1 - \hat{\mathbf{a}}^T \hat{\mathbf{a}}), \end{aligned} \quad (16.3)$$

where  $I$  is the number of experiments (or migration images),  $[\mathbf{x}^{(i)} \mathbf{x}^{(i)T}]$  is the  $N \times N$  outer product matrix and  $\lambda > 0$  is the Lagrange multiplier condition that insures that  $\hat{\mathbf{a}}$  is a unit vector. The data covariance matrix approximated from two migration images is depicted in Figure 16.4d.

The stationary condition  $\partial\epsilon/\partial a_i = 0 \forall i$  for equation 16.3 leads to

$$\frac{1}{I-1} \mathbf{XX}^T \hat{\mathbf{a}} = \lambda \hat{\mathbf{a}}, \quad (16.4)$$

where  $\|\hat{\mathbf{a}}\| = 1$  and  $\frac{1}{I-1} \mathbf{XX}^T$  is the approximation to the data covariance matrix in equation 16.1. The solution to equation 16.4 solves the eigenvalue problem, and the unit eigenvector  $\hat{\mathbf{a}}_n$  with the largest eigenvalue  $\lambda_n$  maximizes the objective function  $\epsilon$ , i.e. the variance, in equation 16.3.

Since the  $N \times N$  matrix  $\mathbf{XX}^T$  is real and symmetric then there are  $N$  real eigenvalues  $\lambda_n$  where the  $N$  eigenvectors  $\hat{\mathbf{a}}_n$  are orthogonal and have unit length. Each eigenvector  $\hat{\mathbf{a}}_n$  defines one of the principal axis of the system. The physical interpretation of  $\lambda_n$  is that it is equal to the sum of the squared values of the projections of the points  $\mathbf{x}^{(i)}$  for  $i \in [1, 2, \dots, I]$  along the  $n^{\text{th}}$  principal axis. These lengths are the distances from the projected points to the origin of the rotated coordinate

system. Therefore, the eigenvector with the largest eigenvalue  $\lambda_n$  defines the eigenvector  $\hat{\mathbf{a}}_n$  with the largest variation of projections  $\mathbf{x}^{(i)}$  onto this axis.

### 16.2.1 Intuitive Dinosaur Example

Assume that 100 hadrosaur teeth are collected and their features  $(w, h, a, s, c)$  vary in width, height, age, sharpness, and color. These features are inserted into a  $5 \times 1$  vector for each tooth so that the input data consist of the  $5 \times 1$  vectors  $\mathbf{x}^{(i)} = (w, h, a, s, c)^{(i)}$  for  $i \in \{1, 2, \dots, 100\}$ . Here,  $(w, h, a, s, c)$  denote the feature values of width, height, age, sharpness and color of a tooth. The goal is to discover a lower-dimensional pattern in these 5-dimensional data.

In this example, the height and width of a tooth are assumed to be linearly proportional to age  $a$ , so that  $h = ah_1 + h_0$  and  $w = aw_1 + w_0$ . Here  $h_0$  and  $w_0$  are the starting values of height and width for a baby tooth. Color and sharpness of a tooth are independent of any other features. Therefore, the  $5 \times 1$  vectors  $\mathbf{x}^{(i)}$  project mainly to the line  $(aw_1 + w_0, ah_1 + h_0, a)$  in the  $w - h - a$  space. The hypotheses that explain this pattern is that a tooth increases in width and height linearly with increasing age, and that tooth color and sharpness are independent random variables.

The above pattern is difficult to discern in a high-dimensional space because we can only visualize 3D plots. However, the PCA analysis of this data will detect these linear correlations and show that only a one-dimensional space described by the line  $(aw_1 + w_0, ah_1 + h_0, a)$  is required to explain the feature variations in these 100 data points. The interpretation of the variable along this line can be ascertained by a biplot analysis discussed in section 16.3.2.

So this is the key idea of PCA: project linearly correlated components to a lower-dimensional space where the components of the new space are decorrelated. In the above example, the  $w - h - a$  components were linearly correlated and so could be projected to the lower-dimensional space of a line described by  $(aw_1 + w_0, ah_1 + h_0, a)$ . This line coincides with PC1 in the PCA coordinate system.

### 16.2.2 Simple Two-Point Example

As a simple example, apply PCA to the two demeaned points  $\mathbf{x}^{(1)} = (1, 1)^T$  and  $\mathbf{x}^{(2)} = (-1, -1)^T$  plotted in Figure 16.5. In this case

$$\begin{aligned}
 \mathbf{X}\mathbf{X}^T &= \begin{pmatrix} x_1^{(1)} \\ x_2^{(1)} \end{pmatrix} \begin{pmatrix} x_1^{(1)} & x_2^{(1)} \end{pmatrix} + \begin{pmatrix} x_1^{(2)} \\ x_2^{(2)} \end{pmatrix} \begin{pmatrix} x_1^{(2)} & x_2^{(2)} \end{pmatrix}, \\
 &= \begin{bmatrix} x_1^{(1)} & x_1^{(2)} \\ x_2^{(1)} & x_2^{(2)} \end{bmatrix} \begin{bmatrix} x_1^{(1)} & x_2^{(1)} \\ x_1^{(2)} & x_2^{(2)} \end{bmatrix}, \\
 &= \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}, \\
 &= \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}.
 \end{aligned} \tag{16.5}$$

The eigenvalues of this matrix can be found by setting the determinant  $|\mathbf{X}\mathbf{X}^T - \lambda\mathbf{I}|$  to zero:

$$\begin{aligned}
 |\mathbf{X}\mathbf{X}^T - \lambda\mathbf{I}| &= \begin{vmatrix} 2 - \lambda & 2 \\ 2 & 2 - \lambda \end{vmatrix}, \\
 &= (2 - \lambda)^2 - 4 = 0,
 \end{aligned} \tag{16.6}$$

where the two eigenvalues are  $\lambda_1 = 4$  and  $\lambda_2 = 0$  and their normalized eigenvectors are, respectively,  $\hat{\mathbf{a}}_1 = (1, 1)/\sqrt{2}$  and  $\hat{\mathbf{a}}_2 = (1, -1)/\sqrt{2}$ .

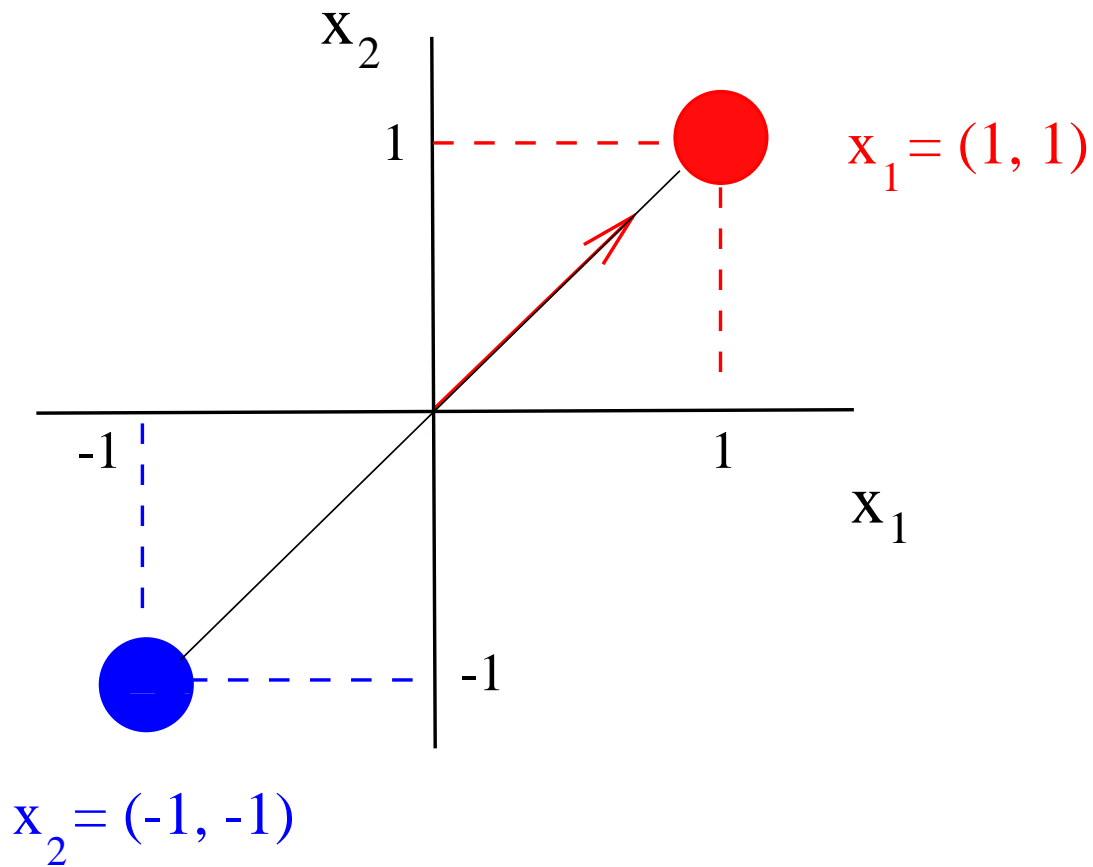


Figure 16.5: The sum of the squared projections  $|(1, 0)^T \mathbf{x}^{(1)}|^2 + |(1, 0)^T \mathbf{x}^{(2)}|^2 = (1)^2 + (-1)^2 = 2$  on the  $x$ -axis is equal to 2. The sum of the squared projections of  $\mathbf{x}^{(1)}$  and  $\mathbf{x}^{(2)}$  on the slanted red unit vector  $(1, 1)/\sqrt{2}$  is even larger at  $(|(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})^T \mathbf{x}^{(1)}|^2 + |(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})^T \mathbf{x}^{(2)}|^2 = 4$ .

The first eigenvector  $\hat{\mathbf{a}}_1$  is parallel to the maximum principal axis and is slanted at  $45^\circ$  to the horizontal as shown in Figure 16.5. It is obvious that the projections  $\hat{\mathbf{a}}_1^{(T)} \mathbf{x}^{(1)} = \sqrt{2}$  and  $\hat{\mathbf{a}}_1^{(T)} \mathbf{x}^{(2)} = -\sqrt{2}$  on the principal axis now have their maximum variation along this line. That is, the sum of the squared lengths of these projections is equal to the eigenvalue  $\lambda_1 = 4$ . The two points have zero projection along the other principle axis vector  $\hat{\mathbf{a}}_2$ , which is consistent with the null eigenvalue  $\lambda_2 = 0$ .

### 16.2.3 Filtering the PCA Components

The original image  $\mathbf{x}^{(i)}$  can be expressed as a weighted sum of eigenvectors<sup>4</sup>:

$$\mathbf{x}^{(i)} = \sum_{n=1}^N (\hat{\mathbf{a}}_n, \mathbf{x}^{(i)}) \hat{\mathbf{a}}_n, \quad (16.7)$$

where  $(\hat{\mathbf{a}}_n, \mathbf{x}^{(i)})$  is the projection of the  $i^{th}$  migration image onto the  $n^{th}$  principal component vector. Here, the increasing value of the eigenvector index  $n \in [1, 2 \dots N]$  is associated with the decreasing magnitude of the eigenvalue. The component  $(\hat{\mathbf{a}}_n, \mathbf{x}^{(1)})$  associated with the largest eigenvalue is the first principal component, and the next one is the second principal component, and so on. Since the eigenvectors are orthogonal then there is zero correlation between the different components. The goal now is to filter the migration image to retain coherent signal and reduce the random additive noise. In this regard, section 16.2.4 presents the PCA workflow and a simple example with outlier noise is shown in Figure 16.6.

The random noise in the migration image is often uncorrelated with itself (and assumed to be uncorrelated with the signal) so it does not contribute much to the element values of the data covariance matrix. This suggests that the eigenvectors associated with small eigenvalues are largely due to the additive noise, while the larger eigenvalues are associated with the coherent reflection image. To filter out this noise, a subjective rule-of-thumb (Jolliffe and Cadima, 2016) is to only keep the eigenvector components from  $n = 1 \dots N_{cutoff}$  in equation 16.7 that contribute about 70% of the energy. Thus we take the upper limit in the sum to be  $N_{cutoff} \ll N$ , where

$$\frac{\sum_{n=1}^{N_{cutoff}} |\lambda_n|}{\sum_{n=1}^N |\lambda_n|} \approx 0.7. \quad (16.8)$$

The standard measure of quality of the  $j^{th}$  principal component is given by the *proportion of total variance*

$$\pi_j = \frac{\lambda_j}{\sum_{i=1}^N \lambda_i}, \quad (16.9)$$

which is the proportion of the variance that each eigenvector represents. The values of  $\pi_j$  can be plotted against the order of the PC components in the scree plot<sup>5</sup> of Figure 16.11b. A reasonable approximation of the original data can often be computed by using a weighted sum of principal components from the top of the curve to the PC associated with the elbow of the curve. Instead of assigning a fixed threshold for the *proportion of total variance* in equation 16.9, "the *elbow* of the graph where the eigenvalues seem to level off is found and factors or components to the left of this point should be retained as significant<sup>6</sup>."

<sup>4</sup>Any  $N \times 1$  vector  $\mathbf{x}$  can be expressed as a weighted sum of the eigenvectors such that  $\mathbf{x} = \sum_{n=1}^N \alpha_n \hat{\mathbf{a}}_n$ . The eigenvectors are orthonormal, where  $(\hat{\mathbf{a}}_m, \hat{\mathbf{a}}_n) = \delta_{mn}$ , so that taking the inner product of  $\mathbf{x}$  with  $\hat{\mathbf{a}}_m$  gives  $(\hat{\mathbf{a}}_m, \mathbf{x}) = \sum_n \alpha_n (\hat{\mathbf{a}}_m, \hat{\mathbf{a}}_n) = \sum_n \alpha_n \delta_{mn} = \alpha_m$ .

<sup>5</sup>[https://en.wikipedia.org/wiki/Scree\\_plot#: :text=A%20scree%20plot%20always%20displays,should%20be%20retained%20as%20significant.](https://en.wikipedia.org/wiki/Scree_plot#: :text=A%20scree%20plot%20always%20displays,should%20be%20retained%20as%20significant.)

<sup>6</sup>[https://en.wikipedia.org/wiki/Scree\\_plot#: :text=A%20scree%20plot%20always%20displays,should%20be%20retained%20as%20significant.](https://en.wikipedia.org/wiki/Scree_plot#: :text=A%20scree%20plot%20always%20displays,should%20be%20retained%20as%20significant.)

### 16.2.4 PCA Workflow

A MATLAB script is used to generate the data shown in Figure 16.6 and the MATLAB code and PCA workflow are described in Box 16.2.1.

#### Code 16.2.1. MATLAB Workflow and Code for PCA

1. Record or create data. Demean the  $D \times 1$  data vectors  $\mathbf{x}^{(i)}$  for  $i = [1, 2, \dots, N]$ . Dividing by standard deviation is an option for each type of random variable. Don't normalize if you wish to preserve true relationship between the different types of data.

```
clear all
data=zeros(30,2);data1=data;
data(:,1) = randn(30,1);
data(:,2) = 3.4 + 1.2 * data(:,1) + 0.2*randn(size(data(:,1)));
% data=data2;
subplot(121);plot(data(:,1),data(:,2),'o');
title('a) Raw data');axis('equal');pause(1)
data(:,1) = data(:,1)-mean(data(:,1));
data(:,2) = data(:,2)-mean(data(:,2));
subplot(121);plot(data(:,1),data(:,2),'o');
title('a) Demeaned data');axis('equal')
```

2. Compute the covariance matrix  $\mathbf{XX}^T$  in equation 16.2.

```
C = cov(data);
```

3. Compute the eigenvectors  $\mathbf{a}_i$  and eigenvalues  $\lambda_i$  of  $\mathbf{XX}^T$ . In MATLAB script below the eigenvectors are columns of "V" and eigenvalues are diagonal components of "D".

```
[V,D] = eig(C);
```

4. Filter the input data by reconstructing the data  $\mathbf{x}^{(i)}$  by a truncated sum of the eigenvectors:

$$\mathbf{x}^{(i)} = \sum_{n=1}^M (\hat{\mathbf{a}}_n, \mathbf{x}^{(i)}) \hat{\mathbf{a}}_n, \quad (16.10)$$

where  $M \ll N$  is hopefully much less than the original dimension of the data. In MATLAB the filtered components along the V(:,1) and V(:,2) axes are

```
x=V(1,1)*data(:,1)+V(2,1)*data(:,2);
y=V(1,2)*data(:,1)+V(2,2)*data(:,2);
subplot(122);plot(x,y,'o');axis('equal');
title('b) PC1 & PC2 Coordinate System')
xlabel('PC2');ylabel('PC1')
```

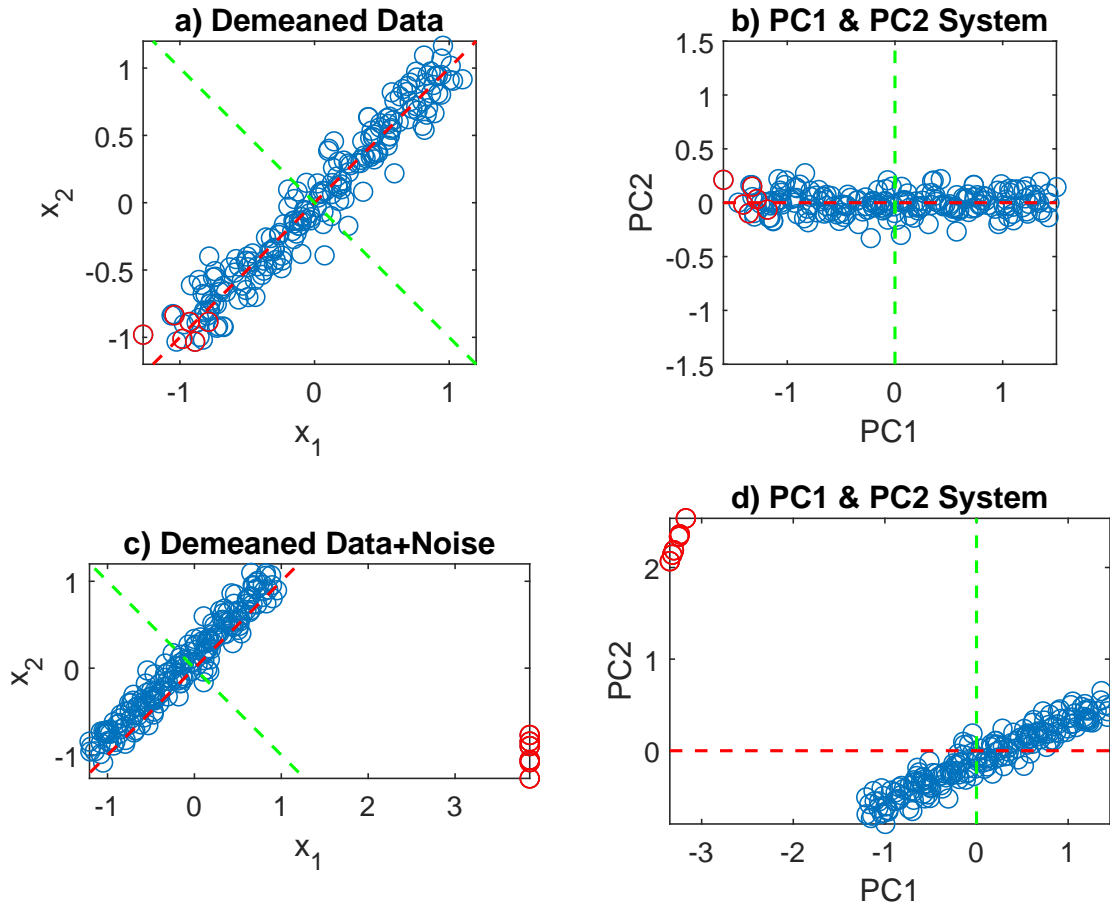


Figure 16.6: Correlated raw data points on the left and same points in PC1 and PC2 coordinate system on the right. The data in a) are mostly correlated along the red-dashed line so they mostly plot along the PC1 axis in b). However, the input data with six noisy outliers (red circles) in c) are not linearly correlated with the blue circles, so the PCA in d) does not accurately decorrelate the data. This reinforces the requirement that outliers in the input data should be eliminated before demeaning and PCA. The red- and green-dashed lines in c) are for reference, and are not rotated to be the dashed lines in d). Partly adapted from <http://mres.uni-potsdam.de/index.php/2017/09/14/principal-component-analysis-in-6-steps/>.



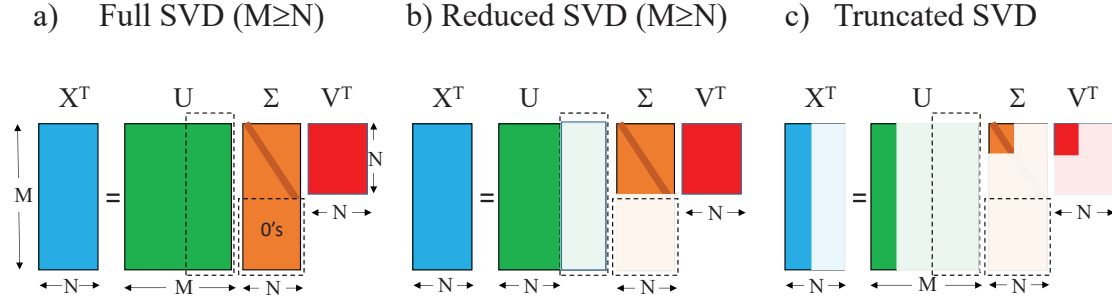


Figure 16.7: Matrix decomposition of the  $M \times N$  matrix  $\mathbf{X}^T$  into a) SVD, b) reduced SVD, and c) truncated SVD matrices. The dashed black rectangle in a) the green matrix indicates the portion of  $\mathbf{U}$  that multiplies the null singular values in the dashed box of the orange matrix  $\Sigma$ . The reduced SVD matrices in b) ignore these null portions of the matrices, while the truncated SVD in c) deliberately zeros out a large band of small singular values along the diagonal of  $\Sigma$ .

### 16.2.5 Singular Value Decomposition

Singular Value Decomposition (SVD) can accomplish the same dimensionality reduction as PCA, and is sometimes less expensive. The SVD can be considered a rank-reduction method in the same class as RPCA (Siahsar et al., 2017), where a low-rank matrix approximates the covariance matrix. An excellent SVD lecture by Gilbert Strang is at [www.youtube.com/watch?v=rYz83XPxiZo](http://www.youtube.com/watch?v=rYz83XPxiZo).

The following steps describe the SVD method.

- Decompose the  $M \times N$  data matrix  $\mathbf{X}^T$  into

$$\mathbf{X}^T = \mathbf{U} \Sigma \mathbf{V}^T, \quad (16.11)$$

where  $M \geq N$  and the  $\mathbf{V}$  is an  $N \times N$  matrix with orthonormal column vectors known as right-singular vectors. The  $M \times M$  matrix  $\mathbf{U}$  also has orthonormal column vectors, but are denoted as left-singular vectors, and  $\Sigma$  is an  $M \times N$  rectangular diagonal matrix of singular values  $\sigma_i$  depicted by the narrow orange band in Figure 16.7a. The diagonal components are sorted in decreasing order  $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_N$ . By orthonormality,  $\mathbf{U}^T \mathbf{U} = \mathbf{I}$  and  $\mathbf{V}^T \mathbf{V} = \mathbf{I}$ .

- The columns in  $\mathbf{V}$  are right-singular vectors equivalent to the eigenvectors of  $\mathbf{X} \mathbf{X}^T$ , and the singular values of  $\mathbf{X}^T$  are equal to the square root of the eigenvalues of  $\mathbf{X} \mathbf{X}^T$ . If the data are demeaned, then the data-covariance matrix can be formed from

$$\begin{aligned} \mathbf{X} \mathbf{X}^T &= \mathbf{V} \Sigma^T \mathbf{U}^T \mathbf{U} \Sigma \mathbf{V}^T, \\ &= \mathbf{V} \tilde{\Sigma}^2 \mathbf{V}^T, \end{aligned} \quad (16.12)$$

where  $\tilde{\Sigma}^2 = \Sigma^T \Sigma$ . We can express the data vector  $\mathbf{x}$  as a truncated sum of weighted singular vectors that have a singular value above a critical threshold. These singular vectors are the column vectors of  $\mathbf{V}$

- The right-singular vector  $\mathbf{v}_i$  in the  $i^{th}$  column of  $\mathbf{V}$  is the  $i^{th}$  principal component vector of PCA, and the singular value  $\sigma_i$  is the square root of the variance along the  $i^{th}$  PC axis. For example, multiply the right of the  $N \times N$  covariance matrix  $\mathbf{X} \mathbf{X}^T$  in equation 16.12 by the

$N \times 1$  singular vector  $\mathbf{v}_1$  to get

$$\begin{aligned} \mathbf{X}\mathbf{X}^T \mathbf{v}_1 &= \mathbf{V}\mathbf{\Sigma}^2 \mathbf{V}^T \mathbf{v}_1 = \mathbf{V} \tilde{\mathbf{\Sigma}}^2 \hat{\mathbf{e}}_1, \\ &= \sigma_1^2 \mathbf{V} \hat{\mathbf{e}}_1, \\ &= \sigma_1^2 \mathbf{v}_1, \end{aligned} \tag{16.13}$$

where  $\hat{\mathbf{e}}_1 = (1, 0, 0, \dots, 0)$  is an  $N \times 1$  unit vector, and  $\mathbf{v}_1$  the is first singular vector, i.e. first column vector of  $\mathbf{V}$ . According to PCA analysis,  $\sigma_1^2$  is the variance of the first PC vector because  $\mathbf{v}_1$  is an eigenvector, not just a right-singular vector, of  $\mathbf{X}\mathbf{X}^T$ .

- The SVD approach is sometimes preferred over PCA because computing singular vectors and singular values can sometimes be less expensive than computing the eigenvalues of  $\mathbf{X}\mathbf{X}^T$  ([https://en.wikipedia.org/wiki/Principal\\_component\\_analysis](https://en.wikipedia.org/wiki/Principal_component_analysis)). If  $N$  is large then the compact SVD methodology is recommended<sup>7</sup>. If only a few singular values are large, then the truncated SVD in Figure 16.7c requires much less memory and computational complexity than that for full or reduced SVD in a) or b). See Exercises 16.7.11 and 16.7.12.
- PCA (or SVD) is sensitive to outliers in the data, so such points should be removed prior to analysis. An alternative is to downweight the noisy data (Kriegel et al., 2008).

## Denoising Seismic Data with SVD

Singular value decomposition, which is equivalent to PCA in that it can decompose  $\mathbf{X}\mathbf{X}^T$  into eigenvectors, can be used to detect and enhance laterally coherent signals in multitrace recordings (Bekara and Van der Baan, 2007). It can be used for dip filtering of seismic reflections, VSP up/down wavefield separation, elimination of the direct waves in ground penetrating radar data (Liu et al., 2017a), and residual statics corrections (Ulrych et al., 1988). It can also be used as a regularized inverse method to improve the spatial resolution of traveltime tomograms (Infante-Pacheco et al., 2020).

SVD is suitable for data where coherent events can be aligned laterally. For example, the reflection events in Figure 16.8a can be time shifted so the early arrivals are aligned, that is flattened, in Figure 16.8b. The data vector  $\mathbf{x}^{(t)}$ , where the time  $t$  plays the role of a sample index, consist of the amplitudes at the fixed time  $t$  for the 5 flattened traces in the red window of Figure 16.8b. Each trace is at a different offset and each example  $\mathbf{x}^{(t)}$  is at a new time value  $t$ .

Figure 16.9a in the red box shows a windowed view of 5 flattened traces from Figure 16.8b. At a fixed time  $t$ , the amplitudes from the five traces are used to form the  $5 \times 1$  vector  $\mathbf{x}^{(t)}$  that surrounds the 3rd trace. Another set of amplitudes at  $t + 1$  forms the  $5 \times 1$  vector  $\mathbf{x}^{(t+1)}$ . This is repeated for all the time values in the red window to form the ensemble of examples  $\mathbf{x}^{(t)}$   $t \in \{t_1, t_2, \dots, t_N\}$ . These data are used to compute the covariance matrix  $\mathbf{X}\mathbf{X}^T$  after demeaning. For the  $5 \times 1$  sample vectors  $\mathbf{x}^{(t)} = (x_1^{(t)}, x_2^{(t)}, \dots, x_5^{(t)})$ , the SVD algorithm is then used to obtain the 5 singular vectors shown in Figure 16.9b. Notice how the first one contains most of the signal in the 3rd trace of Figure 16.9a because the samples for a fixed time are largely redundant, i.e.  $x_1^{(t)} \approx x_2^{(t)} \approx \dots \approx x_5^{(t)}$ , for that time value. This is the reason why we flatten the traces, we want the amplitude values to be nearly the same and largely redundant so they can be approximated by the 1st singular vector. As can be seen in Figure 16.9b, the 2nd and higher-order singular vectors are weak in amplitude and are assumed to represent noise in the traces. After SVD filtering of the 3rd trace in a) we get the leftmost trace in Figure 16.9c. This filtered trace appears similar to the stacked trace on the right where all the traces in the red box in a) are stacked together. This process of estimating the singular vectors and filtering the traces is repeated for the red window sliding over the entire CSG. The final result of using this sliding window SVD filtering method is shown in Figure 16.10.

<sup>7</sup>[https://en.wikipedia.org/wiki/Singular\\_value\\_decomposition#Compact\\_SVD](https://en.wikipedia.org/wiki/Singular_value_decomposition#Compact_SVD)

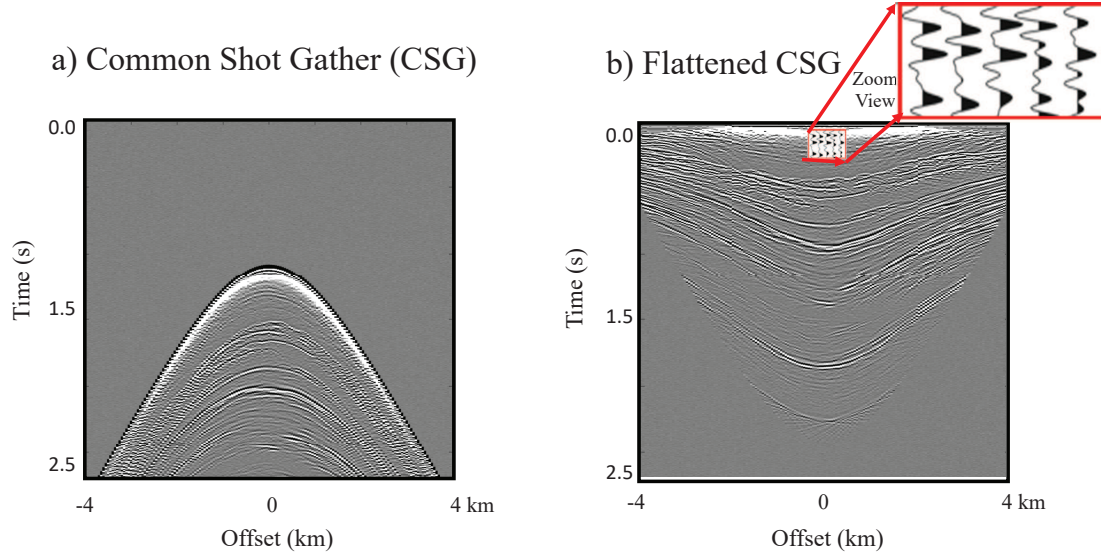


Figure 16.8: a) Common shot gather and b) flattened and windowed portion of shot gather with a zoom view in red box. Results from Yasmin Oshiro.

### Computational Cost of SVD

Sanger (1993) presented an iterative algorithm to find the singular values and vectors for SVD. The SVD of the  $M \times N$  matrix  $\mathbf{X}^T$  is computed by a two-step procedure: 1) the matrix  $\mathbf{X}^T$  is reduced to a bidiagonal matrix, which takes  $O(4MN^2 - 4N^3/3)$  floating-point operations (flops) by a Householder-like algorithm, and 2) the SVD of the bidiagonal matrix is computed by an iterative method such as a QR-type algorithm<sup>8</sup>. It is often the case that the entire SVD is not required when we only need the singular vectors associated with a few large singular values. In this case, there are several reduced SVD methods that can be used to significantly reduce memory and computation costs, such as the reduced SVD in Figure 16.7b or truncated SVD in Figure 16.7c. In the reduced SVD of Figure 16.7b, the *dimmed* column vectors in the green  $\mathbf{U}$  matrix associated with the *dimmed* zero-singular values in  $\mathbf{\Sigma}$  are muted. This can significantly reduce the computational complexity of the computing the SVD. For the truncated SVD illustrated in Figure 16.7c, only a few columns in  $\mathbf{U}$  are retained because just a few singular vectors associated with the largest singular values are required. This can significantly reduce the computational complexity compared to SVD or reduced SVD.

## 16.3 Numerical Examples

We will now demonstrate the use of PCA applied to several examples. One of the examples will be eruption times of Yellowstone earthquakes and their duration.

### 16.3.1 Yellowstone Eruption Data

Figure 16.11a depicts the eruption duration time of Yellowstone earthquakes plotted against their waiting time (WT) between eruptions (Jenset and McGillivray, 2014). In these two coordinates the

<sup>8</sup>[https://en.wikipedia.org/wiki/Singular\\_value\\_decomposition#Numerical\\_approach](https://en.wikipedia.org/wiki/Singular_value_decomposition#Numerical_approach)

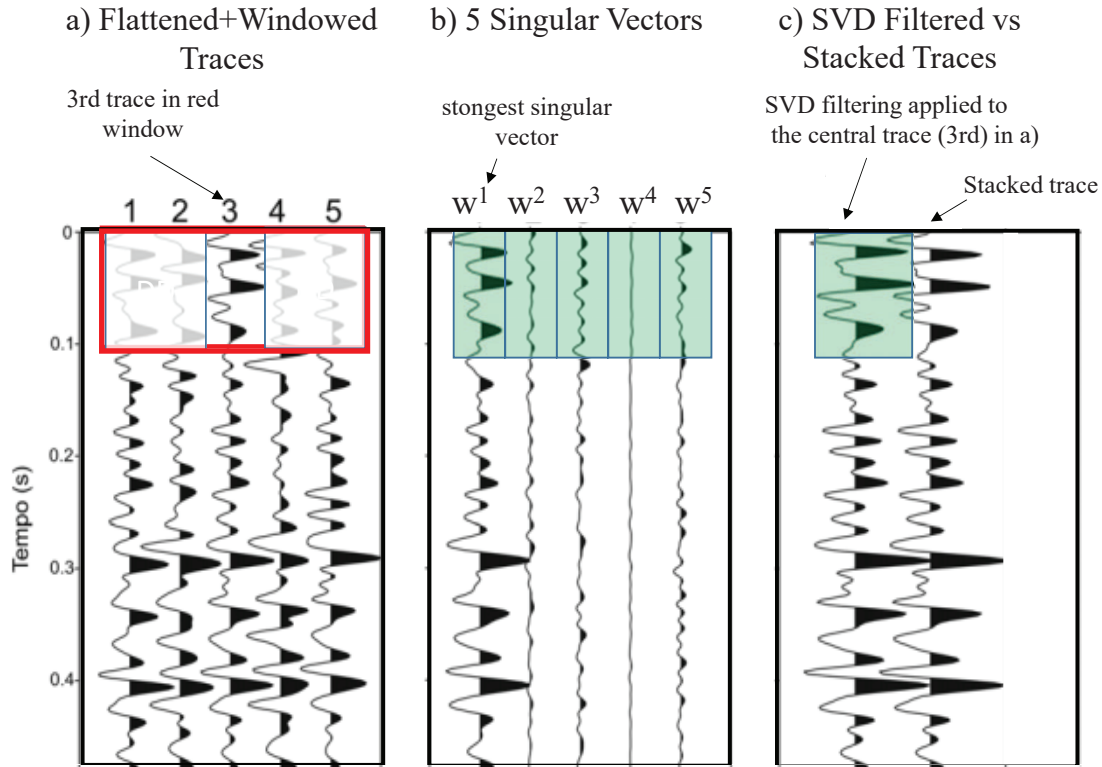


Figure 16.9: a) Windowed and flattened traces from Figure 16.8b, b) singular vectors from the windowed traces in a), and c) the SVD filtered trace on the left and the stacked trace from 5 traces on the right. In practice, the red window size is 5 samples in time and 5 samples in offset. The filtered trace was obtained by estimating the SVs by shifting the red window down in time but centered at the 3rd trace in a). Results from Yasmin Oshiro.

## Raw Data vs Singular Value Vectors

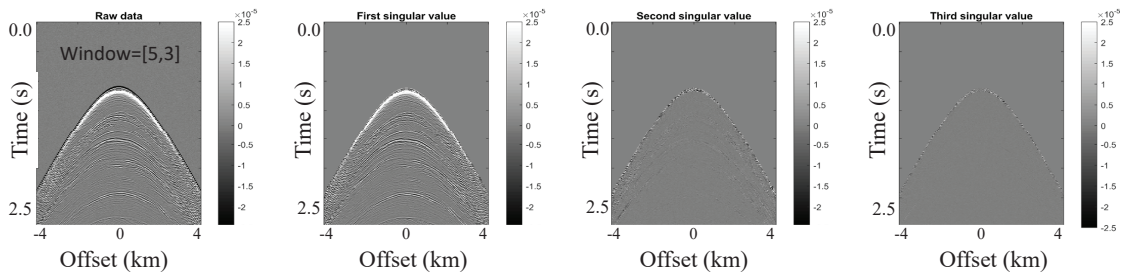


Figure 16.10: a) CSG of the data on the left and the singular-vector representations to the right. Notice that the singular vectors of 2nd-order and higher appear to consist mostly of random noise. Results from Yasmin Oshiro.

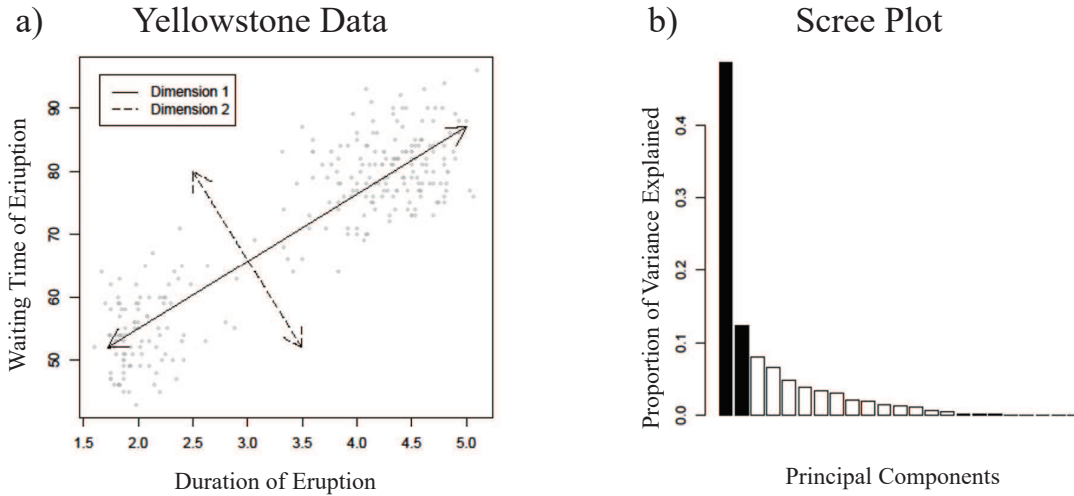


Figure 16.11: a) *Waiting Time for Eruption* plotted against *Duration* for Yellowstone data. The solid and dashed double-sided arrows correspond to the PC1 and PC2 axes, respectively. The scree plot in b) is for an unrelated data set with over 20 principal component vectors. Figures adapted from (Jenset and McGillivray, 2014).

points form a cloud that slants about 45 degrees with respect to the horizontal axis. It is clear that there is a correlation between the waiting time and eruption duration: the longer we wait the bigger the eruption.

The projection of these points along the horizontal axis, the duration of eruption (DUR), approximately covers the range  $1.7 < DUR < 5.1$ . The PCA axes are computed using the PCA workflow described above, and the 1st principal component (PC) axis is plotted as the solid double-sided arrow. It is clear that the solid double-arrow axis does not fully account for all the variation in the data. The remaining variability in the data is accounted for by the projections of the points along the orthogonal PC2 axis plotted as dashed double-sided arrow. These two axes constitute the basis vectors for the subspace that best accounts for the total variation of the data.

Science is all about collecting data and determining mathematical models that explain the data. The simplest model is favored, and so in this case the PC1 axis allows one to predict WT from DUR. That is, the double-arrow PC1 line represented by  $WT = (90 - 50)/(5.5 - 1.5)DUR + b$  is a good approximation for predicting WT from DUR. Here,  $b$  is the intercept WT for  $DUR=1.5$ .

If these data constituted a cloud of points in  $M$ -dimensional space, and the data were only correlated only along the PC1 and PC2 axes, then no other axes are needed to represent these points. This represents a tremendous reduction in data storage, not to mention a simple model that accurately predicts eruption waiting times from DURs.

The scree plot in Figure 16.11b is used to determine the number of PC axes needed to accurately represent the data, which in this case is unrelated to the Yellowstone data. Here, the value of each point along the vertical axis is determined by equation 16.9, which gives the proportion of variance explained by each PC. The first PC accounts for around 48% of the variance and the second one accounts for 12%. The strong drop between the second dark bar and the first light bar suggests that only two PCs are needed for this data set.

### 16.3.2 Biplots and Geochemical Data from a Gold Site

Biplots can be used to understand the physical meaning of the points in the principal component space. For example, Figure 16.12 depicts the geochemical points from an Australian gold site plotted in the PC1 and PC2 subspace. The geochemical data consists of various percentages of Ca, As, K, Ba, Cr, Cu, Ti, Fe, Mn and Ni in a 4-9 meter wide outcrop of quartz breccia in an Australian gold field (Gazley, et al., 2015). Each data sample is assigned a percentage and a  $9 \times 1$  Euclidean unit vector. For example, The Chromium element is assigned the  $9 \times 1$  unit vector  $\hat{\mathbf{e}}_1 = (1, 0, 0, \dots, 0)$  and if the sample  $i = 1$  contains 100% Chromium then  $\mathbf{x}^{(1)} = (100, 0, 0, \dots, 0)$ . Similarly, the potassium sample can be assigned the  $9 \times 1$  Euclidean unit vector  $\hat{\mathbf{e}}_2 = (0, 1, 0, \dots, 0)$  so that if sample  $i = 2$  is 40% Chromium and 60% Potassium then the data sample vector for  $i = 2$  is  $\mathbf{x}^{(2)} = (40, 60, 0, 0, \dots, 0)$ . In general, each of the nine listed elements is assigned a unique  $9 \times 1$  Euclidean unit vector, and the element percent of each sample is assigned as the coefficient  $x_k^{(i)}$  in  $\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots, x_9^{(i)})$ .

The projection of the Ca, As, K, Ba, Cr, Cu, Ti, Fe, Mn and Ni unit vectors onto the PC1-PC2 subspace form the biplot vectors  $\mathbf{v}_i$  shown in Figure 16.12 and defined as

$$\mathbf{v}^{(i)} = (\mathbf{x}^{(i)}, \hat{\mathbf{a}}_1)\hat{\mathbf{a}}_1 + (\mathbf{x}^{(i)}, \hat{\mathbf{a}}_2)\hat{\mathbf{a}}_2, \quad (16.14)$$

where  $\hat{\mathbf{a}}_1$  and  $\hat{\mathbf{a}}_2$  are the first and second principal component unit vectors. Here,  $\mathbf{x}^{(i)}$  is the  $i^{th}$  data sample vector in the original data space and  $(\mathbf{x}^{(i)}, \hat{\mathbf{a}}_1)$  is its projection along the first principal component axis  $\hat{\mathbf{a}}_1$ . Its projection to the 2nd PC axis  $\hat{\mathbf{a}}_2$  is given by  $(\mathbf{x}^{(i)}, \hat{\mathbf{a}}_2)$ .

The geochemical meaning of a biplot vector is deduced by noting that the red Cr vector  $(1, 0, 0, \dots, 0)$  in the input space projects to the point  $\mathbf{y}^{Cr} = (-9, 1)$  in the  $PC1 \times PC2$  plane. This says that the Chromium content is negatively and strongly correlated along the PC1 axis denoted by  $\hat{\mathbf{a}}_1$ ; here, a scaling has been applied to the axes. The projection along the PC2 axis is only 1, which is 9 times weaker than the projection along PC1. Points farther to the left of the origin along the PC1 axis indicate input samples with increasing Chromium content.

The green Ca biplot vector is along the  $-45^\circ$  line and suggests a strong negative correlation of calcium for points along this slanted line. In fact, gold compounds are correlated quite well along this slanted line and suggest that a gold element vector, which was not in the original batch of elements, would have correlated well along this slanted line. Note that the purple potassium element is a point that is in the 4th quadrant.

The lengths of these biplot vectors are proportional to the magnitude of the projection of the original unit vector onto the PC1-PC2 plane. In the Figure 16.12 example, the PC1 axis is strongly correlated with the elements Ca, As, K, and Ba, while the vertical PC2 axis is strongly correlated with the elements Cr, Cu, Fe, Mn and Ni. Gazley et al. (2015) claim that PC1 and PC2 accounts for  $> 60\%$  of the variance in a multivariate dataset, with careful selection of components. They also claim that "that the majority of variation accounted for by PC1 is a function of variation in background rock chemistry, most likely a function of fine scale layering. The variation accounted for by PC2 is likely to show evidence for chemical changes as a result of hydrothermal alteration.". The points with red circles account for the rocks with the highest gold concentrations.

### 16.3.3 Nashville Limestones

Around 200 limestone samples were taken from outcrops near Nashville, Tennessee and their geochemical composition was measured by Theiling et al. (2007). These data were analyzed with PCA and the data points, biplot vectors and scree plots are plotted in Figure 16.13b- 16.13d, while the loadings of the PC vectors are provided in Figure 16.13a<sup>9</sup>. The loadings, i.e. components, for the unit vector  $\mathbf{PC1} = (y_1, y_2, \dots, y_8)$  were computed by taking the dot product of the unit vector

<sup>9</sup><http://strata.uga.edu/8370/handouts/pcaTutorial.pdf> and <http://strata.uga.edu/8370/data/>

## PC1-PC2 Subspace for Geochemical Samples

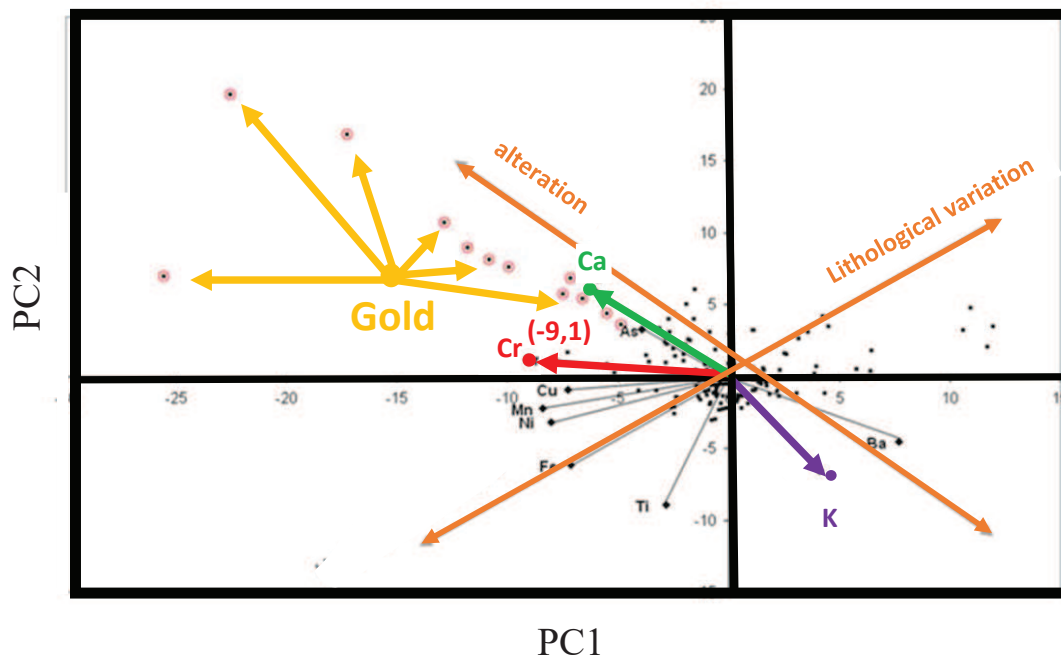


Figure 16.12: Geochemical samples from an Australian gold field plotted in the PC1-PC2 subspace. Samples with red circles indicate a high gold content.

**PC1** with the unit vector for *d13C*, e.g.  $\hat{\mathbf{e}}_1 = (1, 0, 0, \dots, 0)$  to get  $y_1 = -0.17$ . Note that the square of the components in the column vector for **PC1** sum to unity, as it should for a unit vector.

The mineral dolomite is a double carbonate ( $\text{CaMg}(\text{CO}_3)_2$ ) and is rich in magnesium (Mg). The biplot vectors for Mg and *d18O* in Figure 16.13c indicate that data points near the top of the graph and close to the vertical axis are indicative of dolomites. Indeed, this part of the graph is interpreted as dolomite-type rocks in Figure 16.13d. In contrast, limestones are rich in Calcium (Ca) and so the biplot for Ca is near the horizontal axis to the right of the origin in Figure 16.13c and so the interpretation in Figure 16.13d indicates clean limestones to the right and near the horizontal axis.

Limestones can be polluted with clay particles, which are rich in Al, Si and Al. These elements are indicated to be more common for data points along the left side of the PC1 axis as suggested by the biplot vectors in Figure 16.13c. Therefore, the clay-rich limestones are interpreted to be on the left of Figure 16.13d. The biplot vectors indicate that Al, Si and Al are negatively correlated to the PC1 axis, which means larger loadings of clay correspond to the data point being located farther to the left of the origin along the negative PC1 axis.

The non-dolomitized rocks are interpreted to be at the bottom of this graph, as indicated by the biplot vector for  $O_{13}$ . These elements often occur in non-dolomitized rocks. The colored points in Figure 16.13d are associated with either the Carters or the Hermitage formations from which the samples were obtained.

A stand-alone Python lab for these data is available at the CoLab site<sup>10</sup>. This is a lab exercise in the Exercises section at the end of this chapter.

### 16.3.4 PCA Applied to Hyperspectral Images

A digital color camera outputs three  $N \times N$  images from an  $N \times N \times N_b$  color cube, where  $N_b$  is the number of color bands and  $N$  is the number of pixels along one side of a square image. Each image is denoted as a channel. The bands are uniquely assigned one of the red, blue or green colors where the input light is filtered by the appropriate red, green, or blue filter. These images can be blended together to give one color image, which is known as a RGB image. Each color corresponds to a different band of photon frequencies, which are quite wide and overlapping.

To get more information from light, multispectral ( $4 \leq N_b \leq 12$ ) and hyperspectral ( $N_b > 12$ ) cameras<sup>11</sup> record more than three channels. The output is an  $N \times N \times N_b$  data cube, where  $N_b$  can be more than 200 with bands between the far-infrared and ultraviolet regions. A particular problem will demand a particular selection of bands that need to be recorded and processed. The hyperspectral bands can be finely resolved to a width as fine as 10 – 20 nanometers<sup>12</sup>. An example of a hyperspectral image taken by a satellite is shown in Figure 16.14.

A multispectral survey of a portion of south Brazil (see Figure 16.15) was carried out to detect the presence of uranium deposits (de Lima and Marfurt, 2018). The survey airplane carried a gamma ray counter no more than 80 m above the ground to detect the radiation count from surficial radioactive elements that emitted gamma rays by decay of radioactive nuclei near the surface. The target elements were potassium, thorium and uranium so the spectral images were thresholded around the associated energy bands.

Figure 16.16a shows the recorded RGB images for potassium, thorium and uranium channels, which are combined to give the ternary color image. The ternary color chart is shown in the triangle. The plot of energy (frequency) versus counts/s is displayed in Figure 16.16b, and the bands of interest for the target elements are in color. The counts/sec of each channel are shown in Figure 16.17 and are marked by arrows indicating possible economic targets.

<sup>10</sup><https://colab.research.google.com/drive/1mLWN8TdIAMowLsbQ3UQASOeo6-V0qLOz?usp=sharing#scrollTo=iF5JHuyZHmHR>

<sup>11</sup>[https://en.wikipedia.org/wiki/Hyperspectral\\_imaging](https://en.wikipedia.org/wiki/Hyperspectral_imaging)

<sup>12</sup><http://www.adept.net.au/cameras/hyperspectral.shtml>



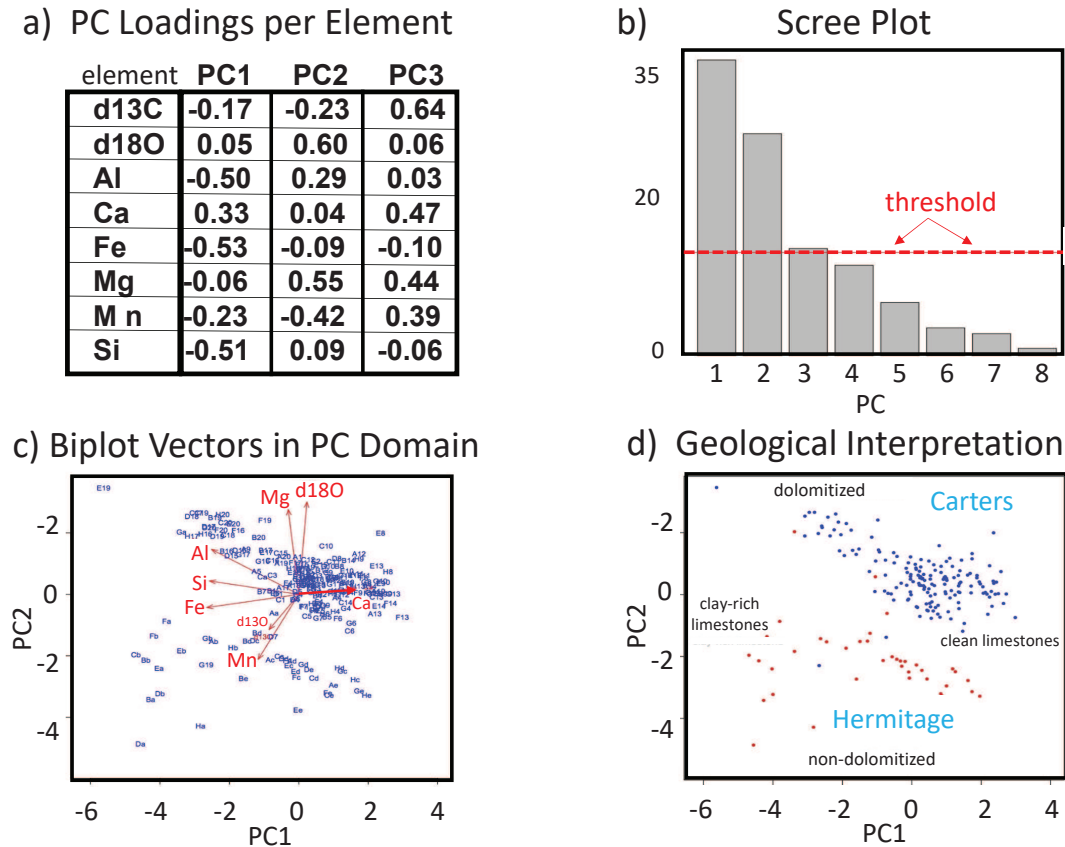


Figure 16.13: a) Table of elemental components, b) scree plot, c) data points plotted in the PC1-PC2 plane along with scaled biplot vectors associated with each of the elemental compositions in a). d) Data points with geological interpretation of the limestone/dolomites. The first two PCs explain about 60% of the data variance. In contrast, PCs with a percent variance less than  $1/8 = 0.125$  (see dashed red line in b) explain less than the average variance associated with all of the PCs. Plots adapted from Stephen Holland's tutorial at <http://strata.uga.edu/8370/handouts/pcaTutorial.pdf>.

## Image Cube from a Hyperspectral Camera

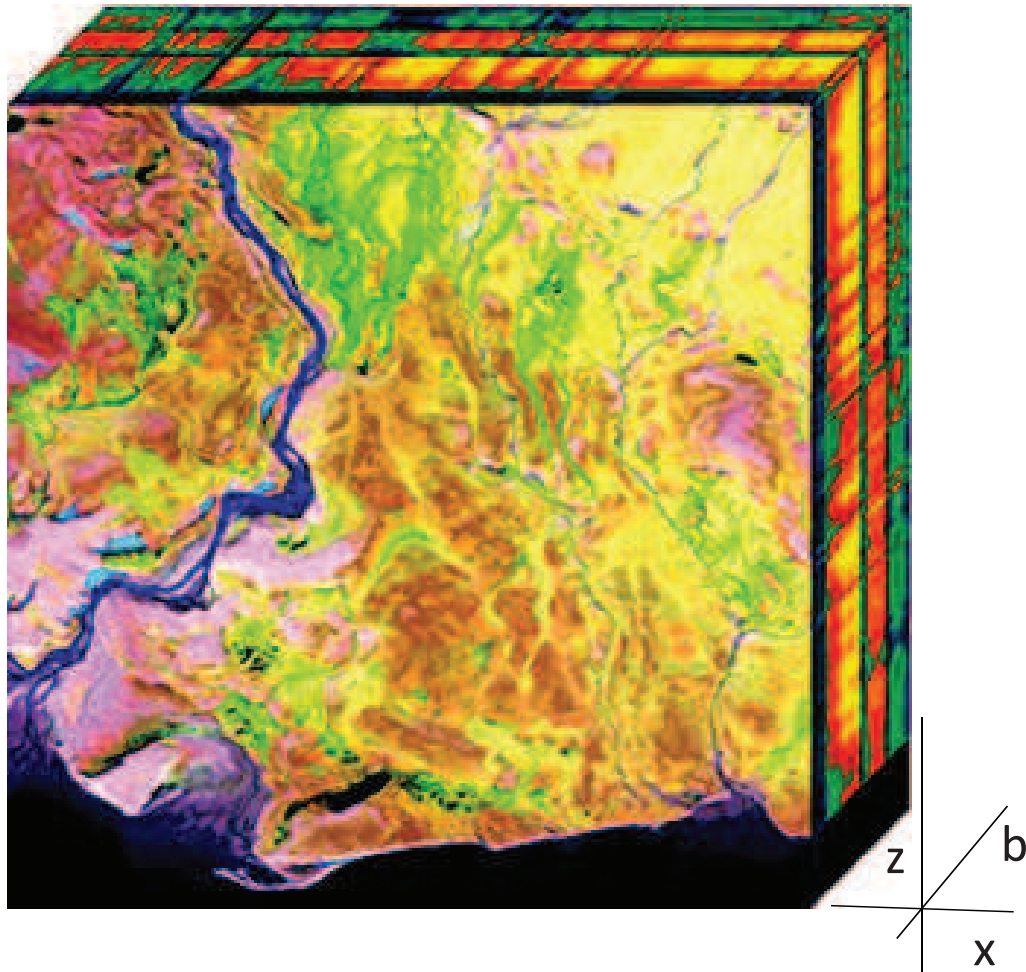
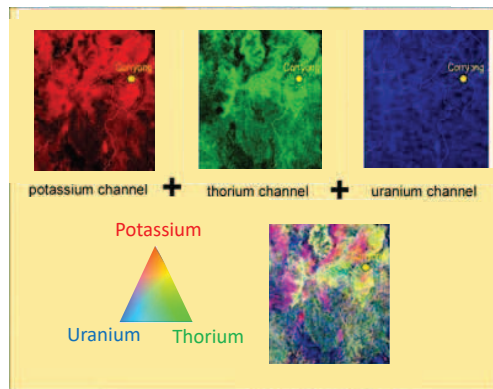


Figure 16.14: Hyperspectral image cube, where  $x$  and  $y$  are the geographic coordinates and  $b$  is the frequency band coordinate. Image from [https://en.wikipedia.org/wiki/Hyperspectral\\_imaging](https://en.wikipedia.org/wiki/Hyperspectral_imaging).



Figure 16.15: (Left) Red survey zone and (right) counts/sec versus gamma-ray energy for responses to uranium, thorium and potassium. Images from de Lima and Marfurt (2018).

#### a) K-Th-Ur Channels



#### b) Counts/sec versus Energy

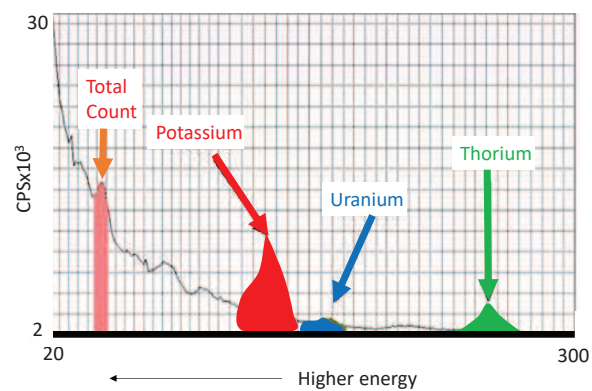


Figure 16.16: a) RGB images of each channel recorded by the gamma-ray spectrometer are above the blended ternary image, and b) depicts the graph of recorded counts/sec versus energy of gamma rays. Images from de Lima and Marfurt (2018).

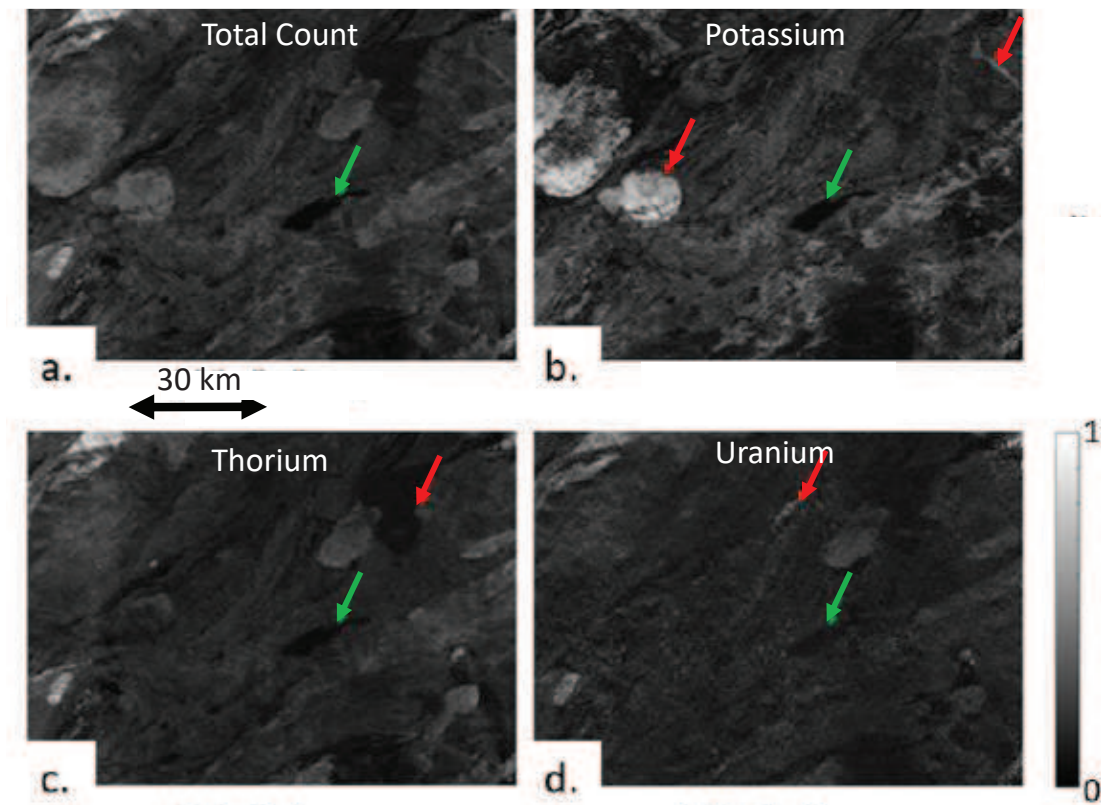


Figure 16.17: Recorded counts/sec for the a) total count (TC), b) potassium (K), c) Thorium (Th) and d) Uranium (Ur) channels. The total count image serves as a reference for the gamma-ray measurements over each pixel. The colored arrows indicate possible economic targets deduced from these images (Lima and Marfurt, 2018).

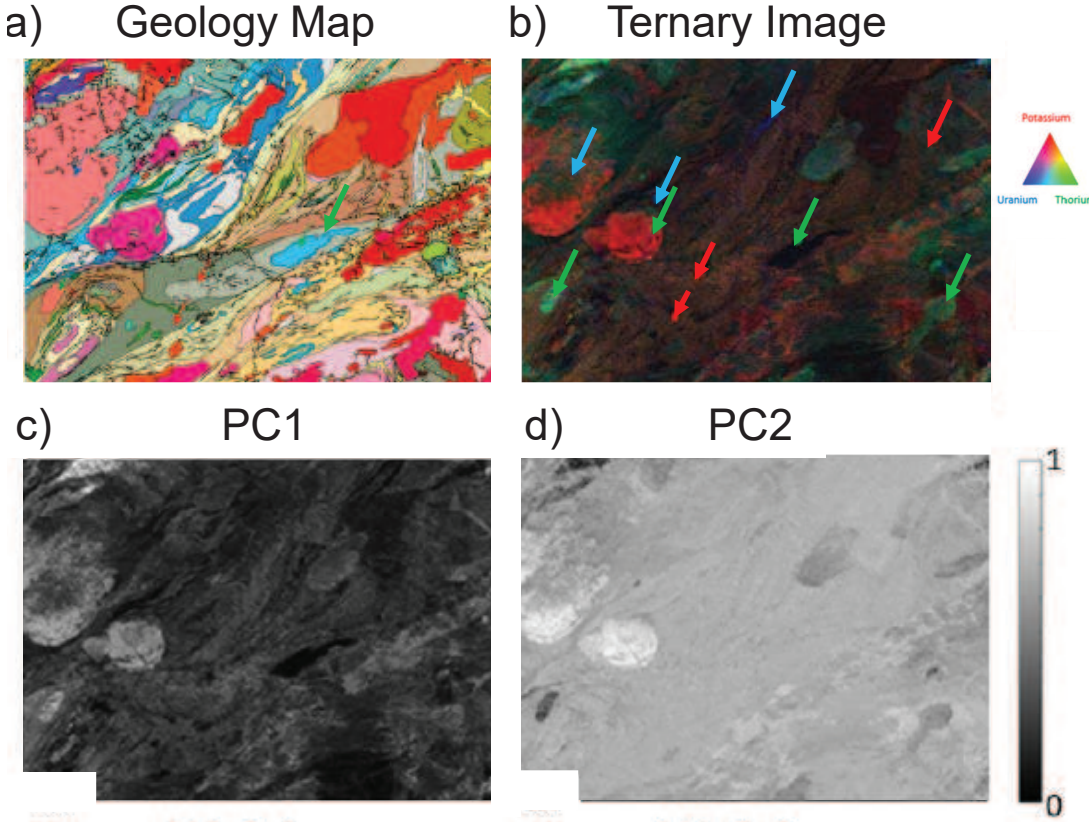


Figure 16.18: a) Geology map, b) ternary image, c) PC1 image and d) PC2 image of the survey area. Each pixel in c) and d) displays the coefficient value associated with the, respectively, 1st and 2nd PC axes (di Lima and Marfurt, 2018). Arrows indicate possible economic targets deduced from these images.

Figure 16.17 displays the images of counts/s for each element, which are not very revealing of anomalous target zones compared to the PC1 and PC2 images in Figure 16.18c-16.18d. These PC images were formed by applying PC analysis to the  $4 \times 1$  vectors of counts/s at each pixel in the recorded data. They are more revealing of possible target opportunities than the original channel images in Figure 16.17.

A K-Means clustering program is applied to the 4-channel data and the PC1 and PC2 components to give the results in Figure 16.19. In this case the spectral distance, i.e.  $d_{jk}$ , of the  $j^{th}$  pixel to its  $k^{th}$  neighbor is

$$d_{jk} = \sqrt{\|\Delta \mathbf{x}_{jk}\|^2}, \quad (16.15)$$

where  $d_{jk}$  is the normalized difference between the  $4 \times 1$  data vectors  $\mathbf{x}^{(j)}$  and  $\mathbf{x}^{(k)}$  at the  $j^{th}$  and  $k^{th}$  pixels, respectively. For Figure 16.19a,  $\mathbf{x}^{(j)}$  is the normalized and demeaned  $4 \times 1$  vector  $(TC_j, K_j, TH_j, Ur_j)$  of counts/s at the  $j^{th}$  pixel. In Figure 16.19b, the four components for the distance calculation were taken from the PC images. For this example,  $K = 9$  and the clusters in the PC1-PC2 plane are more indicative of the potential economic targets than the channel clusters.



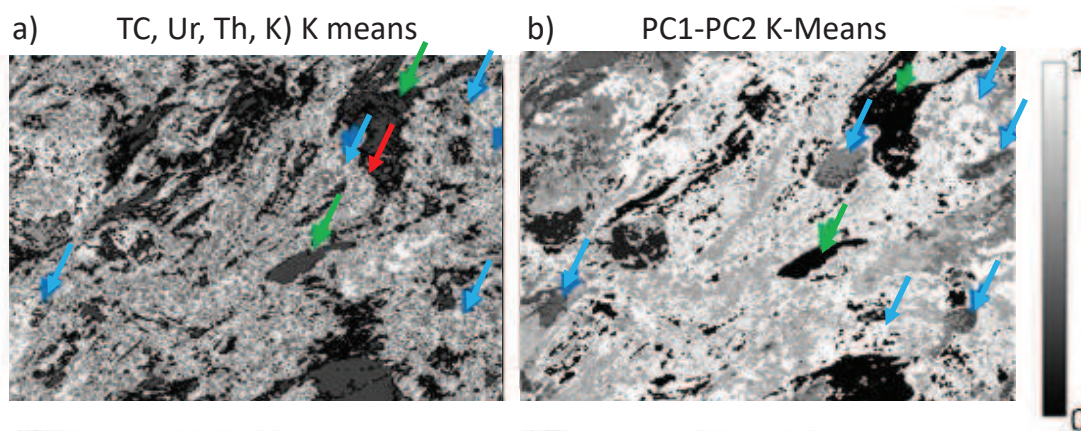


Figure 16.19: The K-Means algorithm is applied to the 4-channel recorded data and the PC1 and PC2 data to give the K-means clusters in a) and b), respectively. Images from de Lima and Marfurt (2018).

Sometimes the Mahalanobis distance is used for the similarity metric, where

$$d_{ij} = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{C}^{-1} (\mathbf{x}_i - \mathbf{x}_j)}, \quad (16.16)$$

where  $\mathbf{C} = \frac{1}{N-1} \mathbf{X} \mathbf{X}^T$  is the data covariance matrix. Division by the data covariance matrix says that the Mahalanobis distance is both unitless and scale-invariant, and takes into account the correlations of the data set. However, in some codes the Mahalanobis distance is computed based on the covariance matrix of the entire data, and thus is mathematically equivalent to the distance in the principal component domain. Some numerical tests suggest that clustering using the distance in the principal component domain gives a much better contrast in the final display, compared to using the regular distance in the original attribute domain<sup>13</sup>. Tests also suggest that K-means does not work well if the Mahalanobis distance per cluster is used compared to forming clusters in the principal component domain of the entire data.

## 16.4 Kernel PCA

The PCA analysis is inappropriate if the components of the data points are not linearly correlated. For example, the sinuous point distributions in Figure 17.13b or concentric points in Figure 16.20a will not allow for a significant reduction in dimension by a PCA rotation of the coordinate axis. In these cases, the correlation between different components is nearly zero so the PCA axes will provide no useful purpose.

In these examples a non-linear PCA-like method can be used to reduce the dimension of the original data. A popular approach is to apply a non-linear transform  $\mathbf{z} = \phi(\mathbf{x})$  to the data to project it to a higher-dimensional space where the blue and red points in Figure 16.20a are now linearly separated in Figure 16.20b using a Gaussian kernel. The kernel trick is used so that a high-dimensional transform can be efficiently used, and is denoted as the kernel PCA (Schölkopf, 1998) method. Instead of finding a new coordinate system that maximizes the margin of labeled points, kernel PCA finds the principal axes coordinate system in the higher-dimensional space that

<sup>13</sup>Communication from Thang N. Ha (ha7675@ou.edu)

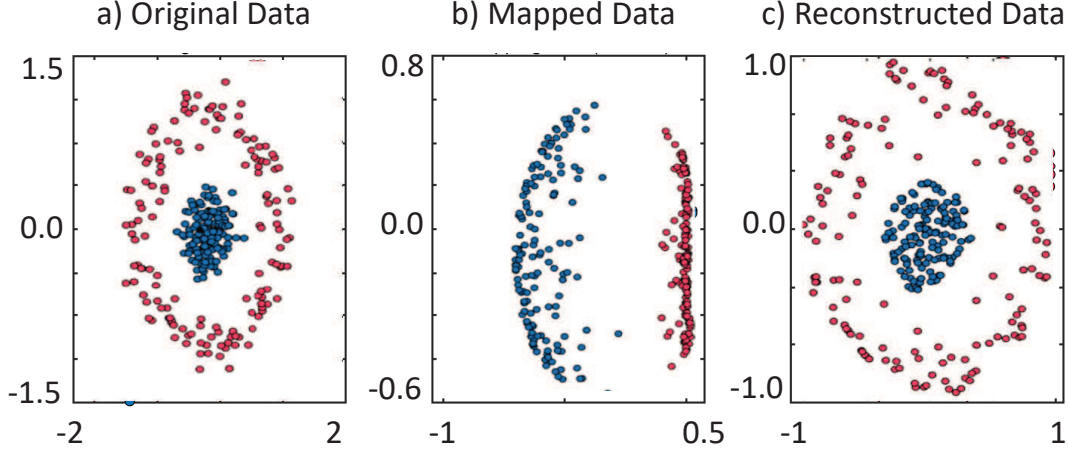


Figure 16.20: Middle image is the result of applying a kernel PCA to the data on the left.

maximizes the variance of unlabeled data along PC axes. Mathematical details for the kernel PCA are described in Schölkopf (1998), Bishop (2006) and Appendix 16.9.

The fourth column of images in Figure 7.11 shows the results of using clustering and a kernel SVM method. In this case a Gaussian kernel is used and correctly separates the two different classes of points. The points in the kernel space can be clustered by a conventional clustering method because they are more separated in this domain. These results are more accurate compared to K-means clustering (Ezuke and Zareian, 2019).

## 16.5 Computational Costs

The computational task of PCA is to find the eigenvalues and eigenvectors of the covariance matrix. If a direct method is used for inverting the  $N \times N$  matrix  $\mathbf{XX}^T$  then the computational cost is  $O(N^3)$ . If the dimension number  $N$  is too big then an iterative method is used to find the eigenvalues and eigenvectors where the computational cost is  $O(KN^2)$ , where  $K$  is the number of iterations. Two popular iterative methods are the Power and the stochastic gradient descent methods (Wu et al., 2018).

The Power method uses the iterative formula

$$\mathbf{a}_1^{k+1} = \mathbf{XX}^T \bar{\mathbf{a}}_1^k, \text{ where } \bar{\mathbf{a}}_1^k = \frac{\mathbf{a}_1^k}{\|\mathbf{a}_1^k\|}, \quad (16.17)$$

where  $\bar{\mathbf{a}}_1^k$  is the  $k^{th}$  iterative estimate of the normalized eigenvector  $\bar{\mathbf{a}}_1$  with the largest eigenvalue. This assumes that the eigenvalues are distinct from one another and  $\mathbf{XX}^T$  is positive definite. As a simple demonstration of this iterative formula, assume  $\mathbf{x} = \bar{\mathbf{a}}_1 + \sum_{i=2}^N \alpha_i \bar{\mathbf{a}}_i$ , where  $\bar{\mathbf{a}}_i$  is the  $i^{th}$  eigenvector of  $\mathbf{XX}^T$ . In this case

$$\mathbf{XX}^T \mathbf{x} = \lambda_1 \bar{\mathbf{a}}_1 + \sum_{i=2}^N \alpha_i \lambda_i \bar{\mathbf{a}}_i = \mathbf{x}. \quad (16.18)$$

Multiplying  $\mathbf{X}\mathbf{X}^T$  by  $\mathbf{x}^1$  gives

$$\mathbf{X}\mathbf{X}^T \mathbf{x}^1 = \lambda_1^2 \bar{\mathbf{a}}_1 + \sum_{i=2}^N \alpha_i \lambda_i^2 \bar{\mathbf{a}}_i = \mathbf{x}^2. \quad (16.19)$$

Repeating this procedure  $K$  times gives

$$\begin{aligned} \mathbf{X}\mathbf{X}^T \mathbf{x}^K &= \lambda_1^{K+1} \bar{\mathbf{a}}_1 + \sum_{i=2}^N \alpha_i \lambda_i^{K+1} \bar{\mathbf{a}}_i, \\ &\approx \lambda_1^{K+1} \bar{\mathbf{a}}_1. \end{aligned} \quad (16.20)$$

because  $\lambda_1^K \gg \lambda_i^K$   $i \in \{2, 3, \dots, N\}$ . If  $\lambda_1$  is slightly larger than  $\lambda_2$  then the convergence rate is very slow. The vector  $\lambda_1^{K+1} \bar{\mathbf{a}}_1$  can be normalized to give  $\bar{\mathbf{a}}_1$ , the eigenvector of  $\mathbf{X}\mathbf{X}^T$  with the largest eigenvalue. This can be plugged into  $\mathbf{X}\mathbf{X}^T \bar{\mathbf{a}}_1 = \lambda_1 \bar{\mathbf{a}}_1$  to give its eigenvalue  $\lambda_1$ .

If  $\bar{\mathbf{a}}_1^k$  is computed, then the eigenvector  $\bar{\mathbf{a}}_2$  with the second largest eigenvalue can be obtained by applying the Power method to

$$\mathbf{a}_2^{k+1} = [\mathbf{I} - \bar{\mathbf{a}}_1 \bar{\mathbf{a}}_1^T] \mathbf{X}\mathbf{X}^T \bar{\mathbf{a}}_2^k \text{ where } \bar{\mathbf{a}}_2^k = \frac{\mathbf{a}_2^k}{\|\mathbf{a}_2^k\|}, \quad (16.21)$$

and  $[\mathbf{I} - \bar{\mathbf{a}}_1 \bar{\mathbf{a}}_1^T] \mathbf{X}\mathbf{X}^T$  is positive semi-definite See Exercise 16.7.3.

The objective function for a single example  $\mathbf{x}$  is

$$\epsilon = \frac{\mathbf{a}^T \mathbf{x} \mathbf{x}^T \mathbf{a}}{\|\mathbf{x}\|^2}, \quad (16.22)$$

which can be used to form the stochastic gradient descent formula (Oja and Karhunen, 1985; Wu et al., 2018)

$$\begin{aligned} \mathbf{a}^{k+1} &= \mathbf{a}^{k+1} + \tilde{\gamma}^k \nabla \epsilon, \\ &= \mathbf{a}^{k+1} + \gamma^k \left( \mathbf{x} \mathbf{x}^T - \frac{|(\mathbf{a}^k, \mathbf{x})|^2}{\|\mathbf{a}^k\|^2} \mathbf{I} \right) \mathbf{a}^k. \end{aligned} \quad (16.23)$$

Equation 16.23 can be used to find the eigenvector PC1 and its eigenvalue, where the iterations proceed as a SGD with a batch size of 1 and one epoch is when all of the examples have contributed to the estimate of  $\mathbf{a}$ . Here,  $\gamma^k = 2\tilde{\gamma}^k / \|\mathbf{a}^k\|^2$  and  $\gamma^k$  is the step length at the  $k^{th}$  iteration (Wu et al., 2018). Equation 16.21 can be used to get PC2 and this procedure repeated to get all of the PCs and eigenvalues. Wu et al. (2018) discuss various tricks for good convergence properties, and Sanger (1993) showed how an iterative algorithm can be used to find the singular values and vectors for SVD. Finally, Fomel et al. (2002) presented an iterative algorithm to get an estimate of the model covariance matrix, which can easily be adapted to getting the inverse to the covariance matrix.

## 16.6 Summary

Data sets are often composed of points in high-dimensional spaces, which are too complex to understand by visual inspection. To reduce this complexity PCA is used to decompose the high-dimensional points into a few subspaces, hopefully just a two- or three-dimensional subspace spanned by the PC1, PC2, and PC3 vectors. This not only saves memory but also facilitates the detection of hidden patterns in the data. The physical meaning of these subspaces can be ascertained by projecting unit vectors of the original axes onto, for example, the PC1 and PC2 unit vectors.



Principal component vectors associated with small eigenvalues are usually associated with noise and so principal component analysis can also be a denoising filter if the noise level is not too high.

A biplot vector for each Euclidean unit vector in the original data space provides a physical interpretation of the points in the principal component subspace. In the gold example, the subspaces were associated with certain combinations of elements, some of which are strongly correlated with the presence of gold.

The problem with PCA is that the data components are assumed to be linearly correlated, which is not always the case. If they are non-linearly correlated such as the point distributions in Figure 17.13b or Figure 16.20a, then a simple rotation of the coordinate axis will not significantly reduce the dimension of the data set. In this case a non-linear PCA analysis can be used such as a neural network PCA (see Chapter 14) or Kernel PCA.

Another problem with PCA is that it cannot fully reconstruct the original data when the observations are corrupted by strong random noise (Candés and Tao, 2005; Candés and Romberg, 2007). An example of the deleterious effects of adding noisy outliers to the data is shown in Figure 16.6c-16.6d. The red circles in Figure 16.6c correspond to outlier points that prevent the PCA from computing the PC1 axis which accounts for the linear trend of blue points. RPCA (Liu et al., 2019a) can, however, recover low-rank data from the data contaminated by complicated but sparse random noise.

A similar problem exists if the level of random noise is too high, as illustrated in Figure 16.21. Here, there are 21 red points in Figure 16.21a that are strongly correlated along the red-dashed line, but are surrounded by 180 randomly located points. In this case, the PCA failed to find the PC1 axis in Figure 16.21b that coincided with the red points.

## 16.7 Exercises

1. Prove that equation 16.21 gives the unit vector for PC2.
2. Show that the estimate of the sample variance  $\hat{\sigma}^2$  is biased as

$$\begin{aligned}
 E[\hat{\sigma}^2] &= \frac{1}{I} \sum_{i=1}^I E[(x_i - \mu)^2], \\
 &= E[x_i^2] - 2E[x_i \hat{\mu}] + E[\hat{\mu}^2], \\
 &= \sigma^2 + \mu^2 - 2\left(\frac{I-1}{I}\mu^2 + \frac{1}{I}(\sigma^2 + \mu^2)\right) + \left(\frac{I^2 - I}{I^2}\mu^2 + \frac{I}{I^2}(\sigma^2 + \mu^2)\right), \\
 &= \frac{I-1}{I}\sigma^2,
 \end{aligned} \tag{16.24}$$

where  $E[x] = \int P(x)xdx$  is the average over the random variable  $\mathbf{x}$  governed by the probability density function  $P(x)$ ,  $\mu$  is the true mean and  $\sigma^2$  is the true variance. Explain why  $1/(I-1)$  must be used in the above formula instead of  $1/I$  to get an unbiased estimate of the variance<sup>14</sup>.

Recall, if  $x_i$  and  $x_j$  are independent random variables for  $i \neq j$ , then  $E[x_i^2] = \sigma^2 + \mu^2$  and

$$E[x_i x_j] = E[x_i]E[x_j] = \mu^2. \tag{16.25}$$

Also, the sample estimates of the mean and variance are given by

$$\hat{\mu} = \frac{1}{I} \sum_{i=1}^I x_i; \quad \hat{\sigma}^2 = 1/I \sum_{i=1}^I (x_i - \hat{\mu})^2. \tag{16.26}$$

<sup>14</sup><https://towardsdatascience.com/why-sample-variance-is-divided-by-n-1-89821b83ef6d>

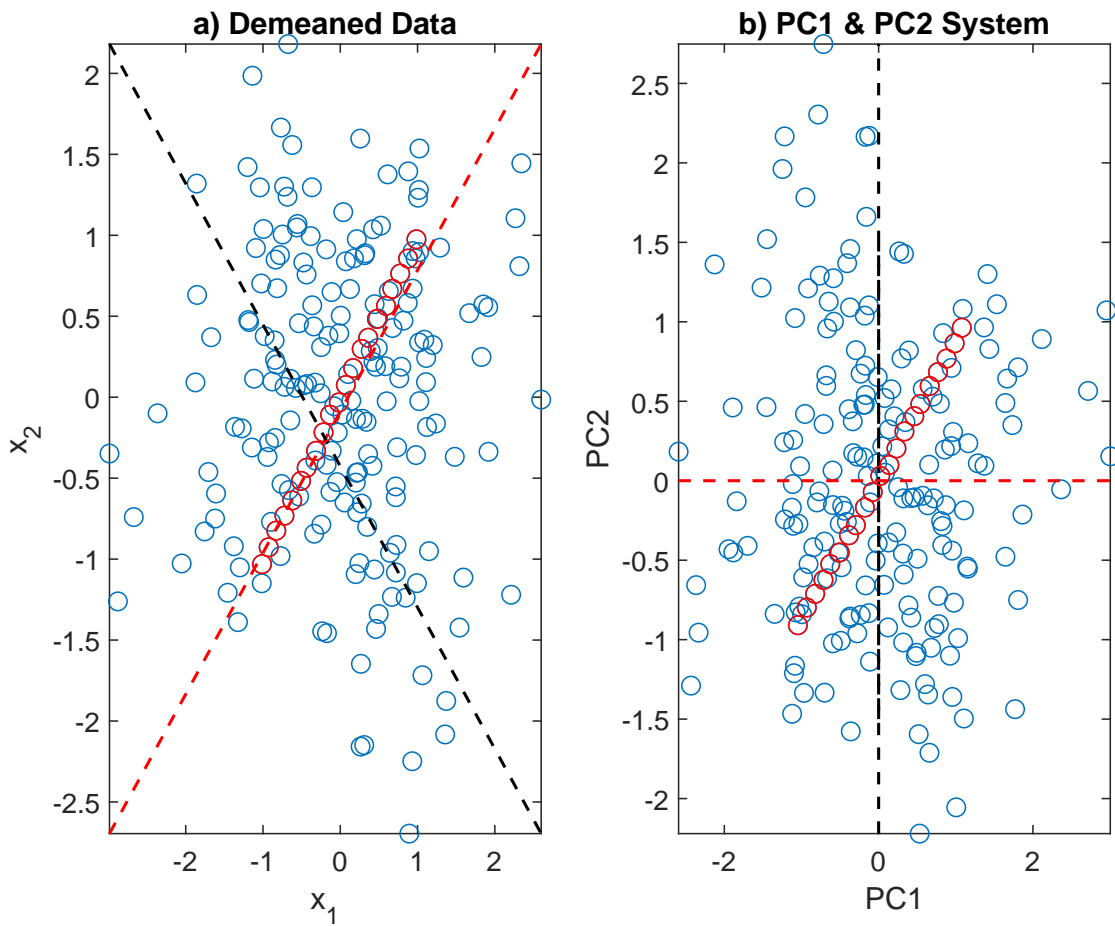


Figure 16.21: a) Linearly correlated points are in red (21 points) and the uncorrelated points (180 points) are in blue. Applying PCA to these data gives the PC coordinate system in b), which does not coincide with the red points. The red- and black-dashed lines in a) are for reference, and are not rotated to be the dashed lines in b).

Show that

$$E[\hat{\mu}] = \frac{1}{I} \sum_{i=1}^I E[x_i] = \mu. \quad (16.27)$$

and

$$E[x_j \hat{\mu}] = \frac{I-1}{I} \mu^2 + \frac{1}{I} (\sigma^2 + \mu^2); \quad E[\hat{\mu}^2] = \frac{I^2 - I}{I^2} \mu^2 + \frac{I}{I^2} (\sigma^2 + \mu^2). \quad (16.28)$$

3. Use the stochastic gradient descent method to solve for  $\mathbf{a}$  using a simple  $2 \times 2$  covariance matrix and compare to the solution by the Power method. Here, the covariance matrix is assumed to be symmetric positive definite.
4. Adjust the MATLAB code in Box 16.2.1 so that the first PC vector is plotted along the horizontal axis.
5. Generate a random set of 200 points evenly distributed around a sphere, where the points are very close to the circle. Compute the covariance matrix and show that the off-diagonal terms are nearly zero. Find the eigenvalues of these data. Explain why they are nearly the same as one another. Generate the scree plot.
6. The kernel PCA is applied to two sinuous curves shown in Figure 16.20. Duplicate their results by downloading the MATLAB code<sup>15</sup>. Now apply a PCA to the data points in the middle panel and replot the points in the PC1-PC2 coordinate system.
7. Generate two narrow ellipses of zero-mean, randomly distributed points. One of the ellipses has a central axis along a line that goes through the origin at an angle of 45 degrees. The other is along the line at an angle of 135 degrees from the horizontal axis. The plot of these points appears as a cross, where the cross is rotated 45 degrees from the horizontal. Now apply the MATLAB PC code to these points. Explain the ambiguous results. Hint: there is a cloud of points whose  $x - y$  components are linearly correlated along the 45 degrees line, and there is another cloud of points with  $x - y$  components linearly correlated along the 135 degrees line. Two clouds of linearly correlated points are not the same as a single cloud with linearly correlated  $x - y$  components.
8. Apply a K-means clustering method for  $k = 2$  to the points in the left and right panels in Figure 16.20. Discuss which domain might be more appropriate for clustering and demonstrate your claims with numerical results.
9. Download the Limestone PCA lab at the CoLab site<sup>16</sup>. Rerun the lab except compare the PCA plot with and without demeaning and with and without normalization of the data.
10. Correlation biplot vectors preserve the correlations among the variables (expressed by the angles between the loading vectors). However, there is some distortion of the distances among the samples. Replot the limestone data except use the correlation, not covariance, matrix to get the data in the PC1-PC2 plane. In this case the correlation matrix is computed by dividing the data components by their standard deviation.
11. Read the truncated SVD documentation at the SciKit site<sup>17</sup>. Go to the CoLab and execute the Python truncated SVD lab with the commands below.

<sup>15</sup>[github.com/iqiukp/Kernel-Principal-Component-Analysis-KPCA](https://github.com/iqiukp/Kernel-Principal-Component-Analysis-KPCA)

<sup>16</sup><https://colab.research.google.com/drive/1mLWN8TdIAMowLsbQ3UQASOeo6-V0qLOz?usp=sharing#scrollTo=iF5JHUYZHmHR>

<sup>17</sup><https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html>

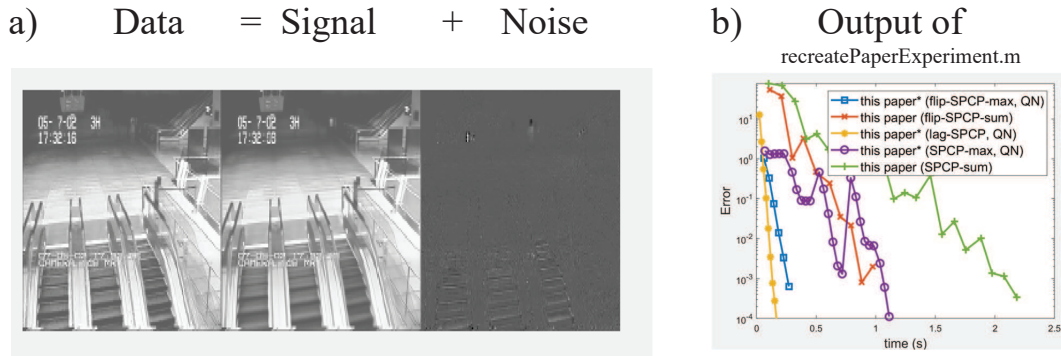


Figure 16.22: Output of RPCA codes a) *demo\_escalatorVideo.m* and b) *recreatePaperExperiment.m* at <https://github.com/stephenbeckr/fastRPCA>.

```
from sklearn.decomposition import TruncatedSVD
from scipy.sparse import random as sparse_random
X = sparse_random(100, 100, density=0.01, format='csr', random_state=42)
svd = TruncatedSVD(n_components=5, n_iter=7, random_state=42)
svd.fit(X)
TruncatedSVD(n_components=5, n_iter=7, random_state=42)
print(svd.explained_variance_ratio_)

print(svd.explained_variance_ratio_.sum())

print(svd.singular_values_)
```

12. Download the truncated SVD codes at the MATLAB<sup>18</sup>. site. Test them on the Yellowstone data.
13. Go to the CoLab site<sup>19</sup> and execute the Python code for classifying a small set of  $2 \times 1$  vectors. Use different input parameters for the SVM. Then apply PCA to these data in the PC domain.
14. Go to the github site <https://github.com/stephenbeckr/fastRPCA> and download the RPCA MATLAB codes. Adjust codes so that a noisy section section is input from the RPCA code rather than the picture of an elevator in Figure 16.22a. Adjust random noise level and explain results. See Aravkin et al. (2014).

## 16.8 Computational Labs

1. Go to the K-means and PCA clustering lab in *LAB1/Chapter.PCA/PCAlab.pdf* and run the PCA lab.
2. Go to the Yellowstone PCA lab in *LAB1/Chapter.PCA/lab1.html* and run the PCA lab.
3. Go to the lab in *LAB1/Chapter.PCA/lab.html* and run the PCA lab.

<sup>18</sup><https://www.mathworks.com/matlabcentral/fileexchange/47132-fast-svd-and-pca>

<sup>19</sup><https://colab.research.google.com/drive/1dM7NgBe720HfCqRGh280CA5X6bs2jOIIm?usp=sharing>

4. Go to the Limestone PCA lab at the CoLab site<sup>20</sup>. item Go to the MATLAB kernel PCA lab at Appendix 16.10. The MATLAB code is described at the CoLab site<sup>21</sup>.
5. Go to the github site <https://github.com/stephenbeckr/fastRPCA> and download the RPCA MATLAB codes. Adjust codes so that a noisy section is input into the RPCA code rather than the picture of an elevator in Figure 16.22a. Adjust random noise level and explain results.
6. Download the truncated SVD codes at the MATLAB<sup>22</sup>. site. Test them on denoising seismic data.

## 16.9 Appendix: Mathematics of Kernel PCA

We now provide the mathematical details for applying PCA to the  $M \times 1$  vectors  $\phi(\mathbf{x}^{(i)})$ ,  $i \in \{1, 2, \dots, I\}$ , where the  $N \times 1$  input vectors  $\mathbf{x}^{(i)}$  are in a lower-dimensional space ( $M > N$ ) than the  $\phi(\mathbf{x}^{(i)})$  vectors. The transformed space, i.e feature space, of  $\phi(\mathbf{x}^{(i)})$  for SVM (see Chapter 7) was employed in order to transform a wiggly margin surface in  $\mathbf{x}$ -space to a hyperplane in the  $\phi(\mathbf{z})$  domain. This is also useful because the points along a curved surfaces in  $\mathbf{x}$  are linearly uncorrelated, so the non-linear transform can map them to a line perpendicular to the PC1 axis in the  $\phi(\mathbf{z})$  domain. Thus the components of the transformed points  $\phi(\mathbf{x})$  will be linearly correlated along the PC1 axis.

Assume the  $N \times 1$  data vectors  $\mathbf{x}^{(i)}$  are demeaned so that the  $N \times N$  covariance matrix  $\mathbf{C}$  in equation 16.4 can be represented by

$$\mathbf{C} = \frac{1}{I-1} \sum_{i=1}^I \mathbf{x}^{(i)} \mathbf{x}^{(i)T}, \quad (16.29)$$

where  $I$  is the number of data samples. Let the non-linear transformation to a higher-dimensional space give  $M \times 1$  vectors  $\mathbf{z}^{(i)} = \phi(\mathbf{x}^{(i)})$  that are demeaned as well. Therefore, the higher-dimensional  $M \times M$  covariance matrix is

$$\tilde{\mathbf{C}} = \frac{1}{I-1} \sum_{i=1}^I \phi(\mathbf{x}^{(i)}) \phi(\mathbf{x}^{(i)})^T, \quad (16.30)$$

where  $M > N$ . The  $M$  eigenvectors of the  $M \times M$  matrix  $\tilde{\mathbf{C}}$  satisfy

$$\tilde{\mathbf{C}} \mathbf{v}_k = \lambda_k \mathbf{v}_k, \quad k \in \{1, 2, \dots, M\}. \quad (16.31)$$

Plugging equation 16.30 into equation 16.31 gives

$$\frac{1}{I-1} \sum_{i=1}^I \phi(\mathbf{x}^{(i)}) \overbrace{[\phi(\mathbf{x}^{(i)})^T \mathbf{v}_k]}^{\alpha_k^{(i)}} = \lambda_k \mathbf{v}_k, \quad k \in \{1, 2, \dots, M\}, \quad (16.32)$$

<sup>20</sup><https://colab.research.google.com/drive/1mLWN8TdIAMowLsbQ3UQASOeo6-V0qLOz?usp=sharing#scrollTo=iF5JHUYZHmHR>

<sup>21</sup>See [https://colab.research.google.com/drive/16\\_Jxp9K6caxF\\_ZaR9JaY2u7pF8K33BWFV?usp=sharing#scrollTo=hmQoAhO7RU9U](https://colab.research.google.com/drive/16_Jxp9K6caxF_ZaR9JaY2u7pF8K33BWFV?usp=sharing#scrollTo=hmQoAhO7RU9U)

<sup>22</sup><https://www.mathworks.com/matlabcentral/fileexchange/47132-fast-svd-and-pca>

which says that the  $M \times 1$  eigenvector  $\mathbf{v}_k$  can be written as

$$\mathbf{v}_k = \sum_{j=1}^I \alpha_k^{(j)} \phi(\mathbf{x}^{(j)}), \quad k \in \{1, 2, \dots, M\}. \quad (16.33)$$

Here,

$$\alpha_k^{(i)} = \frac{1}{I-1} \phi(\mathbf{x}^{(i)})^T \mathbf{v}_k, \quad (16.34)$$

is the  $i^{th}$  component of the  $I \times 1$  vector  $\boldsymbol{\alpha}_k$  for  $k \in \{1, 2, \dots, M\}$ .

The kernel-PCA coordinate system is a new coordinate system in the  $z$ -domain that maximizes the variance of unlabeled data; no labeled data are needed. This compares to the goal of kernel-SVM which is to find the new coordinate system that maximizes the margin of the labeled data. For kernel SVM, labeled data are required

**New coordinate system that maximizes the variance of data components along the principal component axes.** Substituting equation 16.33 into equation 16.32 gives

$$\frac{1}{I-1} \sum_{i=1}^I \phi(\mathbf{x}^{(i)}) \overbrace{\left[ \sum_{j=1}^I \phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(j)}) \alpha_k^{(j)} \right]}^{[\mathbf{K}\boldsymbol{\alpha}_k]_i} = \lambda_k \sum_{j=1}^I \alpha_k^{(j)} \phi(\mathbf{x}^{(j)}), \quad k \in \{1, 2, \dots, M\} \quad (16.35)$$

where the kernel of the  $I \times I$  matrix  $\mathbf{K}$  is  $k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(j)})$ . The kernel trick for SVM forbids any explicit reference to  $\phi(\mathbf{x}^{(i)})$  and only gives expressions in terms of the easily computed kernel  $k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$ . This can be accomplished by multiplying both sides of equation 16.35 by the  $1 \times M$  vector  $\phi(\mathbf{x})^T$  to give

$$\frac{1}{I-1} \sum_{i=1}^I \overbrace{\phi(\mathbf{x})^T \phi(\mathbf{x}^{(i)})}^{\kappa_{xi}} [\mathbf{K}\boldsymbol{\alpha}_k]_i = \lambda_k \sum_{j=1}^I \overbrace{\phi(\mathbf{x})^T \phi(\mathbf{x}^{(j)})}^{\kappa_{xj}} \alpha_k^{(j)}, \quad (16.36)$$

where  $\kappa_{xi} = \phi(\mathbf{x})^T \phi(\mathbf{x}^{(i)})$ . If  $\mathbf{x} = \mathbf{x}^{(i')}$  is evaluated at  $i' \in \{1, 2, \dots, I\}$  then the above equation, after multiplying by  $I-1$ , becomes

$$\mathbf{K}^2 \boldsymbol{\alpha}_k = \lambda_k (I-1) \mathbf{K} \boldsymbol{\alpha}_k. \quad (16.37)$$

which can be rearranged as

$$\mathbf{K}[\mathbf{K}\boldsymbol{\alpha}_k - \lambda_k (I-1) \boldsymbol{\alpha}_k] = 0. \quad (16.38)$$

Here,  $\kappa_{ij}$  represents the coefficient for the  $i^{th}$  row and  $j^{th}$  column of  $\mathbf{K}$ . The solutions to this system of equation are the eigenvectors of

$$\mathbf{K}\boldsymbol{\alpha}_k = \lambda_k (I-1) \boldsymbol{\alpha}_k. \quad (16.39)$$

Thus, the projection of an unknown data point  $\mathbf{x}$  along the  $k^{th}$  principal component vector can be

found by

$$\begin{aligned}
 y_k &= \phi(\mathbf{x})^T \overbrace{\mathbf{v}_k}^{\text{eqn. 16.33}} = \sum_j \alpha_k^{(j)} \phi(\mathbf{x})^T \phi(\mathbf{x}^{(j)}), \\
 &= \sum_j \alpha_k^{(j)} k(\mathbf{x}, \mathbf{x}_j),
 \end{aligned} \tag{16.40}$$

where the examples of points  $(y_1^{(i)}, y_2^{(i)})$  in PC1-PC2 plane are depicted in Figure 16.20b.

For PCA to work, the data must be demeaned. Demeaning the data in the input domain does not imply that they are demeaned in the  $\mathbf{z}$  domain. Therefore, the  $\mathbf{z}$ -domain data must be demeaned as

$$\tilde{\phi}(\mathbf{x}^{(n)}) = \phi(\mathbf{x}^{(n)}) - \frac{1}{I} \sum_{i=1}^I \phi(\mathbf{x}^{(i)}), \tag{16.41}$$

where  $\frac{1}{I-1} \approx \frac{1}{I}$  for  $I \gg 0$ . The corresponding Gram matrix elements are given by (Bishop, 2006)

$$\begin{aligned}
 \tilde{k}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) &= \tilde{\phi}(\mathbf{x}^{(i)})^T \tilde{\phi}(\mathbf{x}^{(j)}), \\
 &= \phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(j)}) - \frac{1}{I} \sum_{l=1}^I \phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(l)}) \\
 &\quad - \frac{1}{I} \sum_{l=1}^I \phi(\mathbf{x}^{(l)})^T \phi(\mathbf{x}^{(j)}) + \frac{1}{I^2} \sum_{n=1}^I \sum_{m=1}^I \phi(\mathbf{x}^{(n)})^T \phi(\mathbf{x}^{(m)}), \\
 &= k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) - \frac{1}{I} \sum_{m=1}^I k(\mathbf{x}^{(m)}, \mathbf{x}^{(j)}) - \frac{1}{I} \sum_{l=1}^I k(\mathbf{x}^{(i)}, \mathbf{x}^{(l)}) + \frac{1}{I^2} \sum_{n=1}^I \sum_{m=1}^I k(\mathbf{x}^{(n)}, \mathbf{x}^{(m)}).
 \end{aligned}$$

This can be represented in matrix notation as

$$\tilde{\mathbf{K}} = \mathbf{K} - \mathbf{1}_I \mathbf{K} - \mathbf{K} \mathbf{1}_I + \mathbf{1}_I \mathbf{K} \mathbf{1}_I, \tag{16.43}$$

where  $\mathbf{1}_N$  is the matrix with all elements equal to  $1/I$ . The matrix  $\tilde{\mathbf{K}}$  that should be evaluated to determine the eigenvectors and eigenvalues in the  $\mathbf{z}$ -domain.

The linear PCA can be applied to the points on the in the  $\mathbf{z}$ -domain. However, if the PCA representation of the points is approximated by a small sum of weighted eigenvectors, then the PCA points cannot always map to the points in the original domain. This is because the principal-component planes in the  $\mathbf{z}$ -domain don't necessarily lie on the manifold of points  $\mathbf{z}^{(i)}$  in this higher-dimensional space (Bishop, 2006). In this case, projections of points  $\mathbf{z}^{(i)}$  onto a PC1-PC2 plane don't necessarily map back to one of the original points  $\mathbf{x}^{(i)}$ . Techniques have been developed to overcome this problem (Bakir et al., 2004).

## 16.10 Appendix: MATLAB Kernel PCA CoLab

The following is the explanation of the MATLAB code<sup>23</sup> for performing Kernel PCA on the data sets in Figure 16.20a and 16.20a. This lab was modified from the original code at the GITHUB repository<sup>24</sup>.

<sup>23</sup>[colab.research.google.com/drive/16-Jxp9K6caxFZaR9JaY2u7pF8K33BWFV?usp=sharing#scrollTo=hmQoAhO7RU9U](https://colab.research.google.com/drive/16-Jxp9K6caxFZaR9JaY2u7pF8K33BWFV?usp=sharing#scrollTo=hmQoAhO7RU9U)

<sup>24</sup><https://github.com/iqiukp/Kernel-Principal-Component-Analysis-KPCA>

There are two main classes are defined in the MATLAB code. The 1st is the 'kernelMatrix' which defines the type of kernel function (e.g. gauss, exponential, etc) being used and the 2nd is the main 'KernelPCA' which performs tasks like Non-linear Dimensionality Reduction (and Reconstruction). The details of these two classes are defined below (with details about the parameters they take as inputs):

- 'KernelMatrix':

Computes the kernel matrix using different kernel functions defined below.

```
kernel = Kernel('type', 'gauss', 'width', 2);
kernel = Kernel('type', 'exp', 'width', 2);
kernel = Kernel('type', 'linear', 'offset', 0);
kernel = Kernel('type', 'lapl', 'width', 2);
kernel = Kernel('type', 'sigm', 'gamma', 0.1, 'offset', 0);
kernel = Kernel('type', 'poly', 'degree', 2, 'offset', 0);
```

- 'KernelPCA':

Performs non-linear dimensionality reduction (and reconstruction). An object is first defined using this class specifying the application and parameter. e.g. 'kpca = KernelPCA(parameter);'. The parameters take on several values defined below.

Structures Description of 'parameter'

application:	dimensionality reduction (dr) or fault detection (fd)
dim:	dimensionality
tol:	tolerance for eigenvalues
alpha:	hyperparameter of the ridge regression that learns the reconstruction
explained:	percent variability explained by principal components
kernel:	kernel function
significanceLevel:	significance level (fault detection)
theta:	experience parameter of fault diagnosis
display:	display the results

## Dimensionality Reduction (and Reconstruction)

The following are the steps to perform dimensionality reduction:

- We first have to load the dataset provided using the command:

```
% load data
load('.\data\circle.mat')
X = data(:, 1:2);
label = data(:, 3);
'''
```

The labels are just extracted to be used for visualization during plotting. The plot for the data is shown in Figure 16.20a. The Kernel Function is next set using:



```
'kernel = Kernel('type', 'gauss', 'width', 0.5);'
```

Here the type of kernel used is "gauss". The very first input to the class 'Kernel' defines the variable (i.e. "type" in this case) and the input that follows it assigns the value to it (i.e. "gauss").

Other Kernel types can also be used and the ones supported are defined in the cell above. Example of others can be polynomial ("poly"), Linear ("linear"), exponential ("exp"), etc.

The next step is to define the parameters to the main "KernelPCA" class and define the object which in turn will be used to perform PCA.

1. First parameters are defined using the command:

```
"parameter = struct('application', 'dr', 'dim', 2, 'kernel', kernel);"
```

2. Here we have set the "application" that we want to perform, i.e. "dr" (which stands for dimensionality reduction). Then the dimensions are defined as '2' and finally we set the "kernel" defined in the previous step.
3. The object named "kpca" is then defined and parameters set in the above step are passed to it:

```
'kpca = KernelPCA(parameter);'
```

4. The final step is to train the KPCA model using the data and visualizing the results.

```
'''
% train KPCA model using given data
X_map = kpca.train(X);
% Visualize the mapping data
Visualization.map(X, X_map, label);
'''
```

5. Plot after performing KPCA is shown in Figure 16.20b.
6. Reconstruction can only be done to the above plotted dataset. It is done using the command :

```
'X_re = kpca.reconstruct;'
```

Plot for the reconstructed data is shown in Figure 16.20c.

Similar dimensionality reduction can be done for the banana dataset by changing the load data command to

```
''load('.\data\banana.mat')''
```

The results are shown in Figure 16.20.

```
% Examples for dimensionality reduction based on KPCA
% -----
```

```
clc
clear all
close all
addpath(genpath(pwd))
```

```

% load data
load('.\data\circle.mat')
X = data(:, 1:2);
label = data(:, 3);
% set kernel function
kernel = Kernel('type', 'gauss', 'width', 0.5);
% parameter setting
parameter = struct('application', 'dr', 'kernel', kernel);
% build a KPCA object
kpca = KernelPCA(parameter);
% train KPCA model using given data
X_map = kpca.train(X);
% reconstruct the mapping data
X_re = kpca.reconstruct;
% Visualize the reconstruction
Visualization.reconstruct(X, X_map, X_re, label);
'''

```

Running the above code gives the output:

```

'''
*** KPCA model training finished ***
dimensionality      = 122
time cost           = 0.2822 s
'''

```

## Chapter 17

# Clustering Algorithms

Cluster analysis is the formal study of methods and algorithms for grouping, or clustering, objects according to measured or perceived intrinsic characteristics or similarity (Jain, 2010). It is similar to logistic regression in grouping data into different classes, except the input data do not have to be labeled and there can be many different clusters. Thus, a clustering algorithm is an unsupervised learning method that avoids the cost of supervised labeling of large data sets. It is often applied to data in order to uncover hidden patterns within a dataset, such as the Yellowstone data in Figure 17.1a that is segmented into the two clusters in Figure 17.1b and three clusters in Figure 17.1c. A table of different unsupervised cluster methods in (Károly et al., 2018) is listed in Figure 17.2.

One of the most widely used clustering algorithms is the K-means cluster method (see Figure 17.3) proposed more than 50 years ago (Steinhaus, 1956; Lloyd, 1982; Ball and Hall, 1965; and MacQueen, 1965). It is simple and very effective in machine learning studies for simple point distributions, and its weighted and regularized versions will be highlighted by geoscience examples in this chapter. For more complex point distributions, other clustering algorithms such as density-based spatial clustering (DBSCAN) and support vector clustering can be used.

### 17.1 Introduction

According to Jain (2010) there are more than a thousand different cluster algorithms, many of them tailored to a different scientific algorithm. They differ in the choice of their definition of distance between points, the objective function, the assignment of probability properties and heuristics. Many of them are unsupervised, although some assign labeling to a small number of points to assist in the final assignment of clusters. In this case this would be a semi- or weakly-supervised learning algorithm.

What is the definition of a cluster of points? According to Jain (2010), we start with a set of  $N$  points, each being a  $D$ -dimensional feature vector, and there are  $K$  groups of these points. The points in any group have a measure of high similarity to one another, but have low similarity to members outside that group. Similarity is an arbitrarily defined quantity, but the points in a specified group in Figure 17.4 are similar, for example, in their close geographic distance to one another.

There are two groups of clustering algorithms: hierarchical and partitional. In partitional clustering, the user defines the fixed number  $K$  of clusters and the algorithm assigns to each point the nearest cluster. The hierarchical algorithms seek to produce a hierarchy of clusters for a range of  $K$  values of which the *best* one is selected.

The labeling of points was first introduced in early chapters on supervised learning. In this case, a learning algorithm was used, such as SVM or logistic regression, to classify the points in

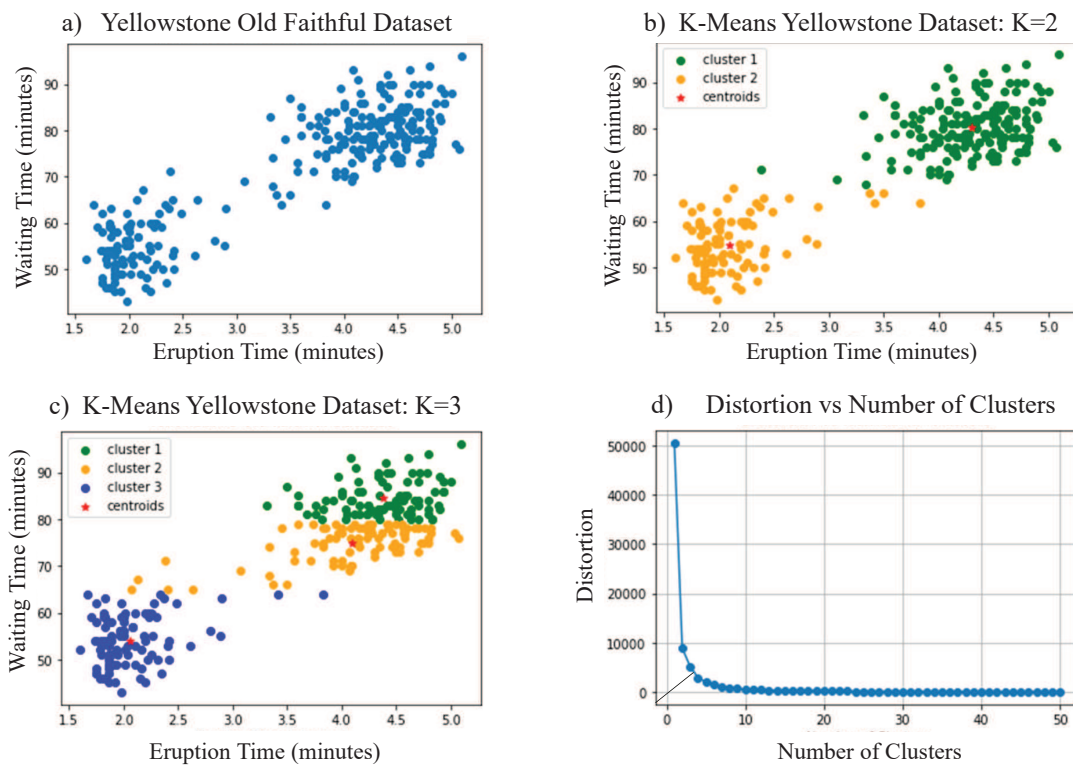


Figure 17.1: Yellowstone data a) as input and labeled by K-means clustering for b)  $K = 2$  and c)  $K = 3$ . Plot in d) is the distortion ( $\propto$  RMS error) versus the number of clusters. The duration time of the eruptions and the waiting time between eruptions are the variables along the, respectively, horizontal and vertical axes in a), b) and c).

Class of Cluster Method	Cluster Method	Reference
Distance Measure	K-means Clustering	Chapter 17.2
	Hierarchical Clustering	Chapter 17.3, Károly et al. (2018)
	Information Weighted Clusters	Chapter 17.4
	Support Vector Machines	Chapter 17.7
	Spectral Clustering	Károly et al. (2018)
	Decision Trees	Károly et al. (2018)
	Fuzzy Clustering	Chapter 17.5
	DBSCAN	Chapter 17.8
	K-Nearest Neighbor Clustering	Chapter 17.9
Class of Cluster Method	Cluster Method	Reference
Neural Network	Self-Organizing Map	Chapter 17.10
	Autoencoders	Chapter 14
	Neural Network	Chapters 4-9
	Generative Models	Chapter 21
Class of Cluster Method	Cluster Method	Reference
Statistical	Expectation Minimization	Chap. 21 & Károly (2018)

Figure 17.2: Classes of cluster methods. Table partly adapted from Károly et al. (2018).

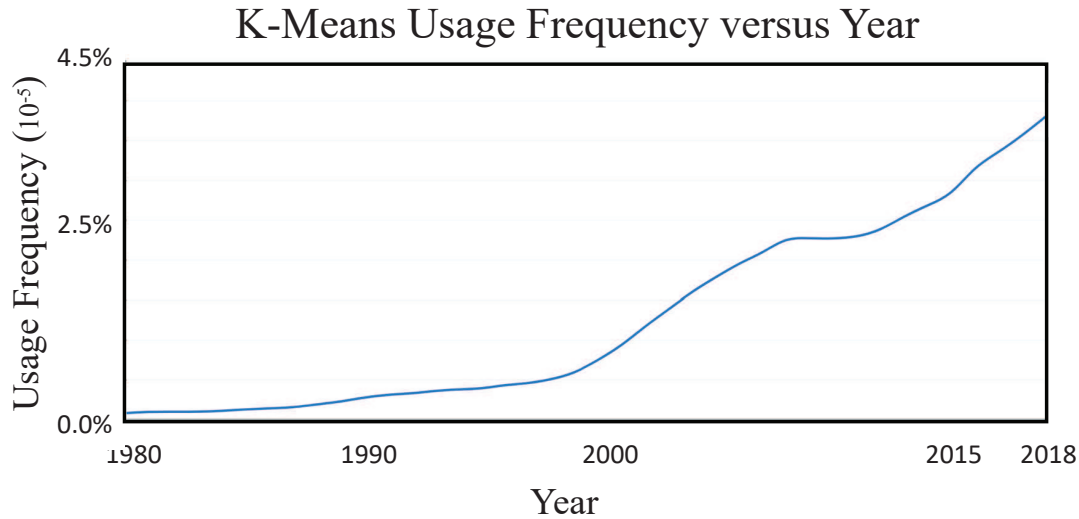


Figure 17.3: Usage of the word *K-means* plotted against calendar year. Courtesy of <https://books.google.com/ngrams/>.

Figure 17.4a. A semi-supervised learning algorithm for the data in Figure 17.4b could be devised so that a few points in each cluster are already labeled so they can be used to automatically classify the unlabeled points. For example, the unlabeled point is assigned the group identity of the nearest labeled point. The points associated with a partially constrained clustering algorithm are depicted in Figure 17.4c. Here, some points are known not to belong to certain groups so a constraint is imposed that is honored by the clustering algorithm. For example, the distance between the points in the dissimilar groups can be used as a soft threshold such that the distance between two points in the same group should not exceed this soft threshold, otherwise its candidacy is assigned a lower grade. Finally, we have the completely unsupervised clustering problem illustrated in Figure 17.4d. In this case, the weighted distance between one point and another is used as a criterion for eligibility in a separate cluster.

The problem shown in Figure 17.4d is the one that is largely addressed in this chapter: clustering unlabeled points by unsupervised learning. Due to its simplicity, empirical success and its widespread use in the machine learning community, this chapter emphasizes the K-means clustering algorithm.

## 17.2 K-means Clustering

A clustering strategy that is widely used for hierarchical or partitional clustering<sup>1</sup> is K-means clustering. It is very simple to use, the number of clusters should be specified by the user, and works well if the clusters consist of points that are nearly uniform in density and are bounded by convex boundaries.

Let  $\mathbf{x}_{ij}$  be  $j^{\text{th}}$   $D \times 1$  feature vector associated with the  $i^{\text{th}}$  cluster, where there are  $K$  clusters.

<sup>1</sup>Partitional clustering defines the number of clusters before running the algorithm. Hierarchical clustering generates a suite of cluster maps during the iterations, each map typically consists of a different number of clusters. The user selects the best after the algorithm is finished.

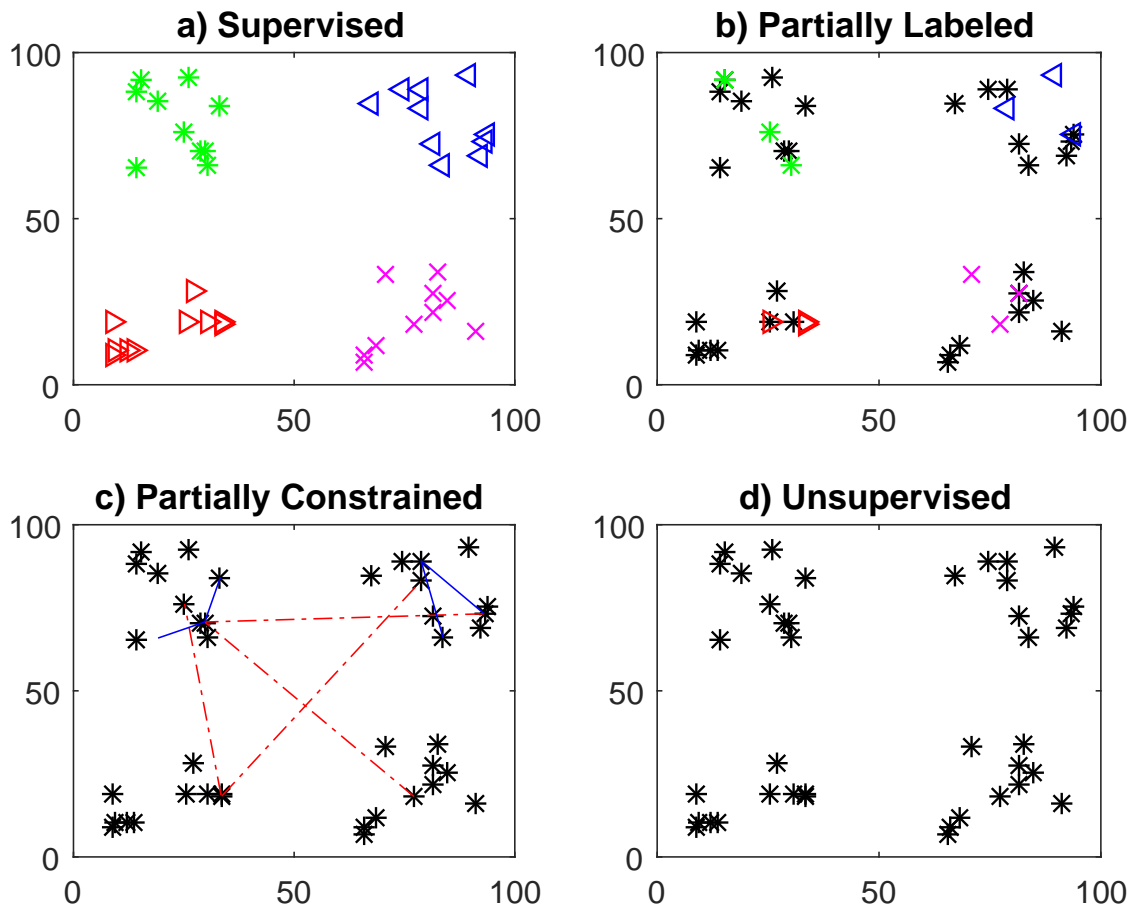


Figure 17.4: Different groups of points: a) Supervised learning where points in different clusters are labeled with different colored symbols, b) partially labeled where only a few non-black points are labeled, c) partially constrained where the dashed red (solid blue) lines forbid (enforce) the points to be in the same cluster, and d) completely unlabeled or unconstrained points. Figure modified from Jain (2010).

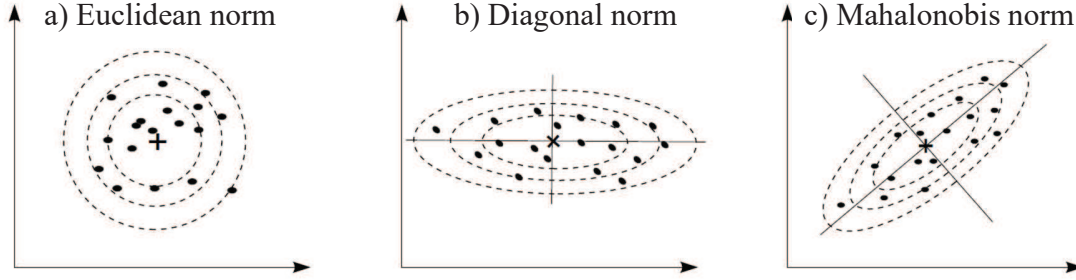


Figure 17.5: Cluster shapes induced by the a)  $L_2$  norm and the Mahalanobis norm for b) diagonal and c) fully-populated  $\Sigma^{-1}$  matrices. The Mahalanobis norm is the same as the  $L_2$  norm except  $\|\mathbf{x}\|_2 = \sqrt{\mathbf{x}^T \mathbf{x}}$  is replaced by  $\sqrt{\mathbf{x}^T \Sigma^{-1} \mathbf{x}}$ , where  $\Sigma^{-1}$  is the inverse to the covariance matrix. Figure modified from <https://homes.di.unimi.it/~valentini/SlideCorsi/Bioinformatica05/Fuzzy-Clustering-lecture-Babuska.pdf>.

The mean centroid point  $\mathbf{C}_i$  of the  $i^{th}$  cluster is defined as

$$\mathbf{C}_i = \frac{1}{m_i} \sum_{j=1}^{m_i} \mathbf{x}_{ij}, \quad (17.1)$$

where  $m_i$  is the number of points in the  $i^{th}$  cluster. The points  $\mathbf{x}_{ij}$  in the  $i^{th}$  cluster are defined as having the smallest distance  $L^p$  distance<sup>2</sup> between that point and  $\mathbf{C}_i$  compared to any other centroid point  $\mathbf{C}_j$  for  $j \neq i$ . Typically K-means uses the  $L_2$  norm where  $p = 2$ , where the choice of the norm index  $p$  and the norm weight can determine the shape of the cluster. For example, the  $L_2$  norm tends to form circular clusters such as the one in Figure 17.5a while the Mahalanobis norm tends to form ellipses with either horizontal or arbitrary orientations shown in Figure 17.5b-17.5c. If the uncertainty, i.e. standard deviation  $\sigma \gg 0$ , is high for a point then it will be downweighted by  $1/\sigma$  and so have little influence in shaping the elliptical contour of its associated Mahalanobis cluster.

The averaged sum of the squared distances between the  $i^{th}$  centroid point  $\mathbf{C}_i$  and the points in the  $i$  cluster is given as

$$d_i = \frac{1}{m_i} \sum_{j=1}^{m_i} \|\mathbf{C}_i - \mathbf{x}_{ij}\|^2. \quad (17.2)$$

The goal is to find the optimal distribution of cluster points  $\mathbf{C}_k$  such that the objective function  $\epsilon$  is minimized:

$$\epsilon = \frac{1}{2} \sum_{i=1}^K \frac{1}{m_i} \sum_{j=1}^{m_i} \|\mathbf{C}_i - \mathbf{x}_{ij}\|^2. \quad (17.3)$$

There are four steps to the K-means clustering algorithm where the number of clusters is specified.

1. Select an initial configuration of  $K$  clusters. Compute the centroid point  $\mathbf{C}_i$  for each cluster.

<sup>2</sup>The  $L^p$  distance between two points  $\mathbf{x} = (x_1, x_2, \dots, x_N)^T$  and  $\mathbf{y} = (y_1, y_2, \dots, y_N)^T$  is  $\|\mathbf{x} - \mathbf{y}\|_p = (\sum_{n=1}^N |x_n - y_n|^p)^{1/p}$ .



2. Compute the distances  $\|\mathbf{C}_i - \mathbf{x}_{ij}\|$ , and reassign points to the nearest cluster. If the elements in each feature vector have different units then they should be normalized, for example, by their standard deviations.
3. Compute the new centroid points from the newly assigned points.
4. Repeat steps 2 and 3 until convergence. This procedure can be adapted to the hierarchical approach where an outer loop can be added where the number of clusters is either increased or decreased.

A problem with the above approach is that it can get stuck in a local minimum. Sometimes this difficulty is remedied by using different starting configurations to determine if they converge to the same answer. This same procedure is used to test for local minima problems with many non-linear optimization methods. Another way to mitigate this problem is to specify different numbers of clusters and choose the final cluster configuration that best suits our goal.

The K-means is a hard clustering method, which means that it will associate each point to one and only one cluster. This does not allow any uncertainty in assigning points to a cluster. K-means clustering usually works well when there are convex clusters of nearly equal size, but otherwise can have difficulty. In addition, there is no uncertainty measure or probability that tells us how much a data point is associated with a specific cluster. To remedy these problems, a soft clustering approach such as Information Weighted Clustering in section 17.4, Fuzzy clustering in section 17.5 or Gaussian Mixtures (Bishop, 2006) can be used. In addition, non-linear methods such as a Kernel PCA (see Chapter 16) can be used to separate the data points, and a suitable clustering method can be used in the transformed space.

### 17.2.1 K-means Cluster Lab with Colab

The Python code for K-means clustering in Box 17.2.1 is setup as a Colab exercise<sup>3</sup>. The Colab implements K-means clustering on a small set of  $2 \times 1$  training points and allows the user to also test soft-margin regularization on the this data set. The Colab code is for the Yellowstone data which must be downloaded<sup>4</sup> and then uploaded to the Colab site.

---

<sup>3</sup>[https://colab.research.google.com/drive/1WRYzkb0Wd5SYyYO\\_OkBYnd7h1tab6N9X#scrollTo=V-r159bWU5J1](https://colab.research.google.com/drive/1WRYzkb0Wd5SYyYO_OkBYnd7h1tab6N9X#scrollTo=V-r159bWU5J1)

<sup>4</sup><http://www.stat.cmu.edu/~larry/all-of-statistics/=data/faithful.dat>

**Code 17.2.1.** *Python Code<sup>a</sup> for K-means Clustering:*

```

import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# Configuration options
num_samples_total = 1000
cluster_centers = [(20,20), (4,4)]
num_classes = len(cluster_centers)

# Generate data
X, targets = make_blobs(n_samples = num_samples_total, centers = cluster_centers, n_features = num_classes,
                        center_box=(0, 1), cluster_std = 2)

np.save('./clusters.npy', X)
X = np.load('./clusters.npy')

# Fit K-means with Scikit
kmeans = KMeans(init='k-means++', n_clusters=num_classes, n_init=10)
kmeans.fit(X)

# Predict the cluster for all the samples
P = kmeans.predict(X)

# Generate scatter plot for training data
colors = list(map(lambda x: '#3b4cc0' if x == 1 else '#b40426', P))
plt.scatter(X[:,0], X[:,1], c=colors, marker="o", picker=True)
plt.title('Two clusters of data')
plt.xlabel('Temperature yesterday')
plt.ylabel('Temperature today')
plt.show()

```

<sup>a</sup><https://www.machinecurve.com/index.php/2020/04/16/how-to-perform-k-means-clustering-with-python-in-scikit/>

The results in Figure 17.1 show the a) Yellowstone data that can be labeled by K-means clustering for b)  $K = 2$  and c)  $K = 3$ . Clearly the  $K = 3$  choice provides a mislabeling of the data. However, the data clusters are distorted by the choice of the axes units, where the vertical axis extends over a range of 60 minutes while the horizontal axis is over the range of only 4 minutes.

The plot of distortion ( $\propto$  misfit error) versus number of clusters is shown in Figure 17.1d. According to the elbow method in section 17.6, the line (solid black line) drawn from the origin to the nearest point of the elbow curve should indicate the optimal number of clusters of around  $K = 3$ . However, in this example it leads to misclassification. This suggests that a trial-and-error approach with visual examination is needed to estimate the optimal number of clusters from the elbow plot in Figure 17.1d. Visual examination of point clusters in a low-dimensional space is easy, but it is difficult in a high-dimensional space. In this case, a self organizing map, discussed in section 17.10, can be used to reduce the points to a low-dimensional space.

## 17.3 Agglomerative Clustering

An effective bottom-up clustering strategy is the agglomerative clustering algorithm described below.

- The bottom-up approach starts with each point being its own distinct cluster.
- At each iteration only two clusters are combined<sup>5</sup> to be a bigger cluster as illustrated in Figure 17.6. The two clusters to be merged are the pair where, for example, the two centroids are closest to one another compared to any other pair of centroids. This is known as the linkage criterion, but there can be many other types of linkage criteria<sup>6</sup>. For example, in

<sup>5</sup>This assumes that all clusters always have unequal distances between them, otherwise more than two clusters can be combined at each iteration if they are equidistant from one another.

<sup>6</sup>[https://en.wikipedia.org/wiki/Hierarchical\\_clustering#Agglomerative\\_clustering\\_example](https://en.wikipedia.org/wiki/Hierarchical_clustering#Agglomerative_clustering_example)

single-linkage clustering the distance between two clusters is determined by a single pair of points: those two points (one in each cluster) are the pair that are closest to each other<sup>7</sup>. The first column of Figure 17.7 depicts the agglomerative cluster results for a single-linkage clustering.

- The above step is repeated until all points are included into one super cluster. The user selects the final number of clusters needed to solve the problem at hand with the *partitioned cluster*.

Scikit-learn<sup>8</sup> has many different cluster programs written in Python, including agglomerative clustering, DBSCAN and K-means. The scikit-learn Python code that applies agglomerative clustering to a point distribution is shown in Box 17.3.1, where the four different linkages are defined as the following.

1. Single Linkage: Two clusters are selected for merging where the two associated centroids are closest to one another compared to any other pair of centroids in different clusters. For  $d(\mathbf{x}, \mathbf{y})$  being the distance between the points  $\mathbf{x}$  and  $\mathbf{y}$  and  $A$  being the set of points in cluster  $A$  and  $B$  being in any other cluster  $B$ , we have the mathematical definition of the single-cluster linkage<sup>9</sup>:

$$\min(d(\mathbf{x}, \mathbf{y}) : \mathbf{x} \in A, \mathbf{y} \in B). \quad (17.4)$$

2. Ward Linkage: Two clusters are selected for merging based on the pair with minimum variance. That is, the formula is the same as for single linkage except the Euclidean distance is squared.

$$\min(d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|^2 : \mathbf{x} \in A, \mathbf{y} \in B). \quad (17.5)$$

3. Weighted Average Linkage:

$$\frac{1}{|A||B|} \sum_{\mathbf{x} \in A} \sum_{\mathbf{y} \in B} d(\mathbf{x}, \mathbf{y}). \quad (17.6)$$

4. Centroid Linkage: The cluster pair with the minimum distance between their associated centroids is selected for the next merging of clusters.
5. Complete Linkage: The link between two clusters contains all element pairs, and the distance between clusters equals the distance between those two elements (one in each cluster) that are farthest away from each other. The shortest of these links that remains at any step causes the merging of the two clusters whose elements are involved ([https://en.wikipedia.org/wiki/Complete-linkage\\_clustering](https://en.wikipedia.org/wiki/Complete-linkage_clustering)).

$$\max(d(\mathbf{x}, \mathbf{y}) : \mathbf{x} \in A, \mathbf{y} \in B). \quad (17.7)$$

---

<sup>7</sup>[https://en.wikipedia.org/wiki/Single-linkage\\_clustering](https://en.wikipedia.org/wiki/Single-linkage_clustering)

<sup>8</sup><https://en.wikipedia.org/wiki/Scikit-learn>

<sup>9</sup>[https://en.wikipedia.org/wiki/Hierarchical\\_clustering#Linkage\\_criteria](https://en.wikipedia.org/wiki/Hierarchical_clustering#Linkage_criteria)

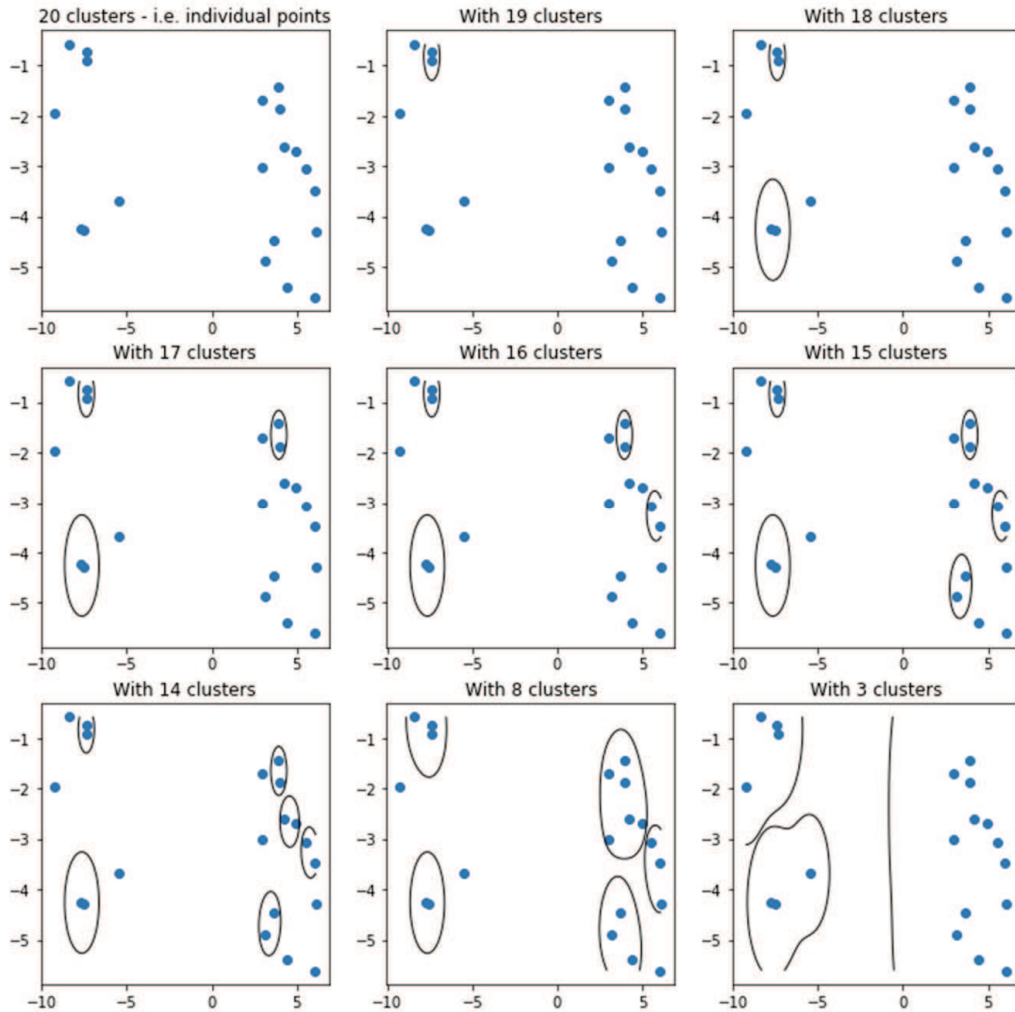


Figure 17.6: Bottom-up clustering of points where, initially, there is one centroid at each point of the input data (upper left plot). At each iteration the number of clusters decreases by one. Note, the horizontal axis is about three times longer than the vertical scale. Illustration from Miller (2020).

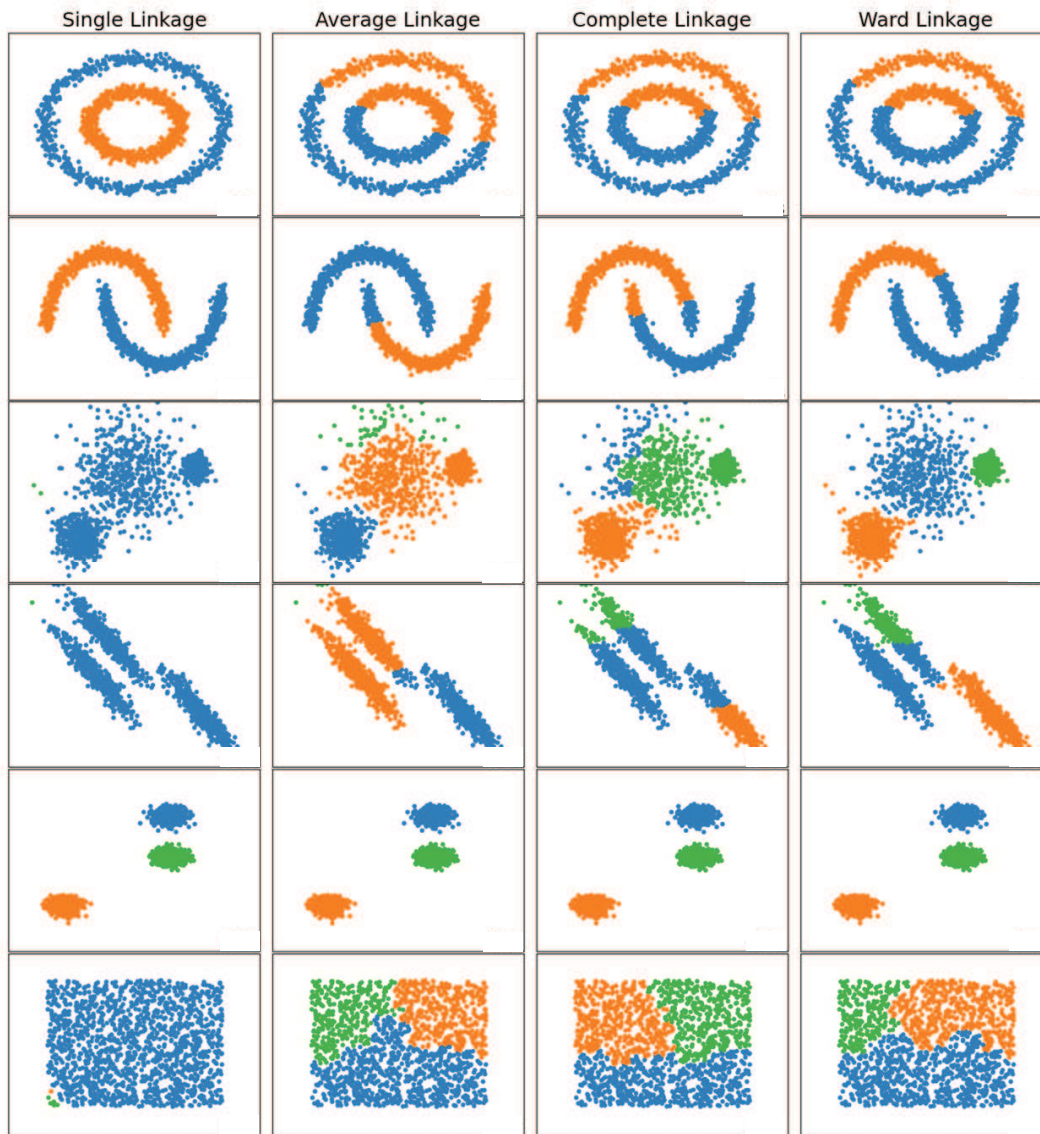


Figure 17.7: Final cluster results for different linkages used for agglomerative clustering. Each row corresponds to a different type of point distribution while each column is for a different linkage. Illustration from Miller (2020) and images computed from the Python code in Box 17.3.1.

**Code 17.3.1.** *Python Code<sup>a</sup> for Agglomerative Clustering:*

```

print(__doc__)
from time import time

import numpy as np
from scipy import ndimage
from matplotlib import pyplot as plt

from sklearn import manifold, datasets

X, y = datasets.load_digits(return_X_y=True)
n_samples, n_features = X.shape

np.random.seed(0)

def nudge_images(X, y):
    # Having a larger dataset shows more clearly the behavior of the
    # methods, but we multiply the size of the dataset only by 2, as the
    # cost of the hierarchical clustering methods are strongly
    # super-linear in n_samples
    shift = lambda x: ndimage.shift(x.reshape((8, 8)),
                                    .3 * np.random.normal(size=2),
                                    mode='constant',
                                    ).ravel()
    X = np.concatenate([X, np.apply_along_axis(shift, 1, X)])
    Y = np.concatenate([y, y], axis=0)
    return X, Y

X, y = nudge_images(X, y)

#-----
# Visualize the clustering
def plot_clustering(X_red, labels, title=None):
    x_min, x_max = np.min(X_red, axis=0), np.max(X_red, axis=0)
    X_red = (X_red - x_min) / (x_max - x_min)

    plt.figure(figsize=(6, 4))
    for i in range(X_red.shape[0]):
        plt.text(X_red[i, 0], X_red[i, 1], str(y[i]),
                color=plt.cm.nipy_spectral(labels[i] / 10.),
                fontdict={'weight': 'bold', 'size': 9})

    plt.xticks([])
    plt.yticks([])
    if title is not None:
        plt.title(title, size=17)
    plt.axis('off')
    plt.tight_layout(rect=[0, 0.03, 1, 0.95])

#-----
# 2D embedding of the digits dataset
print("Computing embedding")
X_red = manifold.SpectralEmbedding(n_components=2).fit_transform(X)
print("Done.")

from sklearn.cluster import AgglomerativeClustering

for linkage in ('ward', 'average', 'complete', 'single'):
    clustering = AgglomerativeClustering(linkage=linkage, n_clusters=10)
    t0 = time()
    clustering.fit(X_red)
    print("%s : %t%.2fs" % (linkage, time() - t0))

    plot_clustering(X_red, clustering.labels_, "%s linkage" % linkage)

plt.show()

```

<sup>a</sup>[https://scikit-learn.org/stable/auto\\_examples/cluster/plot\\_digits\\_linkage.html#sphx-glr-auto-examples-cluster-plot-digits-linkage-py](https://scikit-learn.org/stable/auto_examples/cluster/plot_digits_linkage.html#sphx-glr-auto-examples-cluster-plot-digits-linkage-py)

## 17.4 Information Weighted Clustering

The clustering algorithm can be recast as an iterative optimization problem where the regularized objective function is given by

$$\epsilon = \frac{1}{2} \sum_{i=1}^K \frac{1}{m_i} \sum_{j=1}^{m_i} \|\mathbf{C}_i - \mathbf{x}_{ij}\|^2 + \frac{\lambda}{2} \sum_{i=1}^K \|\mathbf{C}_i - \bar{\mathbf{C}}_i\|^2, \quad (17.8)$$

where  $\lambda > 0$  is the damping parameter and the  $D \times 1$  vector  $\bar{\mathbf{C}}_i$  is a desired centroid point that should be near our final cluster point  $\mathbf{C}_i$ . The gradient of this objective function is

$$\begin{aligned} \nabla_{\mathbf{C}_i} \epsilon = \frac{\partial \epsilon}{\partial \mathbf{C}_i} &= \frac{1}{m_i} \sum_{j=1}^{m_i} (\mathbf{C}_i - \mathbf{x}_{ij}) + \lambda (\mathbf{C}_i^{(k)} - \bar{\mathbf{C}}_i), \\ &= \mathbf{C}_i - \frac{1}{m_i} \sum_{j=1}^{m_i} \mathbf{x}_{ij} + \lambda (\mathbf{C}_i^{(k)} - \bar{\mathbf{C}}_i), \end{aligned} \quad (17.9)$$

where

$$\frac{\partial \epsilon}{\partial \mathbf{C}_i} = \left( \frac{\partial \epsilon}{\partial C_{i1}}, \frac{\partial \epsilon}{\partial C_{i2}}, \dots, \frac{\partial \epsilon}{\partial C_{iD}} \right)^T, \quad (17.10)$$

and  $C_{il}$  is the  $l^{th}$  component of the  $i^{th}$  cluster vector  $\mathbf{C}_i$ . The corresponding steepest descent formula for updating  $\mathbf{C}_i$  is given by

$$\begin{aligned} \mathbf{C}_i^{(k+1)} &= \mathbf{C}_i^{(k)} - \alpha \frac{\partial \epsilon}{\partial \mathbf{C}_i}, \\ &= \mathbf{C}_i^{(k)} - \alpha \frac{1}{m_i^{(k)}} \sum_{j=1}^{m_i^{(k)}} (\mathbf{C}_i^{(k)} - \mathbf{x}_{ij}^{(k)}) - \alpha \lambda (\mathbf{C}_i^{(k)} - \bar{\mathbf{C}}_i), \\ &\quad \text{Average of Points in } i^{th} \text{ Cluster} \\ &= (1 - \alpha) \mathbf{C}_i^{(k)} + \alpha \underbrace{\frac{1}{m_i^{(k)}} \sum_{j=1}^{m_i^{(k)}} \mathbf{x}_{ij}^{(k)}}_{\text{Average of Points in } i^{th} \text{ Cluster}} - \alpha \lambda (\mathbf{C}_i^{(k)} - \bar{\mathbf{C}}_i), \end{aligned} \quad (17.11)$$

where  $k$  is the iteration index in the superscripts of  $(\mathbf{C}_i^{(k)}, \mathbf{x}_{ij}^{(k)}, m_i^{(k)})$ .

The iterative cluster algorithm described by equation 17.11 can be made more robust by applying a weight  $w_{ij}$  to each point in the  $i^{th}$  cluster (Chen, 2018a-b). The weight  $w_{ij}$  can be based on the information content of that point, e.g., points with large-amplitude values in a migration image contain more important information than low-amplitude points in it. The importance of points based on their amplitude values  $A_{ij}$  can be given a soft-max weighting such that

$$w_{ij} = \frac{e^{A_{ij}}}{\sum_{j=1}^{m_j} e^{A_{ij}}} \quad \text{where} \quad \sum_{j=1}^{m_j} w_{ij} = 1. \quad (17.12)$$

Therefore the information-weighted clustering (IWC) formula is

$$\mathbf{C}_i^{(k+1)} = (1 - \alpha)\mathbf{C}_i^{(k)} + \alpha \overbrace{\frac{1}{m_i^{(k)}} \sum_{j=1}^{m_i^{(k)}} w_{ij} \mathbf{x}_{ij}^{(k)}}^{\text{Weighted Average of Points in } i^{\text{th}} \text{ Cluster}} - \alpha \lambda (\mathbf{C}_i^{(k)} - \bar{\mathbf{C}}_i). \quad (17.13)$$

The numerical examples section 17.11 tests the effectiveness of both the standard and regularized cluster algorithms on synthetic data and field data from the Gulf of Mexico.

## 17.5 Fuzzy Clustering

The weighting term in equations 17.12-17.13 can be considered as special weights for fuzzy clustering (Dunn, 1973; Peizhuang, 1983; Moya-Albor et al., 2017), where the weight  $\mu_{kj}$  indicates the degree of membership of the point  $\mathbf{x}_j$  in the  $k^{\text{th}}$  cluster. In K-means clustering the fuzzy weight is binary, it indicates that the point  $\mathbf{x}_j$  either belongs to the  $k^{\text{th}}$  cluster, that is  $\mu_{kj} = 1$ , or it doesn't where  $\mu_{kj} = 0$ . If the degree of membership is fuzzy, each point belongs to a cluster but with fuzzy weights  $0 \leq \mu_{kj} \leq 1$ .

The objective function of the fuzzy clustering algorithm is given by (Bezdek, 2013; Károly et al., 2018)

$$\epsilon = \frac{1}{2} \sum_{k=1}^K \sum_{j=1}^N \mu_{kj}^m \|\mathbf{x}_j - \mathbf{C}_k\|^2, \quad (17.14)$$

where  $m$  is a coefficient to control fuzziness. Cano et al. (2021) specifies  $m = 2$  so that the weight is defined as

$$\mu_{kj} = \frac{\|\mathbf{x}_j - \mathbf{C}_k\|^{-2}}{\sum_{k'=1}^K \|\mathbf{x}_j - \mathbf{C}_{k'}\|^{-2}}, \quad j \in \{1, 2, \dots, N\}, \quad k \in \{1, 2, \dots, K\}. \quad (17.15)$$

Similar to equation 17.12, the weight in equation 17.15 satisfies the constraint

$$\sum_{k=1}^K \mu_{kj} = 1, \quad (17.16)$$

for all values of  $j$  and  $0 \leq \mu_{kj} \leq 1$  (Cano et al., 2021) and is derived in Appendix 17.15. The fuzzy weight  $\mu_{kj}$  becomes stronger as the point becomes closer to the centroid, and weaker the greater the separation distance.

Similar to the *weighted-average-of-points* term in equation 17.13, the centroids will also be weighted-point averages:

$$\mathbf{C}_k = \frac{\sum_{j=1}^N \mu_{kj}^m \mathbf{x}_j}{\sum_{j=1}^N \mu_{kj}^m}, \quad k \in \{1, 2, \dots, K\}. \quad (17.17)$$

Implementation of this algorithm requires the initial specification of the number of clusters, the coefficient of fuzziness, typically  $m \approx 2$ , and an initial degree of random membership for all points. The iterative procedure is similar to that for K-means clustering.



### 17.5.1 Arrival Time Picking and Fuzzy Clustering

One of the applications of unsupervised fuzzy clustering is to pick arrival times of events in seismic data. This avoids the need to have a labeled seismic data set for training, as required by a supervised neural network (Ross et al., 2018) or SVM (Akter et al., 2015). A successful use of fuzzy clustering for picking P- and S-wave arrivals in microseismic data was implemented by Cano et al. (2021) using (Chen, 2020) the fuzzy c-means clustering method (FCM). Cano et al. (2021) successfully tested the FCM on both synthetic data and three-component traces recorded in a well monitoring a hydraulic fracturing experiment (see Figure 17.8a). In their case, they used the  $3 \times 1$  feature vector  $\mathbf{x}^{(i)} = (M(i), P(i), Q(i))^T$  for each time sample  $i \in \{1, 2, \dots, N\}$  of the trace given by  $d(i)$ . Elements of the feature vector  $\mathbf{x}^{(i)}$  for all time samples  $i$  in a trace are illustrated in Figure 17.8b and are defined below.

- $M(i)$  is the mean of the absolute value of the amplitude defined as

$$M(i) = \frac{1}{N} \sum_{i-w}^{i+w} |d(i)|, \quad (17.18)$$

at time sample  $i$  and the half-width window length  $w$ .

- $P(i)$  is the peak power spectral density:

$$P(i) = \max(|D(i, \omega)|^2), \quad (17.19)$$

where  $|D(i, \omega)|$  is the modulus of the discrete short-time FFT of  $d(i)$ .

- $Q(i)$  is the short-time-arrival/long-time-arrival (STA/LTA) ratio defined as

$$Q(i) = \frac{\text{STA}}{\text{LTA}} = \frac{\frac{1}{SW} \sum_{j=i}^{i+SW} |d(j)|}{\frac{1}{LW} \sum_{j=i-LW}^i |d(j)|}, \quad (17.20)$$

where  $SW$  and  $LW$  are the lengths of the short- and long-time windows, respectfully. The dominant period  $T_{dom}$  of the arrival is estimated from the data, which is used to estimate  $w \approx 0.5 \times T_{dom}$ ,  $SW \approx 1.5 \times T_{dom}$  and  $LW \approx 5 \times SW$ .

There are 4 steps in the Cano et al. (2021) method as illustrated by the workflow in Figure 17.9.

- **Signal Detection in Analysis Windows.** Identify coherent events in previously windowed seismic traces using the FCM. Multiple signal intervals (if present) are identified in the analysis windows. The FCM is used to compute the  $i^{th}$  sample's degree  $\mu_{ij}^m$  of membership in a cluster. Only two clusters will be allowed, one for noise and the other for a coherent signal. A threshold level  $\beta$  is set for  $\mu_{ij}^m$  that determines if the  $i^{th}$  sample is part of a coherent event ( $j = 2$ ) or noise ( $j = 1$ ). Cano et al. (2021) state that "Any continuous interval where  $\mu_{ij}^m$  is greater than  $\beta$  for at least  $1.5 \times T_{dom}$  duration is considered to contain a possible wave arrival; therefore, the other remaining intervals are deemed noise and are discarded from further analysis". They set  $\beta$  to be between 1.0 and 2.0 times the mean value of  $\mu_{i2}^m$ . For three-component data the values of  $\mu_{i2}^m$  for all three components are averaged to give a final membership score used to identify the signal intervals. An example of the membership scores for three components recorded at a single level is shown in Figure 17.8c, and the averaged or stacked result is shown in Figure 17.8d.
- **Wave-type Classification of Events.** Polarization analysis of three-component data is used to identify the type of event in a signal interval: S wave, P wave or unidentified signal.

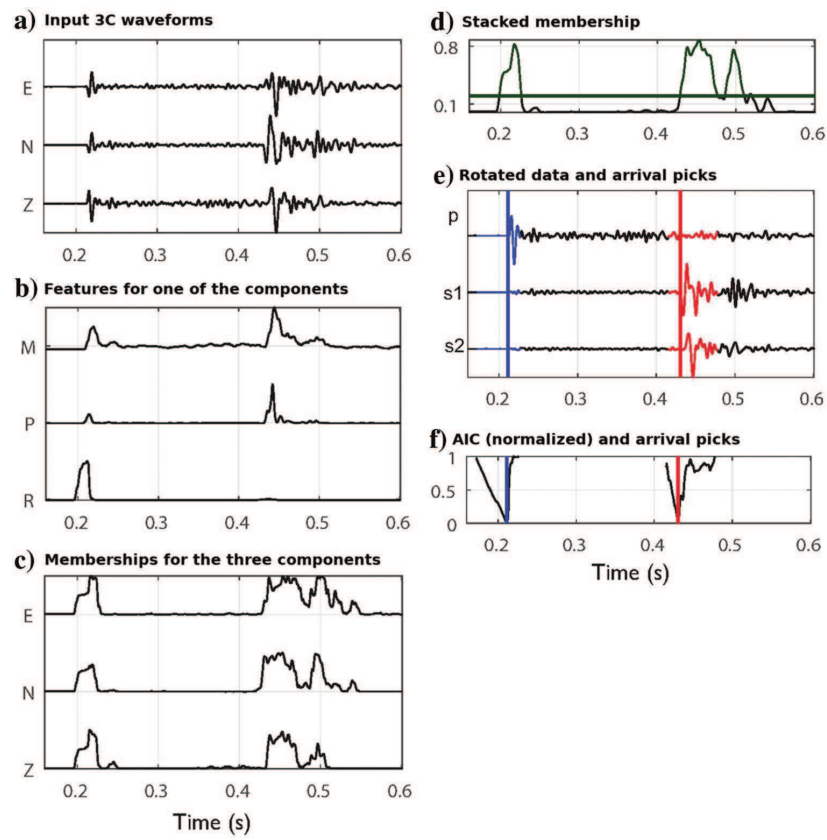


Figure 17.8: Results from the traveltime picking workflow of Cano et al. (2021).

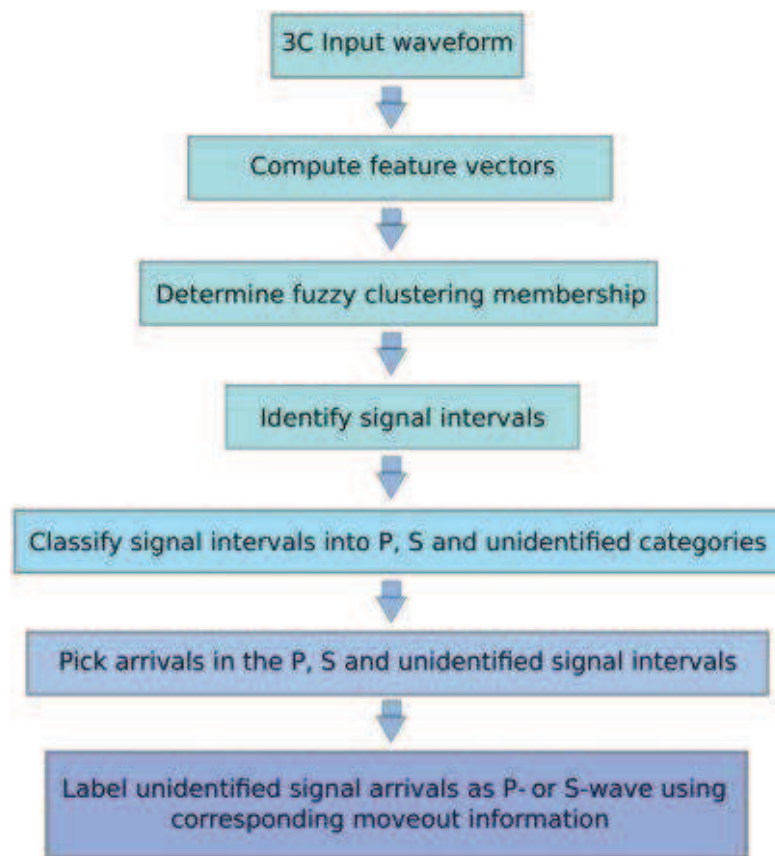


Figure 17.9: Workflow of Cano et al. (2021) method for picking P- and S-wave events in microseismic data.

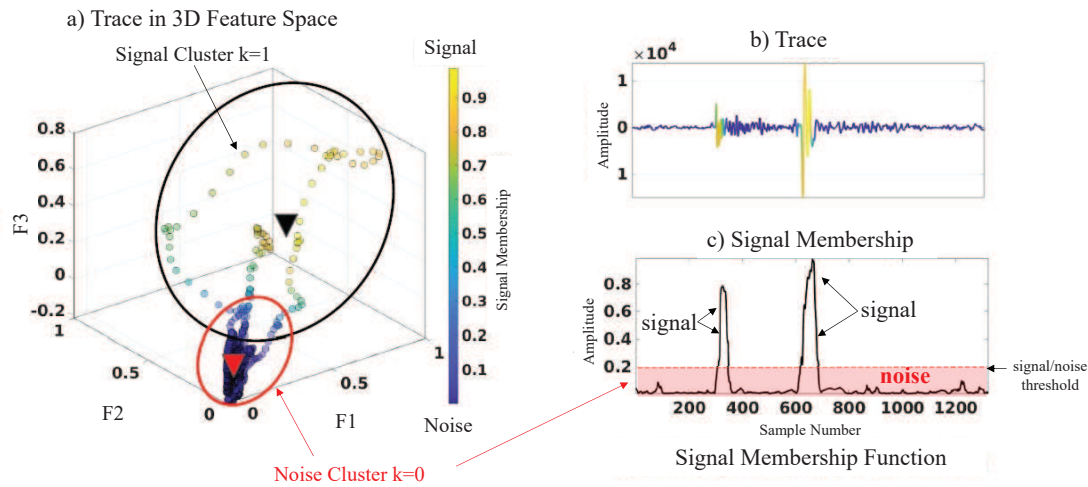


Figure 17.10: Seismic trace plotted in a) 3D feature coordinates and in b) time where the signals are marked in yellow. c) depicts the signal membership function with the horizontal dashed line representing the amplitude threshold between noise (red zone) and signal. Figure adapted from Eduardo Cano's PPT presentation.

A linearity analysis is used to determine if an event is a P or S arrival. For example, particle motion perpendicular (parallel) to the angle of incidence can be classified as S (P) events after rotation of the axes so that one axis is along the angle of incidence. The rotated components and labeled events (red indicates an S arrival) are shown in Figure 17.8e.

Note that the signals represented by the signal cluster index of  $k = 1$  can be separated in time. This is illustrated in Figure 17.10b where both the P- and S-arrivals in yellow are separated in time but are part of the same  $k = 1$  signal cluster in 3D feature space. The features of the trace in b) are illustrated by the wiggly-line drawing in a), which is plotted w/r to the 3D feature coordinates  $(F1, F2, F3) = (M(i), P(i), Q(i))$  described in equations 17.18-17.20. It is obvious in this space that the P- and S-arrivals are bunched together with the largest amplitude-like features and are far from the origin in the signal cluster denoted by  $k = 1$ . In contrast, the trace's noise cluster, denoted by  $k = 0$ , are clumped together near the origin. In summary, points in the same cluster can be separated from one another in the time domain, but they are clumped together in the feature space. This concept is also true for points originally plotted in spatial coordinates, but if the distance metric contains non-geographical coordinates then the earthquakes from the same cluster class can be geographically separated from one another, as shown in Figures 17.36-17.37. Another example is the seismic horizon map in Figure 21.14, where clusters with the same color are geographically separated from one another.

- **Arrival Time Picking.** The arrival time of the signal is picked using the Akaike information criterion (Akaike, 1973). The AIC algorithm is applied to the P (S) interval on the p (s) components to pick the P-wave (S-wave) arrival time, as illustrated in Figure 17.8f. The AIC picked arrival times in the S interval are averaged for the two perpendicular components, and the time associated with the minimum AIC value is the picked arrival time of a coherent event.
- **Unidentified Pick Classification.** A semblance-like criterion is used to estimate the move-out pattern of the unidentified picks as either that of P or S arrivals. The moveout patterns

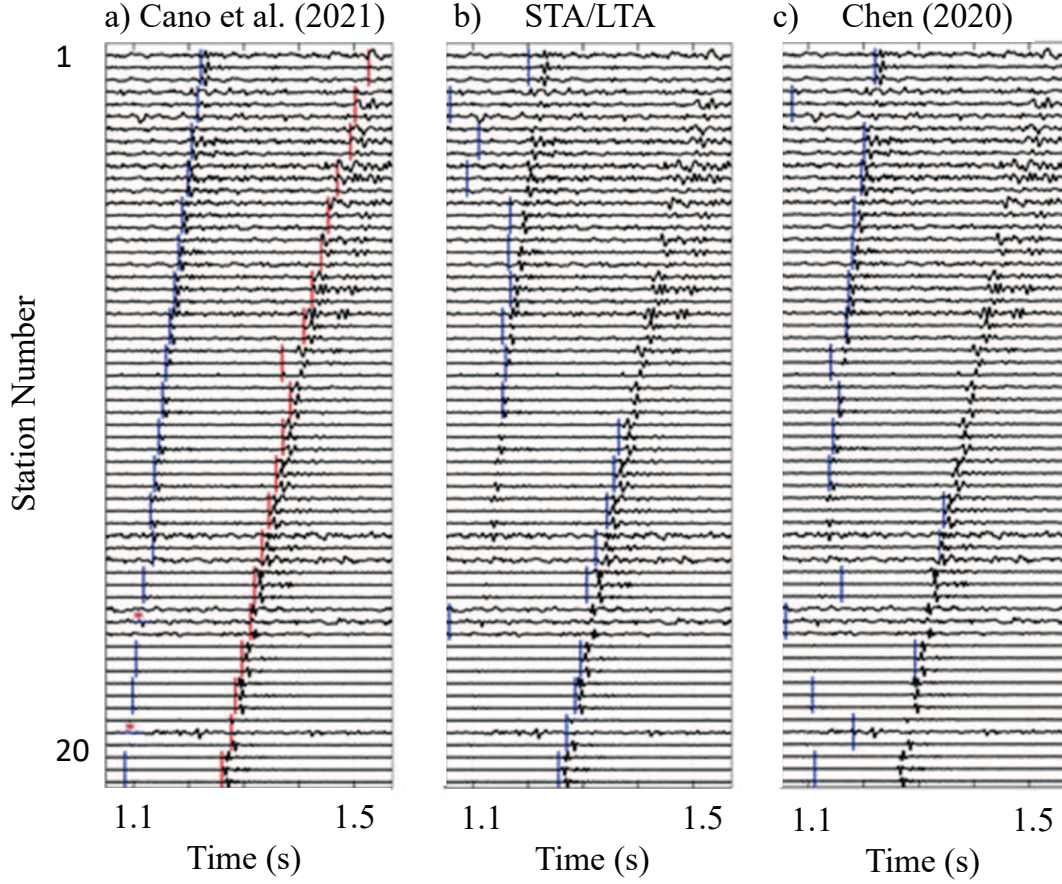


Figure 17.11: P (blue bars) and S (red bars) arrivals picked by the a) Cano et al. (2021), b) STA/LTA and c) Chen (2020) methods. Figure from Cano et al. (2021).

of P and S arrivals are obtained from a database of known events.

An example of their picking results for the hydraulic fracturing data is shown in Figure 17.11 where the P- and S-arrivals are picked by the a) Cano et al. (2021), b) LTA/STA and c) Chen (2020) methods. The blue bars denote the picks for P arrivals and the red ones denote the picks for the S arrivals. For these data, the Cano et al. (2021) method picks the P and S arrivals much more accurately than the STA/LTA or Chen (2020) methods.

## 17.6 Elbow Method for Determining the Number of Clusters

The number of clusters is sometimes determined by using the elbow method (Thorndike, 1953; Goutte et al., 1999).

1. Define the initial number of clusters as  $K = 2$  and the initial two centroid points.
2. Use equation 17.13 to recompute the new cluster centroids  $\mathbf{C}_i$  from the old centroids. If there are  $k$  clusters, determine the number of points  $m_i^k$  in the  $i^{th}$  cluster. Then find the sum of

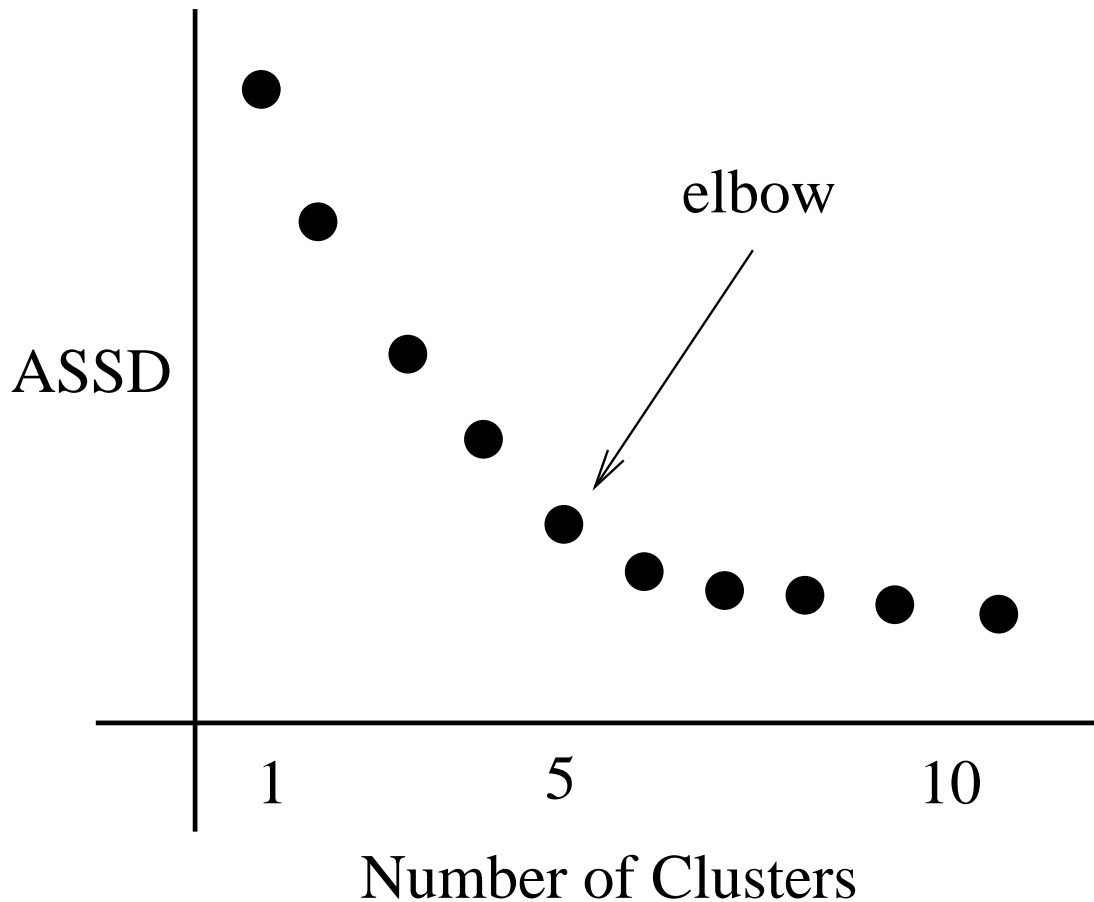


Figure 17.12: Average sum of squared distances (ASSD) plotted against the number of clusters. The *elbow* cannot always unambiguously determine the correct number of clusters ([https://en.wikipedia.org/wiki/Elbow\\_method\\_\(clustering\)](https://en.wikipedia.org/wiki/Elbow_method_(clustering)) ).

the squared distances  $SSD_i^k$  between the centroid of the  $i^{th}$  cluster and the  $m_i^k$  points in that cluster. Compute the average of the variances  $SSD_i^k$  as  $ASSD^k = \sum_{i=1}^k SSD_i^k / m_i^k$ .

3. Plot  $ASSD^k$  against  $k$  as shown in Figure 17.12.
4. Increase the number of centroids by one and repeat steps 2-3 until enough points are computed to show an elbow in the curve. The choice of the new centroid is up to the user, but setting it to be the average point of the computed centroids is one strategy. The first few clusters show a large change in  $ASSD^k$  with respect to a change in the number  $k$  of clusters, and will segregate data over a wide range of values. The decrease in  $ASSD^k$  value will diminish with an increase in iteration number and the number of clusters, leading to a flattened curve. The number of clusters is chosen at this point of flattening, hence the "*elbow criterion*". The point at which the curve starts to flatten is often the point closest to or near the origin, as shown in Figure 17.12. Instead of plotting average variance one can plot the percentage of variance.

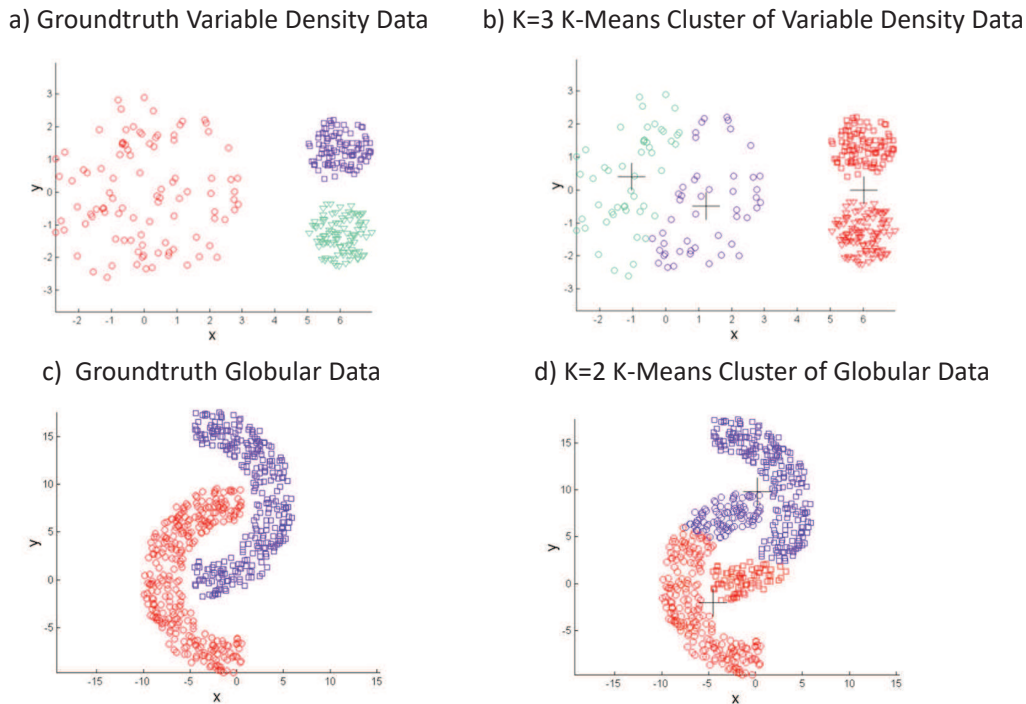


Figure 17.13: Point distributions with variable densities of points with different classes in a) will confuse a K-means clustering algorithm to give the clusters in b). The same problem exists with points of different classes that form sinuous patterns in the second row of figures. These sinuous patterns indicate a non-linear correlation among the components of the data vector, which violates the linearity assumption of PCA analysis (See Chapter 16). Illustration adopted from Ben-Hur et al. (2001).

## 17.7 Support Vector Clustering

The problem with K-means clustering is that it doesn't work well when the data types are distributed with different densities as in Figure 17.13a. In this case, K-means clustering will fail as seen in Figure 17.13b where the two balls of red points on the right are incorrectly labeled because they are close together compared to the other labeled points. If the data points of different types are distributed as sinuous and non-convex shapes as illustrated in Figure 17.13c, then K-means cluster will also fail and label according to how close the points are to one another as shown in Figure 17.13d. More generally, non-convex clusters of points will not always be accurately clustered by K-means clustering. Finally, if the dimensionality of the data is too high then the expense of K-means clustering can become unaffordable. In this case principal component analysis in Chapter 16 can be used to reduce the dimensionality of the data prior to K-means clustering.

To mitigate the problems of variable point density and non-convex clusters, Ben-Hur et al. (2001) developed unsupervised support vector clustering (SVC). The SVC method attempts to find hidden structure in unlabeled data (Ben-Hur et al., 2001) and is partly based on the kernel SVM method except no labeled data are needed. The output consists of clustered points with distinct labels. Its advantage over K-means clustering is that the data points of one type do not have to be evenly clumped together, but can be distributed over a large extended region such as the non-convex



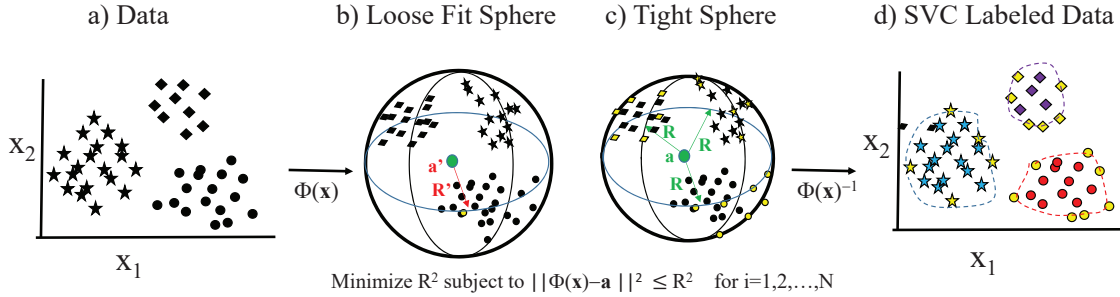


Figure 17.14: Steps for unsupervised support vector classification of points in a) to the different classes in d). Illustration adopted from Ben-Hur et al. (2001).

globs in Figure 17.13.

There are six steps to unsupervised SVC (Ben-Hur et al., 2001; Statnikov et al., 2011).

1. Input the data with, for example, three unlabeled classes as indicated by the stars, dots, and diamonds in Figure 17.14a.
2. Transform the data  $\mathbf{x}$  to a higher dimensional feature space by  $\Phi(\mathbf{x})$  such that there is a hypersphere at the center  $\mathbf{a}'$  with the radius  $R'$  that loosely encloses all of the points  $\mathbf{z} = \Phi(\mathbf{x})$ . See Figure 17.14b.
3. Define the Primal problem as finding the tightest fitting hypersphere with the smallest radius  $R^*$  that tightly fits all of the points  $\Phi(\mathbf{x}^{(i)})$ , i.e.,

$$(R^*, \mathbf{a}^*) = \arg \min_{R, \mathbf{a}} \|\Phi(\mathbf{x}^{(i)}) - \mathbf{a}\| \leq R^2, \quad (17.21)$$

where  $\mathbf{a}^*$  is the center of the optimal hypersphere. The support vectors associated with the primal problem are computed by solving the reduced dual problem using a Gaussian kernel  $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|^2 / \sigma^2)$ . The reduced dual problem with soft-margin constraints is given by

$$\begin{aligned} \min_{\alpha} \mathcal{L} &= \min_{\alpha} \left[ \sum_i \alpha_i K(\mathbf{x}^{(i)}, \mathbf{x}^{(i)}) - \sum_i \sum_j \alpha_i \alpha_j K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \right], \\ &\text{subject to } 0 \leq \alpha_i \leq C, \end{aligned} \quad (17.22)$$

where the number of allowable errors is controlled by the value of  $C$ . The value of  $\sigma$  will determine the number of clusters. Smaller values of  $\sigma$  will lead to fewer clusters, while larger values will lead to a larger number of clusters.

4. Schölkopf et al. (1999, 2001) and Tax and Duin (1999) showed that the solution to equation 17.21 can characterize the support vectors of a high-dimensional distribution. The contours enclosing these data points can be used to define their class, which is interpreted as the cluster boundaries by Ben-Hur et al. (2001). These support points lie on the surface of the tightest fitting hypersphere, which also define the support vectors that delineate the cluster boundaries in data space, as illustrated by the yellow symbols in Figure 17.14c and Figure 17.14d.
5. We know that the support vectors lie on the hypersphere's surface, but we don't know which cluster they belong to. To determine their membership, Ben-Hur et al. (2001) use a geometric approach involving the tightest-fitting radius  $R^*$ , based on the following observation: given a



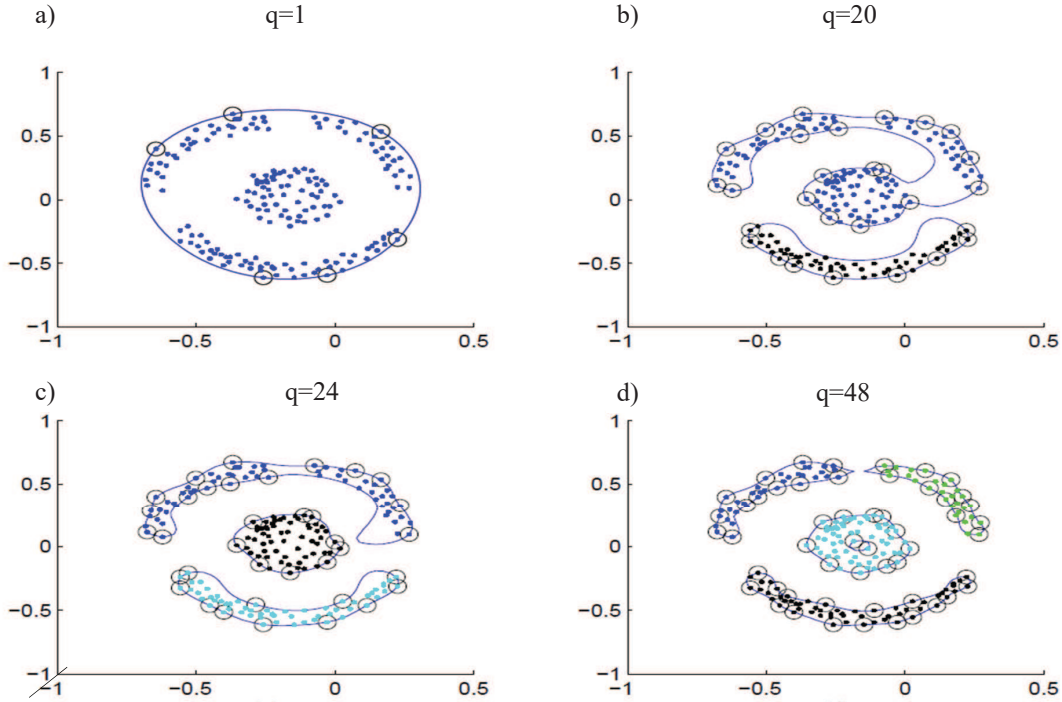


Figure 17.15: Unsupervised clustering of points for a)  $q = 1$ , b)  $q = 20$ , c)  $q = 24$ , and d)  $q = 48$ , where  $q = 1/\sigma^2$  in the Gaussian kernel and  $C = 1$ . Illustration adopted from Ben-Hur et al. (2001).

pair of data points that belong to different clusters, any path that connects them must exit from the hypersphere in feature space. Such a path contains a segment of points  $\mathbf{y}$  such that  $R(\mathbf{y}) > R^*$ , where  $R(\mathbf{y})$  is measured from the center at  $\mathbf{a}^*$  to  $\mathbf{y}$ . Therefore, if the points between the original pair of points jump outside the hypersphere with radius  $R^*$  then this indicates that the pair of points belong to different clusters. They will be labeled as such in both feature space and the final output domain in Figure 17.14d. This test is done for many pairs of points until they are all labeled with the correct class.

To facilitate this labeling of points they created the  $N \times N$  adjacency matrix  $\mathbf{A}$ . Each element  $A_{ij}$  denotes whether the pair of points  $\mathbf{x}_i$  and  $\mathbf{x}_j$  belong to the same class in feature space. That is,  $A_{ij} = 1$  indicates that  $\mathbf{x}_i$  and  $\mathbf{x}_j$  belong to the same class if all points between the line connecting  $\mathbf{x}_i$  and  $\mathbf{x}_j$  are within or on the optimal sphere with radius  $R^*$ . Otherwise  $A_{ij} = 0$ .

6. The yellow points in Figure 17.14c can be inverse transformed by  $\Phi(\mathbf{x}^{(i)*})^{-1}$  to get the support vectors  $\mathbf{x}^{(i)*}$  in the data domain. These support vectors are indicated as yellow symbols in Figure 17.14d and so the margin boundaries can then be computed. Each closed margin boundary defines a separate class of points.

As an example, the two rings of points in Figure 17.15a are clustered by the SVC procedure described above. For  $q = \sigma^{-2} = 1$ , the Gaussian kernels have a wide width so only one cluster is detected in Figure 17.15a. As  $q$  gets larger the Gaussian kernel becomes narrower and more localized around a few points so the SVC will detect more clusters as illustrated in Figures 17.15b-17.15d.

## 17.8 DBSCAN

Density-based spatial clustering (DBSCAN) groups together points that are closely packed together (Ester et al., 1996). It excludes points, denoted as outliers, that lie alone in low-density regions. The DBSCAN cluster is a contiguous region of dense data points separated from other clusters by a region with a low density of data points, such as the three clusters in Figure 17.15c. It is able to find arbitrary shaped clusters and clusters with noise (i.e. outliers) and, unlike K-Means clustering, the number of clusters is determined by the algorithm.

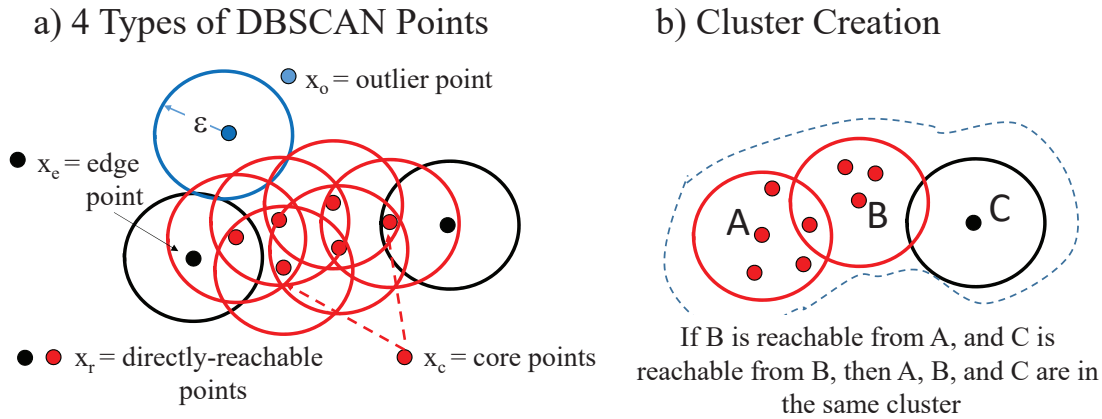


Figure 17.16: a) For  $N^{minpts} = 4$ , all red points are core points because the circle surrounding these points in an  $\epsilon$  radius contain at least 4 points (including the point itself). b) All the red and black points are  $\epsilon$ -reachable from one another, so they form a single cluster. The edge points in a) are edge points but not core points because they are  $\epsilon$ -reachable to at least three neighbors. The blue point in a) is neither a core point nor directly-reachable and so does not belong to the cluster in b). See <https://commons.wikimedia.org/w/index.php?curid=17045963>.

Three classes of points in Figure 17.16a are defined for DBSCAN: core points, directly-reachable points and outliers (<https://en.wikipedia.org/wiki/DBSCAN>).

- A point  $\mathbf{x}_c$  is a *core point* if at most  $N^{minpts}$  points are within the distance  $\epsilon$  of it (including  $\mathbf{x}_c$ ), as indicated by the red points in Figure 17.16. The user defines the values of  $\epsilon$  and  $N^{minpts}$ .
- A point  $\mathbf{x}_r$  is defined as a *directly-reachable point* from  $\mathbf{x}_c$  if point  $\mathbf{x}_r$  is within distance  $\epsilon$  from the core point  $\mathbf{x}_c$ . Points are only said to be directly reachable from core points. Both the black and red points in points in Figure 17.16a are directly reachable points, aka density-reachable points.
- If two core points  $\mathbf{x}_A$  and  $\mathbf{x}_B$  in Figure 17.16b are directly-reachable (neighbors), and if  $\mathbf{x}_B$  and  $\mathbf{x}_C$  are directly-reachable (neighbors), then  $\mathbf{x}_A$ ,  $\mathbf{x}_B$  and  $\mathbf{x}_C$  belong to the same cluster. This concept is called density connectivity. By this approach, the algorithm creates a cluster. Each cluster contains at least one core point; non-core points can be part of a cluster, but they form its "edge" (see black points in Figure 17.16a), since they cannot be used to reach more points.
- All points not reachable from any other point are *outliers or noise points*.

Unlike a K-means clustering algorithm, DBSCAN will be able to properly cluster the blue, black and light-blue points in Figure 17.15c if the correct values of  $N^{minpts}$  and  $\epsilon$  are selected.

As an example, Figure 17.17a depicts three clusters of points, and Figure 17.17b-17.17d are the DBSCAN results for different values of  $\epsilon$ . Figure 17.17c is the correct choice of  $\epsilon$  and larger values starts to inappropriately blend different groundtruth clusters together. The DBSCAN algorithm

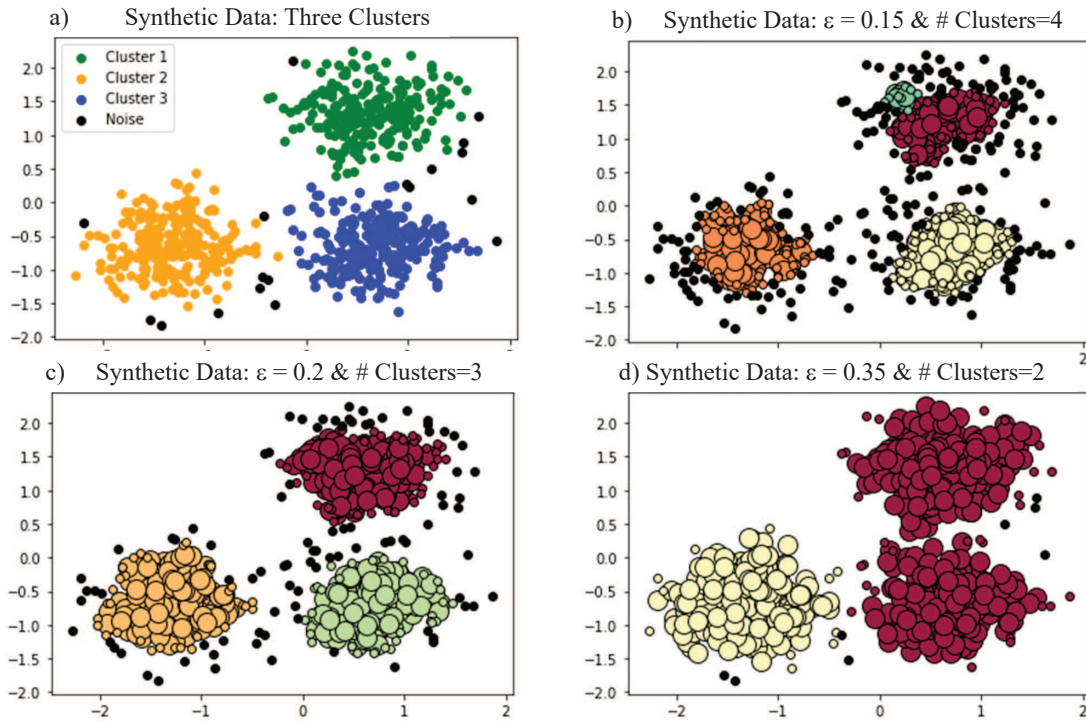


Figure 17.17: a) Groundtruth points grouped into three clusters, and DBSCAN results for b) an estimate of four clusters ( $\epsilon = 0.15$ ), c) three clusters ( $\epsilon = 0.2$ ) and d) two clusters ( $\epsilon = 0.35$ ).

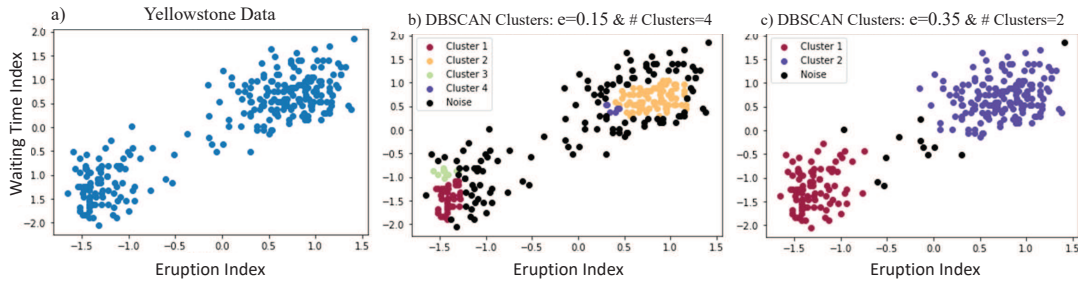


Figure 17.18: a) Yellowstone data and DBSCAN results for an estimate with b) four clusters ( $\epsilon = 0.15$ ) and c) two clusters ( $\epsilon = 0.35$ ).

can be summarized with the following steps<sup>10</sup>.

1. Specify values for  $\epsilon$  and  $N^{minpts}$ .
2. Find the points in the  $\epsilon$  neighborhood of every point, and identify the core points with more than  $N^{minpts}$  neighbors.

<sup>10</sup><https://en.wikipedia.org/wiki/DBSCAN>

- The benefits of DBSCAN are that its computational complexity can be as low as  $O(N \log N)$  and as high as  $O(N^2)$  depending on how the algorithm is implemented (<https://en.wikipedia.org/wiki/DBSCAN>). Here,  $N$  is the number of data points. There is no need to assign the number of clusters. The DBSCAN algorithm is often robust to noise or outliers in the data. If the proper  $\epsilon$  value is selected for the Yellowstone data in Figure 17.18, the DBSCAN algorithm performs as well as the K-Means algorithm. This assumes that the K-means method chose the correct number of clusters.

A liability of DBSCAN is that the resulting clusters depend on the distance measure used as well as the values of  $\epsilon$  and  $N^{minpts}$ . DBSCAN can have difficulty defining clusters with different densities and with high dimensionality datasets that suffer from the *curse of dimensionality* ([https://en.wikipedia.org/wiki/Curse\\_of\\_dimensionality](https://en.wikipedia.org/wiki/Curse_of_dimensionality)).

### 17.8.1 DBSCAN Lab with Colab

The Python code for DBSCAN clustering is in Boxes 17.8.1 and 17.8.2.

#### Code 17.8.1. *Python Code<sup>a</sup> for DBSCAN:*

```
#####
# Generate sample data
centers = [[1, 1], [-1, -1], [1, -1]]
X, labels_true = make_blobs(n_samples=750, centers=centers, cluster_std=0.4,
                             random_state=0)

# Preprocessing data
X = StandardScaler().fit_transform(X)

# plot the clusters
plt.scatter(X[labels_true == 0,0],X[labels_true == 0,1], color='green', label='Cluster 1')
plt.scatter(X[labels_true == 1,0],X[labels_true == 1,1], color='orange', label='Cluster 2')
plt.scatter(X[labels_true == 2,0],X[labels_true == 2,1], color='blue', label='Cluster 3')

plt.legend(scatterpoints=1)
plt.title("Concentric synthetic data")
plt.show()
#####
# Compute DBSCAN
db = DBSCAN(eps=0.3, min_samples=10).fit(X)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_

# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)

print('Estimated number of clusters: %d' % n_clusters_)
print('Estimated number of noise points: %d' % n_noise_)
print("Homogeneity: %0.3f" % metrics.homogeneity_score(labels_true, labels))
print("Completeness: %0.3f" % metrics.completeness_score(labels_true, labels))
print("V-measure: %0.3f" % metrics.v_measure_score(labels_true, labels))
print("Adjusted Rand Index: %0.3f"
      % metrics.adjusted_rand_score(labels_true, labels))
print("Adjusted Mutual Information: %0.3f"
      % metrics.adjusted_mutual_info_score(labels_true, labels))
print("Silhouette Coefficient: %0.3f"
      % metrics.silhouette_score(X, labels))
```

<sup>a</sup>[https://colab.research.google.com/drive/1L\\_OLxY73G1T6GsZkIu-SxMowPsQ6zTzt?usp=sharing#scrollTo=b1E2iAIQb4l](https://colab.research.google.com/drive/1L_OLxY73G1T6GsZkIu-SxMowPsQ6zTzt?usp=sharing#scrollTo=b1E2iAIQb4l)

**Code 17.8.2.** Python Code ([https://colab.research.google.com/drive/1L\\_OlXy73G1T6GsZkIu-SxMowPsQ6zTzt?usp=sharing#scrollTo=mb1E2iAIQb4l](https://colab.research.google.com/drive/1L_OlXy73G1T6GsZkIu-SxMowPsQ6zTzt?usp=sharing#scrollTo=mb1E2iAIQb4l)) for DBSCAN (cont.):

```
% Below is the output from previous commands
% Estimated number of clusters: 3
% Estimated number of noise points: 18
% Homogeneity: 0.953
% Completeness: 0.883
% V-measure: 0.917
% Adjusted Rand Index: 0.952
% Adjusted Mutual Information: 0.916
% Silhouette Coefficient: 0.626

# plot the clusters
plt.scatter(X[labels == 0,0],X[labels == 0,1], color='green', label='Cluster 1')
plt.scatter(X[labels == 1,0],X[labels == 1,1], color='orange', label='Cluster 2')
plt.scatter(X[labels == 2,0],X[labels == 2,1], color='blue', label='Cluster 3')
plt.scatter(X[labels == -1,0],X[labels == -1,1], color='black', label='Noise')

plt.legend(scatterpoints=1)
plt.title("Concentric synthetic data")
plt.show()

# #####
# Plot result
import matplotlib.pyplot as plt

# Black removed and is used for noise instead.
unique_labels = set(labels)
colors = [plt.cm.Spectral(each)
          for each in np.linspace(0, 1, len(unique_labels))]
for k, col in zip(unique_labels, colors):
    if k == -1:
        # Black used for noise.
        col = [0, 0, 0, 1]

    class_member_mask = (labels == k)

    xy = X[class_member_mask & core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=14)

    xy = X[class_member_mask & ~core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=6)

plt.title('Estimated number of clusters: %d' % n_clusters_)
plt.show()
```

## 17.9 K-Nearest Neighbors Algorithm

One of the simplest classification and regression methods is the K-nearest neighbors algorithm (K-NN). It is a supervised method that interrogates, for example, the classes of the  $k$  neighbors nearest to the test point, and assumes the class of the majority (Altman,1992; [https://en.wikipedia.org/wiki/K-nearest\\_neighbors\\_algorithm](https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm)). The training set consists of the points previously labeled or assigned numbers in the case of regression. For example, if the  $k = 7$  nearest neighbors consisted of 2 democrats, 1 socialist, and 4 republicans, then the test point assumes the identity of the majority class, a republican. If  $k = 1$  then the test point assumes the identity of the nearest neighbor. The Euclidean distance metric is most often used to measure *distance*, but if other metrics are used then the units must be normalized (Pirayonesi and El-Diraby, 2020; Hastie et al., 2001) for optimal results.

If the input points are characterized by a property value, instead of a class type, the property value of the test point is the average property value of the k-nearest neighbors. Weights can be assigned to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. A common weighting scheme is to assign the weight as  $\frac{1}{d}$ , where  $d$  is the distance to the neighbor, similar to linear interpolation.

A problem with K-NN is an unbalanced data set where there are many more members of one class than another (Coomans and Massart, 1982). To overcome this problem, one can incorporate (or create) more data to balance the number of points in each class. Another possible remedy is to

weight the class (or value, in regression problems) of each of the  $k$  nearest points by the inverse of the test-neighbor distance  $\frac{1}{d}$ .

If the number of components is too large then a gradient method can suffer from the curse of dimensionality ([https://en.wikipedia.org/wiki/Curse\\_of\\_dimensionality](https://en.wikipedia.org/wiki/Curse_of_dimensionality)) when a Euclidean distance metric is used for the objective function. This is also a problem for K-NN. That is, the volume of a high-dimensional space grows exponentially with dimension but the distance between neighbors grows much less. In fact, as the dimension grows the distance to the nearest neighbor approaches that to the farthest neighbor (Beyer et al., 1999; Durrant, R. and A. Kabán, 2009). Beyer et al. (1999) show an example where the distinction between the farthest and nearest neighbors blur with as few as 15 dimensions. This means that an objective function or nearest neighbor metric with an  $L_2$  distance metric will not compute large differences between distant neighbors in a high-dimension data space. This can slow down convergence with a gradient optimization method or result in poor clustering with K-NN. In this case the data should be skeletonized to a smaller set of data and/or model. This is called feature selection in the ML community.

## 17.10 Self-organizing Map

A self-organizing map (SOM) is an unsupervised learning algorithm that produces clusters from a high-dimensional data set and maps these clusters onto a lower dimensional space, such as a 2D map ([https://en.wikipedia.org/wiki/Self-organizing\\_map](https://en.wikipedia.org/wiki/Self-organizing_map)). Unlike K-means and DBSCAN for high-dimensional input vectors, a SOM can produce the clustered points in a convenient 2D space that can be easily inspected for its effectiveness in separating points from different clusters. This reduction in dimension is similar to the goal of a non-linear PCA where high-dimensional data are approximated by a low-dimensional representation of the data. The SOM was originally developed by Kohonen (1982) and has a variety of implementations in the geosciences (Roy et al., 2011; Roy et al., 2013; Hussein et al., 2020; Hussein et al., 2021; Ramdani, 2022).

Some key ideas associated with the SOM algorithm are illustrated in the Figure 17.19a example, where several solar systems surround three newly-arrived asteroids. These asteroids are in a transitory state but will eventually be captured by one of the solar systems. Which solar system will gravitationally attract which asteroid? This question can be answered intuitively by an alternating variables approach: freeze the gravitational pull of all the solar systems except a randomly selected one such as solar system  $A$ . In this example, the distance  $r_{A1}$  of asteroid #1 is closest to  $A$  and it is pulled more closely to  $A$  while the neighboring asteroids #2 and #3, with distances  $r_{A1} \ll r_{B2} < r_{C1}$ , are barely nudged towards  $A$ . This is the first iteration of the alternating variable procedure that leads to the configuration in Figure 17.19b. The longest white arrow in Figure 17.19b indicates that asteroid #1 moved the farthest compared to the other ones after the first iteration.

The next iteration in Figure 17.19c freezes the gravitational attractions of  $A$  and  $C$ , and activates that of  $B$ . The asteroids are then pulled towards  $B$  with a force inversely proportional to the square of their distance from  $B$ . Again, the configuration of asteroids becomes readjusted so that the asteroid #2 closest to  $B$  moves more than the others. This freezing and unfreezing of the gravitational attractions of the solar systems is repeated, as in Figure 17.19d, until the asteroids are captured by their closest solar systems. The captured asteroid in each system is used to classify it as belonging to the greediest solar system. This clustering procedure is similar to the SOM workflow presented below.

1. The first step of the SOM method is to define the number  $N$  of *neurons*  $\mathbf{w}^{(n)}$  for  $n \in \{1, 2, \dots, N\}$  that will be used to identify the number of clusters in the  $M$  input points  $\mathbf{x}^{(m)}$  for any  $m \in \{1, 2, \dots, M\}$ , where typically  $M \gg N$ . The components of  $\mathbf{w}^{(n)}$  are given random values, and the dimension of  $\mathbf{w}^{(n)}$  is the same dimension as the input points



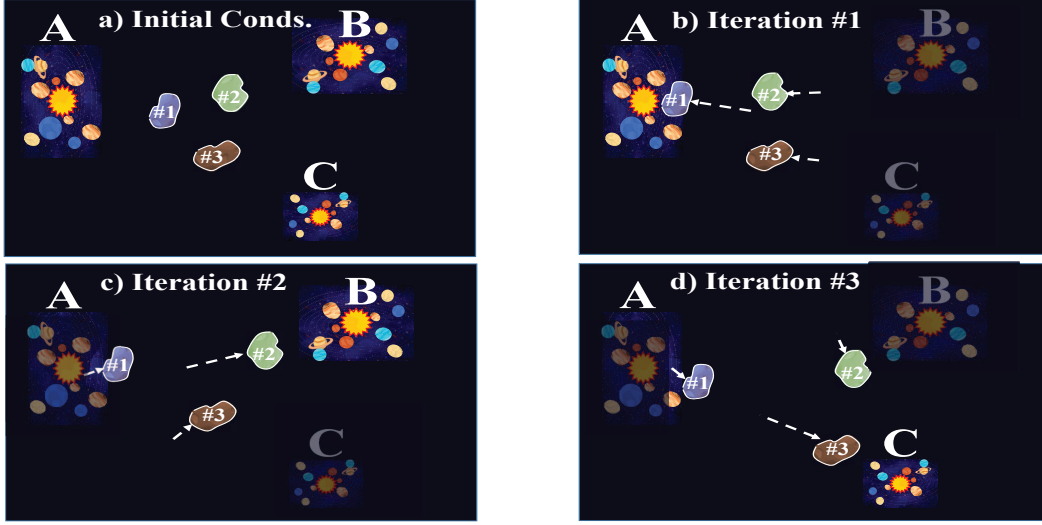


Figure 17.19: Configurations of a) initial state of the asteroids #1, #2, #3 and later configurations of these asteroids after the b) first, c) second and d) third iterations. At each iteration, the gravitational pull of only one of the solar systems is briefly activated.

$\mathbf{x}^{(m)}$ . The neuron weights  $\mathbf{w}^{(n)}$  are analogous to the positions of the asteroids in Figure 17.19, but in general the components of  $\mathbf{w}^{(n)}$  characterize the different features of the input data.

The initial positions of  $\mathbf{w}^{(n)}$  can be on a semi-uniform grid of points in data space as shown by the filled-black circles in Figure 17.20a. Other options are to assign random locations to the neurons, which is discouraged by Kohonen (2005). A viable alternative is to apply a PCA to the input data and only select a few dominant eigenvectors to be the initial vectors of the neurons (Kohonen, 2005).

2. Compute the distance between all of the neurons and a randomly selected input point  $\mathbf{x}^{(m)}$ :

$$d_{nm} = \|\mathbf{w}^{(n)} - \mathbf{x}^{(m)}\| \text{ for } n \in \{1, 2, \dots, N\}. \quad (17.23)$$

Select the neuron  $\mathbf{w}^{(n^*)}$  which is closest to  $\mathbf{x}^{(m)}$ :

$$n^* = \arg \min_n d_{nm}. \quad (17.24)$$

This neuron is denoted as the winning neuron, or winner at this iteration.

3. Move the winning neuron  $\mathbf{w}^{(n^*)}$  and its close neighbors  $\mathbf{w}^{(n)}$  toward the  $m^{th}$  point  $\mathbf{x}^{(m)}$  using the update formula

$$\Delta \mathbf{w}^{(n)} = \eta(t) T_{n^*,n}(t) (\mathbf{x}^{(m)} - \mathbf{w}^{(n)}), \quad (17.25)$$

where  $\eta(t)$  is the step length, aka the learning-rate factor, defined as

$$\eta(t) = \eta_0 \exp\left(-\frac{t}{\tau}\right); \quad 0 < \eta(t) < 1. \quad (17.26)$$

Here,  $t$  is the iteration index and the Gaussian neighborhood function  $T_{n^*,n}(t)$  (Kohonen,

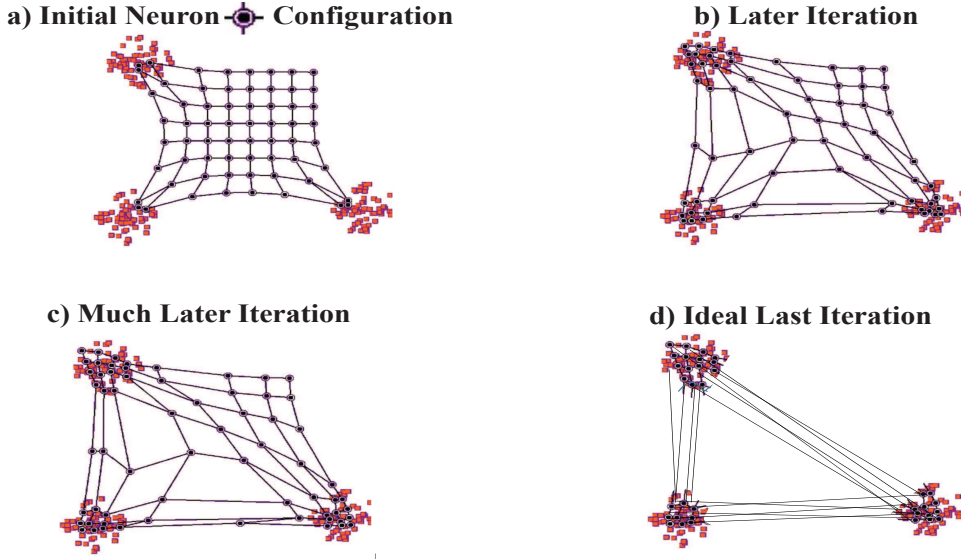


Figure 17.20: 2D configurations of a) initial state, b) later iteration, c) much later iteration and d) ideal final iteration of the neurons  $\mathbf{w}^{(n)}$  denoted as black-filled circles. The input points  $\mathbf{x}^{(m)}$  in data space are denoted by red-filled circles. The proximity of the input points to the bunched neurons at the final iteration determines their class type. Pictures adapted from the movie in [https://en.wikipedia.org/wiki/Self-organizing\\_map#cite\\_note-KohonenMap-1](https://en.wikipedia.org/wiki/Self-organizing_map#cite_note-KohonenMap-1).

2005) is defined as

$$T_{n^*,n}(t) = \exp\left(-\frac{\|\mathbf{w}^{(n^*)} - \mathbf{w}^{(n)}\|^2}{2\sigma(t)^2}\right), \quad (17.27)$$

where

$$\sigma(t) = \sigma_0 \exp\left(-\frac{t}{\tau_0}\right). \quad (17.28)$$

The Gaussian neighborhood function  $T_{n^*,n}(t)$  plays a role that is similar to the gravitational force exerted by the solar system in Figure 17.19. For example, if the neighboring neuron  $\mathbf{w}^{(n)}$  is extremely close to the winning neuron  $\mathbf{w}^{(n^*)}$ , then  $T_{n^*,n}(t) \approx 1$  so that the neighbor neuron  $\mathbf{w}^{(n)}$  is pulled with the same force towards  $\mathbf{x}^{(m)}$  as the winning neuron  $\mathbf{w}^{(n^*)}$ . On the other hand, if the neighbor is far from the winning neuron, then  $T_{n^*,n}(t) \approx 0$  and the  $n^{th}$  neuron does not move at that iteration. Being far from the winning neuron also means that is much farther from the target data point  $\mathbf{x}^{(m)}$  and so it is not pulled to it.

The hyperparameters  $(\tau_0, \sigma_0, \eta_0)$  are determined by trial and error, where  $\sigma_0$  determines the width of the neighborhood around the winner at  $\mathbf{w}^{(n^*)}$  where its neighbors will move towards the selected input point. For example, if  $\sigma >> 0$  then all neighbors will move towards the selected input point, just like the solar system example.

4. The newly computed neurons are then used to iteratively repeat steps 2-4 until acceptable convergence. The neurons  $\mathbf{w}^{(i)}$  iteratively change location with each iteration in the SOM method.

## Reduction of Dimensionality

High dimension of  $\mathbf{w}^{(i)}$  reduced to single scalar value  $\Delta_{nn}$ .

Lateral Distance

$$\Delta_{nn'} = \|\mathbf{w}^{(n)} - \mathbf{w}^{(n')}\|$$

$$\Delta_{12} = 2$$

$$\Delta_{1x} = 2.5$$

$$\Delta_{2x} = 1$$

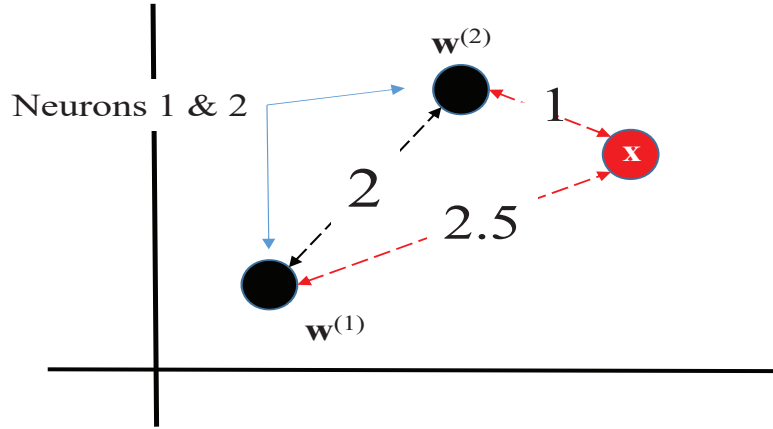


Figure 17.21: High-dimensionality of neurons  $\mathbf{w}^{(n)}$  can be reduced by computing the Euclidean lateral distance  $\Delta_{nn'} = \|\mathbf{w}^{(n)} - \mathbf{w}^{(n')}\|$  between nodes  $\mathbf{w}^{(n)}$  and  $\mathbf{w}^{(n')}$ . These nodes can then be plotted on a 2D plane such that they are separated by the lateral distance  $\Delta_{nn'}$ . The dimension of the data point  $\mathbf{x}$  can be reduced in the same way except it must be located a distance  $\Delta_{nx}$  from the  $n^{th}$  node and a distance  $\Delta_{n'x}$  from the  $n'^{th}$  node. Partly adapted from [https://www.youtube.com/watch?v=g8O6e9C\\_CfY](https://www.youtube.com/watch?v=g8O6e9C_CfY).

5. For ease of inspection, the high 3D dimension of the neurons are projected to an appropriate 2D plane. More generally, the final neurons  $\bar{\mathbf{w}}^{(n)}$  with arbitrary high dimension can be mapped to a 2D plane by computing the lateral distance

$$\Delta_{nn'} = \|\bar{\mathbf{w}}^{(n)} - \bar{\mathbf{w}}^{(n')}\|, \quad (17.29)$$

between the  $n$  and  $n'$  neurons. These nodes are placed in the 2D plane in Figure 17.21 such that they are separated by their lateral distance. Similarly, the lateral distance  $\Delta_{xn} = \|\bar{\mathbf{w}}^{(n)} - \bar{\mathbf{x}}\|$  between the  $\mathbf{x}$  data point and the  $\bar{\mathbf{w}}^{(n)}$  neuron is also computed. Then,  $\mathbf{x}$  is located at the distance  $\Delta_{xn}$  from the  $\bar{\mathbf{w}}^{(n)}$  neuron and the distance  $\Delta_{xn'}$  from the  $\bar{\mathbf{w}}^{(n')}$  neuron.

Kohonen (2005) suggests the following batch computational scheme for efficient execution of the SOM.

- Set the number of neurons and initialize the values of their components.
- For each neuron, collect a list of the input samples  $\mathbf{x}^{(m)}$ , which belong to its neighborhood. Move the neurons towards this mean vector. Compute, for each new neuron vector, the mean over the respective list.
- Repeat step 2 a few times.

The pseudo-code for SOM is shown below.

```

1. Get M data points  $x(i)$  for the input
2. Define number N of neurons  $w(n)$  and their distribution.
3. Define the neighborhood function  $T(n*,n)$ 
4. Initialize  $w(j)$ 
5 for iter=1:niter
    for i=1:M
         $n*=\operatorname{argmin}(\|x(i)-w(n)\|^2)$ 
        for n=1:N
             $w(n) = w(n) + \alpha T(n*,n) (x(i)-w(n))$ 
        end
    end
end
end

```

### 17.10.1 MATLAB SOM Examples

Three examples will be used to illustrate the application of a SOM to simple data.

#### Ten $4 \times 1$ Input Vectors

The first MATLAB example (see <https://www.mathworks.com/help/deeplearning/ug/irs-clustering.html>) demonstrates the clustering capability of a SOM applied to the ten input  $4 \times 1$  vectors shown in the top row of Figure 17.22. The input  $4 \times 1$  vectors are plotted in the  $x_1 - x_2 - x_3$  volume shown in Figure 17.22a, where the 4<sup>th</sup> dimension is not shown. Pairs of points are clustered around one of the three vertices  $(1, 0, 0, 0)$ ,  $(0, 1, 0, 0)$ , and  $(0, 0, 1, 0)$ ; and four points are clustered around  $(0, 0, 0, 1)$ . The starting configuration of four neuron vectors  $\mathbf{w}^{(n)}$ , each with dimension  $4 \times 1$ , are defined as the round circles in b), and after 200 iterations the weights are clustered around, not surprisingly, the data-space vertices (red circles) in d). Figure 17.22c indicates the number of input points associated with each of the four neurons  $\mathbf{w}^{(n)}$ .

After 200 iterations, the neuron's weights take on their final form as shown in Figure 17.22e-17.22h. As expected, the final neuron vectors are roughly proportional to  $(1, 0, 0, 0)$ ,  $(0, 1, 0, 0)$ ,  $(0, 0, 1, 0)$ , and  $(0, 0, 0, 1)$ . The MATLAB code that generated these images is below

```

% Solve a Clustering Problem with a Self-Organizing Map
% Script generated by Neural Clustering app
% Created 21-May-2021 10:15:01
%
% This script assumes these variables are defined:
%
% irisInputs - input data characterizes different flowers.
load iris_dataset
% irisInputs - 4x1 input data characterized by points clustered around vertices
irisInputs=10*[1 1 0 0 0 0 0 0 0 .002; ...
               0 .0001 1 1 0 .0001 0 0 0 .002;...
               0 .0001 0 0 1 .9 0 0 0 0;...
               0 -.0001 0 0 0 .001 1 1.0001 1 1]

x = irisInputs;

```

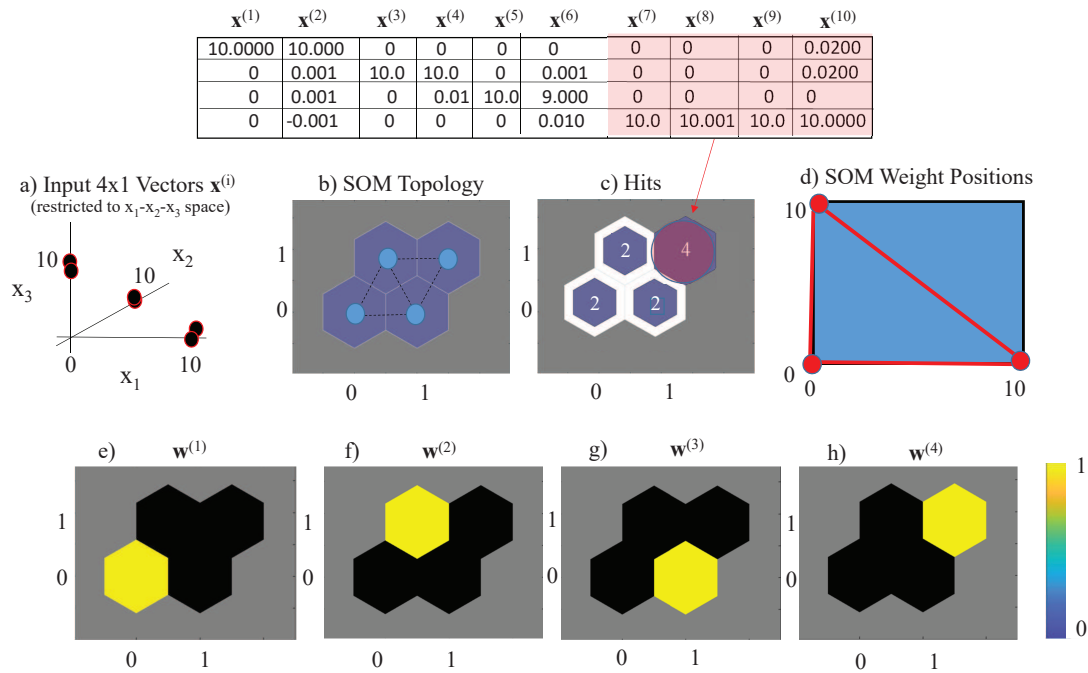


Figure 17.22: MATLAB results for applying the SOM algorithm to the ten  $4 \times 1$  input vectors  $\mathbf{x}^{(i)}$  for  $i \in \{1, 2, \dots, 10\}$  shown in the top row. Six input points  $\mathbf{x}^{(i)}$  are plotted in the  $x_1 - x_2 - x_3$  domain in a), and the other 4 (not shown) are clustered around the point  $\mathbf{x} = (0, 0, 0, 1)$ . We assign four  $4 \times 1$  neurons  $\mathbf{w}^{(i)}$  for  $i \in \{1, 2, 3, 4\}$  with unknown weights, where the 2D SOM topology of the 4 neurons is shown in b); each blue circle denotes the location of a neuron. c) denoted the number of input points  $\mathbf{x}^{(i)}$  that are captured by each neuron; there are 4 input points (denoted by the red highlight in the top row) that form a cluster around the upper-right neuron after 200 iterations. After training with 200 iterations, the four  $4 \times 1$  neuron vectors  $\mathbf{w}^{(i)}$  are plotted in e)-h).

```

% Create a Self-Organizing Map
dimension1 = 2;
dimension2 = 2;
net = selforgmap([dimension1 dimension2]);

% Train the Network
[net,tr] = train(net,x);

% Test the Network
y = net(x);

% View the Network
view(net)

% Plots
% Uncomment these lines to enable various plots.
figure, plotsomtop(net)
figure, plotsomnc(net)
figure, plotsomnd(net)
figure, plotsomplanes(net)
figure, plotsomhits(net,x)
figure, plotsompos(net,x)

```

## 1D Map Space

A simple example of a SOM with a 1D map space is shown in Figure 17.23. There are 6 nodes in the one-dimensional map space in Figure 17.23e. The neighborhood function is defined as  $T_{n,n'} = \exp(-\|\mathbf{w}^{(n)} - \mathbf{w}^{(n')}\|^2) - 0.4$ , which means  $T_{n,n'}$  could be negative. The locations of the six neuronal nodes are indicated by the colored dots in Figure 17.23a. At the first SOM iteration,  $\mathbf{x}^{(1)}$  is the target data point and so we find the nearest neuron as  $\mathbf{w}^{(n^*)}$ . Equation 17.25 is used to update all the nodes as shown by the move vectors in Figure 17.23b and the new positions of the neurons in Figure 17.23c. After updating the neurons,  $\mathbf{x}^{(2)}$  in Figure 17.23c is the new target data point and Figure 17.23d shows the final neuronal configuration after a sufficient number of iterations. To classify a new input point, simply find its nearest neuron  $\mathbf{w}^{(n)}$  and it will adopt its class.

To visualize the six neurons of the SOM, a U-matrix is used, which is defined as  $U_n = \sum_{neighbors} \|\mathbf{w}^{(n)} - \mathbf{w}_{neighbor}\|^2$ . In Figure 17.23g, we define  $U_n = \|\mathbf{w}^{(n)} - \mathbf{w}^{(n+1)}\|^2$  where there are valleys and peaks. Valleys represent a closely-packed distribution of neurons, where the number of valleys help us decide the number of clusters in data space. In contrast, the peaks represent low density of neurons, so it marks the boundaries of different clusters, which are used to extract texture information in the images. This visualization procedure can be used for a 2D or higher-dimensional SOMs.

## 2D Map Space

The input data consist of 150  $4 \times 1$  vectors, where each vector component characterizes a certain feature of an iris-like flower. The SOM goal is to identify the different classes of iris-like flowers.

Applying a SOM with 25 neurons to this data set gives the hit map shown in Figure 17.24a, where the two lower-left neurons are surrounded by at least 20 input data points. The final configuration of the 25 neurons is depicted by the red dots in c) after 200 iterations. The SOM neighbor distances are plotted in b), which suggest that the boundary separating two clusters is described by the dashed black lines in b) and c). The boundary obtained from the neighbor distance plot is similar to the

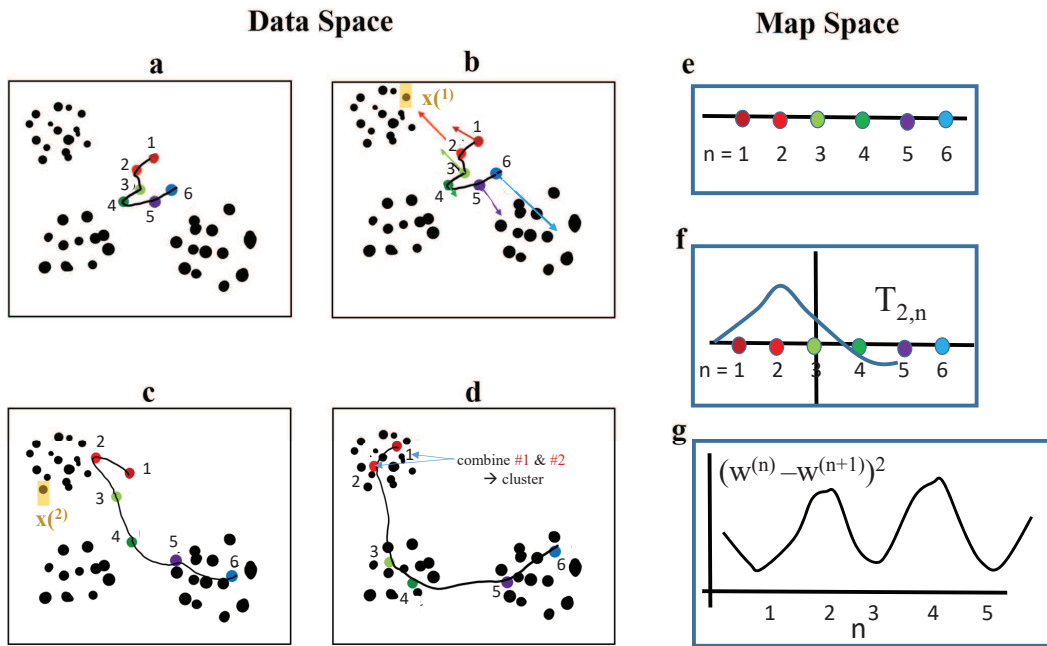


Figure 17.23: Example of the SOM procedure for a 1D map space. a) Black dots represent data points  $\mathbf{x}^{(i)}$  and the colored dots are neuron vectors  $\mathbf{w}^{(n)}$  that represent the colored node points in the 1D map space shown in e). b) Update directions of all the nodes  $\mathbf{w}^{(k)}$  in data space are indicated by arrows when selecting  $\mathbf{x}^{(1)}$  as the first target point. After updating, the locations of feature vectors  $\mathbf{w}^{(k)}$  are shown in c) and a new data point  $\mathbf{x}^{(2)}$  is selected for the next iteration. After many iterations, the distribution of  $\mathbf{w}^{(n)}$  for the SOM is shown in d). f) displays the shape of the neighborhood function  $T_{2,n}$  and g) depicts the U-matrix for the well-trained SOM in d).

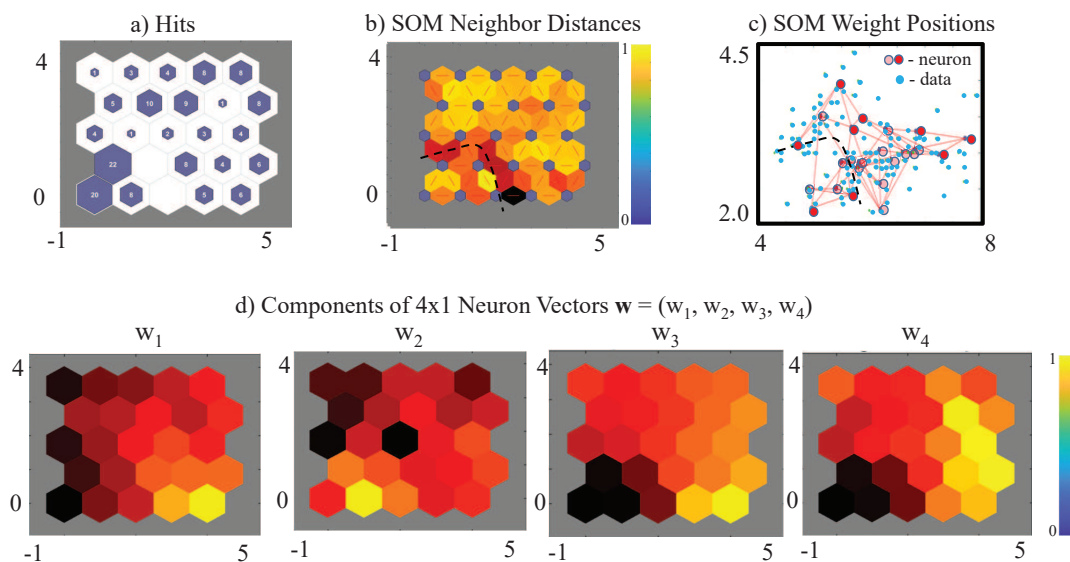


Figure 17.24: Results of a SOM applied to the Iris data set. The input data (blue dots in c) consisted of 150  $4 \times 1$  vectors. The lower row of plots depicts each component  $w_i$  of the 25 neuron vectors which characterizes a specific feature of an Iris-like flower. These  $4 \times 1$  neuron vectors are mapped to the red dots in the 2D space shown in c). The black dashed lines in b) and c) roughly delineate the 2D boundary between different clusters, similar to the U-matrix boundary in Figure 17.24g.

U-matrix boundary in Figure 17.23g.

## 17.11 Numerical Examples

Four examples of applying clustering to geophysical data are presented. The first applies K-means clustering with softmax weighting for automatic picking of the stacking velocity from semblance panels (Yilmaz, 2001). The centroids of each cluster are then connected to form the stacking velocity curve. The second estimates the layers associated with velocity models generated by VAE. The third example applies DBSCAN to earthquake locations in order to identify the location of faults. The final example shows how a self-organizing mapping method can be used to extract the important microporosity parameters measured in borehole specimens.

### 17.11.1 Picking Stacking Velocities by K-means Cluster Analysis

Seismic data generated by a single shot (see Figure 27.6a) are most often too noisy to be used for imaging the subsurface properties. In particular, the primary reflections are degraded by both random and coherent noise. To increase the SNR level of the primaries, primary reflections from the same subsurface reflection point at  $\mathbf{x} = (x_o, z)$  are generated by shots at different locations. The traces where the midpoints of the sources and receivers have the same midpoint coordinate<sup>11</sup>  $x_o$  are assembled into a *common midpoint gather* (CMG) as illustrated in Figure 27.6b. The *common*

<sup>11</sup>For pedagogical convenience, we assume a 2D survey geometry and a layered medium where the sources and receivers are along a line on a horizontal free surface.



*midpoint* reflections from the same reflector are shifted in time so they all align with one another (see Figure 27.6c), and then they are stacked together to get the stacked trace in Figure 27.6d with an increased SNR. This alignment time is the two-way zero-offset time  $T_o$  of a zero-offset primary reflection reflecting at  $\mathbf{x}$ ; here, zero-offset refers to the fact that the separation between the source and receiver has zero offset and both are located at  $(x_o, 0)$  on the horizontal free surface. This assumes a layered medium and we use semblance analysis (Yilmaz, 2001) to find the best way to time shift and stack the traces to get the primary reflection with the best SNR.

As illustrated in Figure 17.25, the goal of semblance analysis is to find the stacking velocity  $V_{stack}$  as a function of two-way zero-offset traveltime  $T_o$  such that the hyperbola

$$t(x, V_{stack}) = \sqrt{\frac{x^2}{V_{stack}^2} + T_o^2}, \quad (17.30)$$

best fits the reflections in a common midpoint gather. Here, the midpoint coordinate is  $x_o = (x_r - x_s)/2$  and the offset between the source and receiver is  $x = x_r - x_s$ . The primary reflections in each trace  $d(x, t)$  approximately follow the traveltimes in equation 17.30 for the correct values of  $V_o$  and  $T_o$ , so the stacking procedure uses the semblance equation given by

$$S(V_{stack}, T_o) = \frac{\sum_x d(x, t(x, V_{stack}))}{\sum_x d(x, t(x, V_{stack}))^2}, \quad (17.31)$$

where  $S(V_{stack}, T_o)$  is the semblance panel associated with the midpoint coordinate  $x_o = (x_r - x_s)/2$ . To generate a smooth semblance panel, the trace amplitudes are typically stacked together along a half-period wide strip centered about the moveout curve described in equation 17.30. The black curve in Figure 17.25 joining the peak values of the semblance energy is picked as the best estimate of the stacking velocity  $V_{stack}(T_o)$  as a function of zero-offset traveltime  $T_o$ . This optimal stacking velocity produces the optimal "stacked" trace at zero-offset, where optimal corresponds to reflection events with the highest signal-to-noise ratio.

The problem with picking the semblance curve in Figure 17.25 is that it can easily get confused by strong noise or by free-surface multiples that stack incoherently. In this case, semblance curves must be manually picked by hand which can be extremely time consuming for large 3D data sets. To mitigate this expense, (Chen, 2018a-b) proposed a bottom-up method for K-means clustering to automatically group the large energy events in  $S(V_{stack}, T_o)$  into different clusters.

Chen proposed the following algorithm.

1. Set the initial number of clusters  $K=4$ . Assign the centroid point of the  $k$  cluster to be  $(V_o^{(k)}, T_o^{(k)})$ . These  $K$  points are selected at equi-spaced  $T_o$  intervals along an initial semblance curve.
2. Apply K-means clustering for several iterations until a stable group of  $K$  clusters has emerged.
3. Increment the value of  $K$  by 1 and repeat step 2. The value of  $K$  is incrementally increased until the residual misfit in equation 17.3 does not decrease by a significant amount.
4. The centroids of the final cluster are used to form the final semblance curve.

As an example, the bottom-up K-means algorithm is applied to the common midpoint gather (CMG) in Figure 17.26a. Here, the data are computed for a five-layer velocity model and the semblance spectrum is shown in Figure 17.26b. The bottom-up K-means clustering algorithm is used to pick the clusters for  $1 \leq K \leq 10$ , and the centroids are used to pick the points for the semblance curve. Six clusters, with centroids denoted by the red dots in Figure 17.27a, were found to be optimal before the decrease in the  $L_2$  residual became negligible in b). These centroids are connected by the dashed magenta line, which agrees with the one manually picked from the semblance plot.

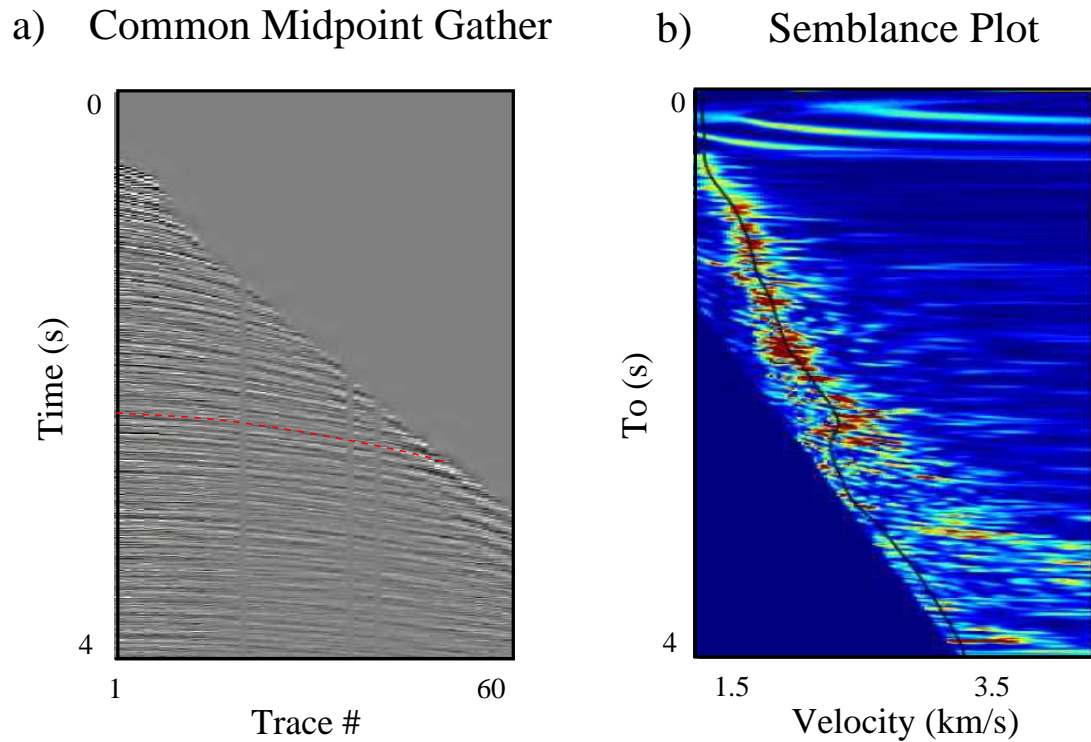


Figure 17.25: a) Common midpoint gather of traces assembled from traces recorded by a seismic reflection survey. For a dense survey, a shot can be located at each source position. The traces in a) are summed and weighted along different hyperbolas (e.g., the red dashed curve) described by equation 17.3 for different values of  $V_{stack}$  and  $T_o$  to get the semblance plot shown in b). The black line in b) describes the stacking velocity curve that is used to estimate the velocity as a function of depth. Images extracted from <http://www.ahay.org/RSF/book/geo384s/hw3/nmo/fig/cmp700.png>.

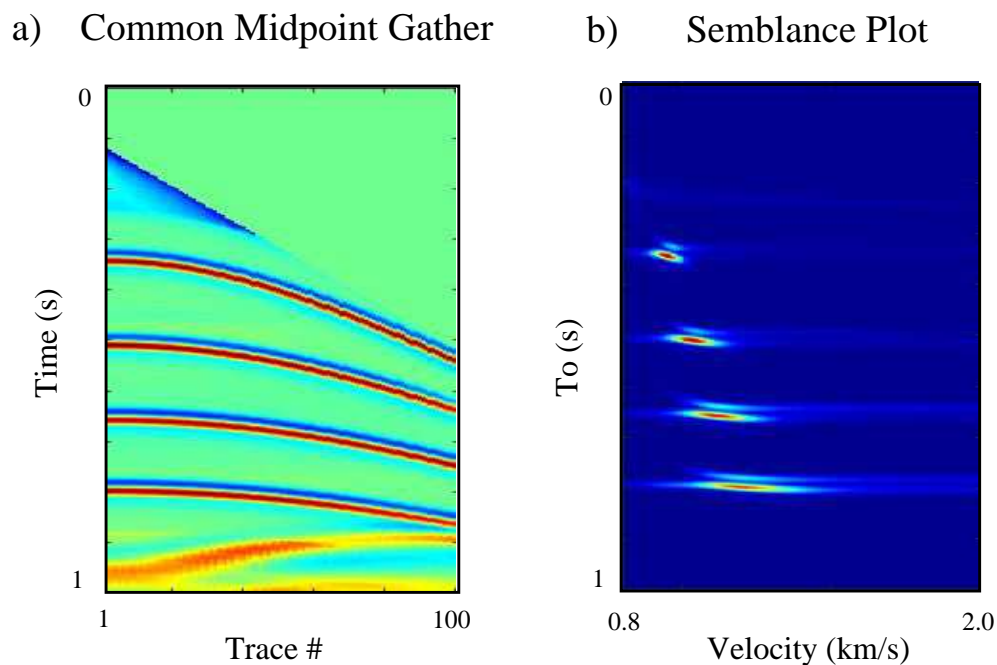


Figure 17.26: a) Common midpoint gather (CMG) for a 5-layer velocity model and b) semblance spectrum (Chen, 2018a-b).

A more challenging example is to estimate the semblance curve from data generated from the Marmousi model in Figure 17.28a. In this case, there are 100 traces in most of the common midpoint gathers, and the semblance curve is shown in Figure 17.28c. Here, the number of clusters was selected to be  $K = 12$  because this represented the largest cluster number before the error decrease became small. A plot of misfit values against cluster numbers is shown in Figure 17.28d.

A final example is for the field data shown in Figure 17.29a. The semblance spectrum by the K-means algorithm is shown in Figure 17.29b and agrees well with the one estimated by a human interpreter. The misfit error plotted against  $K$  in Figure 17.29c and the *elbow criterion* suggests that  $K = 5$  is the optimal number of clusters, although  $K = 12$  was selected for to provide a high-resolution final semblance curve.

One of the problems with semblance analysis for marine data is that free-surface multiples can contribute to strong semblance energy at stacking velocities near that of the water velocity. In this case constraints should be imposed to steer the K-means picking algorithm away from choosing the semblance curve associated with multiples. See Exercises 2-3.

The information weighted clustering (IWC) algorithm described in equation 17.13 is now applied to the GOM data. The starting cluster points are shown as 15 red dots along the lines in Figure 17.30b and 17.30d. The IWC algorithm converged to the red dots intersecting the black curves in Figure 17.30b and Figure 17.30d. These auto-picked curves ignored the multiples at the lower semblance velocity because the IWC algorithm weighted the clusters points with the amplitude of the semblance values in Figure 17.30b and 17.30d. The semblance values for the primaries appear to be much larger than those for the perceived multiples.

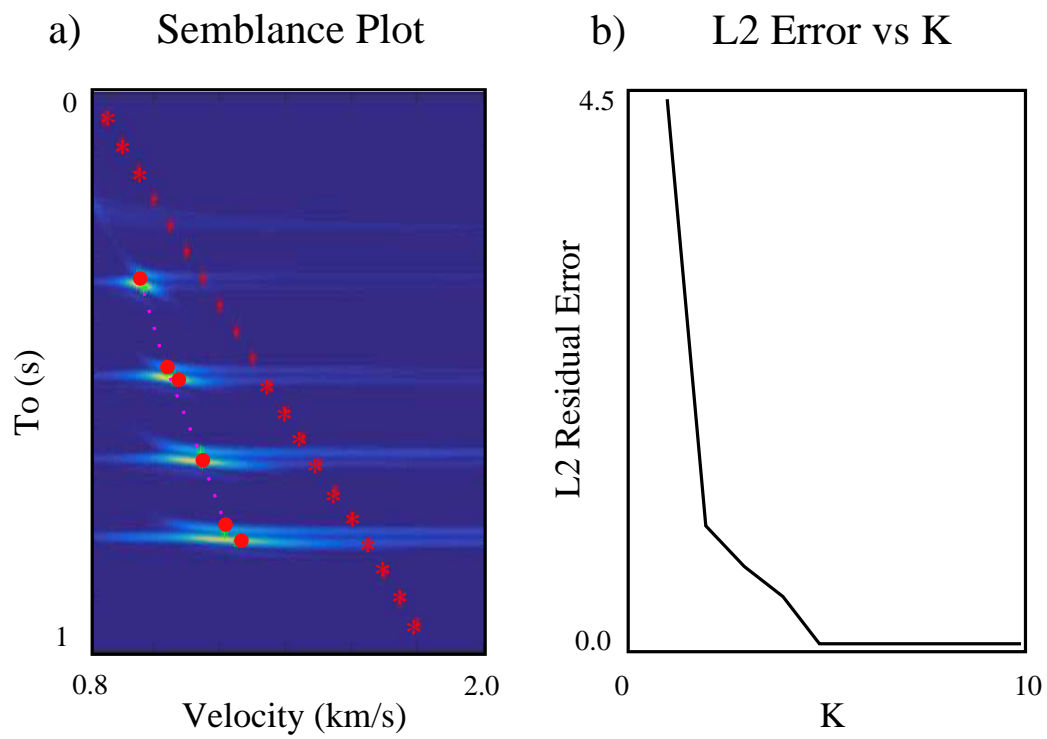


Figure 17.27: a) Final cluster configuration where the centroid of each cluster is marked by a red dot. The red stars are the initial starting model and b) denotes the sum of the squared residuals plotted against the number of clusters  $K$  (Chen, 2018a-b).

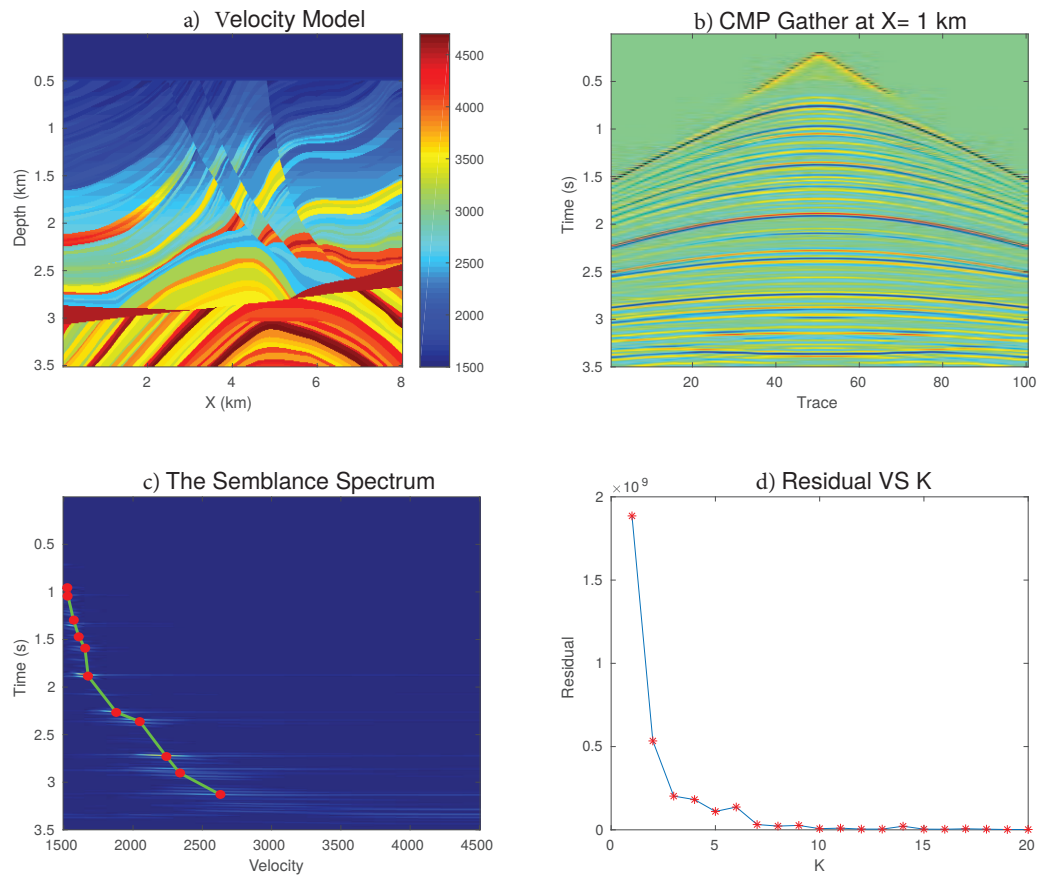


Figure 17.28: a) Marmousi model, b) common-midpoint gather, c) semblance spectrum with centroids picked by the K-means algorithm, and d) plot of  $L_2$  norm of residuals vs  $K$  (Chen, 2018a).

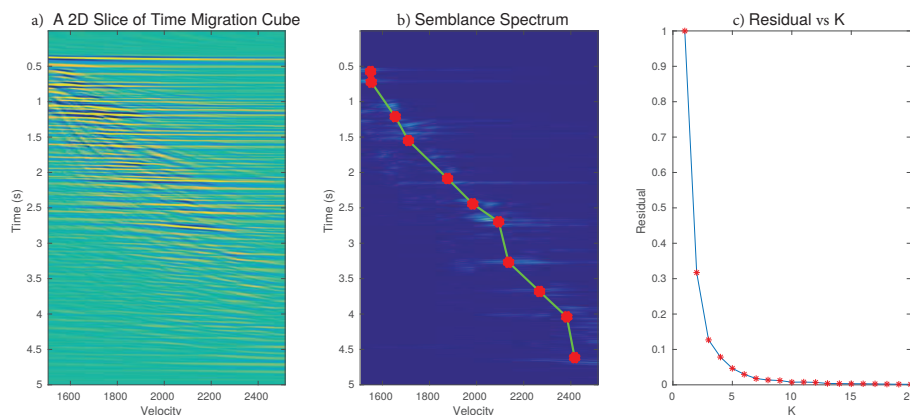


Figure 17.29: a) A 2D slice of the time migration cube of Gulf of Mexico (GOM) data at  $X = 1$  km and b) semblance spectrum with the semblance curve picked from the centroids of the final cluster configuration. c) Misfit error plotted against  $K$ , the number of clusters (Chen, 2018a).

### 17.11.2 Generating Models with Sharp Impedance Contrasts

The variational autoencoder (VAE) is a generative model that can generate the observation and target  $(\mathbf{x}, \mathbf{y})$ , or the observation  $\mathbf{x}$  given the target  $\mathbf{y}$ . It can learn the probability distribution of a training set of images, and then generate thousands of never-before-seen images that are similar, but more blurry (see Chapter 14 and Berthelot et al. (2018)).

A geophysical application of a VAE is to generate new velocity models from a training set of realistic velocity models. These new velocity models can then use a seismic forward modeling code to generate a never-before-seen set of seismic data for purposes of full waveform inversion (see Chapter 26.1.2). Unfortunately, the new velocity models obtained by VAE are smooth velocity models that are missing sharp velocity contrasts. Such contrasts are needed in order to render realistic velocity models.

To overcome this problem we can use VAE to generate smooth velocity models by training with velocity models having sharp interfaces. The VAE generated models are then passed through a K-means clustering algorithm to generate velocity models with sharp interfaces. The VAE and clustering workflow is shown in Figure 17.31, where the training set consisted of 2036 velocity models from the F3 Interpretation dataset (Silva et al., 2019). The number of epochs for the VAE was 50 epochs with an MSE loss of 0.577. Figure 17.32 depicts a seismic section and an interpreted velocity model from the F3 data (Silva et al., 2019).

After training, the VAE is used to generate an additional 8000 velocity models. Figure 17.33 depicts some of these training models. The output models from the VAE are shown as the first column of panels in Figure 17.34a-c. These VAE velocity models are characterized by very low resolution, as if a low-pass filter was applied to the original velocity model with sharp interfaces.

To mitigate this problem Gautam used K-means clustering to generate the VAE+cluster models shown in the right column of panels in Figure 17.34a-c. The clustering algorithm clearly delineated the sharp velocity boundaries to generate more realistic velocity models with sharp interfaces. However, if the VAE model is too complex with non-convex boundaries then the clustering algorithm can produce an unrealistic layered model, similar to the problems seen in Figure 17.13b and 17.13d. In this case a DBSCAN or SVC algorithm might be more appropriate if the layer boundaries form non-convex surfaces.

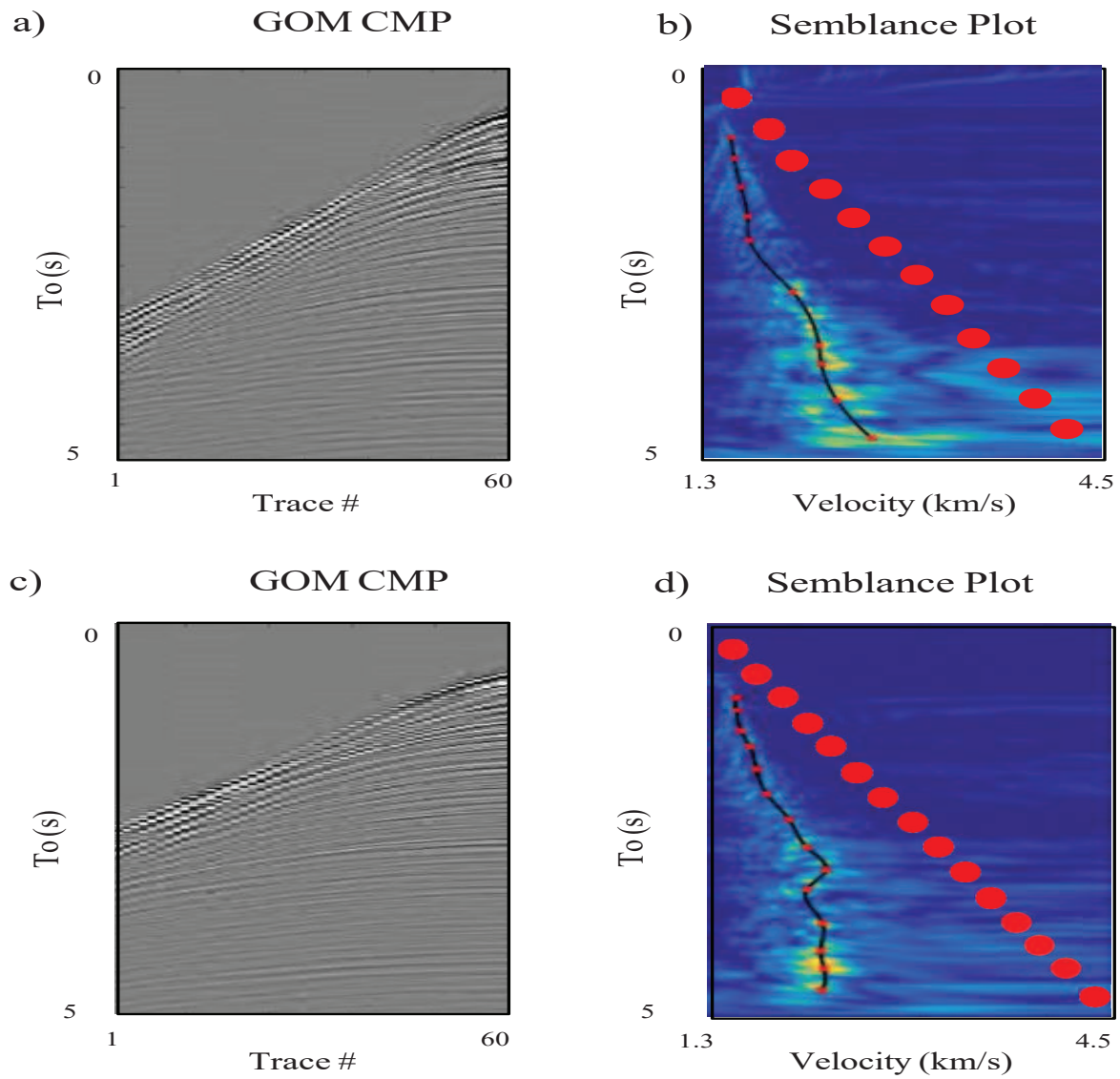


Figure 17.30: Common midpoint gathers and (right column) semblance images picked by the IWC algorithm where the large red points along a line are the initial cluster points and the black line is through the final red cluster points (Chen, 2018b). Note, there are 15 initial cluster points and the final number is 14.

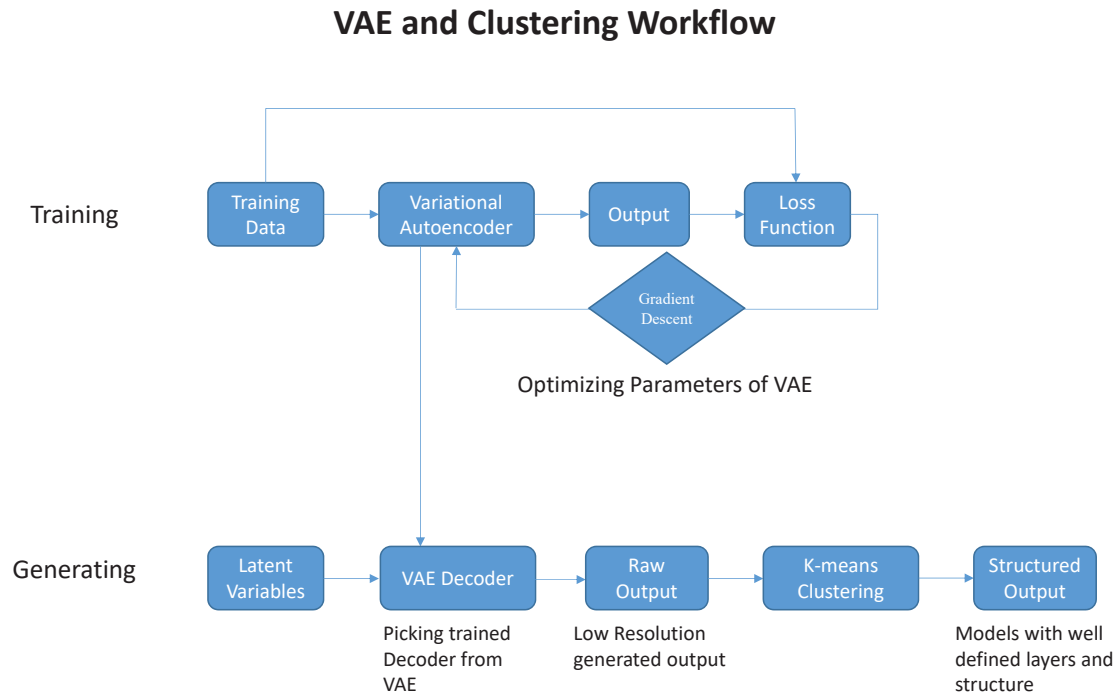


Figure 17.31: VAE+clustering workflow.

### 17.11.3 DBSCAN Applied to Earthquakes

Mapping earthquake epicenters and identifying their associated faults that break the surface is one of the fundamental tasks of earthquake monitoring stations throughout the world. Accurate assessment of a fault's geometry, gaps, growth rate, and properties can be used to update the hazard analysis of a region. The more accurate this assessment the more reliability in the prediction of hazard analysis<sup>12</sup>. One of the methods for delineating different faults from one another is the use of a clustering method applied to epicentral locations. For example, Setyorini et al. (2017) used K-means clustering to classify earthquakes along the coastline of Sumatra. They tested a different number of clusters and selected the one with the greatest variance change of the distance function. In their distance function they used both geographic distance between pairs of earthquakes and the difference in magnitudes as well.

Unfortunately, many earthquakes cluster along long sinuous faults, so they don't satisfy the ideal K-means conditions where points from different clusters should have nearly the same density and form round-like clusters of points. For example, many strike slip faults in California are located along linear fault scarps and have a low density of earthquakes around the non-fault areas and a high density along the fault scarps. In this case, DBSCAN is a more suitable clustering method.

To demonstrate this approach, Shi, Jiang, and Ning from Peking University used DBSCAN to cluster earthquakes around the Sichuan and Yunnan provinces in China. They first used DBSCAN clustering to separate epicenters associated with different fault zones, and then used DBSCAN again to cluster different faults in the same cluster. In the second DBSCAN test, they incorporated wave-form information into the distance metric so that earthquakes from faults with different orientations could be separated.

<sup>12</sup><https://earthquake.usgs.gov/>



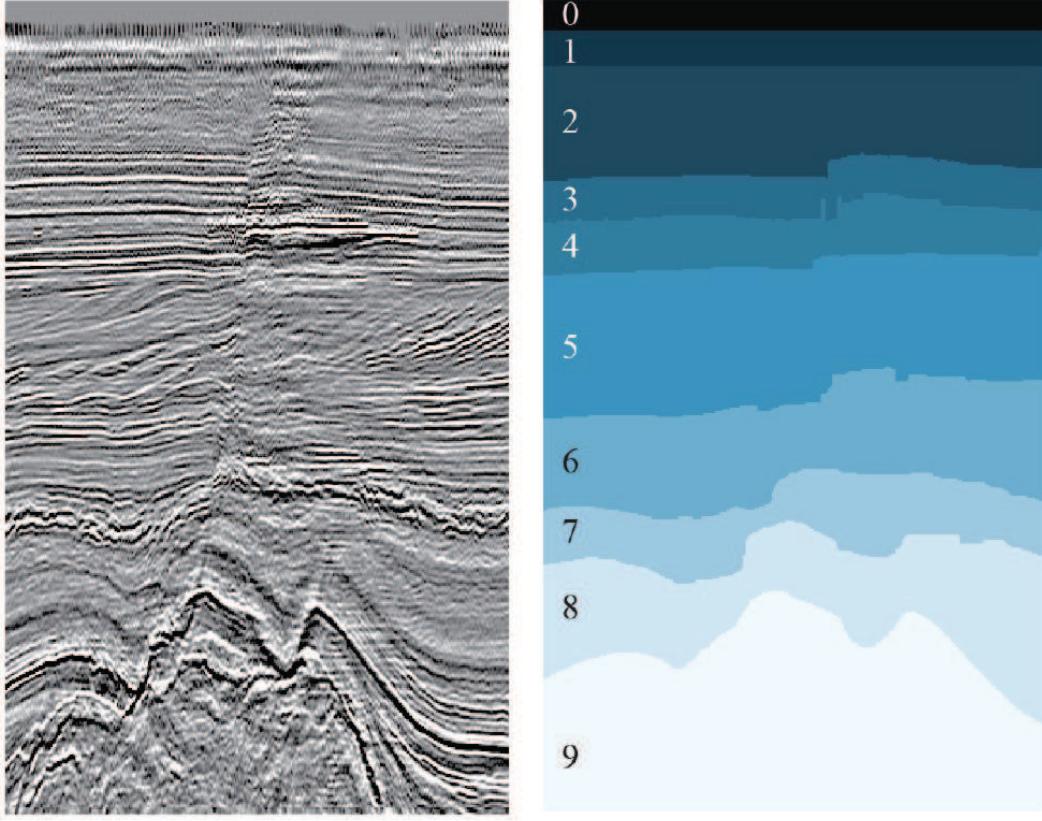


Figure 17.32: Examples of a (left) seismic section and (right) velocity model from the F3 data set (Silva et al., 2019).

The pseudo-code of the DBSCAN method is shown in Figure 17.35. Here, they used the geographic distance between the epicenters of two earthquakes for their distance metric:

$$d(Eq1, Eq2) = \sqrt{(x_1^{(1)} - x_1^{(2)})^2 - (x_2^{(1)} - x_2^{(2)})^2}, \quad (17.32)$$

where  $\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, x_3^{(i)})$  is the location of the  $i^{th}$  earthquake and  $d(Eq1, Eq2)$  is the geometrical distance between their locations.

The second metric also uses the cross-correlation information:

$$d(Eq1, Eq2) = \sqrt{(x_1^{(1)} - x_1^{(2)})^2 - (x_2^{(1)} - x_2^{(2)})^2 + \alpha^2 \frac{\sum_i [1 - Corr(R_i^{(1)}, R_i^{(2)})]^2}{N_s}}, \quad (17.33)$$

where  $\alpha$  is the weight balancing the physical distance and cross-correlation metrics.  $R_i^{(k)}$  is the trace of the  $k = 1, 2$  earthquake recorded at station  $i$ , and  $N_s$  is the number of stations which record both earthquakes. The normalized zero-lag correlation of these two traces is given by  $Corr(R_i^{(1)}, R_i^{(2)})$ .

For the following examples, the earthquake catalog contains 60,000 earthquakes which are located with the IASP91 earth model. The earthquake detection method is called Array-assisted Phase Picker.

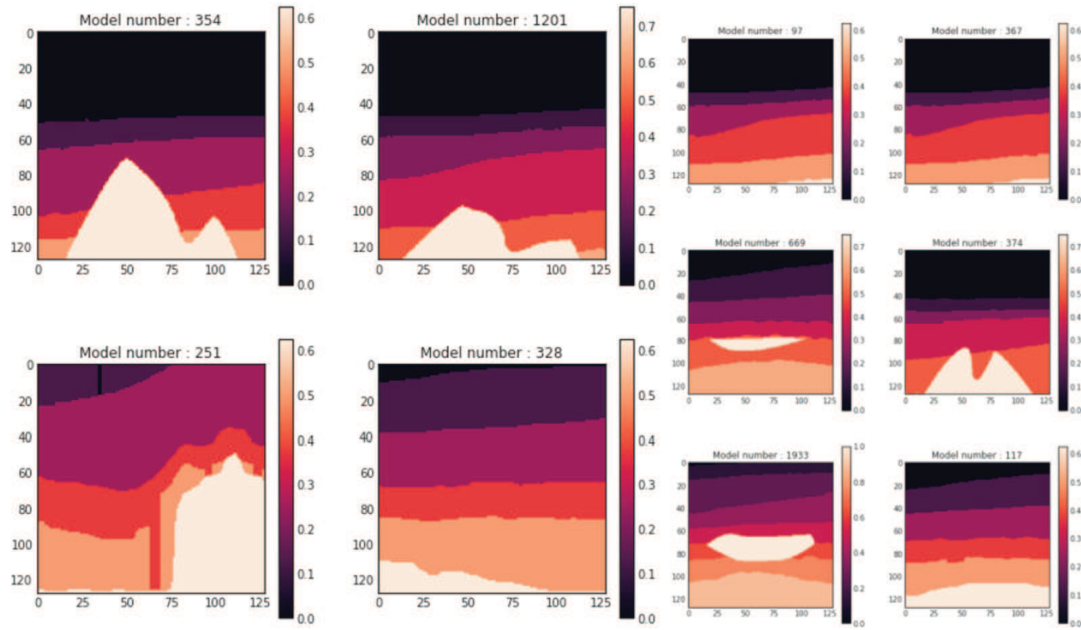


Figure 17.33: Some examples of the VAE training set. Figures from Tushar Gautam.

## VAE Output after Clustering

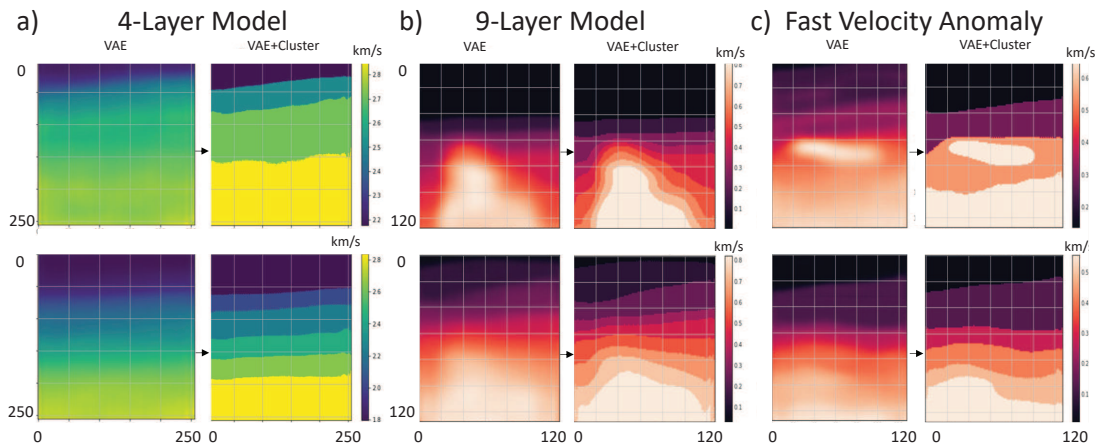


Figure 17.34: Some VAE and VAE+Clustering results. Figures courtesy of Tushar Gautam.

### DBSCAN 1

There are many possible faults in Figure 17.36, but the faults are obscured by too many points. Therefore, we need to apply another clustering of the points within the large cluster. In this case, trial and error testing suggests that the DBSCAN1 algorithm should use the values  $\epsilon = 10$  and

---

$P_i$	:Data points $P_i$
$\epsilon$	: Radius
minPts	: min neighbors
$D_{i,j}$	: Distance between $P_i$ and $P_j$
$L_i$	: cluster of $P_i$

---

```

1. for  $P_i$  in Database:
2.   if  $L_i$  not 0:                                # marked, go to next point
3.     continue
4.    $S = \{P_j | D_{i,j} < \epsilon\}$ ;  $N = \text{num}(S)$ 
5.   if  $N < \text{minPts}$ :                               # noise, go to next point
6.      $L_i = -1$ ; continue
7.    $k = k + 1$                                        # create a new cluster
8.   for  $P_j$  in  $S$ :                                   # check all points in set
9.      $S_j = \{P_l | D_{l,j} < \epsilon\}$ ;  $N_j = \text{num}(S_j)$ 
10.    if  $N_j < \text{minPts}$ :
11.       $L_j = k$ ; continue                         #  $P_j$  does not have enough neighbors, go to next point
12.    else:
13.       $L_j = k$ 
14.       $S = S \cup S_j$ ;  $N = N + N_j$                 # it has enough neighbors, add neighbors of this point into set

```

---

Figure 17.35: Pseudo-code for DBSCAN.

$\epsilon = 20$ .

## DBSCAN 2

A second DBSCAN method is applied to the points in the largest cluster in Figure 17.36 to give the results in Figures 17.37-17.39. Points within the green oval of Figure 17.37 are displayed in Figure 17.38. In this case, equation 17.33 is used to calculate the distance metric of DBSCAN. Here the following constraints are used:

$$\begin{aligned} \alpha &= 5 \text{ if } |\mathbf{x}^{(1)} - \mathbf{x}^{(2)}| < 15 \text{ km;} \\ \alpha &= 0 \text{ if } |\mathbf{x}^{(1)} - \mathbf{x}^{(2)}| \geq 15 \text{ km.} \end{aligned} \quad (17.34)$$

Although some clusters appear to contain related fault scarps such as the one in Figures 17.39, the biggest cluster does not completely isolate entire fault systems. A third round of clustering was performed but this did not work well.

## DBSCAN 2 with Crosscorrelation

In this cluster, there are only about 8,000 events where the computational cost is much lower than processing 20,000 events. So we add information of full waveform rather than only P- and S-wave traveltimes. We choose the weights as:

$$\begin{aligned} \alpha &= 3 \text{ if } |\mathbf{x}^{(1)} - \mathbf{x}^{(2)}| < 10 \text{ km;} \\ \alpha &= 0 \text{ if } |\mathbf{x}^{(1)} - \mathbf{x}^{(2)}| \geq 10 \text{ km} \end{aligned} \quad (17.35)$$

The DBSCAN parameters are set to be  $\epsilon = 5$  km and  $\text{MinPts} = 10$ , and the results are shown in Figure 17.40. The biggest cluster is shown in Figure 17.41. There is an earthquake gap in Figure 17.41b, where there are a few shallow events. Along  $AA'$  there are many fault branches near

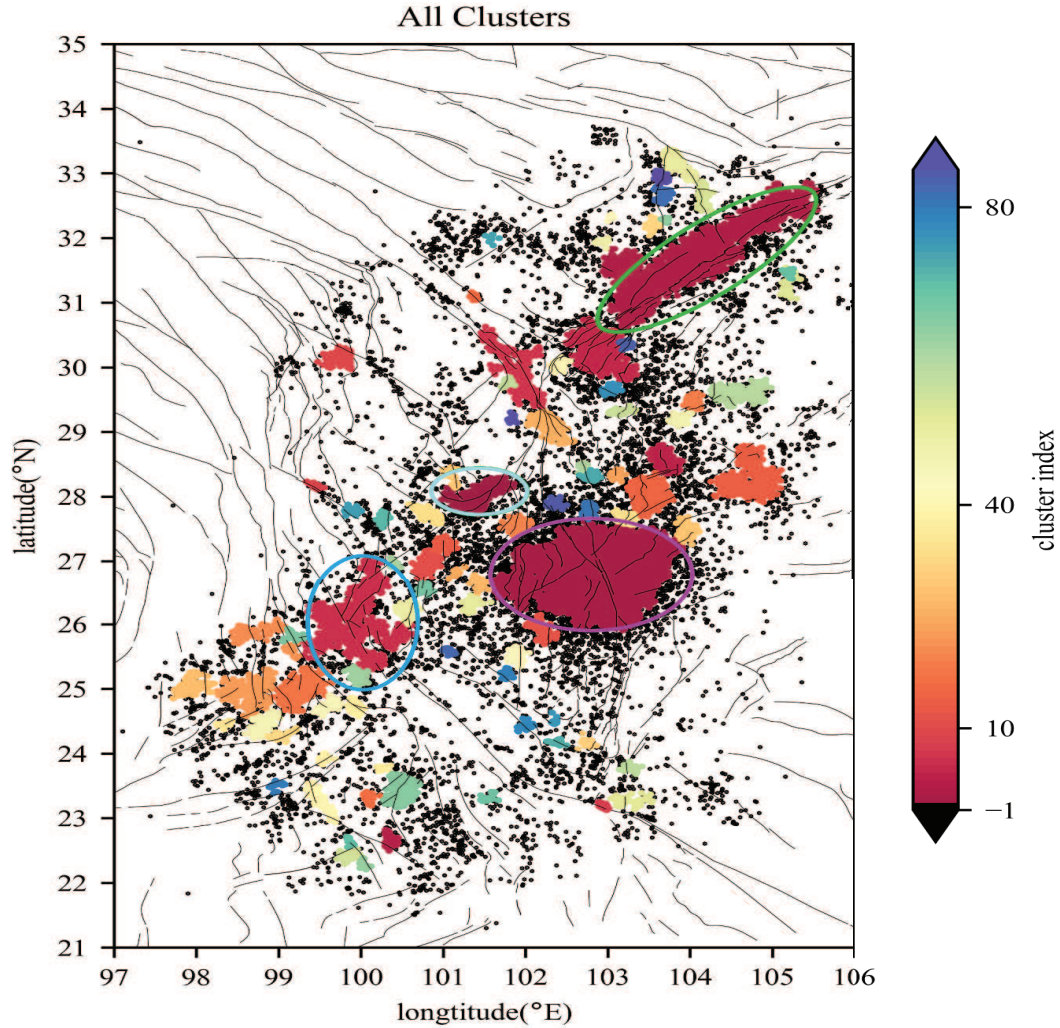


Figure 17.36: DBSCAN result after clustering 60000 earthquakes with 88 clusters, including a noise cluster. The purple oval is the first cluster which has 23575 earthquakes, the green oval contains 8856 earthquakes and the cyan oval contains 2231 earthquakes. The light blue oval could contain many faults, and the black lines are the observed faults.



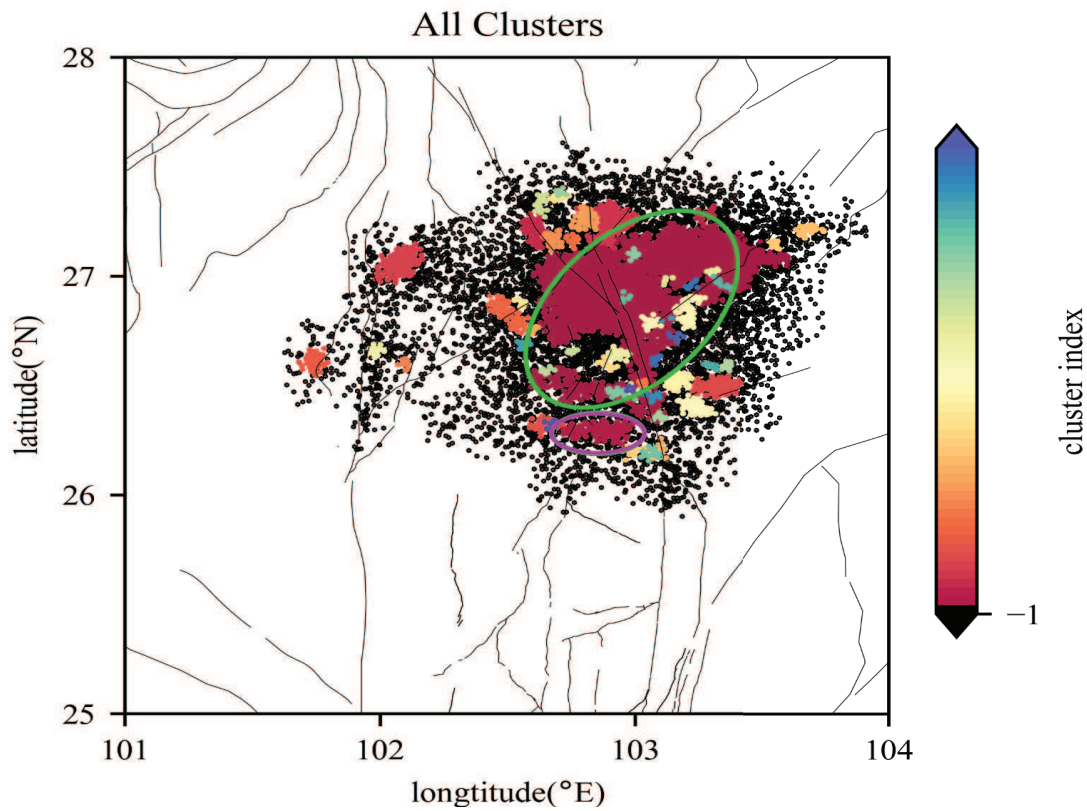


Figure 17.37: Second DBSCAN result after applying DBSCAN to the largest cluster in Figure 17.36. There are 55 clusters and the green oval is the biggest cluster that clusters 13236 events, and the purple oval indicates a possible fault. Here,  $\epsilon = 5$  km and  $MinPts = 10$ .

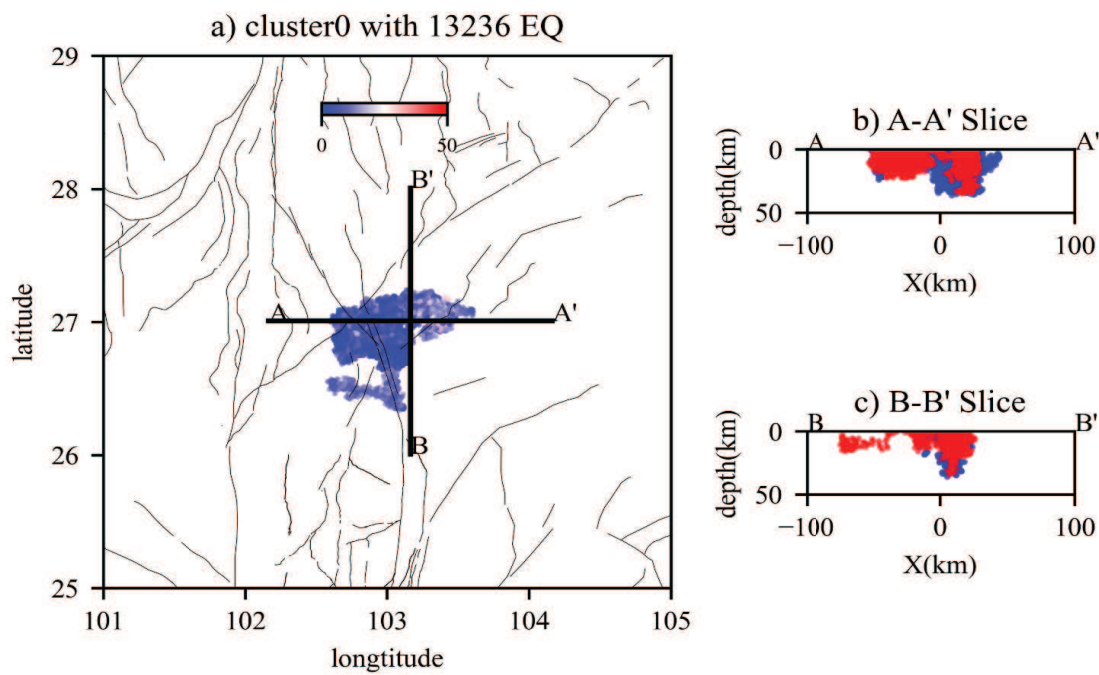


Figure 17.38: Points within the green oval of Figure 17.37. a) Distribution of all epicenters, where the colors indicate hypocentral depths between 10 km to 50 km; b) slice of line AA'. All events are displayed that are within 10 km from AA'. The red points are events from the North, and the blue points indicate from the South. c) slice of line BB'. All events are displayed that are within 10 km from BB', where the red points are events from the West, and the blue points are from the East.

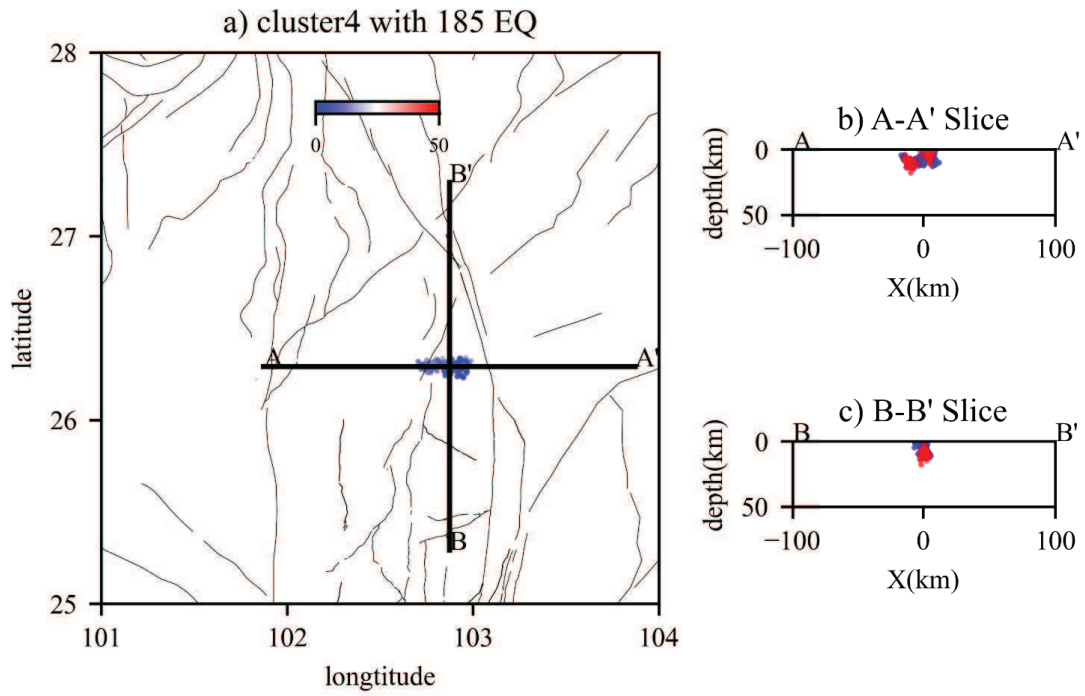


Figure 17.39: Epicenters within the small purple oval (just below the large green oval) in Figure 17.37.

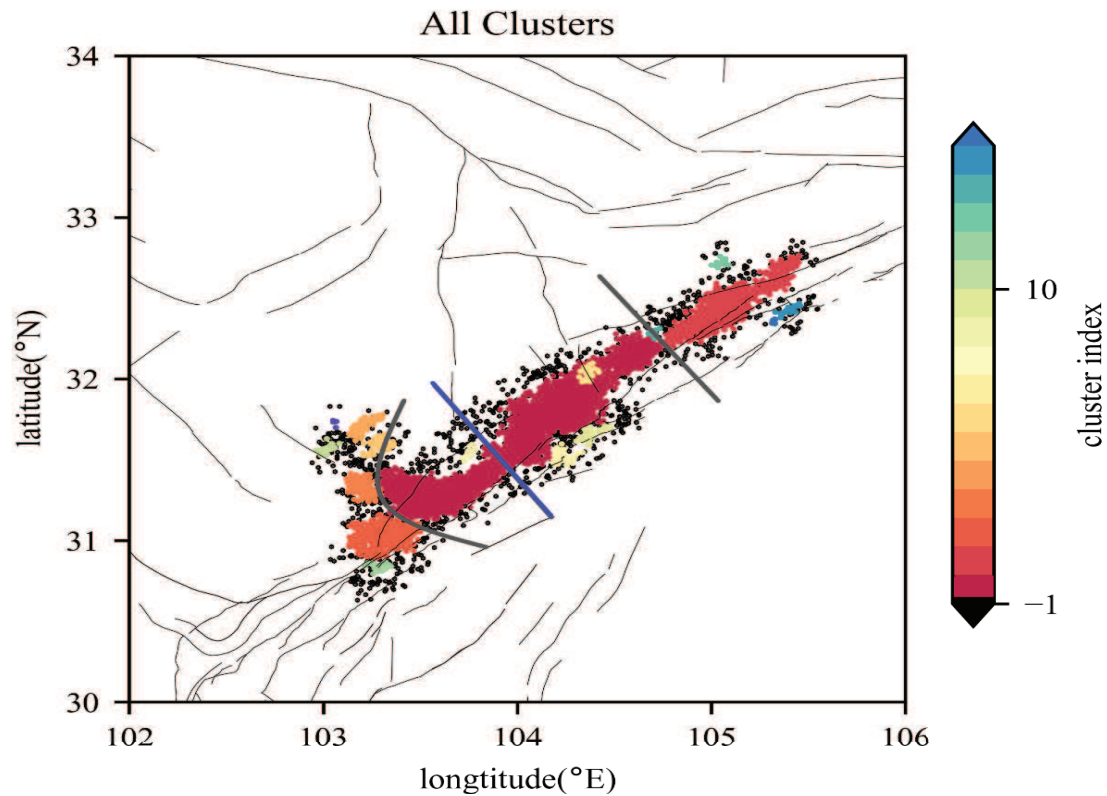


Figure 17.40: Second DBSCAN result of green oval in Figure 17.36, where there are 18 clusters. The colored thick lines are used to mark different regions of this fault zone by a human interpreter, which mostly agree with this result.

the Earth's surface, and only a few in the middle part. So most of the earthquakes are in the middle and are related to the main fault, which is deeper.

In summary, DBSCAN can be used to approximately delineate faults from epicentral locations in earthquake catalogs. It can be sequentially applied to first discover large clusters of many earthquake epicenters, then each cluster can be further divided into sub-clusters that can delineate the fault patterns. It is demonstrated that the waveform information can be used to assist in separating different faults from one another.

#### 17.11.4 Self-Organized Maps and Classification of Porosity in Rocks

Rock porosity is defined as the volume of pore space divided by the total volume in the rock. The pore space can be filled with a combination of gases and fluids, and so porosity is a valuable property that is used to assess the economic viability of gas and oil reservoirs. For carbonates, microporosity is the metric used to characterize their pore size.

Characterizing pore sizes in carbonates is not as straightforward as that in clastics. This is because carbonate pore geometries range from macro to sub-micron sizes (Lucia 2007, Hasiuk et al 2016, Vahrenkamp et al, 2014). Unfortunately, the lowest resolution limit of specific image-acquisition methods is used to characterize microcrystals in carbonates (Baechle et al, 2008; Kaczmarek et al, 2015). As an example, Choquette and Pray (1970) defined  $62\ \mu\text{m}$  pore diameter as the



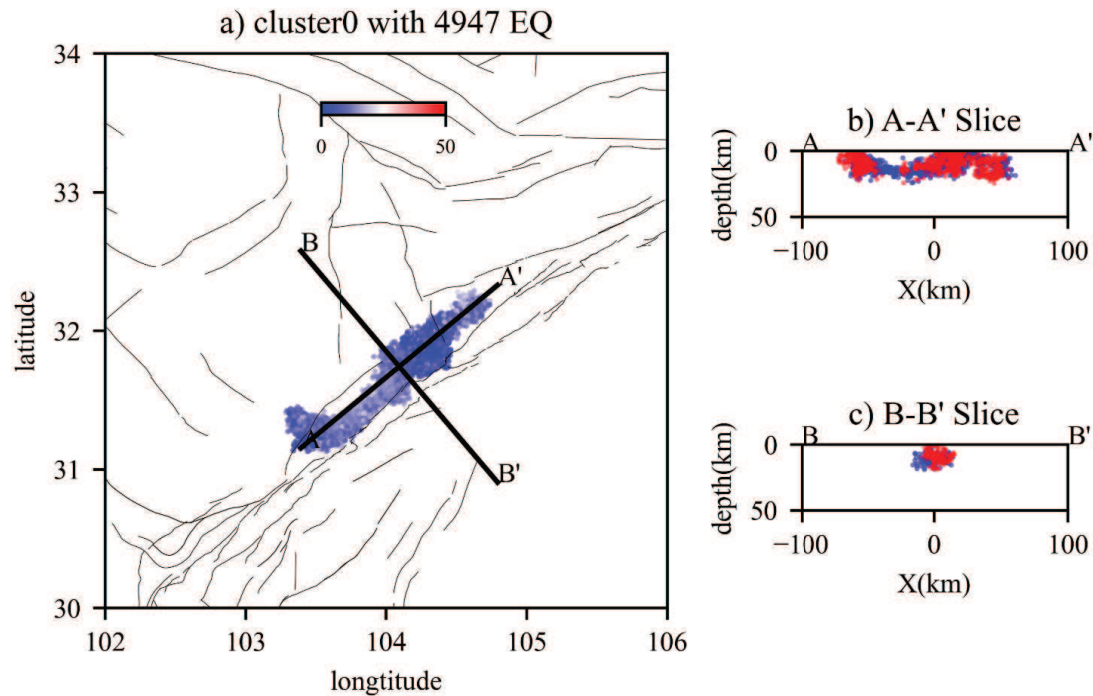


Figure 17.41: Points associated with the biggest cluster in Figure 17.40.

upper limit of measurable microporosity. In contrast, Baechle et al (2008) defined microporosity as rocks with a pore diameter of less than  $6 \mu m$  because of the limits of his imaging method. Others (Cantrell and Hagerty, 1999; Lonoy, 2006) defined microporosity as porosity with a pore diameter of less than  $10 \mu$ .

This non-uniform definition of microporosity can lead to incorrect characterization of carbonate reservoirs with microporosity (Ehrenberg et al, 2008; Baechle et al 2008; Farouk et al., 2014), even though they are typically characterized by low permeability and high-water saturation in wireline log measurements (Ehrenberg et al, 2008; Baechle et al 2008; Vanorio and Mavko, 2011; Deville de Periere et al, 2011). Therefore, a more uniform definition of microporosity and their relationships with petrophysical properties are important for an accurate description of the reservoir.

### Characterization of Microporosity by SOM

To provide a more objective measure of microporosity in carbonate rocks, we use the unsupervised method of self-organizing maps (SOMs) to extract the important microporosity parameters measured in borehole specimens. The SOM method is a type of artificial neural network method<sup>13</sup> that is trained with unsupervised learning to produce a low-dimensional representation of the high-dimensional input data. It was introduced by Kohonen (1982) to create low-dimensional views of high-dimensional input data. The low-dimensional representations can be used as input to unsupervised clustering methods (Ultsch, 2007) to classify data according to a sparse set of important parameters. The full description of the SOM is in section 17.10.

The SOM method is applied to data from the Upper Jubaila Formation outcrop in Riyadh,

<sup>13</sup>[https://en.wikipedia.org/wiki/Self-organizing\\_map](https://en.wikipedia.org/wiki/Self-organizing_map)

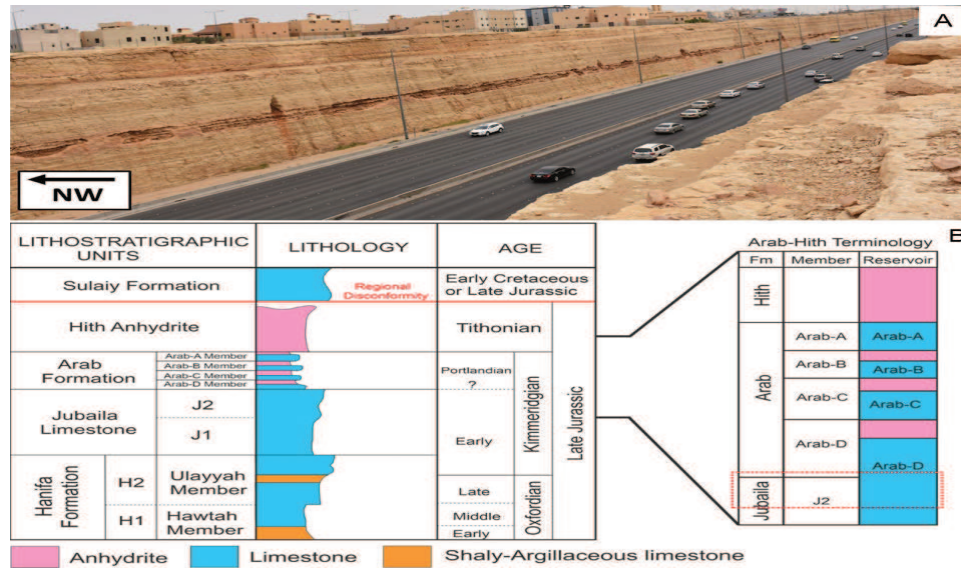


Figure 17.42: a) Roadcut in Riyadh, Saudi Arabia, which exposes an approximately 26 m thick part of the Upper Jubaila Formation as an analogue to the lower part of the Arab-D reservoir (Jacquemyn et. al., 2018). b) Generalized lithostratigraphy and chronostratigraphy of the Late Jurassic sedimentary deposits in Saudi Arabia (left). The red dashed box represents the studied interval in the Upper Jubaila Formation, (right) which is equivalent to the lower part of the Arab-D reservoir. Figures modified from Al-Mojel et al. (2020) and Jacquemyn et al. (2018).

Saudi Arabia (Figure 17.42). This outcrop is an analogue for the lower part of the petroliferous Late Jurassic Arab-D reservoir (Jacquemyn et al, 2018). Based on the result of mercury injection capillary pressure data (Figure 17.43), this outcrop is dominated by porosity with pore diameter distribution of less than  $10 \mu m$ . Cantrel and Hagerty (1999) defined  $10 \mu m$  as the upper limit of microporosity in their subsurface microporosity study of the Arab Formations. Thus, this outcrop pore system is dominated by microporosity and a suitable analogue to investigate the correlation between microporosity with petrophysical properties.

The aim is to establish a SOM methodology that links micrometer-scale morphology of microcrystals in carbonates with macro-scale petrophysical properties in the well. These results can be used for upscaling microporosity to the reservoir grid scale (Ramdani, 2022).

## SOM Results

The main dataset for this study consist of 106 cylindrical core plugs from a 35 m-long core that was drilled behind the outcrop cliff (Figure 17.44). Petrographic observations for lithofacies identification were performed on the thin sections made from the trim end of the core plugs, and used the modified Dunham classification (Embry and Klovan, 1971) to describe depositional textures of the investigated thin sections. Micrometer-scale petrographic observations were performed using Scanning Electron Microscopy (SEM) on 37 selected core plug samples. The SEM imaging strategy in Deville de Periere et al (2011) and the microtexture classification of Kaczmarek et al (2015) are used to characterize the microcrystals that host microporosity. Two quantitative parameters modified from microtexture classification of Kaczmarek et al (2015) and Deville De Periere et al

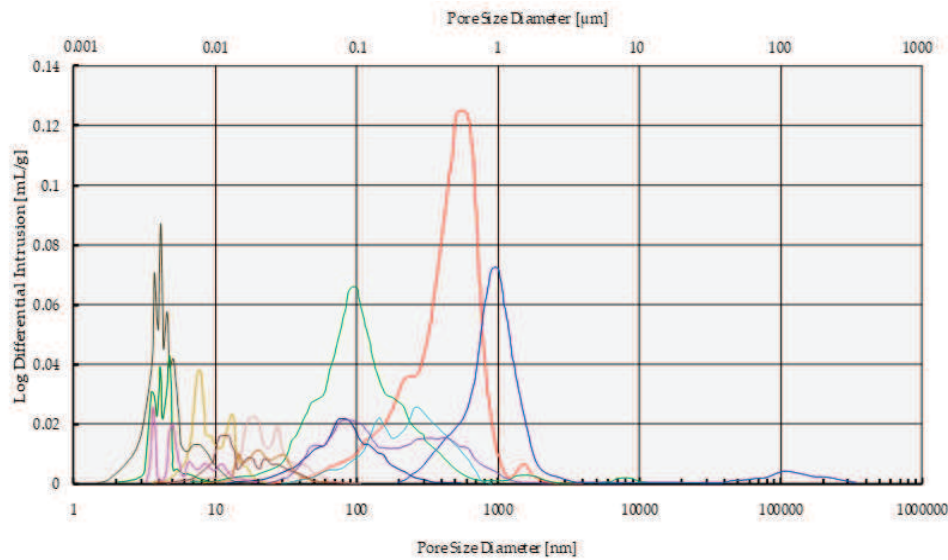


Figure 17.43: Mercury injection capillary pressure data from the studied outcrop shows that this outcrop is dominated by porosity with a pore diameter distribution of less than  $10\ \mu\text{m}$  (Finkbeiner et. al., 2018).

(2011) were also determined: *roundness index* and *fuse index*. The *roundness index* determines how close each individual microcrystal is to a perfect circle. This parameter uses the aspect ratio-corrected circularity method by Takashimizu and Iiyoshi (2016) to approximate the Krumbein-Sloss roundness chart (Krumbein and Sloss, 1963). An angular-shaped microcrystal will be assigned a *roundness index* of close to 0 and a spherical one will have a *roundness index* value of 1. The fraction of granular microcrystals in the investigated SEM image is defined as *fuse index*. A fuse index value equal to 1 means all microcrystals are granular while 0 says that all microcrystals are indistinguishable and fused together.

Spectral gamma ray measurements were also performed to measure the percent of K (Potassium), ppm of U (Uranium) and ppm of Th (Thorium) of the whole intact core. Total gamma ray counts in API units were calculated using Luthi's (2001) empirical formula. Helium porosimetry and gas permeametry were used to measure grain density, bulk density, porosity and permeability of the core plugs. Powder X-ray diffraction was performed to obtain the mineral composition of the plugs, from which four main mineral constituents were defined: calcite, dolomite, clay minerals (kaolinite and montmorillonite), and quartz. Ultrasonic P-wave velocity measurements (300 kHz) under non-saturated (dry) and unconfined condition were also performed to the core.

The self-organizing map (SOM) is used to perform unsupervised classification by reducing the ten-dimensional space of petrophysical properties to a smaller dimension. The ten petrophysical properties of each sample in the well are the following: porosity, permeability, gamma ray, bulk density, grain density, P-wave velocity, calcite fraction, dolomite fraction, quartz fraction, and clay minerals fraction. Four discrete SOM classes are identified from this process. Since there was no explicit external supervision nor a priori information imposed to run the SOM, the four classes identified in this process were based on the natural grouping tendencies of the input dataset.

The result of thin section analysis, SEM imaging, lab-scale petrophysical measurements and SOM classes from the well is displayed in Figure 17.45. The SOM classes are shown to be superior compared to the classical depositional-texture based rock typing for petrophysical properties characterization. This claim is validated in the crossplot in Figure 17.46 which shows the weakness



Figure 17.44: Drilling rig next to the carbonate outcrop (Ramdani, 2022). The core penetrates 35 m into the Upper Jubaila Formation in this outcrop.

of the standard classification method. Here, the median values of *roundness index* with *fuse index* color-coded by depositional texture are plotted. A linear correlation coefficient of 0.87 is observed between the *roundness index* and *fuse index*, and implies a positive relationship between sphericity and granularity of microcrystals. Angular microcrystals tend to be more fused/fitted while rounded microcrystals tend to be more granular. However, no apparent arrangement can be seen between microtexture and depositional texture. All depositional textures are dispersed almost equally in granularity-angularity space which implies that both sphericity and granularity of microcrystals are independent of depositional texture.

Figure 17.47a shows a crossplot between the *roundness index* and the *fuse index* color-coded by the SOM classes. In contrast to depositional texture, the SOM clusters the input data into four classes and these four classes have a strong correlation with microtexture. Class 1 is located at the highest *roundness index* and *fuse index* region which corresponds to median *roundness index* of  $> 0.6$  and a *fuse index* of  $> 0.8$ . Class 3 is located at the lowest value of *roundness index* and *fuse index* space which corresponds to a median *roundness index* value  $< 0.2$  and *fuse index*  $< 0.4$ . The majority of the data falls in Class 2 and Class 4 with some minor overlap between them. Class 2 corresponds to a median *roundness index* of  $0.6 - 0.4$  and a *fuse index* of  $0.8 - 0.4$  while Class 4 corresponds to a *roundness index* of  $0.5 - 0.2$  and a *fuse index* of  $0.6 - 0.4$ . The SEM images of microcrystals of these SOM classes are shown in Figure 17.47b. The SOM classes 1 and 2 tend to be more rounded and granular, class 3 tends to be more subhedral to anhedral, and class 3 tends to be more anhedral and tight-fitted. These results suggest that the four SOM-derived classes can be utilized as a proxy for texture of microcrystals.

Figure 17.48 shows the porosity-permeability crossplots color-coded by SOM class a) and depositional facies b). Porosity-permeability crossplot in this study is relatively predictable with relatively straight-forward log-linear relationship. Porosity-permeability also shows a better arrangement when sorted by SOM classes as opposed to depositional texture. Class 1 is located at the highest porosity and permeability space (porosity  $12\%$  and permeability  $> 1mD$ ). Class 2 occupies the upper half of porosity and permeability space (porosity  $8\% - 12\%$  and permeability  $0.1 mD - 1 mD$ ). Classes 3



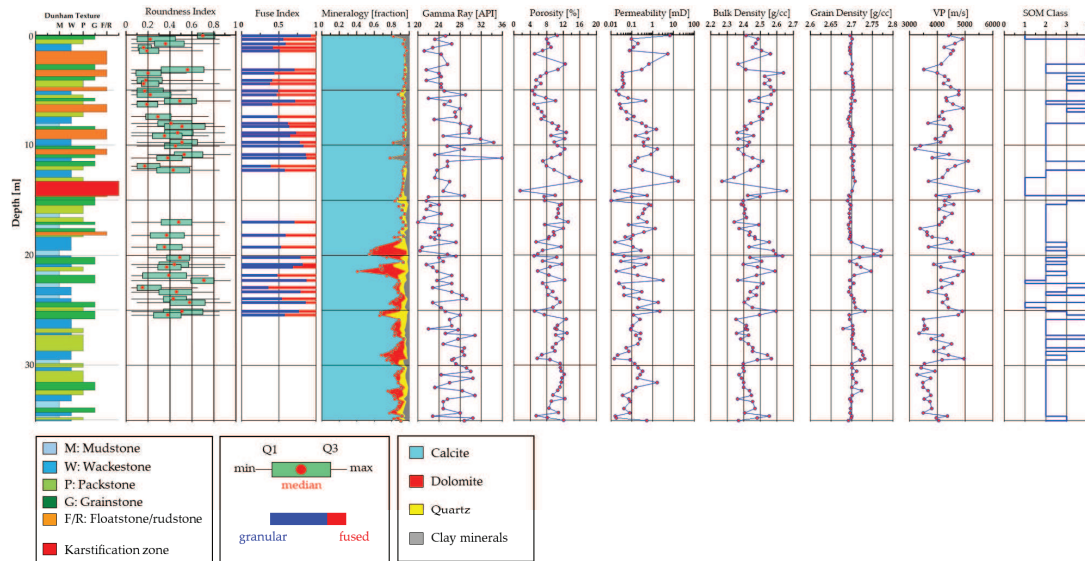


Figure 17.45: Compiled results of depositional texture, microtexture parameters, lab-scale petrophysics measurements, and Self-Organizing Map (SOM) clusters of the well (Ramdani, 2022).

and 4 occupy the lower part of porosity and permeability space. Class 3 corresponds to the porosity  $< 8\%$  and permeability  $< 0.05$  mD while Class 4 corresponds to the porosity of  $< 10\%$  and permeability  $< 0.2$  mD. In contrast to SOM class, this data clustering characteristics cannot be seen if we sort porosity-permeability to depositional texture. Most of the depositional texture are dispersed in all range of porosity and permeability. Porosity and permeability have a better correlation with microtexture and SOM class rather than depositional texture.

Sorting granularity and sphericity into depositional facies resulted with little to no apparent control of initial depositional texture to the texture of microcrystals. The sphericity of microcrystals is associated with granularity but almost independent of depositional texture. On the other hand, the SOM clustered all lab-measured properties into four discrete classes. The arrangement of these four classes has a strong association with sphericity and granularity, even though these four classes were generated purely from lab-measured petrophysical properties without any external supervision or forced input.

Based on the SOM classes, we interpret that granularity and sphericity are the principal components and/or the dominant controlling variables over all lab-measured properties in this well. This suggests that machine-learning based unsupervised classification might be used as a proxy for microtexture in the subsurface microporous reservoirs.

The influence of granularity and sphericity to the reservoir quality is apparent and can be seen clearly in Figure 17.48 where the SOM classes (as a proxy to microtexture) cluster the data better than depositional texture. A simple log-linear relationship between porosity and permeability follows the trend of microcrystals granularity and sphericity. Moreover, this log-linear relationship by-passes multiple depositional texture in a way so that it is possible to represent multiple depositional facies with a single continuous log-linear porosity-permeability relationship. SOM classes 4 and 3 represent tight (lower porosity-permeability) microcrystals while Class 1 and Class 2 represents porous (higher porosity-permeability) microcrystals. Discounting two obvious outlier points, the majority of the dataset belongs to Class 2 and Class 4 where the bulk of log-linear relationship between porosity and permeability is formed. Class 1, which represents the roundest and contains the most granular

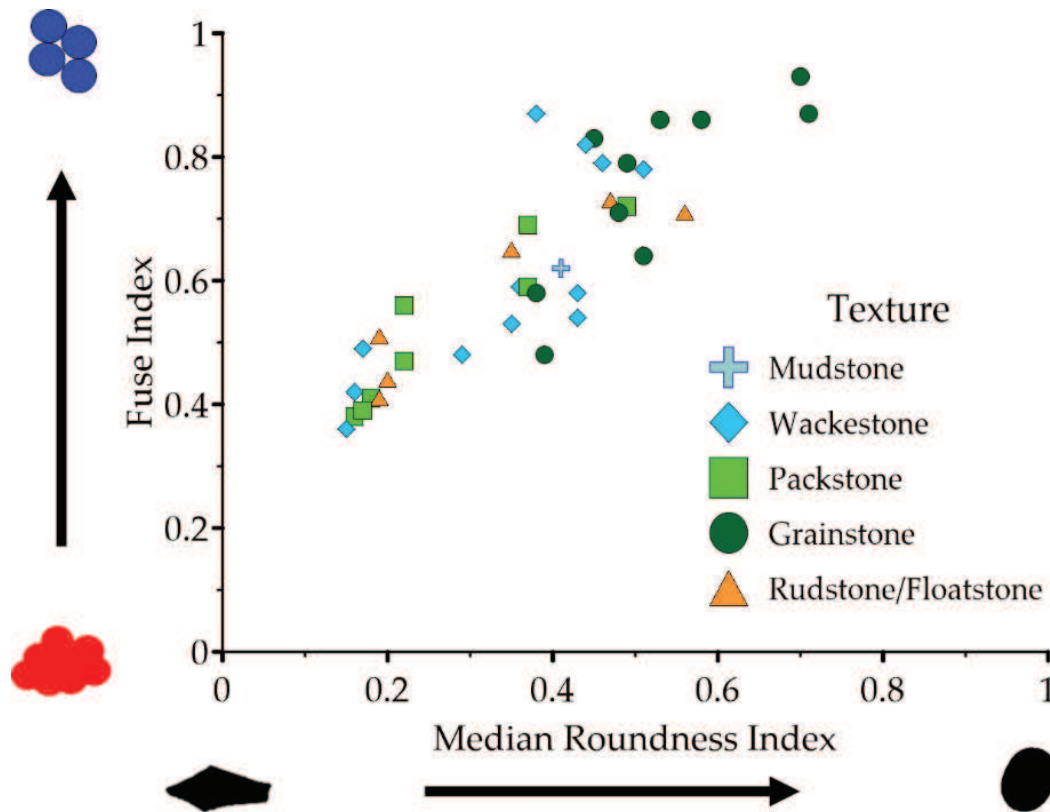


Figure 17.46: Crossplot between median values of *roundness index* with *fuse index* color-coded by depositional texture from samples with SEM images. The symbols of any color are largely distributed all along the line instead of clumping around a small portion of the line (Ramdani, 2022).

microcrystals, is noticeably located at the highest porosity and permeability space. The porosity range of Class 1 is relatively similar with the upper member of Class 2 but with significantly higher permeability. This makes Class 1 a higher permeability continuation of Class 2. Class 3, which represents the most angular and contains the most tight/fused microcrystals, resides in the lower permeability space of Class 4. They have a similar porosity range with mid-to-lower membership in Class 4 but with a lower permeability value. These results demonstrate the applicability of using SOMs as a method for linking multi-dimensional petrophysical properties with the texture of microcrystals.

### 17.11.5 Self-Organized Maps and Classification of Seismic Horizons

Another SOM example is that by de Matos et al. (2010) where they used both 1D and 2D SOM procedures to detect geological patterns in horizon slices of seismic data. The horizon slice is that of instantaneous amplitudes taken from a series of 2D seismic sections, one of which is shown as the inset in the upper right of Figure 17.49. This horizon, marked by the black arrow in Figure 17.49, is over a subsurface reservoir. They used a SOM to compute a large number of SOM classes and mapped each class to an HSV color model. Displaying these colors on a map allowed them to

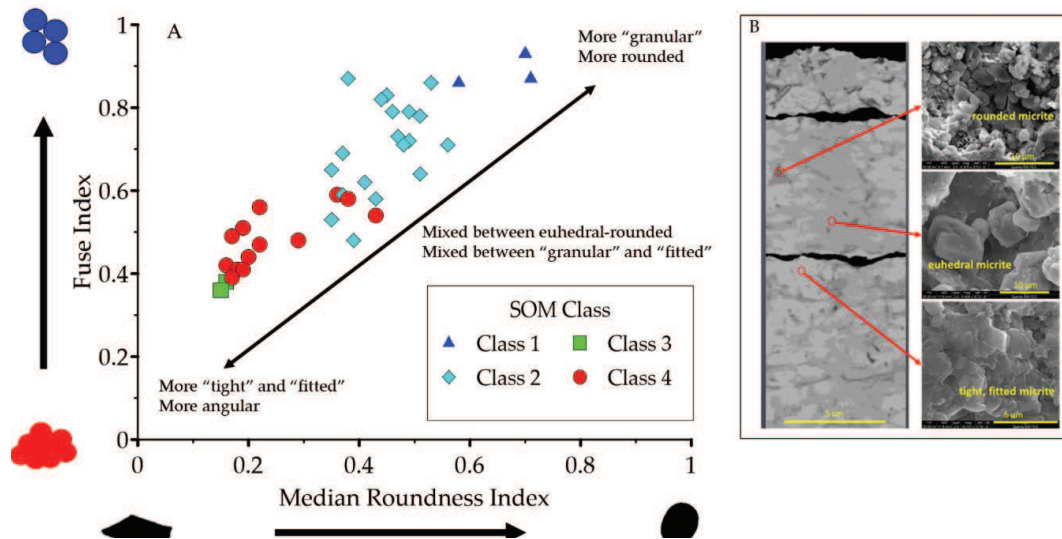


Figure 17.47: a) Crossplot between median value of *roundness index* with *fuse index* color-coded with the Self-Organizing Map (SOM) class from samples with SEM images. The SOM classes show a data grouping that follows microtexture. b) Medical CT and SEM photomicrograph images from Finkbeiner et al (2018) from the same core data used in this study. The three SEM photomicrographs show the tendencies of rounded microcrystals to be more granular and angular microcrystals to be tighter/fused (Ramdani, 2022).

conclude "...that the channel is clearly delineated and the relationship among waveforms in the 2D SOM colorbar helps to interpret the geology."

### 17.11.6 Self-Organized Maps and Fault Detection

Another application of SOM is for fault detection in seismic records. Hussein et al. (2020) identified 8 combinations of 16 seismic attributes for useful fracture and fault detection. However, this large number of attributes and their combinations are impractical for fault detection by an interpreter. To remedy this problem, Hussein et al. (2020) used SOM and PCA analysis to reduce the fault features to one classification volume. They showed that the SOM volume provided efficient and noticeably more accurate detection of faults compared to conventional interpretation techniques.

## 17.12 Summary

Unsupervised cluster methods group, or cluster, objects according to measured or perceived intrinsic characteristics or similarity among the input points. There are over a thousand such methods, but K-means is one of the most popular for simple point distributions. More sophisticated clustering methods such as SVC or DBSCAN can be used if the point distributions are too irregular or are bounded by non-convex boundaries.

The advantage of unsupervised clustering is there is no manual labeling of the data prior to clustering. As demonstrated by the semblance examples, this approach can avoid manual labor in estimating the initial velocity model from semblance panels. For semblance analysis, any final choice of stacking velocity curves should be vetted by a skilled interpreter.

If the data have strong coherent noise then the clustering algorithm can erroneously pick the wrong velocity models. In this case constraints can be imposed to avoid the characteristic low velocities of free-surface multiples. Another possibility is a semi-supervised clustering algorithm because a few labeled data serve as groundtruth anchors that can steer the algorithm away from incorrect cluster memberships. The IWC algorithm can weight each cluster point with a weight that is proportional to the information content of that point. This is a type of Fuzzy clustering.

The fuzzy clustering method for traveltimes picking has the following advantages over other methods in that it is robust by exploiting the different characteristics of signal and noise as exposed by the trace features, is flexible, and is straightforward to implement. The method only requires an estimate of the signal's dominant period to compute the trace features and a membership threshold to separate signal from noise. Moreover, the user can define different features depending on the target signal.

A drawback of fuzzy clustering is that it requires an analysis window it partitions the trace into signal and noise intervals. Further work is required to determine the type of detected signal (e.g., P or S waves). An extra burden is that a picking method is required to determine the onset times of events, since fuzzy clustering only defines a suitable window for picking.

Self-organizing maps are another approach to clustering, except they present the clustered points in a lower-dimensional space, typically two or three dimensions. This makes it much easier to visualize the separability of the cluster geometries compared to examining many 2D slices of the clusters for points in high-dimensional spaces.

A different geoscience problem might require a clustering algorithm tuned to that problem. In that case, one of the many other variations of cluster algorithms might provide the algorithm most suited for solving your problem.

## 17.13 Exercises

1. Download the semblance NMO and clustering program<sup>14</sup>. Now run "K-means.m" for different numbers of clusters and find the best number of clusters that give the most accurate semblance curve.
2. Devise a regularization procedure that uses the semblance curve picked from the previous CMG to constrain the semblance curve for the new CMG. Show the results.
3. Devise other features in addition to amplitude, time and  $V_{NMO}$  that might increase the accuracy of the picked semblance curve. Implement them in the MATLAB code and show results.
4. Devise the MATLAB code for fuzzy clustering and compare results using softmax clustering. Show results.
5. Download the DBSCAN code (Ester et al., 1996) in Python<sup>15</sup> and compare its performance against the K-Means clustering code for clustering Yellowstone data.
6. Execute the SOM MATLAB code in section 17.10 after loading the iris flower data using the command `load iris_dataset`. Use different numbers of clusters and discuss the meaning of the resulting plots. What is the minimum number of clusters in this data set?
7. Why are there only three red circles in Figure 17.22d when there are 4 neurons assigned in Figure 17.22a? What happened to the missing neuron? Try using the MATLAB code with only three neurons for the starting number of neurons to come up with an answer.

<sup>14</sup>LAB1/Chapter.Book.Cluster/Chapter.Cluster1/lab.html

<sup>15</sup><https://www.kaggle.com/fazilbtopal/popular-unsupervised-clustering-algorithms>



## 17.14 Computational Labs

1. Download the Python code in Box 17.3.1 and insert the Yellowstone data as the input. Compare its performance against that of K-means clustering.
2. Download the semblance NMO and clustering program<sup>16</sup> and execute the MATLAB code for finding the semblance curves. Follow the instructions.
3. Same as previous exercise except adjust the clustering procedure to incorporate a robust regularization constraint that is updated with each new CMG. The adaptive constraint is the priori semblance curve that is the average of a select number of previously computed curves.
4. Same as the first computational exercise except incorporate new features in finding the best semblance curve. Use some relevant textures<sup>17</sup> as new features.
5. Break DBSCAN. Go to the Colab for DBSCAN [https://colab.research.google.com/drive/1L\\_OLxY73G1T6GsZkIu-SxMowPsQ6zTzt?usp=sharing#scrollTo=mb1E2iAIQb4l](https://colab.research.google.com/drive/1L_OLxY73G1T6GsZkIu-SxMowPsQ6zTzt?usp=sharing#scrollTo=mb1E2iAIQb4l) and change the point distribution so that DBSCAN fails. Validate the claim that suggests DBSCAN does not work well when clusters have very different point densities.
6. Arrival time picking by Fuzzy Clustering. Go to the site <https://github.com/evcano/fcm-aic-arrival-picker> and test the sensitivity of the Fuzzy Clustering picker by using different values of hyperparameters such as window length, dominant period and the coefficient  $m$  that controls the degree of fuzziness.

## 17.15 Appendix: Derivation of the Fuzzy Clustering Weight

The formula is now derived for the weight given in equation 17.15. The starting point is the objective function in equation 17.14, except the constraint in equation 17.16 is accounted for by a Lagrange multiplier term

$$\epsilon = \frac{1}{2} \sum_{k=1}^K \sum_{j=1}^N \mu_{kj}^m \|\mathbf{x}_j - \mathbf{C}_k\|^2 - \sum_{j=1}^N \lambda_j \left(1 - \sum_{k=1}^K \mu_{kj}\right), \quad (17.36)$$

where  $\lambda_j$  is the Lagrange multiplier for the  $j^{th}$  point.

Setting the gradient  $\nabla_{\mathbf{C}_k} \epsilon$  of equation 17.36 to zero gives

$$\nabla_{\mathbf{C}_k} \epsilon = \sum_{j=1}^N \mu_{kj}^m (\mathbf{x}_j - \mathbf{C}_k) = 0. \quad (17.37)$$

Solving for  $\mathbf{C}_k$  provides the formula for the weighted mean

$$\mathbf{C}_k = \frac{1}{\sum_{j=1}^N \mu_{kj}^m} \sum_{j=1}^N \mu_{kj}^m \mathbf{x}_j. \quad (17.38)$$

Setting the gradient  $\nabla_{\lambda_j} \epsilon$  of equation 17.36 to zero yields

$$\nabla_{\lambda_j} \epsilon = 1 - \sum_{k=1}^K \mu_{kj} = 0, \quad (17.39)$$

<sup>16</sup>LAB1/Chapter.Book.Cluster/Chapter.Cluster1/lab.html

<sup>17</sup><https://arxiv.org/ftp/arxiv/papers/1904/1904.06554.pdf>

or rearranging gives the constraint

$$\sum_{k=1}^K \mu_{kj} = 1. \quad (17.40)$$

Finally, the weights can be determined by setting the gradient  $\nabla_{\mu_{kj}} \epsilon$  of equation 17.36 to zero to get

$$\nabla_{\mu_{kj}} \epsilon = m \mu_{kj}^{m-1} \|\mathbf{x}_j - \mathbf{C}_k\|^2 + \lambda_j = 0. \quad (17.41)$$

or by rearranging we get

$$\mu_{kj}^{m-1} = -\frac{\lambda_j}{m} \frac{1}{\|\mathbf{x}_j - \mathbf{C}_k\|^2}. \quad (17.42)$$

Taking each side to the  $1/(m-1)$  power gives

$$\mu_{kj} = -\frac{\lambda_j^{1/(m-1)}}{m^{1/(m-1)}} \frac{1}{(\|\mathbf{x}_j - \mathbf{C}_k\|^2)^{1/(m-1)}}. \quad (17.43)$$

Summing both sides over the index  $k$  and using the constraint in equation 17.40 gives

$$\begin{aligned} \sum_{k=1}^K \mu_{kj} &= -\frac{\lambda_j^{1/(m-1)}}{m^{1/(m-1)}} \sum_{k=1}^K \frac{1}{(\|\mathbf{x}_j - \mathbf{C}_k\|^2)^{1/(m-1)}}, \\ &= 1. \end{aligned} \quad (17.44)$$

Defining

$$\alpha_{kj} = \frac{1}{(\|\mathbf{x}_j - \mathbf{C}_k\|^2)^{1/(m-1)}}, \quad (17.45)$$

where the dependency of  $\alpha_{kj}$  on  $m$  is silent, and solving for  $\lambda_j^{1/(m-1)}$  gives

$$\lambda_j^{1/(m-1)} = -\frac{m^{1/(m-1)}}{\sum_{k=1}^K \alpha_{kj}}. \quad (17.46)$$

Inserting equation 17.46 into equation 17.43 gives

$$\begin{aligned} \mu_{kj} &= \frac{1}{(\|\mathbf{x}_j - \mathbf{C}_k\|^2)^{1/(m-1)}} \frac{m^{1/(m-1)}}{m^{1/(m-1)} \sum_{k'=1}^K \alpha_{k'j}}, \\ &= \frac{1}{(\|\mathbf{x}_j - \mathbf{C}_k\|^2)^{1/(m-1)} \underbrace{\sum_{k'=1}^K \alpha_{k'j}}_{\text{eqn. 17.45}}}, \\ &= \frac{(\|\mathbf{x}_j - \mathbf{C}_k\|^{-2})^{1/(m-1)}}{\sum_{k'=1}^K (\|\mathbf{x}_j - \mathbf{C}_{k'}\|^{-2})^{1/(m-1)}}. \end{aligned} \quad (17.47)$$

Setting  $m = 2$  gives equation 17.15.

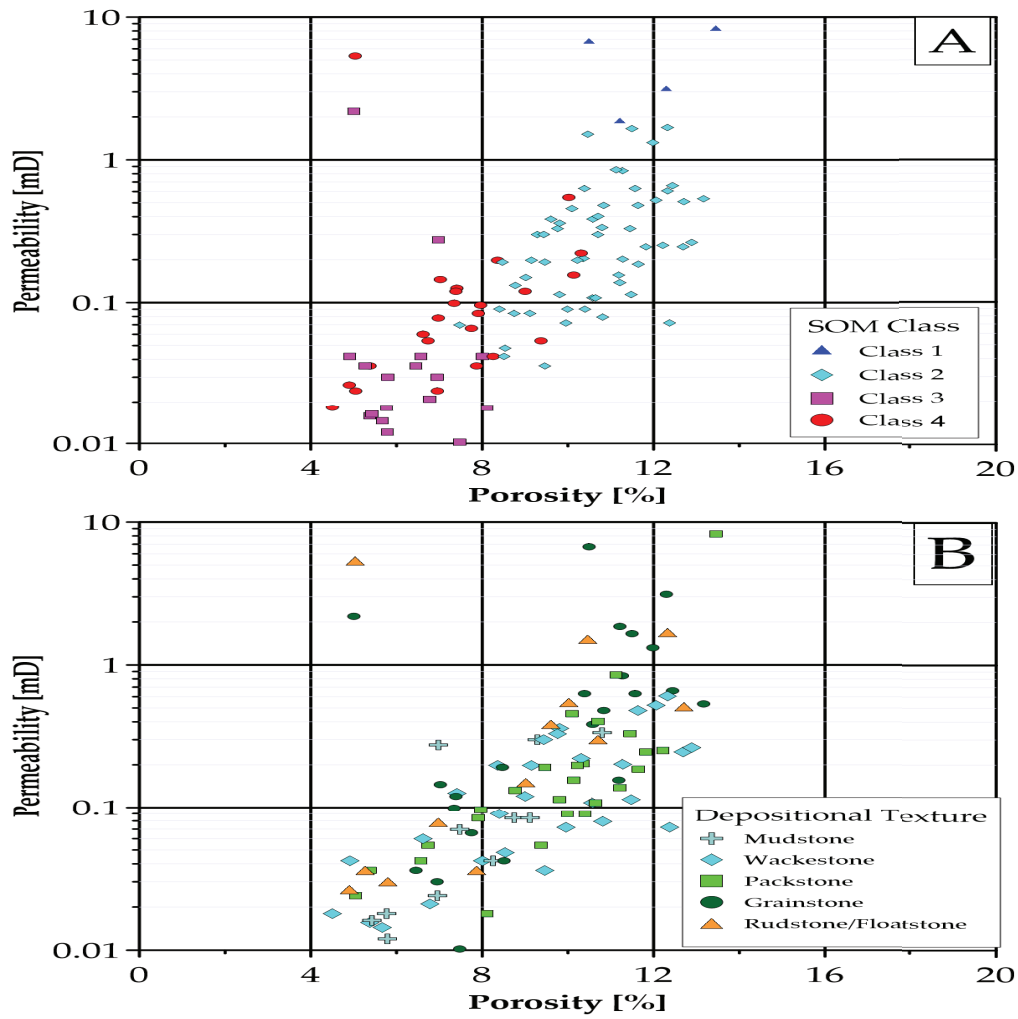


Figure 17.48: Porosity-permeability crossplots color-coded by SOM class a) as a proxy for microtexture and b) depositional facies. Porosity-permeability is better clustered if the dataset is sorted with SOM classes rather than depositional facies (Ramdani, 2022).

## Horizon Slice

### 2D Seismic Section

Horizon →

256 Classes

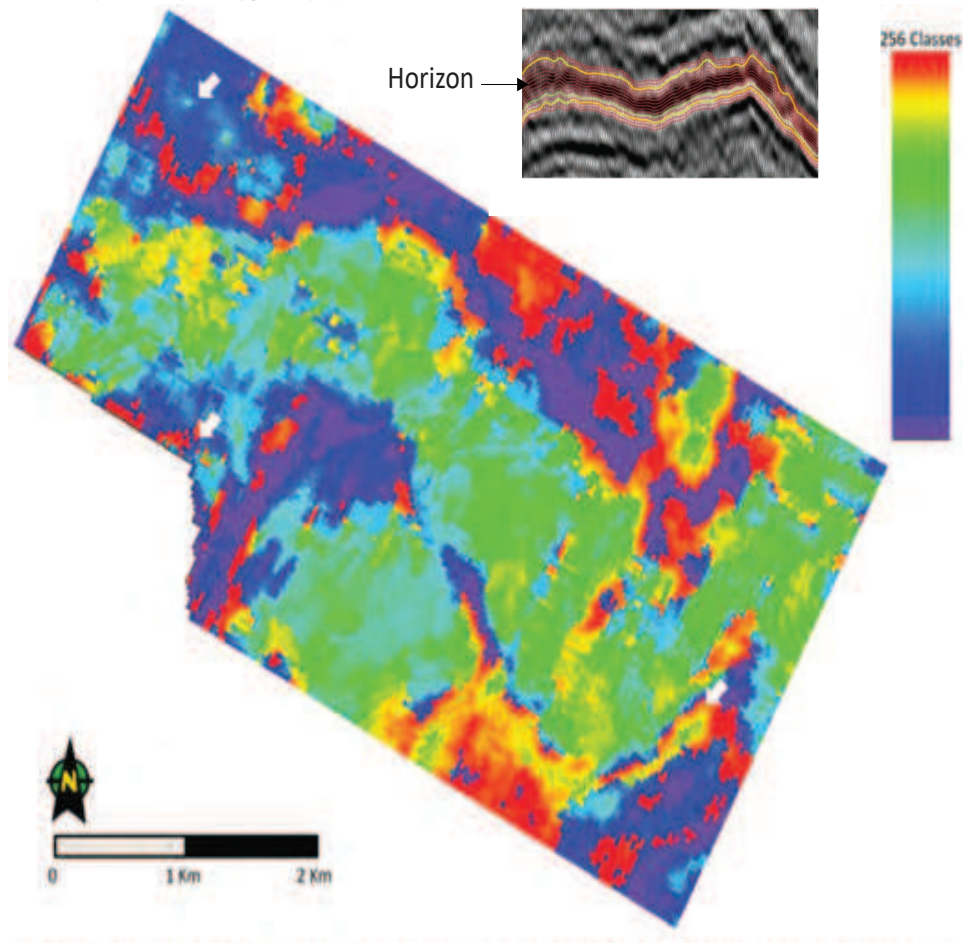


Figure 17.49: 2D seismic section as inset in upper right, and the 1D SOM of the horizon amplitudes, marked by the arrow in the inset, with 256 classes. Each class is mapped to a different color in the color scale on the right. Figure adapted from de Matos et al. (2010).

## Chapter 18

# Generative Adversarial Networks

Generative Adversarial Networks (GANs) are generative algorithms composed of two neural networks, one is called a generator and the other is a discriminator. The GAN goal is to create realistic and unique images by learning the important features of real ones. The discriminator judges whether an input image is realistic or not, and the generator creates fake images that will eventually, with sufficient training, fool the discriminator into thinking they are realistic images. After sufficient training, the generated *fake* images appear to be real and so, for example, can be used as augmented training data for a supervised learning algorithm. GANs can be used for many other applications such as transforming coarsely sampled images into more highly resolved images, interpolating seismic data, and translating seismic sections into attribute sections.

### 18.1 Introduction

The GAN was originally developed by Goodfellow et al. (2014) as a means for generating a huge number of realistic images for later use in, for example, a supervised learning algorithm. It created unique images similar to the ones in a large training set of sample images. After adequate training, believable images were characterized by a probability distribution similar in shape to the training data. A fake image is often characterized by a smooth blend of features from several training images. The original GAN implementation used multiperceptrons (Goodfellow et al., 2014), but now many implementations are with CNNs.

A simple schematic for a GAN is shown in Figure 18.1, where one of the neural networks is labeled as the discriminator and the other as the generator. The goal is to compute the generator's weights which can generate realistic images that fool the discriminator into judging these to be real images. By real, we mean that the generated images  $G(\mathbf{z})$  in Figure 18.1b have important features similar to those in the batch of real images  $\mathbf{x}$  in Figure 18.1a. The input  $\mathbf{z}$  to the generator is the random noise image in Figure 18.1c, which is transformed by the generator into the *fake* image  $G(\mathbf{z})$  of a seismic fault in Figure 18.1b.

Shortly after its introduction, the Facebook director Yann Lecun claimed "Generative Adversarial Networks is the most interesting idea in machine learning in the last ten years" (Singh, 2018). The original application of GANs was to develop unique plausible renderings of numerals from the MNIST data set, and also to create new faces from a training set of faces. The multiperceptron algorithm was plagued by uncertain convergence, blurry outputs and image size limitations, but such problems were mitigated by later improvements such as using a deep CNN (Radford et al., 2015), regularization (Che et al., 2017), incorporating conditional data into the input of the generator (Isola et al. 2018; Reed et al., 2018), feature matching (Salimans et al., 2016) and a multiscale approach where low-resolution features are used for training at the early iterations, and then higher resolution images are generated at the later iterations (Karras et al., 2017).

## Generative Adversarial Network (GAN)

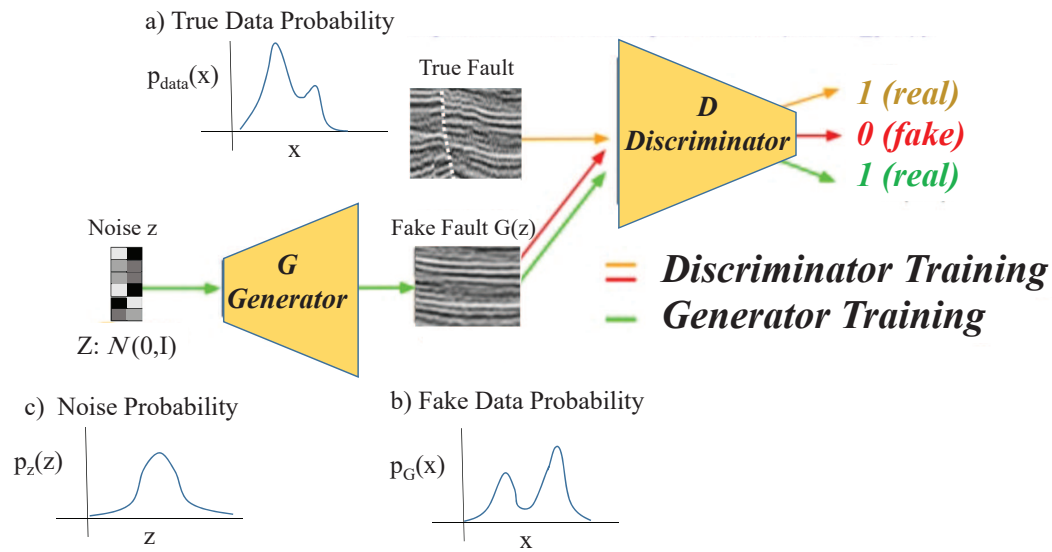


Figure 18.1: Schematic diagram for the GAN architecture where the generator's produces outputs that the discriminator judges to be either fake ( $Y = 0$ ) or real ( $Y = 1$ ). The probability distributions are plotted for the a) real image  $\mathbf{x}$ , b) generator output  $G(\mathbf{z})$  and c) noise  $\mathbf{z}$  images. Successful GAN training, where  $\mathbf{x}$  belong exclusively to true images, will reshape the probability distribution of the generator's output images in b) to be nearly the same as that of the training data in a).

CRITERIA	VANILLA GAN	CGAN	LAPGAN	DCGAN	AAE	GRAN	INFOGAN	BIGAN
Learning	Supervised	Supervised	Unsupervised	Unsupervised	Supervised, semi-supervised and unsupervised	Supervised	Unsupervised	Supervised and unsupervised
Network Architecture	Multilayer perceptrons	Multilayer perceptrons	Laplacian pyramid of convolutional networks	Convolutional networks with constraints	Autoencoders	Recurrent convolutional networks with constraints	Multilayer perceptrons	Deep multilayer neural networks
Gradient Updates	SGD with k steps for D and 1 step for G	SGD with k steps for D and 1 step for G	No updates	SGD with Adam optimizer for both G and D	SGD with reconstruction and regularization steps	SGD updates to both G and D	SGD updates to both G and D	No updates
Methodology / Objective	Minimize value function for G and maximize for D	Minimize value function for G and maximize for D conditioned on extra information	Generation of images in coarse-to-fine fashion	Learn hierarchy of representations from object parts to scenes in both G and D	Inference by matching posterior of hidden code vector of autoencoder with prior distribution	Generation of images by incremental updates to a "canvas"	Learn disentangled representations by maximizing mutual information	Learn features for related semantic tasks and use in unsupervised settings
Performance Metrics	Log-likelihood	Log-likelihood	Log-likelihood and human evaluation	Accuracy and error rate	Log-likelihood and error-rate	Generative Adversarial Metric (proposed)	Information metric and representation learning	Accuracy

Figure 18.2: Different GANs and their characteristics. Table from Hitawala (2018).

Later, Karras et al. (2018) developed a style-based GAN that leads to an automatically learned, unsupervised separation of high-level attributes of a human face and its stochastic variations of, for example, uncombed hair or eyebrows. It also allows for intuitive control of which features are to be mixed as well as interpolation of images by interpolation of the latent space.

There are now dozens of GAN architectures and applications (Hussain, 2020; Alqahtani et al., 2021), some of which are described in Figure 18.2 and Appendix 18.8. The rapid developments of GANs by the computer science community are now being tested for their effectiveness in solving geophysical problems. Navarro et al. (2020) used a deep convolutional GAN (DCGAN) to transform seismic amplitude sections into seismic attributes (Chopra and Marfurt, 2006). They used supervised training to train the GAN and then applied the trained GAN model to new seismic data. Figure 18.3 depicts their DCGAN architecture, where the seismic section on the upper left conditions the generator to produce accurate attribute (i.e., energy, coherence, phase) sections on the upper right. They found that this procedure was more than an order-of-magnitude faster compared to the traditional method for converting amplitude sections into seismic attributes.

GANs were also used to improve seismic resolution (Wang et al., 2017; Zhang et al., 2019), interpolation of seismic traces (Wang et al., 2018c), and noise attenuation (Si et al., 2019). Details for improving resolution and seismic-to-attribute translation are provided in section 18.5. A comprehensive review of GAN architectures is in Alqahtani et al. (2021).

The next sections presents the mathematical formulation of a GAN and a pseudo-code for its implementation. Numerical examples of applying GANs to seismic data are then provided, and a summary finally concludes this chapter.

## 18.2 Theory of GANs

The theory of a GAN is now described for the architecture illustrated in Figure 18.1. The next three subsections present the mathematical details for the discriminator, generator and objective functions. An excellent video description of GAN by Ganapathy Krishnamurthi is at <https://www.youtube.com/watch?v=MKedB9qOH4>.

For a GAN,  $D(\mathbf{x})$  and  $G(\mathbf{z})$  are probability functions so they can only take on values between 0 and 1. A high-probability value of  $D(\mathbf{x}) \approx 1$  means that the discriminator judges the input image  $\mathbf{x}$  to be real, consequently  $\ln D(\mathbf{x}) = \ln 1 = 0$  is maximized. If the input to the discriminator is the *generated image*  $G(\mathbf{z})$ , then a high-probability value for the discriminator  $D(G(\mathbf{z})) \approx 1$  means that

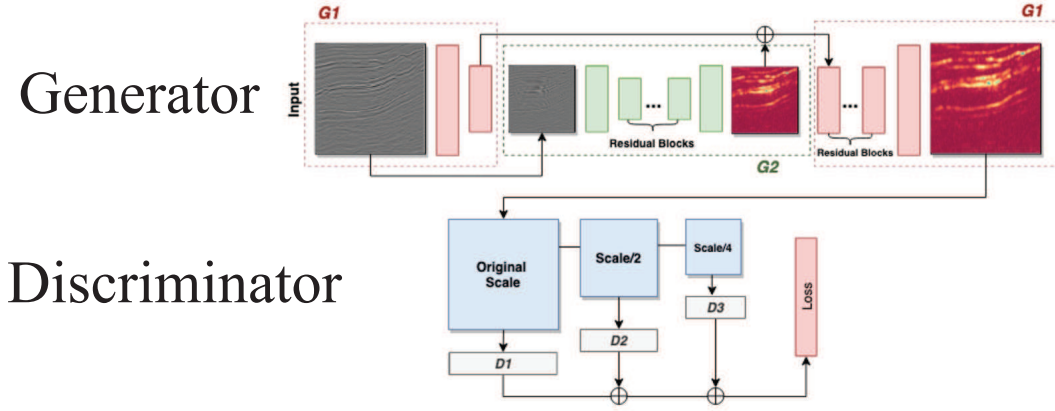


Figure 18.3: Architecture for DCGAN used by Navarro et al. (2020) for mapping seismic sections (upper left) to attribute sections (upper right). The seismic section is the input constraint that conditions the supervised generator to produce accurate attribute sections.

$\ln[1 - D(G(\mathbf{z}))] \approx \ln 0 = -\infty$  is minimized. In this case, we can say that the generator produced an image that has fooled the discriminator into thinking it is real.

### 18.2.1 GAN Objective Function and Minimax Algorithm

The objective function for a GAN is the sum of the objective functions for the discriminator and generator:

$$\epsilon = E_{\mathbf{x} \sim p(\mathbf{x})_{data}} \ln D(\mathbf{x}) + E_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} \ln(1 - D(G(\mathbf{z}))), \quad (18.1)$$

where the optimal coefficients are found by a minimax<sup>1</sup> procedure:

$$\{\mathbf{w}_D^*, \mathbf{w}_G^*\} = \underset{\mathbf{w}_G}{\operatorname{argmin}} \underset{\mathbf{w}_D}{\operatorname{argmax}} [\overbrace{E_{\mathbf{x} \sim p(\mathbf{x})_{data}} \ln D(\mathbf{x}) + E_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} \ln(1 - D(G(\mathbf{z})))}^{\epsilon}]. \quad (18.2)$$

Here,  $E_{\mathbf{z} \sim P(\mathbf{z})}$  ( $E_{\mathbf{x} \sim p_{data}(\mathbf{x})}$ ) indicates the average of sampling from the probability distribution  $P(\mathbf{z})$  ( $p_{data}(\mathbf{x})$ ). Typically,  $p_{\mathbf{z}}(\mathbf{z})$  is selected to be a Gaussian distribution of much lower dimension than the  $N \times N$  images  $\mathbf{x}$  and  $G(\mathbf{z})$ . This minimax formulation is commonly known as the *vanilla GAN* solution (Farni and Ozdaglar, 2020).

As will be shown below, there will be two steps to the iterative minimax algorithm. The first step is maximization of  $\epsilon$  w/r to the  $\mathbf{w}_D$  weights so that the discriminator correctly identifies the data images  $\mathbf{x}$  as true images, i.e.  $D(\mathbf{x}) \approx 1$ , and any images  $G(\mathbf{z})$  produced by the generator to be false, i.e.  $D(G(\mathbf{z})) \approx 0$ . The second step is the adversarial step, where the weights  $\mathbf{w}_G$  of the *adversarial generator* are trained to produce images  $G(\mathbf{z})$  that fool the discriminator into judging them to be true images, i.e.  $D(G(\mathbf{z})) \approx 1$ . That is, the discriminator thinks the true images only

<sup>1</sup>Minimax is a decision rule used in game theory, AI etc. for minimizing the possible loss for a worst case (maximum loss) scenario (<https://en.wikipedia.org/wiki/Minimax>). Later, we will replace the generator's cost function  $\min_{\mathbf{w}_G} \ln[1 - D(G(\mathbf{z}))]$  with its equivalent  $\max_{\mathbf{w}_G} \ln D(G(\mathbf{z}))$ , but still refer to the algorithm as a minimax method.



originate from the data batch governed by  $p_{x \sim p_{data}}(x)$  yet it can be fooled, e.g.  $D(G(\mathbf{z})) \approx 1$ , by counterfeit images  $D(G(\mathbf{z}))$  produced by a smart generator.

Computing the optimal weights in equation 18.1 consists of three steps in the iterative minimax procedure.

1. **Train the Discriminators**  $D(\mathbf{x})$  and  $D(G(\mathbf{z})) \rightarrow \mathbf{w}_D$ . The weights  $\mathbf{w}_G$  for the discriminators  $D(\mathbf{x})$  and  $D(G(\mathbf{z}))$  are trained by a *gradient ascent* method using the update formula with a positive sign in front of the gradient:

$$\mathbf{w}_D = \mathbf{w}_D + \alpha \nabla_{\mathbf{w}_D} \overbrace{\frac{1}{M} \sum_{m=1}^M \{\ln D(\mathbf{x}^{(m)}) + \ln[1 - D(G(\mathbf{z}^{(m)}))]\}}^{\epsilon}, \quad (18.3)$$

where  $M$  is the number of samples and  $\epsilon$  is interpreted as

$$\begin{aligned} \epsilon &= \frac{1}{M} \sum_{m=1}^M \{\ln D(\mathbf{x}^{(m)}) + \ln[1 - D(G(\mathbf{z}^{(m)}))]\}, \\ &\approx \underbrace{E_{\mathbf{x} \sim P_{data}} \ln D(\mathbf{x})}_{\substack{D(\mathbf{x}) \approx 1 \text{ recognizes } \mathbf{x} \text{ authentic} \\ \text{so } \ln D(\mathbf{x}) \text{ is maximized}}} + \underbrace{E_{\mathbf{z} \sim P(\mathbf{z})} \ln[1 - D(G(\mathbf{z}))]}_{\substack{D(G(\mathbf{z})) \approx 0 \text{ recognizes } G(\mathbf{z}) \text{ as fake} \\ \text{so } \ln[1 - D(G(\mathbf{z}))] \approx \ln 1 \text{ maximized}}}. \end{aligned} \quad (18.4)$$

Training the discriminator in this step requires a maximization of both terms on the rightside of equation 18.3 in order for the discriminator to recognize images drawn from  $P_{data}$  to be authentic and  $G(\mathbf{z})$  images to be fake.

2. **Train the Generator**  $G(\mathbf{z}) \rightarrow \mathbf{w}_G$ . The  $\mathbf{w}_G$  must now be adjusted so that the discriminator  $D(G(\mathbf{z})) \approx 1$  gets fooled and thinks the generated image  $G(\mathbf{z})$  is as authentic as the input images  $\mathbf{x}$ . In this step,  $\mathbf{w}_D$  is fixed from the previous step and the optimal  $\mathbf{w}_G$  minimizes the objective function  $\epsilon$  in equation 18.4 so that the discriminator is fooled. The  $\ln D(\mathbf{x})$  term in equation 18.4 will not contribute to the gradient w/r to  $\mathbf{w}_G$  because it is not a function of  $\mathbf{w}_G$ . Therefore the update procedure for  $\mathbf{w}_G$  is a *steepest descent method* where there is a negative sign in front of the gradient, i.e.,

$$\mathbf{w}_G = \mathbf{w}_G - \alpha \sum_{m=1}^M \nabla_{\mathbf{w}_G} \underbrace{\ln[1 - D(G(\mathbf{z}^{(m)}))]}_{\substack{D(G(\mathbf{z}^{(m)})) \approx 1 \text{ fools the discriminator} \\ \text{so } \ln[1 - D(G(\mathbf{z}^{(m)})) \text{ minimized}}}, \quad (18.5)$$

because  $D(G(\mathbf{z})) \approx 1$  indicates that the discriminator foolishly thinks  $G(\mathbf{z})$  is an image as authentic as  $\mathbf{x}$ .

3. Repeat steps 1 and 2 using a new batch of  $\mathbf{x}$  and  $\mathbf{z}$  samples until the *fake* images appear to be realistic renderings indistinguishable from the real images. This is now a well-trained GAN that can be used to generate huge numbers of realistic images by drawing from  $P_{\mathbf{z}}(\mathbf{z})$ .

Minimizing  $\ln[1 - D(G(\mathbf{z}))]$  w/r to  $\mathbf{w}_G$  is equivalent to maximizing  $\ln D(G(\mathbf{z}))$  w/r to  $\mathbf{w}_G$ , i.e.,

$$\arg \min_{\mathbf{w}_G} E_{\mathbf{z} \sim P(\mathbf{z})} \ln[1 - D(G(\mathbf{z}))] = \arg \max_{\mathbf{w}_G} E_{\mathbf{z} \sim P(\mathbf{z})} \ln D(G(\mathbf{z})), \quad (18.6)$$

because  $D(G(\mathbf{z})) = 1$  minimizes the leftside expression as well as maximizing the rightside one. In practice, the rightside expression is preferred when computing updates for  $\mathbf{w}_G$  in equation 18.5 because it leads to a more robust convergence (Goodfellow, 2016). In this case there will be two

gradient ascent loops for carrying out the minimax procedure of equation 18.1: the outer loop will update the weight  $\mathbf{w}_G$ , and the inner loop will update the weight  $\mathbf{w}_D$ , as will be described in Code 18.3.1.

This is known as a gradient optimization method that alternates between updating two different sets of parameters, similar to that used for the AAE in Chapter 14. Goodfellow et al. (2014) claim that "This results in  $D$  being maintained near its optimal solution, so long as  $G$  changes slowly enough.". They also recommend a variation of the algorithm: *Rather than training  $G$  to minimize  $\ln(1 - D(G(z)))$  we can train  $G$  to maximize  $\ln D(G(z))$ . This objective function results in the same fixed point of the dynamics of  $G$  and  $D$  but provides much stronger gradients early in learning.*. To achieve stable convergence, the DCGAN architecture of Radford et al. (2015) is recommended.

### 18.3 Pseudo-Code for a GAN

The minimax pseudo-code (Goodfellow et al., 2014) for a GAN is given below.

**Code 18.3.1.** *Minimax Pseudo-code for a GAN*

```

for  $i = 1 : iter.global$       (Global Loop over the GAN)
  for  $id = 1 : iter.discrim$  (Discriminator Ascent Loop)
    Sample minibatch of  $M$  noise samples
       $\{\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \dots, \mathbf{z}^{(M)}\}$  from  $p_G(\mathbf{z})$ 
    Sample minibatch of  $M$  noise samples
       $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(M)}\}$  from  $p_{data}(\mathbf{x})$ 
    
$$\mathbf{w}_D = \mathbf{w}_D + \frac{\alpha}{M} \nabla_{\mathbf{w}_D} \sum_{m=1}^M [\ln D(\mathbf{x}^{(m)}) + \ln(1 - D(G(\mathbf{z}^{(m)})))]$$

  end
  for  $id = 1 : 1$  (Generator Ascent Loop)
    Sample minibatch of  $M$  noise samples
       $\{\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \dots, \mathbf{z}^{(M)}\}$  from  $p_G(\mathbf{z})$ 
    
$$\mathbf{w}_G = \mathbf{w}_G + \frac{\beta}{M} \nabla_{\mathbf{w}_G} \sum_{m=1}^M \ln D(\mathbf{z}^{(m)})$$

  end
end

```

(18.7)

where the last do loop replaced both  $\ln[1 - D(G(\mathbf{z}^{(m)}))]$  in equation 18.5 and the associated steepest descent algorithm with, respectively,  $\ln D(\mathbf{z}^{(m)})$  and the steepest ascent algorithm. Many practical types for implementing a GAN are in Goodfellow (2016).

#### Remarks:

- **Convergence.** Goodfellow et al. (2014) show that the global minimum of the objective function is reached when  $p_g(\mathbf{x}) = p_{data}$  (see Exercise 18.7.6). The gradient algorithm can

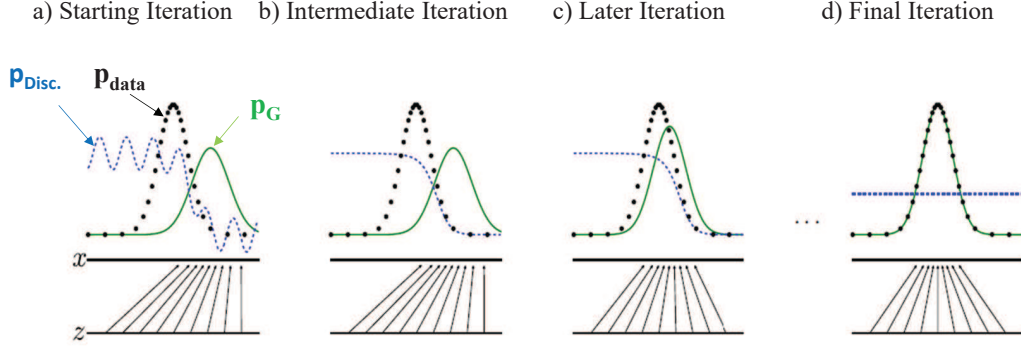


Figure 18.4: Probability distributions plotted against the domain for the true data images  $p_{data}$  (dotted black lines), the generator's images  $p_G$  (green lines), and the discriminator's output  $p_{Disc.}$  (blue dashed lines) for early to later iterations of the minimax method. The probability distributions are plotted for the a) early, b)-c) intermediate, and d) final iterations, while the lower horizontal line is the domain from which  $z$  is sampled. The axis for  $x$  is the domain for the training images. After several iterations,  $G$  and  $D$  will be optimized where  $p_G = p_{data}$  in d), i.e., the discriminator is unable to differentiate between the two distributions, i.e.  $D(x) = D(G(z)) = 1/2$ . Figure adopted from Goodfellow et al. (2014).

also converge to the global minimum if both  $G$  and  $D$  have enough capacity. The condition  $p_g(\mathbf{x}) = p_{data}$  makes sense because generating a suite of believable models is equivalent to generating a probability distribution  $p_G(\mathbf{x})$  (see Figure 18.1b) for the generative images that approximately matches the probability distribution  $p_{data}(\mathbf{x})$  of the actual images in Figure 18.1a. At the initial iterations of the GAN, these probability distributions are strongly mismatched as illustrated by the green and black curves in Figure 18.4a-18.4b. The later iterations in Figure 18.4c-18.4d show a much closer in agreement, which suggests that the generator's images are now believable by the discriminator. In practice, stable convergence of GANs is still one of its biggest challenges of GANs (Alqahtani et al., 2021).

- The generator's goal is to create a fake image  $G(\mathbf{z})$  in Figure 18.1b from a random vector input  $\mathbf{z}$  so that the discriminator judges  $D(G(\mathbf{z})) \approx 1$  to be a realistic output. But this is a contradiction because  $G(\mathbf{z})$  is always fake, i.e. it is not from the training set of images  $\mathbf{x}$ . This is a zero-sum adversarial game similar to that of an art counterfeiter trying to fool the art inspector. For example, if the batches of  $\mathbf{x}$  only contained images of masterpieces by Van Gogh, then the discriminator should be trained to recognize input images as true  $Y = 1$  paintings if they contain the post-impressionist style of Van Gogh. If the generator is also well trained by a GAN then it will generate images of unique paintings that appear to be realistic renderings of paintings with the van Gogh style.
- **Nash Equilibrium.** GAN training searches for the Nash equilibrium point for a two-player non-cooperative game (Salimans et al., 2016). The discriminator seeks to maximize its cost function  $\epsilon^{discrim.} = \ln D(\mathbf{x}) + \ln[1 - D(G(\mathbf{z}))]$  w/r to the weights  $\mathbf{w}_D$  in equation 18.7 while the adversarial generator minimizes its cost function  $\epsilon^{gen.} = \ln[1 - D(G(\mathbf{z}))]$  w/r to the weights  $\mathbf{w}_G$ . GANs are designed to reach a Nash equilibrium at which each player cannot reduce their cost without changing the other players' parameters.

The Nash equilibrium is a point  $(\mathbf{w}_D^*, \mathbf{w}_G^*)$  such that the objective function in equation 18.1

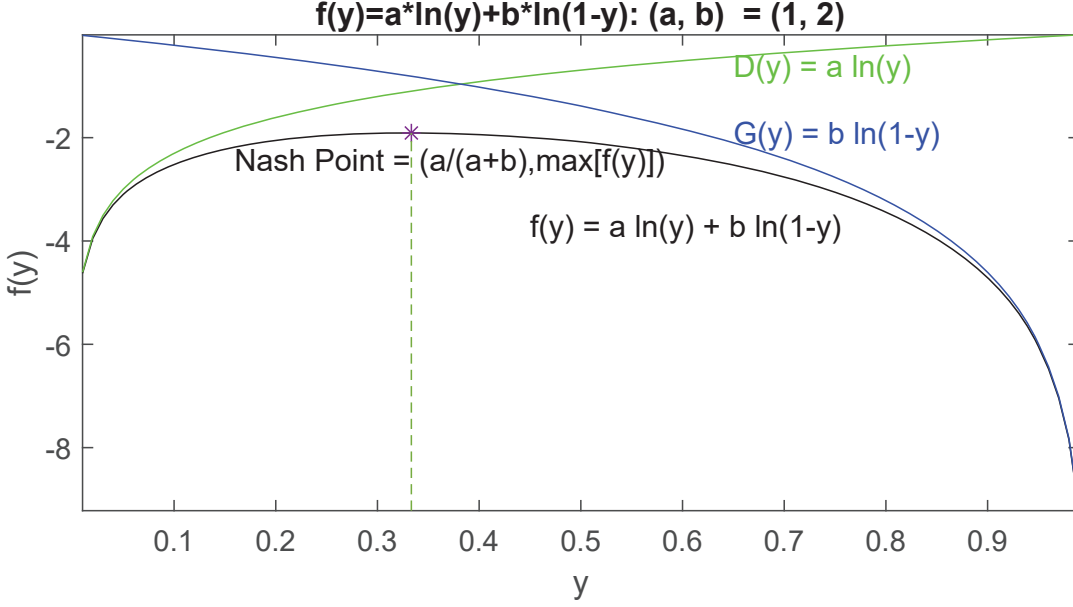


Figure 18.5: Plots of the cost functions for the green discriminator  $D(y) = a \ln y$ , blue generator  $G(y) = b \ln(1 - y)$  and total cost function  $f(y) = a \ln y + b \ln(1 - y)$ .

achieves the following condition (Farnia and Ozdaglar, 2020):

$$\epsilon(\mathbf{w}_D, \mathbf{w}_G^*) \leq \epsilon(\mathbf{w}_D^*, \mathbf{w}_G^*) \leq \epsilon(\mathbf{w}_D^*, \mathbf{w}_G), \quad (18.8)$$

where  $\epsilon^{discrim.}$  is at a maximum with respect to variations in  $\mathbf{w}_D$  and  $\epsilon^{gen.}$  is at a minimum with respect to variations in  $\mathbf{w}_G$ .

Figure 18.5 demonstrates a simple example of the 1D cost function

$$f(y) = a \ln(y) + b \ln(1 - y), \quad (18.9)$$

where the discriminator  $D(y) = a \ln(y)$  and generator  $G(y) = b \ln(1 - y)$  compete to find the stationary point at  $y = a/(a + b)$  (see Exercise 18.7.7). At this point, a move to the right increases the value of the green discriminator curve but it decreases the value of the blue generator curve. However this moves points downhill from the stationary point at  $y = a/(a+b)$ . There is only one variable  $y$  here so only the lower bound portion of equation 18.8 is applicable.

The conditions in equation 18.8 describe that for a 2D function  $f(x, y)$  with a saddle point, as shown in Figure 18.6 for  $f(x, y) = xy$ . Starting from the saddle point at  $(0, 0)$  and moving along a line slanted at the azimuthal angle  $45^\circ$  w/r to the x-axis decreases the function value  $f(x, y) = xy$ ; but moving along a line at angle  $-45^\circ$  increases the function value. This satisfies the conditions for the Nash equilibrium point at  $(0, 0)$  if  $x$  is  $\mathbf{w}_g$  and  $y$  is  $\mathbf{w}_D$ .

Unfortunately, finding Nash equilibria is a very difficult problem, and 1st-order gradient optimization methods often fail to converge to the stationary point, as demonstrated in Figure 3.4. Algorithms exist for specialized cases, but Farnia and Ozdaglar (2020) claim that the Nash equilibrium point might not exist for GANs. However, there are successful modifications to GANs that allow them to get useful results. This is usually the case when the GANs are regularized by, for example, batch normalization or spectral normalization of the generator, or by adding a regularization term to the objective function (Farnia and Ozdaglar, 2020).

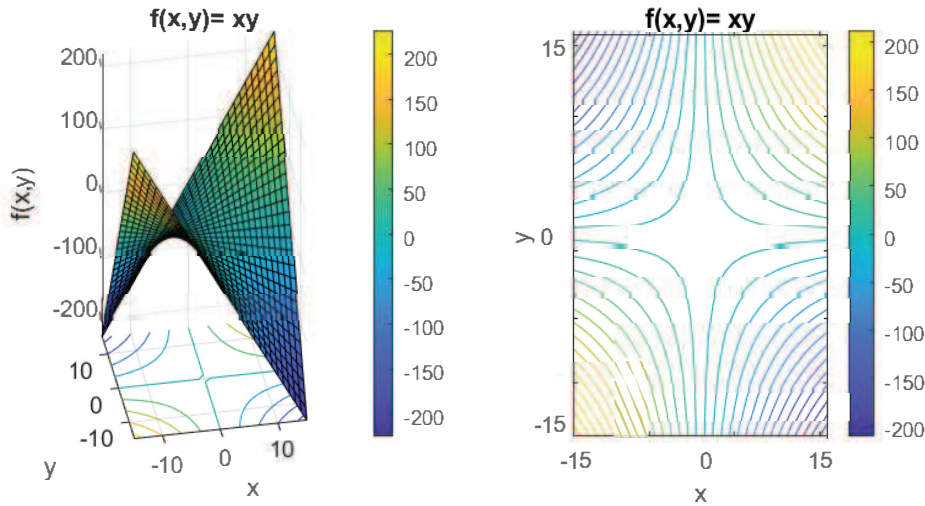


Figure 18.6: Plot of the cost function  $f(x,y) = xy$  which plots out as a saddle with a saddle point at  $(0,0)$ .

- **Convolutional GAN.** Early tests showed that a GAN (Goodfellow, 2014; Salimans et al., 2016) can sometimes fail to converge to a believable generated image<sup>2</sup>. This motivated the development of partial remedies to improve the quality of GAN images (Denton et al., 2015; Radford et al., 2015; Che et al., 2016; Springenberg, 2016; Salimans et al., 2016; Reed et al., 2016; Arjovsky et al., 2017; Khoash et al., 2018; Isola et al., 2018; Karras et al., 2017). Radford et al. (2015) achieved excellent generated images (see Figure 18.7) by using a DCGAN (see the generator shown in Figure 18.17), where fully connected layers, pooling and unpooling are eliminated. Pooling layers are replaced with strided convolutions. They also used ReLU functions in the generator except for a Tanh in the last layer, leaky ReLUs in the discriminator, batch normalization with mini-batches of input data, and an Adam optimizer with a stochastic gradient descent. Batch normalization stabilizes learning by normalizing the input to each unit to have zero mean and unit variance. Radford et al. (2015) reported that batch normalization applied to all the layers resulted in sample oscillation and model instability. This was avoided by not applying batchnorm to the generator output layer and the discriminator input layer. Results from their tests showed that the manifold  $G(\mathbf{z})$  of the generator varied smoothly between interpolated images of a bedroom in Figure 18.8.

Salimans et al. (2016) introduced a new objective function  $\|E_{\mathbf{x} \sim p_{data}} D(\mathbf{x}) - E_{\mathbf{z} \sim p_z} D(G(\mathbf{z}))\|_2^2$  that requires the generator to generate data that matches the statistics of the real data. In this case, the discriminator is forced to honor the statistics that are worth matching. Their empirical results demonstrated that this could sometimes lead to convergence when the standard GAN failed.

- **Mode Collapse.** Mode collapse is characterized by the output of the generator collapsing to a single oscillating mode as the iterations proceed. Radford et al. (2015) partly mitigated this mode collapse problem with DCGAN, which includes batch normalization.

<sup>2</sup>Radford et al. (2016) states that 'Generative Adversarial Networks (Goodfellow et al., 2014) generated images suffering from being noisy and incomprehensible.'

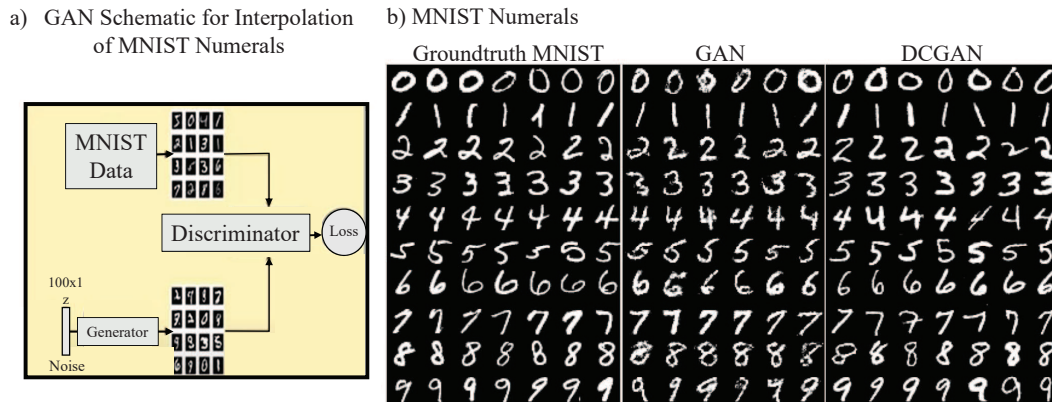


Figure 18.7: a) GAN schematic for interpolating MNIST numerals and b) Groundtruth, GAN and DCGAN images of MNIST numerals computed by Radford et al. (2015). The GAN images are computed using fully connected neural networks for the generator while a CNN is used for the DCGAN images. Schematic partly adapted from GAN Youtube video <https://www.youtube.com/watch?v=MKedB9qOHi4> by Ganapathy Krishnamurthi.

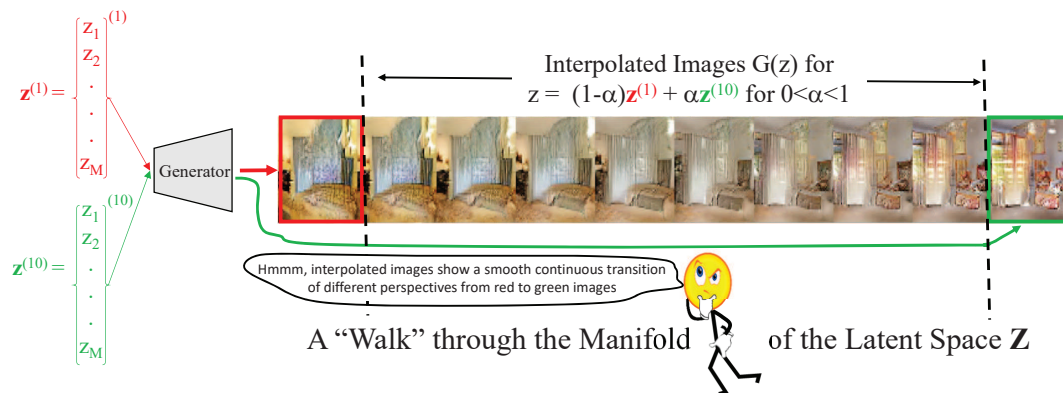


Figure 18.8: Interpolation of the input vectors  $z^{(1)}$  and  $z^{(10)}$  resulted in believable and smoothly varying images  $G(z)$ . Radford et al. (2015) denoted this interpolation exercise as walking through the manifold of latent space.

### Simple Example: Dipping Faults & Noise

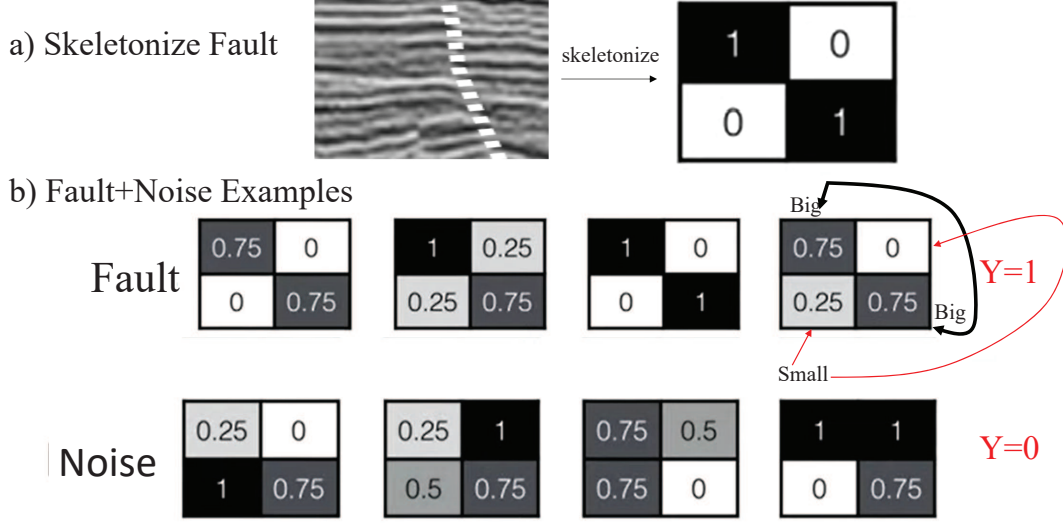


Figure 18.9: a) Seismic section showing an image of a dipping fault (dashed line) skeletonized into a  $2 \times 2$  approximation. The true fault and noise images are shown in b), where darker (brighter) intensity values correspond to higher (lower) gray-level values.

## 18.4 Simple Fault Example

A simple seismic fault example is used to illustrate the steps in applying a GAN to seismic data. Figure 18.9a depicts a  $45^\circ$  dipping fault skeletonized into a  $2 \times 2$  image of the fault. Examples of skeletonized faults and noise are shown in Figure 18.9b. There are three steps in training a GAN to produce believable  $2 \times 2$  renderings of actual dipping faults.

1. **Generator.** The NN for the generator is assumed to have one hidden layer with four nodes as shown in Figure 18.10. Here, the  $4 \times 1$  weight matrix is  $\mathbf{w}_G = (1, -1, -1, 1)^T$  and the bias vector is  $\mathbf{b}_G = (1, 0.4, -1, 1)^T$ . In this case, the output of the generator is the  $4 \times 1$  output vector

$$\mathbf{y} = \sigma(\mathbf{w}_G z + \mathbf{b}_G) = \sigma(1.7, -0.3, -1.7, 1.7)^T, \quad (18.10)$$

where  $z = 0.7$  is the scalar input value from the random number generator. The component values of  $\mathbf{y}$  are mapped to the pixels in the  $2 \times 2$  image on the right of Figure 18.10. The dark gray colors in the image correspond to large numbers that are nearly 1 and the light shading corresponds to numbers that are almost 0. The generated image produces a reasonable rendering of a  $45^\circ$  dipping fault. If the output did not appear to resemble a  $45^\circ$  dipping fault then the gradient descent method will adjust the coefficients to form a more believable fault.

2. **Discriminator.** The architecture of the discriminator is shown in Figure 18.11, where the  $2 \times 2$  input images represent the a) true and b) fake faults. In this example, the NN weights  $\mathbf{w}_D = (1, -1, -1, 1)$  and  $b_D = (-1)$  are correctly tuned to distinguish the true fault ( $\sigma(\mathbf{w}_D^T \mathbf{x} + \mathbf{b}_D) = 0.73$ ) from the false fault  $\sigma(\mathbf{w}_D^T \mathbf{x} + \mathbf{b}_D) = 0.15$ ).
3. **GAN.** Combining the discriminator with the generator is the GAN model shown in Figure 18.1. The NN weights are adjusted until a stationary point of the objective function in



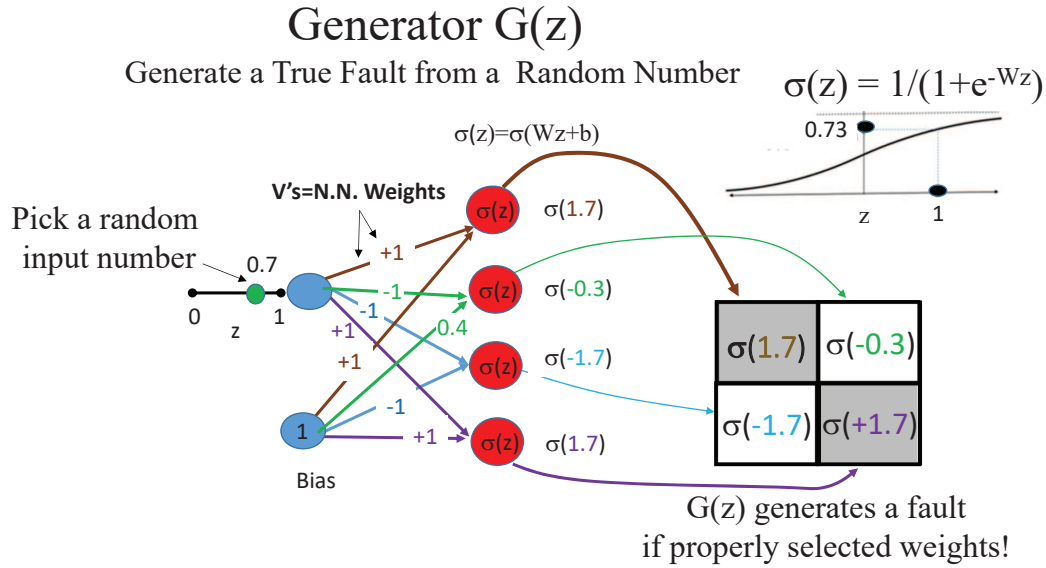


Figure 18.10: Simple generator architecture with 1 hidden layer consisting of 4 nodes. The output number for each node is associated with the intensity value of a pixel in the  $2 \times 2$  image on the right.

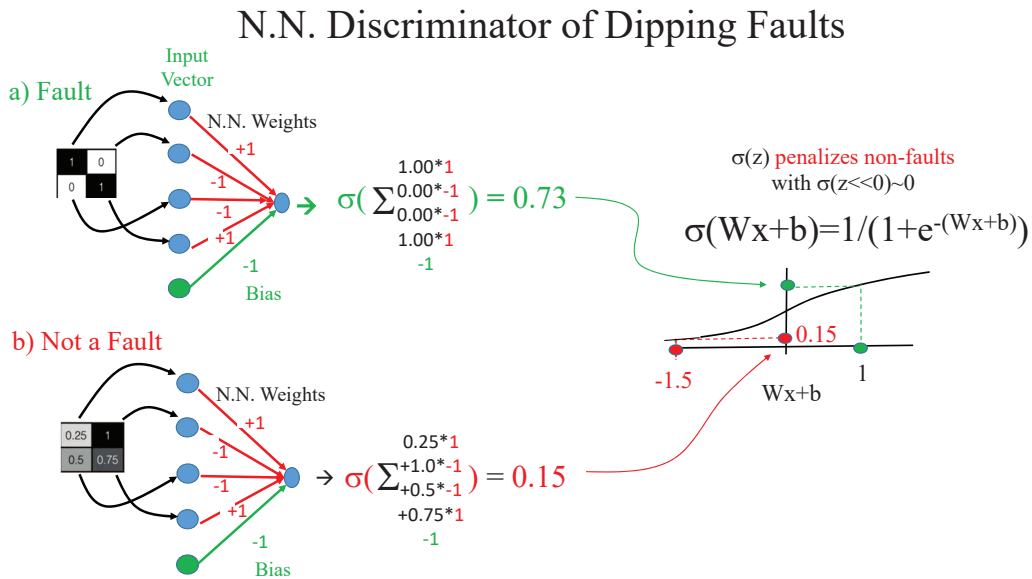


Figure 18.11: Simple discriminator architecture with one hidden layer that produces a)  $\sigma(\mathbf{w}_D \mathbf{x} + \mathbf{b}) = 0.73$  for a true fault input  $\mathbf{x}$  and b)  $\sigma(\mathbf{w}_D \mathbf{x} + \mathbf{b}) = 0.15$  for a false fault input. In this case the discriminator predicts with a high probability of 0.73 that  $\mathbf{x}$  in a) is a true fault (i.e.  $Y = 1$ ) and b) is likely a false fault (i.e.  $Y = 0$ ) with probability  $1 - 0.15 = 0.85$ .



equation 18.1 is found.

## 18.5 GAN Examples in Geoscience

Several examples of applying a GAN to seismic data are now presented. The first one is for transforming seismic sections into attribute sections (Navarro et al., 2020) and the second one is for increasing the resolution of seismic images by a GAN.

### 18.5.1 Seismic-to-Attribute Sections by a GAN

Navarro et al. (2020) used a DCGAN architecture to transform seismic sections to attribute sections. This is somewhat similar to the DCGAN that translates images from one scene to another, as done with the conditional GAN for day-to-night translations, and black-and-white photos to color photos and image-to-image translation (Reed et al., 2018; Isola et al., 2018; Alqahtani et al., 2021). Navarro et al. (2020) used supervised training to train the GAN and then inserted new seismic data into the trained model for real-time translation.

Figure 18.3 depicts the DCGAN architecture of Navarro et al. (2020), where the seismic section on the upper left is the input to the generator to produce accurate attribute (e.g., coherence, phase, or energy) sections on the upper right. The following attributes are used for the attribute sections:

- **Coherence:** This attribute can be used to detect discontinuities along horizons (see section 5.8).
- **Energy:** Squaring the amplitude of each sample in a seismic trace and summing over a small-time interval gives the energy as a function of time. This attribute can highlight lateral changes and detect geologic objects in the sections.
- **Instantaneous Phase:** The instantaneous phase as a function of time can be computed by taking the Hilbert transform of the trace (Yilmaz, 2001). This attribute enhances spatial continuity and readily reveals the discontinuity of reflections across a fault.

Attributes are features in the seismic data that highlight important geology that are otherwise hidden in the recorded traces.

The goal of Navarro et al. (2020) is to replace the direct transformations of seismic sections to attribute sections with real-time DCGAN calculations. They used the multiscale strategy of Wang et al. (2017) where the scale of the generators  $G = (G1, G2)$  and discriminators  $D = (D1, D2, D3)$  transition from coarse-to-fine. The generator is designed to promote aggregation between the global and local information during the image synthesis task, while the discriminator  $D$  aims to distinguish real attributes from generated ones. Their GAN will be a supervised one where the  $k^{th}$  training pair  $(\mathbf{s}^{(k)}, \mathbf{a}^{(k)})$   $k \in \{1, 2, \dots, K\}$  consists of the seismic  $\mathbf{s}^{(k)}$  and attribute  $\mathbf{a}^{(k)}$  sections. They trained this DCGAN using the objective function in equation 18.1 except the generator's input is the seismic section rather than a random vector.

Navarro et al. (2020) state the following: *To differentiate real and synthesized images, the discriminator needs a large receptive field. This leads to deeper networks with larger convolution kernels, which may promote overfitting. To deal with this problem, three discriminators with identical network structures are used. Each discriminator operates in different lower-resolution scales, improving the ability of final discriminator to distinguish real and synthesized samples..* A well-trained DCGAN should be able to take input seismic sections  $\mathbf{s}$  and quickly produce accurate attribute sections  $\mathbf{a}$ .

### Generator

The generator at the top of Figure 18.3 consists of the global generator  $G1$  and the local enhancer  $G2$ . The input to the residual blocks in  $G2$  is the element-wise sum of the feature maps from  $G2$  and

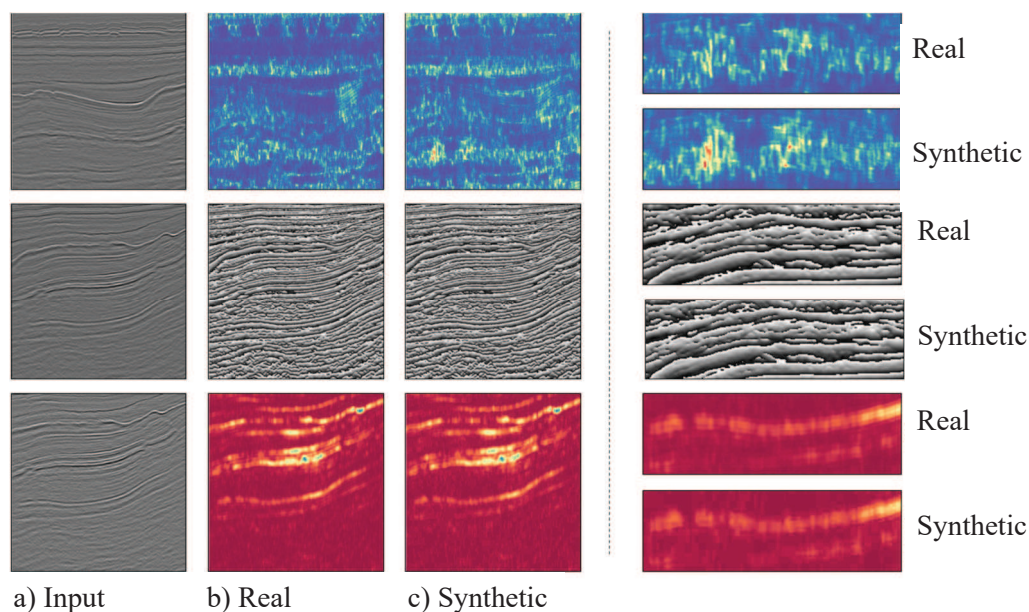


Figure 18.12: Results of a DCGAN trained to compute Semblance (top), Instantaneous Phase (middle) and Energy (bottom) seismic attributes. The overall image structure is well captured. Local features present minor differences in expanded regions of interest (right panels). Figure from Navarro et al. (2020).

the last feature map from  $G1$ . This produces an effective blending of the global and local information in the seismic section.

### Discriminator

The discriminator at the bottom of Figure 18.3 needs a large receptive field in order to differentiate real and synthesized images. Thus, deeper networks are required with larger convolution kernels, which might lead to overfitting. To mitigate this problem, three discriminators with identical network structures are employed. Each discriminator is associated with a different scale, which improves the performance of the final discriminator ( $D1, D2, D3$ ) to distinguish real and synthesized samples.

### Results

Navarro et al. (2020) applied their DCGAN procedure to seismic sections such as those shown in Figure 18.12. The results are shown in the DCGAN attribute images in Figure 18.12c, which are compared to the groundtruth attribute sections in Figure 18.12b. They found that the differences between the GAN attribute sections and the groundtruth sections are minor. They also found that their results were superior to those from Geng et al. (2019) who used a U-Net CNN architecture in a traditional supervised learning role. In this case, Navarro et al. (2020) claimed better performance for the objective of translating seismic sections to attribute sections. After training, they found that this procedure was more than an order-of-magnitude faster compared to the traditional method for converting amplitude sections into seismic attributes.

### 18.5.2 Seismic Resolution Enhancement by GANs

Zhang et al. (2019) used a GAN to enhance the seismic resolution of 3D field data. Their goal was to use training examples from 64% of the recorded seismic data as the labeled data for higher resolution. Figure 18.13 depicts the architecture of the DCGAN where the input data to the generator consist of low-resolution seismic sections. The generator's output consists of the seismic sections that might be equivalent to those processed by a traditional high-resolution method. The role of the discriminator is to judge the output of the generator as either a true or false high-resolution image.

Zhang et al. (2019) used two objective functions for training their GAN: the pre-training objective function  $\epsilon^{pre}$  is for pre-training the generator's weights separate from the standard GAN training. The pre-training objective function is

$$\epsilon^{pre} = \underbrace{-\sum_{k=1}^K \ln D(G(\mathbf{x}_{low\ res}^{(k)}))}_{\text{typical generator loss}} + \underbrace{\sum_{k=1}^K \|G(\mathbf{x}_{low\ res}^{(k)}) - \mathbf{x}_{hi\ res}^{(k)}\|^2}_{\text{mean-square error}}, \quad (18.11)$$

where there are  $K$  training examples and  $\mathbf{x}_{low\ res}^{(k)}$  is the  $k^{th}$  training example at low resolution as depicted by the input images in the upper left images in Figure 18.13. The high-resolution image  $\mathbf{x}_{hi\ res}^{(k)}$  that the generator must produce is depicted on the upper-right of Figure 18.13. This is a regularizer that prevents mode collapse<sup>3</sup> (Srivastava et al., 2017; Hong, 2019; Pei et al., 2021). The mean-square error term in  $\epsilon^{pre}$  encourages the generator's output image  $G(\mathbf{x}_{low\ res}^{(k)})$  to be closest to the high-resolution groundtruth image  $\mathbf{x}_{hi\ res}^{(k)}$  in the  $L_2$  norm sense. Since training pairs of output and input images are needed then this is a supervised GAN. Once the generator is pre-trained, then an alternating gradient method is used to solve for the minimax solution of equation 18.2.

The trained GAN was then applied to 36% percent of the field data to enhance their resolution. These images were then compared to the ones computed by a conventional resolution-enhancement method. The results in Figure 18.14 show comparable enhancements in resolution, but the GANs was able to recover higher frequency details as suggested in the third column of Figure 18.14. A standard resolution enhancement procedure was used to increase resolution for the b) images, and the c) images were those enhanced by the GAN procedure. Zhang et al. (2019) claim the GAN images contained higher-frequency details.

Figure 18.15 shows the spectral comparisons among the raw, standard resolution-enhanced, and GAN-enhanced traces. The enhanced traces show significant widening, but both the GAN and conventional resolution methods show similar widening of the bandwidth. A benefit of the GAN might be that it is computationally more efficient in implementation once proper training is done. These results are somewhat consistent with those of Christian et al. (2017) who achieved higher resolution in photographs by a similar GAN procedure.

## 18.6 Summary

The theory of GANs is presented. At each cycle of iterations, the discriminator  $D(\mathbf{x})$  undergoes a small number of iterations to adjust its weights to correctly classify as  $Y = 1$  a mini-batch of real data examples  $\mathbf{x}^{(k)}$   $k \in \{1, 2, \dots, K\}$ ; the task of  $D(\mathbf{x})$  is to also classify fake images  $G(\mathbf{z})$  as fake images. Then the generator is allowed 1 or 2 iterations to update its weights to produce a *fake* image  $G(\mathbf{z})$  that hopefully fools the discriminator into thinking it is *real*. The input  $\mathbf{z}$  to the generator is a small random vector compared to the large-dimension output image, so  $G$  can be thought of as

<sup>3</sup>Mode collapse often results in generated images being significantly similar, even though their corresponding latent vectors are quite different (Hong, 2019).

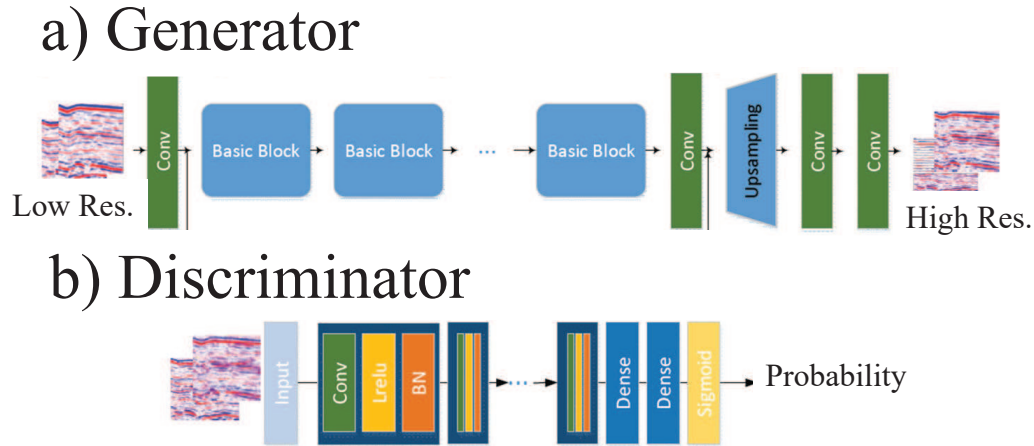


Figure 18.13: Conditional GANs architecture to transform a) low-resolution input images on the left to high-resolution sections on the right. The discriminator in b) is used to make sure these transformations are valid. Figure from Zhang et al. (2019).

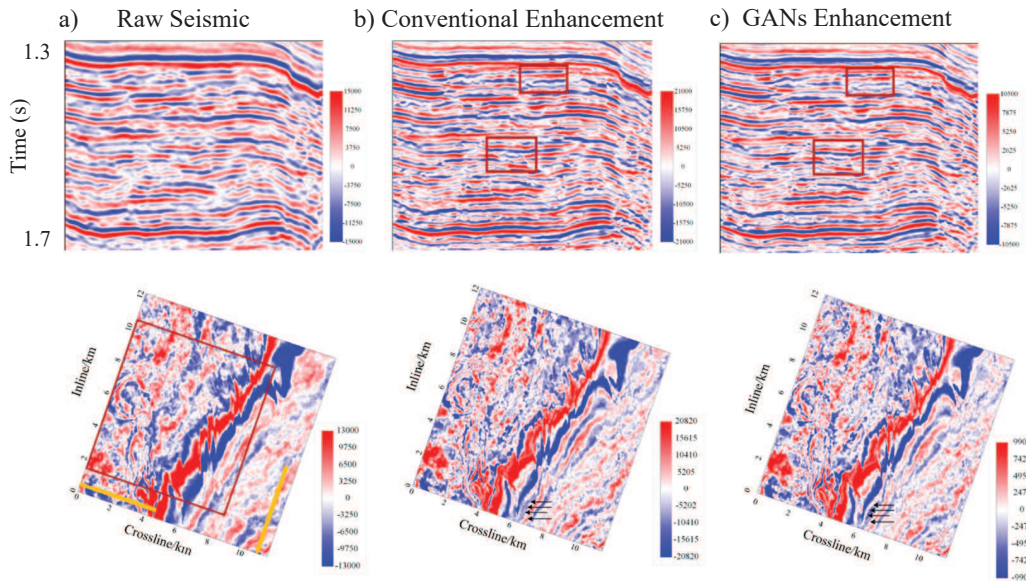


Figure 18.14: Top row contains vertical seismic sections' from the original data and bottom row contains depth slices a) before resolution enhancement, b) after conventional resolution enhancement, and c) GANs enhancement. Figure from Zhang et al. (2019).

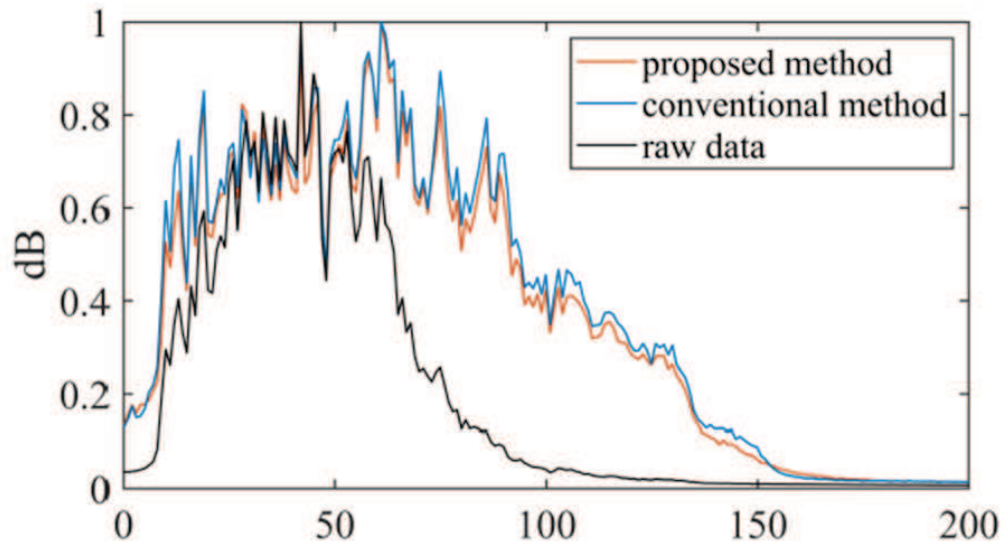


Figure 18.15: Frequency spectra associated with traces from the (black) raw seismic section, (blue) processed by a conventional resolution enhancement procedure, and c) (orange) processed by GANs. Figure from Zhang et al. (2019).

a decoder. The discriminator is rewarded for correct classifications and the generator is rewarded for generating examples that fooled the discriminator. The next cycle of the game, denoted as alternating SGD, repeats until believable fake images are created.

The computer science community is largely responsible for the development of GANs and the geoscience community is testing its effectiveness in solving geoscience problems. The biggest problem with GANs is robust convergence in trying to solve an adversarial optimization problem, where the objective functions of the generator and discriminator are fighting one another to achieve a stationary solution. Some successful remedies that mitigate the convergence problem are DCGAN, conditional GANs, and adding regularization terms such as proximal operator to the Value function. The result is that there are many applications of GANs in computer vision (Brownlee, 2019b) such as generating photographs of human faces, background scenes and objects. There are also GANs for image-to-image translation of, e.g., day scenes to night scenes for training AI driving systems, text-to-image translation, interpolation and editing of incomplete images, poor-to-high resolution of photographs, aging faces of young people, and 2D-to-3D object generation. Successful examples in geoscience include transforming seismic sections into attribute sections, interpolation of seismic data, and the calculation of high-resolution seismic images.

The advantages of a GAN over standard supervised training networks and generative modelers are the following (Alqahtani et al., 2021).

- No need for standard data augmentation methods such as rotation, scaling or stretching. The GANs images are, ideally, seamless blends of those in the training set.
- There is no need to define the shape of the probability distribution for the generator. This shape gradually emerges after a number of iterations to adopt the shape of the training set. Also, the computation of GANs can easily be adapted to parallel computers because it produces *data all at once, not pixel by pixel as autoregressive models*. (Alqahtani et al., 2021).
- GANs does not need to approximate the likelihood or requires assumptions about the prob-

ability distribution. Instead, GAN plays an adversarial game between the generator and the discriminator, and aims to find fake models with the same probability distribution as seen in the training data.

- GANs produce images with much sharper detail than VAEs. The generator tries to fool the discriminator to recover the real data distribution, and this includes generating high-frequency parts to deceive the discriminator.

The disadvantages of GANs include the following.

- Training of GANs are typically not very robust (Hitawala, 2018; Hussain, 2020). Decreasing the discriminator's cost function can cause an increase of the generator's cost function and vice versa. Thus, iterative solutions of a vanilla GAN can often fail to converge to a stable solution unless proper regularization is implemented.
- Mode collapse prevents the GANs from generating believable images. It is characterized by the output of the generator collapsing to a single oscillating mode as the iterations proceed. Radford et al. (2015) partly mitigated this mode collapse problem with DCGAN, which includes batch normalization. Other partial remedies include a new objective function or adding different components to the GAN model (Che et al., 2016; Ghosh et al., 2018; Farnia and Ozdaglar, 2020). Arjovsky et al. (2017) proposed using a Wasserstein distance metric rather than a KL-based divergence or energy-based one and multiple GANs were suggested to cover the different modes of the probability distributions (Tolstikhin et al., 2017). However, mode collapse still remains a major problem and is a current topic of research.
- The optimizer of GANs can lie on a saddle point and 1st-order gradient methods cannot converge to such points (Lee et al., 2016).

## 18.7 Exercises

1. The geophysicist often faces the problem of finding a model that minimizes both a least-squares data misfit function  $\epsilon^{data}$  and a model penalty function  $\epsilon^{penalty}$  based on a priori information about the surrounding geology, such as constraints from a well log. In this case the goal is to find a model that minimizes the hybrid cost function

$$\epsilon = \epsilon^{data} + \lambda \epsilon^{penalty}, \quad (18.12)$$

where the scalar  $\lambda > 0$  determines the degree of importance in honoring the model penalty term. The a priori model is somewhat different than the actual model, so if  $\lambda$  is too large then there will be too large of a  $L^2$  data misfit error in order to reduce the value of the penalty term. Thus, there is an adversarial relationship between these two cost functions. Derive a least squares cost function similar to equation 18.12 for the problem solved by a GAN, where one of the cost functions seeks to minimize the error in generating an authentic version of the true image  $\mathbf{x}$ . Discuss how the optimal value of  $\lambda$  can be determined. Outline the benefits and liabilities of solving the GAN problem by an iterative least squares method compared to the minimax method.

2. Same as previous question except use a  $L^1$  data misfit rather than a  $L^2$  data misfit function.
3. For joint inversion, the misfit function is a sum of two different data misfit functions, such as the joint inversion of gravity and magnetic data in Astic and Oldenburg (2018) or seismic and electromagnetic data in Columbo and Rovetta (2018). To encourage structural consistency between the two different models, such as the seismic velocity model and the electromagnetic parameter model, Columbo and Rovetta (2018) added to the data misfit functions two types

of structural regularization terms; one of them is the standard cross-gradient and the other is a summative gradient. These regularization terms rewarded seismic and electromagnetic models with similar spatial gradients. Devise a GAN objective function with a structural regularization term that rewards fake images with similar structural features as the actual images. Discuss how to optimize the value of the weight for the structural regularization term.

4. Test the effectiveness of a GAN on the MNIST data set using Colab at <https://colab.research.google.com/github/pytorch/xla/blob/master/contrib/colab/DC-GAN.ipynb>. Add the structural regularization term to the objective function and test different strategies for finding the optimal value of the structural weight.
5. The simple cross-entropy loss function for supervised learning is given by

$$\epsilon = y \ln g(\mathbf{x}) + (1 - y) \ln(1 - g(\mathbf{x})), \quad (18.13)$$

where  $g(\mathbf{x})$  is a sigmoid function and  $y \in \{0, 1\}$ . Equation 18.13 appears to be similar to a simplified version

$$\epsilon = -\ln D(\mathbf{x}) + \ln[1 - D(G(\mathbf{z}))], \quad (18.14)$$

of the GAN objective function in equation 18.1. However, equation 18.13 does not require a minimax solution appropriate for adversarial training. Explain why this is true.

6. Analytically show that the stationary point  $y^*$  of  $f(y)$  in equation 18.9 is  $y^* = a/(a + b)$ .
7. From Goodfellow et al. (2014), prove

$$D_G^*(\mathbf{x}) = \frac{P_{data}(\mathbf{x})}{P_{data}(\mathbf{x}) + P_G(\mathbf{x})}. \quad (18.15)$$

Show how this is equivalent to constraining the probability distribution of the final image to be the same as that of the data. This idea is illustrated in Figure 18.4.

8. Determine the Hessian matrix  $H$  associated with  $f(x, y) = xy$ . What are the eigenvalues of  $H$  at the stationary point  $(0, 0)$ . Is this a positive definite matrix?
9. Use one of the gradient descent algorithms in Chapter 3 to find the stationary point for  $f(x, y) = xy$ . Then use the proximal regularized Value function

$$V(x, y) = \max_{\tilde{y} \in y} V(x, y) - \|\tilde{y} - y\|^2 \quad (18.16)$$

in Farnia and Ozdaglar (2020) to test for improved convergence. Comment on the effectiveness of regularization in finding the stationary point.

10. Repeat the above exercise except now compare the convergence of Wasserstein GAN, f-GAN and the second-order Wasserstein GAN (see Farnia and Ozdaglar, 2020) in finding the Nash equilibrium point for  $f(x, y) = xy$ .
11. Write a DCGAN code in MATLAB to generate  $4 \times 4$  images of faults. Compare the performance with and without batch normalization.

## 18.8 Appendix: GAN Examples

Some applications of GANs are listed below.



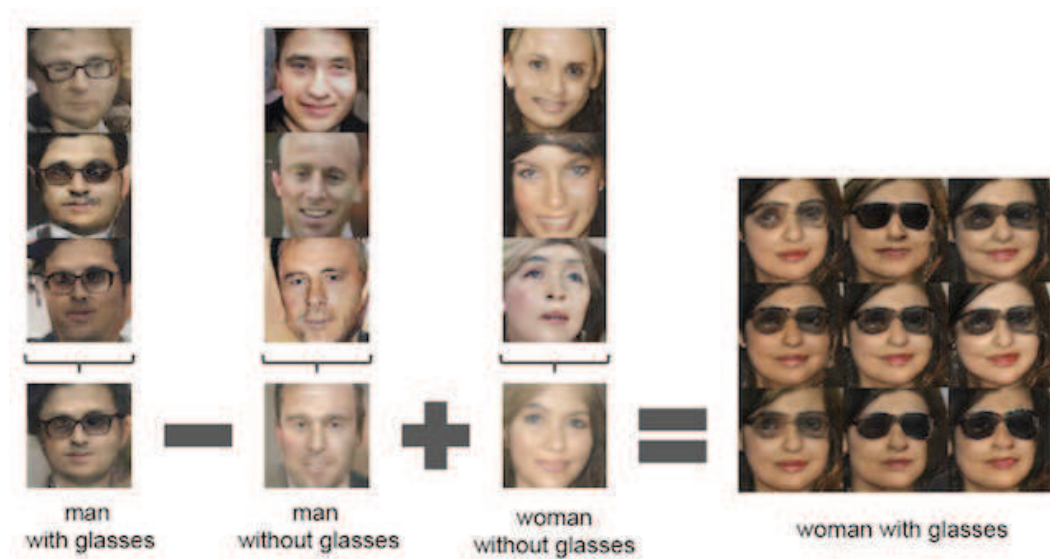


Figure 18.16: Columns of faces where the leftmost column contains images of males with glasses and the third column is of females without glasses. Training by a GANs can extract the glasses and insert them onto the images of females. Figure from Radford et al. (2015).

1. **Data Augmentation GANs.** One of the problems in training a CNN is that there is sometimes not enough examples to train on, and each example must be carefully labeled. A traditional data augmentation technique takes images from the original training set and distorts them in various ways, such as stretching, scaling and/or rotation, to create new training images. This type of augmentation is not fully effective because it does not account for the entire universe of possible input images and the distorted images are still strongly characterized by identifiable parts from the original images. In contrast, GANs blend features from many different images to give unique images that appear to be real. In fact, it is often difficult for a human observer to be able to pinpoint which training images contributed to the GANs image.

Unfortunately, early versions of GANs suffered from stability and convergence problems. To partly mitigate these problems, Radford et al. (2015) replaced the multiperceptron architecture with strided convolutional layers for the discriminator and fractional striding for the generator. They also introduced batch normalization in the hidden layers and used ReLU activations in the generator and LeakyReLU functions in the discriminator. Radford et al. (2015) also developed an image arithmetic so that features from different images can be combined to give an output image with the targeted features. Figure 18.16 shows results of a DCGAN where the common feature of glasses in the leftmost column of male images can be added to the 3rd column of glassless females to get the rightmost matrix of female images with glasses. The architecture of a DCGAN model is illustrated in Figure 18.17.

In a related work, Makhzani et al. (2015) proposed an Adversarial Autoencoder (AAE) that consists of two networks, one of them is a traditional autoencoder and the other one is an autoencoder that matches the posterior distribution of the latent representation to an arbitrary prior distribution. Chapter 14 provides more details on the AAE.

Denton et al. (2015) proposed a sequence of coarse-to-fine convolutional networks within a Laplacian pyramid framework. This approach, denoted as Laplacian GAN (LAPGAN)



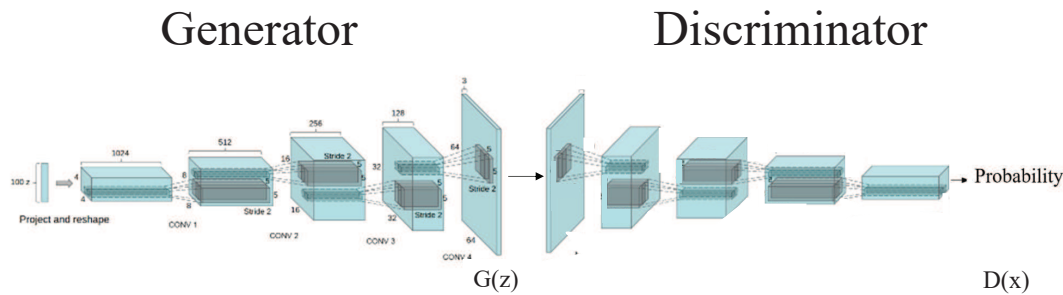


Figure 18.17: DCGAN architectures for the generator and discriminator models. Figure partly adapted from Radford et al. (2015).

in Figure 18.2, allowed them to exploit the multiscale structure of natural images, building a series of generative models with their characteristics listed in Figure 18.2. Each of the generative models in the Laplacian pyramid captures the image structure at a particular level.

2. **Image-to-Image-Translation and Conditional GANs.** Isola et al. (2018) developed an image-to-image translation using a conditional GANs (CGANs). Goodfellow (2017) states *Using labels in any way, shape or form almost always results in a dramatic improvement in the subjective quality of the samples generated by the CGAN model.*

Some CGAN applications include translation of aerial photos into Google maps, transforming skeletonized street scenes with cars into realistic photos (Reed et al., 2018), and translating hand-drawn sketches into realistic photographs of objects. The CGANs conditions the generator and discriminator models with extra information such as a vector  $\mathbf{c}$  of class labels of the input, so CGANs can be used as a supervised learning algorithm, or as a semi-supervised learning method (Springenberg, 2016) if only some of the training examples are labeled with different classes. The input to the conditioned discriminator takes both the reference image vectors  $\mathbf{x}$  as well as extra information vectors  $\mathbf{c}$  to identify true examples for each class. The conditioned generator takes in both the latent vector  $\mathbf{z}$  and an information vector  $\mathbf{c}$  as well.

3. **Text-to-Image-Translation.** Zhang, et al. (2016) and Reed et al. (2016) developed a GANs that translates text into scenes described by the text.
4. **Super-resolution Images.** Ledig et al. (2017) present a GAN that upscales input images by a factor of 4. In their super-resolution GANs (SRGANs), they use a perceptual loss function which consists of an adversarial loss and a content loss. The discriminator network is trained to differentiate between the super-resolved images and original photo-realistic images. The content loss emphasizes perceptual similarity instead of similarity in pixel space.

Karras et al. (2017) also introduced a multiscale procedure where the starting images were of low resolution. Starting from a low-resolution model and input images, new layers with higher-resolution capabilities are added to the model as training progresses. This increases the convergence speed in transforming a low-resolution image into a super-resolution image.

There are many applications for SRGANs, but an example developed by Bin et al. (2017) created photos of human faces. This procedure is used in a multi-scale workflow (Wang et al., 2018b) where lower resolution images are upgraded to slightly high-resolution ones, which could be used as the starting image for the next higher resolution image.

5. **GANs Interpolation.** Interpolating images corrupted by missing data or information is also denoted as inpainting. Pathak et al. (2016) filled in missing parts of photographs by a GANs

procedure. This type of interpolation can be used for interpolating geophysical data recorded by a sparse distribution of sensors.

6. **Other Variants of GANs.** Incorporating autoencoders into the GANs model tends to stabilize the iterative solution and prevent mode collapse (Alqahtani et al., 2021). In the standard GANs, the starting point for the generator is to feed a latent space vector with a small dimension into the generator. Thus, the GAN sometimes finds it difficult to find a network model that maps the output images with a much higher dimension  $\mathbf{x}$  back into  $\mathbf{z}$ . However, an autoencoder can effectively learn the model coefficients for both decoder  $p(\mathbf{z}|\mathbf{x})$  and encoder  $p(\mathbf{x}|\mathbf{z})$  operations, and so combining it with a GAN might mitigate the problem of mode collapse. Donahue et al. (2016) and Dumoulin et al. (2016) both introduce autoencoder-like models into the GAN architecture to develop, respectively, the Bidirectional Generative Adversarial and Adversarially Learned Inference Networks. New GANs architectures and applications are still being rapidly developed, so Figure 18.2 (Hitawala, 2018) is only a partial listing of them.

Part V

Bayesian Analysis



## Chapter 19

# Sampling a Probability Distribution

The main motivation for developing efficient sampling strategies of a probability distribution is that Bayesian analysis of a problem often requires plugging in different models into the posterior probability distribution  $p(\mathbf{m}|\mathbf{d}) \propto p(\mathbf{d}|\mathbf{m})p(\mathbf{m})$  (see Chapters 28 and 20). If enough samples  $\mathbf{m}$  are used then the contours associated with  $p(\mathbf{m}|\mathbf{d})$  can be used to estimate the optimal model  $\mathbf{m}^*$  that maximizes  $p(\mathbf{m}|\mathbf{d})$  for a given  $\mathbf{d}$ . These contours also provide the associated mean and variance of  $\mathbf{m}^*$ . For multidimensional models, the sampling is prohibitively expensive unless an efficient sampling strategy is employed.

A probability distribution is given in either analytic, e.g. Gaussian density function, or numerical form. The goal is to efficiently draw samples from this distribution such that the frequency of drawn samples around some point honors the occurrence probability from the given distribution. For example, flipping an unfair coin, i.e.  $p(heads) = 0.7$  and  $p(tails) = 0.3$ , and sampling from the probability distribution will tend to give the correct probability distribution as the number of coin flips increases. This is illustrated in the Figure 19.1 example where the normalized histogram of coin flips tends to the correct probability with an increase in the number of trials. In this case, a uniform distribution  $U(0,1)$  is approximated by a random number generator in MATLAB, and any number  $x \sim U(0,1)$  drawn between 0.0 and 0.7 is assigned a heads, otherwise it is a tails. In this case, 20 random numbers are sequentially generated to give Figure 19.1a, which is an unacceptable accuracy in drawing numbers that approximate the actual  $p(x)$ . If the experiment is rolling an unfair dice, then the values of  $x$  are assigned to one of 6 unique intervals between 0 and 1, with the length of each non-overlapping interval equal to the probability  $p(x)$ . For more complicated multivariate PDFs, this sampling strategy cannot be easily employed so we now present the more sophisticated sampling methods that are used to sample multivariate probability distributions.

### 19.1 Sampling a Probability Distribution

Sampling a probability density function  $f_X(x)$  is defined as drawing independent samples  $x$  from a given probability distribution  $f_X$ , i.e.  $x \sim \mathcal{N}(\mu, \sigma)$  for a Gaussian distribution. For example, the inverse transform (Chapter 1 in Gillespie, 1992) specifies the probability value  $f_X(x) = 2/9$  at the red star in Figure 19.2a and estimates the value  $x = 1$  at the green star.

Sampling is needed when the geoscientist is given, for example, the PDF of measured rock parameters  $\mathbf{m} = (\phi, \gamma)$  for, as an example, the porosity  $\phi$  and  $\gamma$  values of a rock type in Figure 20.15. This PDF is then used to design statistically appropriate rock models so that synthetic geophysical data  $\mathbf{d} = G(\mathbf{m})$  can be forward modeled to compute realistic seismic data. To insure accurate

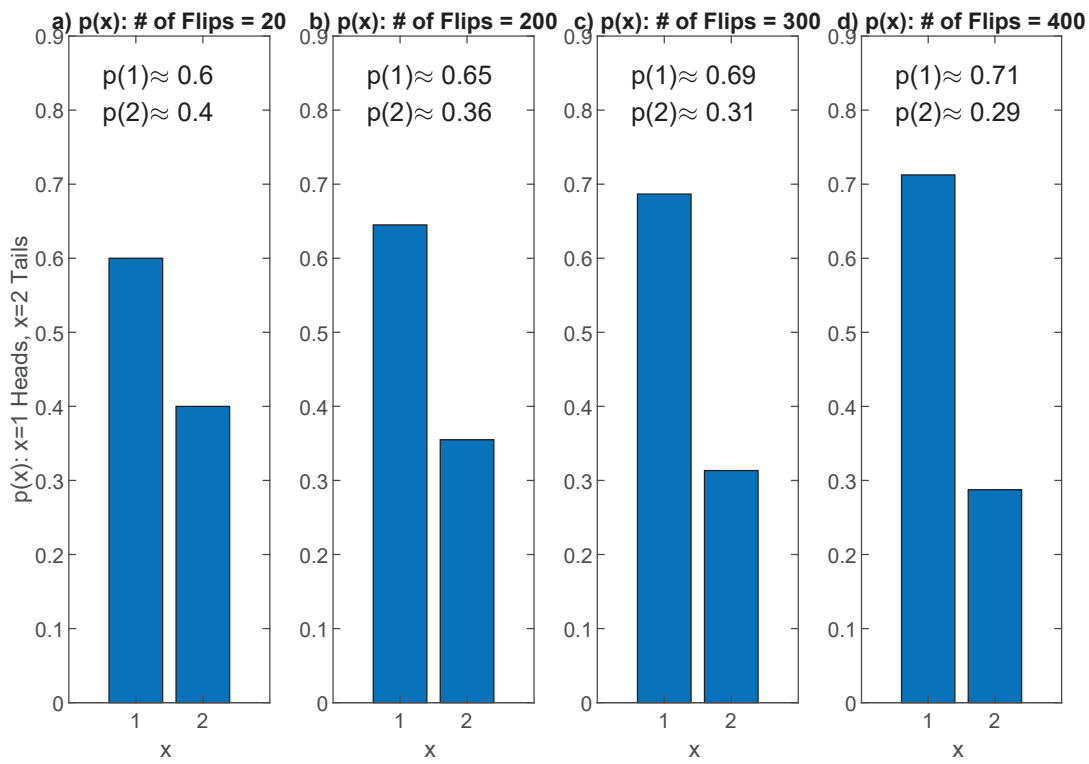


Figure 19.1: Normalized histograms for the results of flipping an unfair coin a) 20, b) 200, c) 300 and d) 400 times. The actual PDF of the coin flip is  $p(x = 1) = 0.7$  for heads and  $p(2) = 0.3$  for tails.

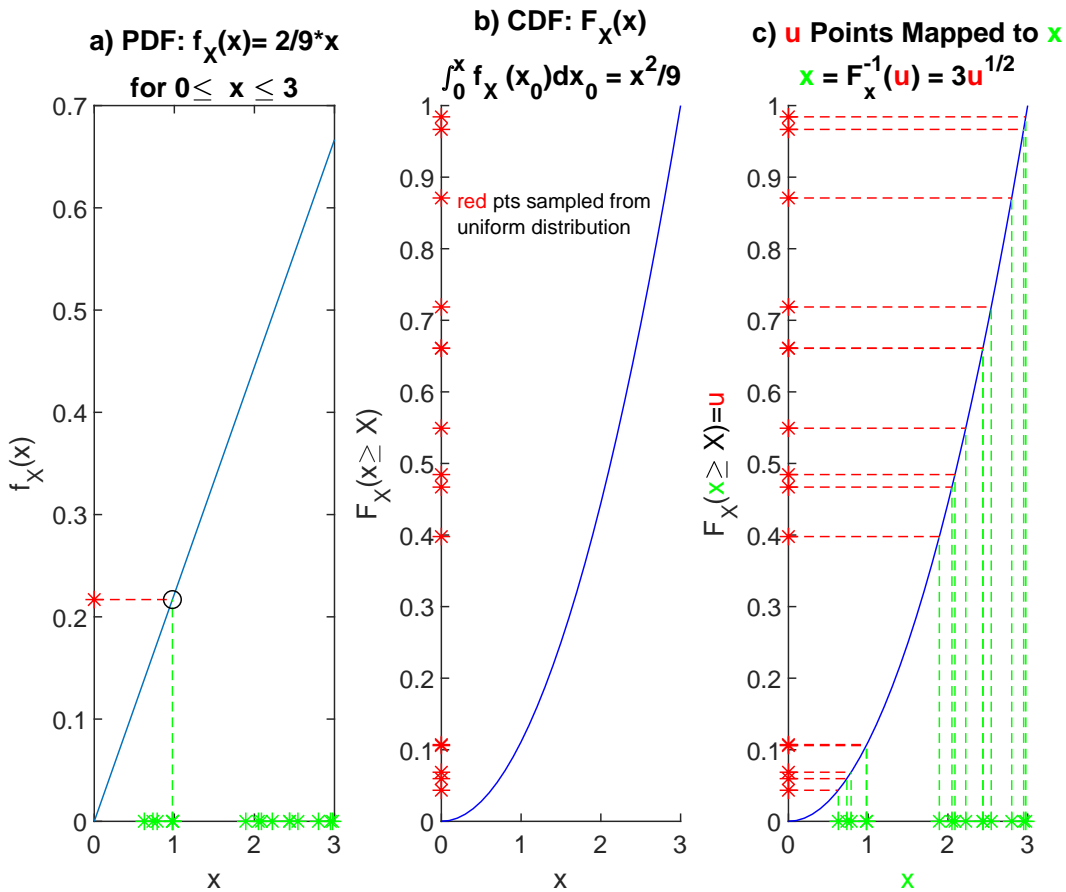


Figure 19.2: Sampling a PDF by an inverse transform. a) PDF  $f_X(x)$  with the most likely sampled green points associated with the highest probability and b) cumulative probability distribution with a 1-1 mapping between points on vertical axis mapped to points on horizontal axis. c) Inverse sampling strategy where points  $u$  uniformly sampled along vertical axis are mapped by  $F_X(u)^{-1}$  to points  $x$  along the horizontal axis.

modeling, the distribution of the model parameters  $\mathbf{m} = (\phi, \gamma)$  must be consistent with the PDF of the recorded rock data. Therefore, sampling a PDF to generate a cloud of points  $\mathbf{m}$  is the reverse process of GMM, which generates weighted combinations of Gaussians that fit a cloud of points. Sampling is also needed for estimating the parameters that maximize the posterior PDF in Bayes' formula. Proper and efficient sampling of the parameters also provide an accurate estimate of the posterior PDF.

Two simple sampling strategies for a PDF  $f_X(x)$  are the inverse transform and the acceptance-rejection sampling methods (Chapter 1 in Gillespie, 1992). In practice, a combination of the two methods can be used for realistic PDFs. Later we will introduce the Markov chain methods that are more efficient for sampling high-dimensional distributions.

## 19.2 Sampling a PDF by the Inverse-Transform

The inverse transform can be used when a simple PDF is defined so that an analytic cumulative probability distribution (CDF) can be derived. Figure 19.2a depicts a simple linear PDF  $f_X(x) = 2x/9$  and its CDF

$$\begin{aligned} F_X(x \geq X) &= \int_0^x f_X(x_0) dx_0, \\ &= \int_0^x \frac{2x_0}{9} dx_0, \\ &= x^2/9. \end{aligned} \tag{19.1}$$

is shown in Figure 19.2b. Defining  $u = F_X(x \geq X) = x^2/9$  allows for the inverse mapping to be  $x = 3\sqrt{u}$ , where non-negative values of  $x$  are inadmissible. This inverse mapping allows a uniform sampling  $u \sim U(0, 1)$  of red points along the vertical  $u$  axis in Figure 19.2b to have a 1-1 mapping<sup>1</sup> to the CDF curve in Figure 19.2c. The intersection of the red dashed lines with the blue CDF curve can be projected downward along the green dashed lines to give the sampled green points along  $x$  in Figure 19.2c.

As expected, the greatest densities of green points in Figure 19.2a and Figure 19.2c are associated with the highest values of the probability density function  $f_X(x)$ . In general, if the CDF of  $f_X(x)$  has an analytic inverse then  $f_X(x)$  can be sampled in this way so that the distribution of sampled points is consistent with the PDF.

## 19.3 Sampling a PDF by Acceptance-Rejection

Many PDFs can be too complicated to easily get analytic inverses of their CDFs. For these cases, the adaptive acceptance-rejection (AR) method can be used. The AR method is described in Figure 19.3 for the 1D PDF  $f_X(x)$ , where there are two steps.

1. Define the red box in Figure 19.3a that encloses the blue PDF denoted as  $f_X(x)$ . This PDF does not need to be normalized.
2. Randomly  $N$  assign points  $(x_i, z_i)$  in the red box using a uniform distribution  $U(0, 1)$  for both the  $x$  and  $z$  coordinates. Only accept the points that are at or fall below the blue PDF curve, as illustrated by the yellow points in Figure 19.3b. Note that the highest density of accepted points are the ones with the highest probability.

---

<sup>1</sup>Points along the vertical axis do not have a 1-1 mapping to a Gaussian PDF curve, but they do have a 1-1 mapping to its CDF.



## Sample a Probability Distribution by Acceptance-Rejection

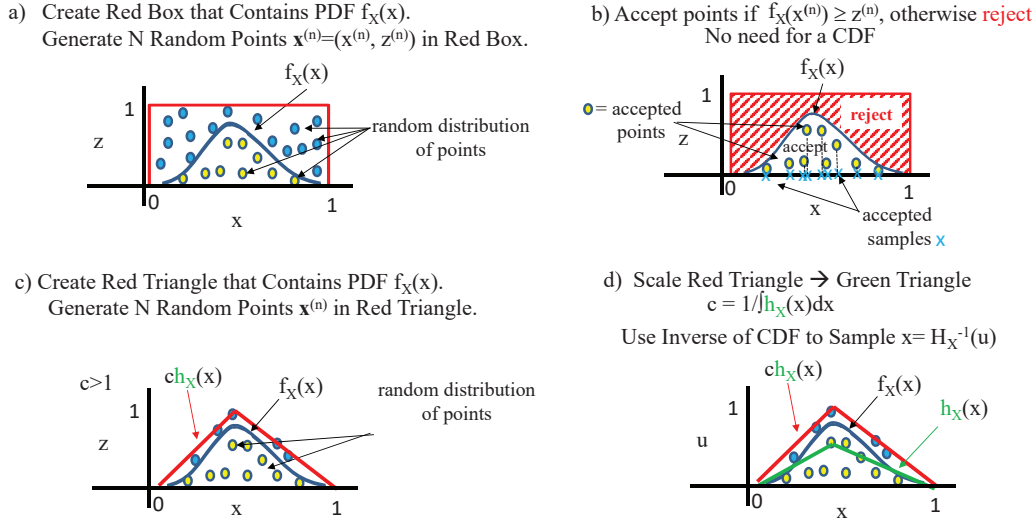


Figure 19.3: Acceptance-rejection method. a) shows the red box that contains the Gaussian PDF and a random distribution of blue and yellow points. b) depicts the accepted yellow points  $(x_i, z_i)$  that are at or fall below the PDF curve  $f_X(x_i) \geq z_i$ . The blue crosses in b) represent the horizontal components of the yellow accepted points. c) depicts the red triangle where many fewer random points are enclosed, and d) depicts the scaled green triangle which integrates to 1. See Jan Dufek's Youtube at <https://youtu.be/p9EHmdtaBBA> viewed 11/01/2021.

For example, choose a point  $x_i$  from a uniform distribution  $U(0, 1)$ , say  $x_i = 0.7$ . Then roll the dice again; that is, select a number from a uniform distribution and assign it to  $z_i$ , say  $z_i = 0.2$ . Accept  $\mathbf{x}_i = (x_i = 0.7, z_i = 0.2)$  if  $f_X(x_i) \geq z_i = 0.2$ , otherwise reject  $\mathbf{x}_i$ .

In summary,  $N$  numbers are generated along the horizontal  $x_i \sim U(0, 1)$ ;  $i \in (1, 2, \dots, N)$  axes from  $U(0, 1)$ . For each  $x_i$  another number  $z_i$  is drawn from  $U(0, 1)$  and the accepted values for  $x_i$  satisfy the following conditions for  $i \in (1, 2, \dots, N)$ :

$$x_i = \begin{cases} \text{if } z_i \leq f_X(x_i) & \text{accept} \\ \text{if } z_i > f_X(x_i) & \text{reject} \end{cases}. \quad (19.2)$$

The disadvantage of this method is that there is a large area of rejection (denoted by the slanted red lines in Figure 19.3b) above the low-probability part of  $f_X(x)$ . This means that a large number of random points is required in order to sufficiently sample a number of points  $x$  with low probability. In high dimensions it can be computationally too costly to evaluate  $f_X(\mathbf{x})$  for a large number of reject points. If  $N$  is the number of random points in the 2D plane for acceptable sampling, then  $O(N^D)$  random vectors must be generated and evaluated with  $f_X(\mathbf{x})$  for points  $\mathbf{x}$  in an  $N$ -dimensional space. This cost of cursed dimensionality is computationally prohibitive for sampling high-dimensional PDFs.

To reduce this cost, the size of the rejection zone in Figure 19.3a can be *adaptively* reduced by enclosing the PDF within the red triangle function  $ch(x)$  in Figure 19.3c, where  $c \geq 1$  is a constant.

The triangle function  $ch(x)$  is not a PDF because it has an area greater than 1. To force it to have a unit area, we divide  $ch(x)$  by  $c$  to give the scaled PDF  $h(x)$  illustrated by the green triangle in Figure 19.3d where  $c$  is defined as

$$c = \frac{1}{\int_{-\infty}^{\infty} h(x) dx} = 1. \quad (19.3)$$

The CDF  $H_X(x)$  of  $h_X(x)$  and its inverse  $H_X(x)^{-1}$  can now be analytically generated by the inverse transform method because each line segment of the green triangle  $h_X(x)$  has an invertible CDF (see Figure 19.2). More complicated PDFs can be enclosed by adaptively linking together a series of piecewise lines to enclose  $f_X(x)$ .

The uniformly generated points  $z_i$  along the vertical axis, starting at  $z = 0$  and ending at the apex of the green triangle, are mapped by  $H_X(z_i)^{-1}$  to the corresponding points  $x_i$  along the horizontal axis. These points  $(x_i, z_i)$  are then used to compare the scaled version  $cz_i$  to  $f_X(x_i)$  for either acceptance or rejection, as summarized below.

1. Define the green triangle  $ch_X(x)$  that closely encloses the PDF  $f_X(x)$  s.t.

$$f_X(x) \leq ch_X(x) \quad \forall x, \quad (19.4)$$

where  $c \geq 1$  and  $c = \sup_x [f_X(x)/h_X(x)]$ .

2. Divide  $ch_X(x)$  by  $c$  to get the green triangle function  $h_X(x)$  in Figure 19.3d.
3. Calculate the analytic CDF  $H_X(x)$  and its inverse transform

$$H_X(\overbrace{H_X(x)}^z)^{-1} = x. \quad (19.5)$$

4. Uniformly sample  $N$  points along the vertical axis to get  $z_i \sim U(0, z_{upper})$  for  $i \in \{1, 2, \dots, N\}$ . Here,  $z_{upper}$  is the height of the green triangle. These  $z_i$  values are used to get the  $x_i = H_X(z_i)^{-1}$  samples along the horizontal axis.
5. At each  $x_i$ , we use the acceptance criterion

$$z_i ch(x_i) \leq f_X(x_i), \quad (19.6)$$

otherwise we reject  $x_i$ . For high-dimensional points, knowledge of the green triangle's boundary  $ch(x_i)$  and  $z_i$  avoids having to expensively evaluate  $f_X(x_i)$  for a large number of rejectable points  $x_i$ .

An example of the acceptance-rejection method is shown in Figure 19.4, where the PDF is the Gaussian function in Figure 19.4a, and the green points  $(x, z)$  in Figure 19.4b are randomly obtained from a 2D uniform distribution. The points which satisfy the acceptance condition  $f_X(x) \geq z$  are displayed in Figure 19.4c and the associated histogram is in Figure 19.4d with the scaled Gaussian function. The red points along the horizontal axes in a) and d) are the horizontal coordinates of the accepted points.

The efficiency of the acceptance-rejection method is evaluated by the formula

$$\frac{\int f_X(x) dx}{\int ch(x) dx} = \frac{1}{c}, \quad (19.7)$$

which is the area of the original PDF divided by the area of triangular PDF  $ch(x)$ . Since  $c \geq 1$  then the optimal triangular PDF that is a close approximation to  $f_X(x)$  should have  $c \approx 1$ .

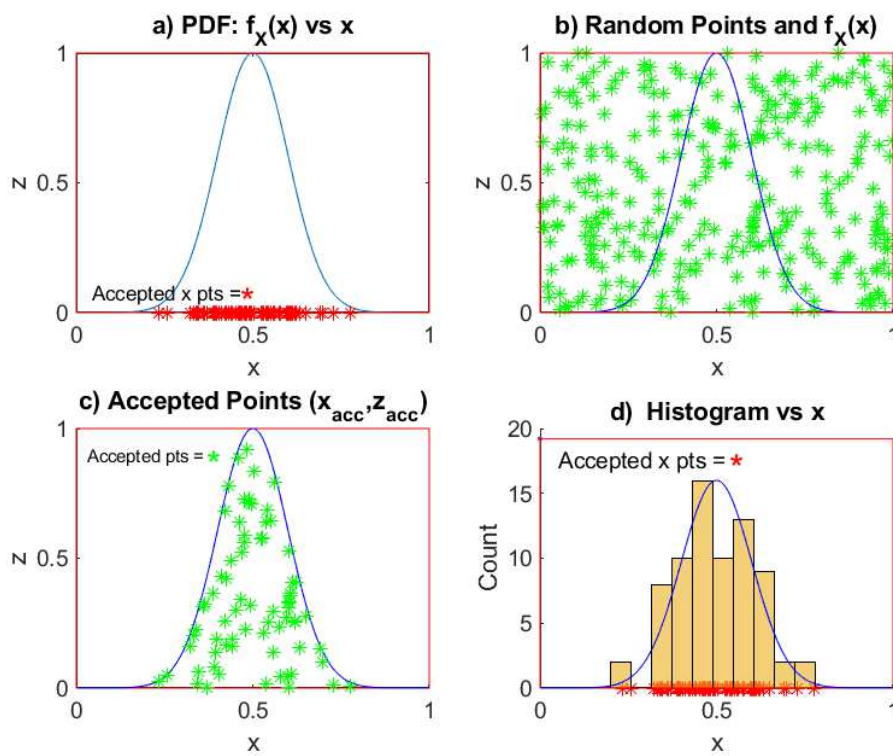


Figure 19.4: a) Gaussian PDF  $f_X(x)$  described by the blue curve, b) green points  $(x, z)$  taken from a 2D uniform distribution, c) accepted points  $(x_{acc}, z_{acc})$  from b) s.t.  $f_X(x) \geq z$ , and d) histogram of accepted points. The red points in a) and d) are the horizontal coordinates  $x_{acc}$  of the accepted points  $(x_{acc}, z_{acc})$  in c).

## 19.4 Markov Chain

A Markov chain is a stochastic model that describes a sequence of stochastic events where the probability of the event at time  $t + 1$  only depends on the state attained at the previous time  $t$  ([https://en.wikipedia.org/wiki/Markov\\_chain](https://en.wikipedia.org/wiki/Markov_chain)). For example, assume that the probability of tomorrow being sunny  $p(S_{t+1})$  depends on a weighted combination of probabilities for today's weather being sunny  $p(S_t)$  and rainy  $p(R_t)$ :

$$p(S|S)p(S_t) + p(S|R)p(R_t) = p(S_{t+1}), \quad (19.8)$$

where each increment of the index  $t$  represents a day. Similarly, the probability of it being rainy tomorrow is given as

$$p(R|S)p(S_t) + p(R|R)p(R_t) = p(R_{t+1}), \quad (19.9)$$

where the weights  $p(S|S)$ ,  $p(S|R)$ ,  $p(R|S)$ , and  $p(R|R)$  are conditional probabilities that are time invariant. The probabilities  $p(R_t)$  and  $p(S_t)$  are the fractions of the current day which are rainy and sunny, respectively. The RVs  $R$  (rainy) and  $S$  (sunny) are denoted as states of the system, and there are as many states as needed for the specified problem. Combining equations 19.8 and 19.9 into a system of equations gives

$$\overbrace{\begin{bmatrix} p(S|S) & p(S|R) \\ p(R|S) & p(R|R) \end{bmatrix}}^{\text{transition matrix}} \begin{Bmatrix} p(S_t) \\ p(R_t) \end{Bmatrix} = \begin{Bmatrix} p(S_{t+1}) \\ p(R_{t+1}) \end{Bmatrix}, \quad (19.10)$$

where the  $2 \times 2$  matrix is the time-invariant transition matrix.

Another example of a Markov chain is illustrated in Figure 19.5. The diagram in Figure 19.5a depicts the conditional probabilities of eating grapes (G), lettuce (L) and cheese (C) at  $t + 1$ , where the ovals indicate the states of the system. Today's state is indicated by the subscript  $t$  and tomorrow's state is indicated by the subscript  $t + 1$ . For example, if you only ate grapes today then this state is described by the  $3 \times 1$  vector  $\mathbf{x}_t = (1 \ 0 \ 0)$  in Figure 19.5b. The rays emanating from the grape oval in Figure 19.5a say that the conditional probabilities for eating grapes, lettuce and cheese the next day are  $p(G|G) = 0.1$ ,  $p(L|G) = 0.5$  and  $p(C|G) = 0.4$ , respectively. This string of conditional probabilities comprises the first column of the  $3 \times 3$  transition matrix in b). The middle column consists of the conditional probabilities for eating lettuce today, which is the state given by  $\mathbf{x}_t = (0 \ 1 \ 0)$ , and the last column consists of the conditional probabilities for the state  $\mathbf{x}_t = (0 \ 0 \ 1)$  of only eating cheese today. The vector on the right denotes the state  $\mathbf{x}_{t+1} = (0.1 \ 0.4 \ 0.5)$ , which suggests a more balanced diet the next day that favors cheese. This system of equations is more compactly written as  $\mathbf{A}\mathbf{x}_t = \mathbf{x}_{t+1}$ , where  $\mathbf{A}$  is the transition matrix.

The state that describes your menu choices  $n$  days from now is obtained by taking the product of  $n$  transition matrices  $\mathbf{A}^n \mathbf{x}_t = \mathbf{x}_{t+n}$ . These states are displayed in Figure 19.5c, where the states change along the first row of images, but stabilize along the second row. This is because the Markov chain, after a sufficiently long time, has reached a stationary state where the probability vector  $\mathbf{x}_t$  no longer changes at future times. That is,

$$\mathbf{A}\mathbf{x} = \mathbf{x}, \quad (19.11)$$

which is characteristic of the second row of images in Figure 19.5c where  $\mathbf{x} = (0.335 \ 0.3110 \ 0.3541)$ . Not all Markov chains have a stationary distribution, but stationarity is guaranteed to exist for Markov chains with the ergodic property (Gillespie, 1992).

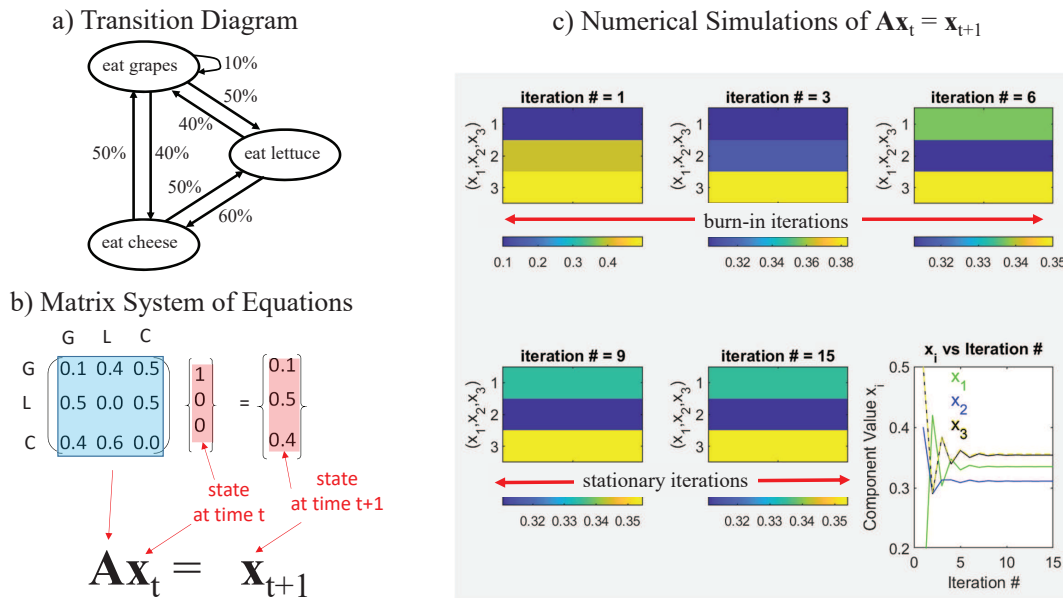


Figure 19.5: a) Diagram of probabilities (in percent) for tomorrow's food you will eat (tip of arrow) that depends on what you ate today (tail of arrow). b) System of equations that mathematically describe the transition from today's state  $\mathbf{x}_t$  to tomorrow's state  $\mathbf{x}_{t+1}$ . c) Images of  $\mathbf{x}_{t+n}$  for different numbers  $n$  of iterations for  $\mathbf{A}^n \mathbf{x}_t = \mathbf{x}_{t+n}$ . Samples from the burn-in iterations should be discarded because the probability distribution, e.g.  $x_1 = p(G_{t=3})$ , of the *grape* state is not that of the desired one, e.g.  $x_1 = p(G_{t=15})$ , computed during the stationary iterations. Here,  $x_1 \rightarrow p(G_{t=15})$  is the probability of eating grapes after the 15<sup>th</sup> iteration. Image in a) inspired by Ben Lambert's video at <https://www.youtube.com/watch?v=U561HGMWjcw> viewed in November, 2021.

## 19.5 Metropolis Markov Chain Monte Carlo Sampling

The computational cost of the standard acceptance-rejection method is  $O(N^D)$  evaluations of the multivariate PDF, where  $D$  is the dimension of  $\mathbf{x}$  and  $N$  is the number of uniformly sampled points along one axis. This is very inefficient because many evaluations of candidate samples are used to admit a small number of samples in low-probability regions. To reduce this cost, an iterative Markov chain Monte Carlo (MCMC) sampling method is used to sample the PDF (Gillespie, 1992; Bishop, 2006). Instead of generating high-dimensional clouds of samples with uniform density, MCMC efficiently generates high-density clouds of samples near high-probability regions and low-density clouds around low-probability regions. It iteratively selects candidate points for acceptance or rejection using a proposal transition distribution  $g(x'|x)$ . Here we can sometimes select  $g(x'|x) = p(x'|x)/C$  to be proportional to the original distribution  $p(x)$  where  $C$  is a sometimes difficult-to-compute<sup>2</sup> normalization factor.

- The MCMC algorithm consists of a Markov chain where the decision to move to the next sample  $x_{i+1}$  only depends on the previous sample  $x_i$ . This is a 1st-order Markov chain method, where higher-order MC methods use information from two or more preceding steps to compute the new sample (Shamshad et al., 1992).
- The MCMC is a Monte Carlo algorithm because it uses a random number generator to decide the next move. The simplest one is known as the random walk Monte Carlo method. The conditional proposal distribution can be symmetric such that  $g(x|x') = g(x'|x)$ , which leads to the Metropolis MCMC. If it is asymmetric then this is called the Hastings-Metropolis MCMC. These proposal distributions should have the same support as the target distribution  $p(x)$ .
- An effective transition probability is  $g(x'|x) \propto \exp(-\frac{1}{2}\|x - x'\|^2/\sigma^2)$ , so that samples closer to  $x$  are more likely to be visited next. A ratio of  $g(x|x')/g(x'|x)$  is the same as the ratio of the actual transition probabilities  $p(x|x')/p(x'|x)$  if the target and proposal distributions are proportional to one another.

The iterative MCMC algorithm generates samples  $x_i$  using the following steps illustrated in Figure 19.6.

1. Choose a starting point  $x_0 \sim \mathcal{N}(x, \sigma^2)$  and center the enclosing distribution  $\tilde{g}(x) = \mathcal{N}(x, \tilde{\sigma}^2)$  at  $x_0$ . Re-center the enclosing distribution at any new point  $x_i$  that is updated. Conveniently use  $g(x_{i+1}|x_i) \approx \exp(-\frac{1}{2}\|x_{i+1} - x_i\|^2/\sigma^2)$ .
2. Choose the next points  $x_i$  for  $i \in \{1, 2, 3, \dots\}$  with the following code:

$$\begin{aligned}
 & \text{for } i = 0 : \text{niter} \\
 & \quad \tilde{x}_{i+1} \sim g(x_{i+1}|x_i) \\
 & \quad u \sim U(0, 1) \\
 & \quad \text{if } u \leq A(x_i, \tilde{x}_{i+1}) = \min(1, \frac{p(\tilde{x}_{i+1})g(x_i|\tilde{x}_{i+1})}{p(x_i)g(\tilde{x}_{i+1}|x_i)}), \\
 & \quad \quad x_{i+1} = \tilde{x}_{i+1} \\
 & \quad \text{else} \\
 & \quad \quad x_{i+1} = x_i \\
 & \text{end}
 \end{aligned} \tag{19.12}$$

At each iteration, the algorithm picks a candidate point  $\tilde{x}_{i+1}$ , and if it is accepted according to  $u \leq A(x_i, \tilde{x}_{i+1})$  in equation 19.12 then  $x_{i+1} = \tilde{x}_{i+1}$ . The new starting point is  $x_{i+1}$  and the mean of

<sup>2</sup>Computing  $C$  for high-dimensional points  $\theta$  can be computationally prohibitive in practice.

## Metropolis Markov Chain Sampling

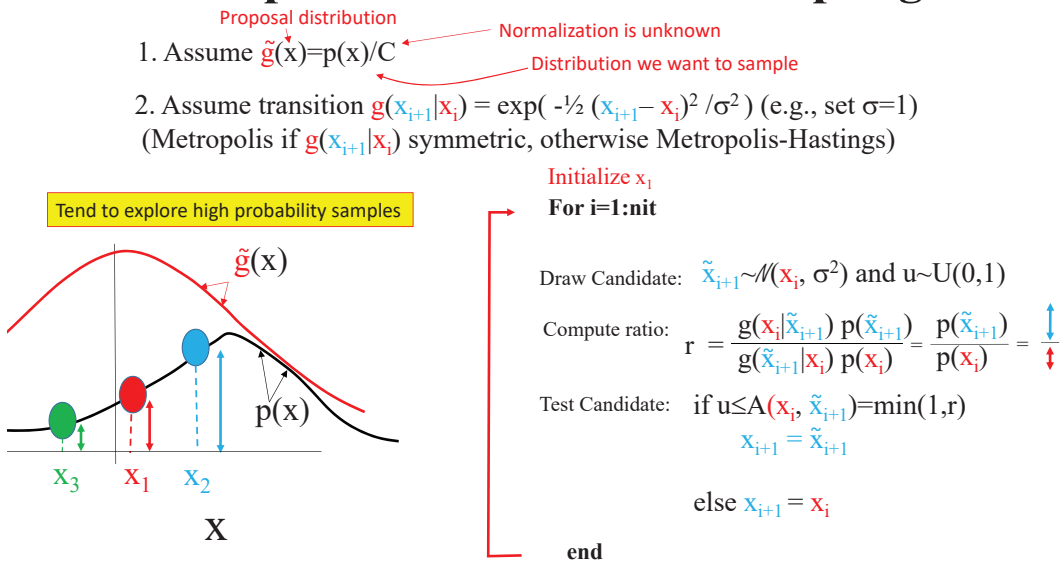


Figure 19.6: Metropolis workflow algorithm where samples  $x_i$  are obtained using the proposal distribution  $\tilde{g}(x) = \mathcal{N}(x, 1)$  centered at  $x$  where the proposed transition distribution  $g(x'|x_i) \approx \exp(-\frac{1}{2}\|x_i - x'\|^2 / \sigma^2)$  is symmetric. The Metropolis algorithm with wider Gaussians, i.e.  $\sigma \gg 1$ , searches far from the starting sample  $x_i$  to avoid correlated samples, but this can slow down convergence to a stationary state.

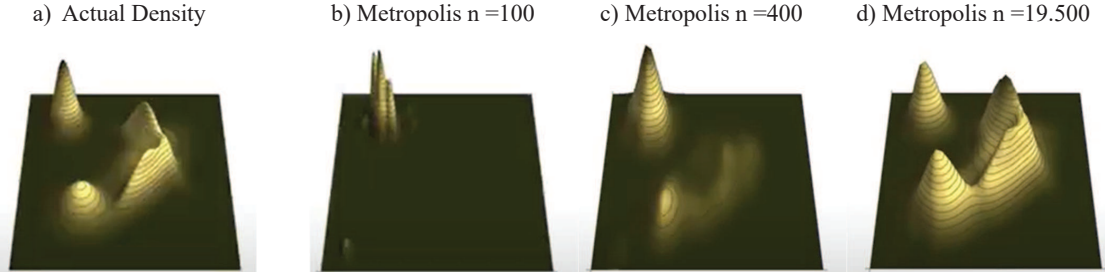


Figure 19.7: a) Actual density function  $f(x, y)$  and density function reconstructed from the MCMC samples after b) 100, c) 400 and d) 19,500 iterations. Images extracted from Ben Lambert's video at <https://www.youtube.com/watch?v=U561HGMWjcw> viewed in November, 2021.

the proposal Gaussian is shifted to  $x_{i+1}$  and the iterations proceed until convergence to a stationary state.

The Metropolis algorithm generates a sequence of sample values  $x_i$  in such a way that, as more and more sample values are produced, the distribution of values more closely approximates the target distribution  $p(x)$ . This is illustrated in Figure 19.7 where the target distribution is in a), and b)-e) show the distributions reconstructed from MCMC samples for different numbers of iterations. By the 19,500 iteration, the MCMC sampled points adequately reconstruct the original distribution. Unlike the Gibbs algorithm, the Metropolis algorithm can reject points and search along any diagonal direction that is not along a coordinate axis.

Note that the acceptance ratio  $A(x_i, \tilde{x}_{i+1})$  indicates the probability ratios of the new and old samples, according to the distribution  $p(x)$ . If we attempt to move to a point  $\tilde{x}_{i+1}$  that is more probable than that of the current point  $x_i$ , we will always accept the move. Otherwise  $x_{i+1} = x_i$ . Thus, we will tend to stay in (and return large numbers of samples from) high-density regions around  $p(x_i)$ , while only occasionally visiting low-density regions. Intuitively, this is why this algorithm works and returns samples that follow the desired distribution with density  $p(x)$ .

For example, if the variance  $\sigma$  of the proposal distribution is wide enough then this means it has a good chance to find a candidate sample far from the starting point where many correlated points can lurk. However, if  $\sigma$  is too large then the search for samples is inefficient and can leave gaps in the estimated histogram as illustrated in Figure 19.8c. If  $\sigma$  is too small then the sampling is inefficient as well, spending too many iterations around the starting point as illustrated in Figure 19.8a. But if  $\sigma$  is just right, then the histogram in Figure 19.8b provides an accurate estimate of the actual  $p(x)$ .

### 19.5.1 Detailed Balance Equation

The Metropolis algorithm seems intuitively reasonable in the sense that it prefers to sample regions where there is a high probability in the actual distribution, yet will allow for some sampling in low-probability regions. This is the parsimonious acceptance-rejection strategy of the Metropolis algorithm. However, we must ensure that the algorithm, with enough iterations, samples points with a probability that honors the target probability and becomes stationary.

The stationary character is embodied by the *detailed balance equation*:

$$\overbrace{p(x)g(x'|x)a(x'|x)}^{x \rightarrow x'} = \overbrace{p(x')g(x|x')a(x|x')}^{x' \rightarrow x}, \quad (19.13)$$

where  $a(x'|x)$  is the acceptance-rejection operator that decides if the proposed new state  $x'$  should



## Goldilocks Variance For Metropolis Algorithm

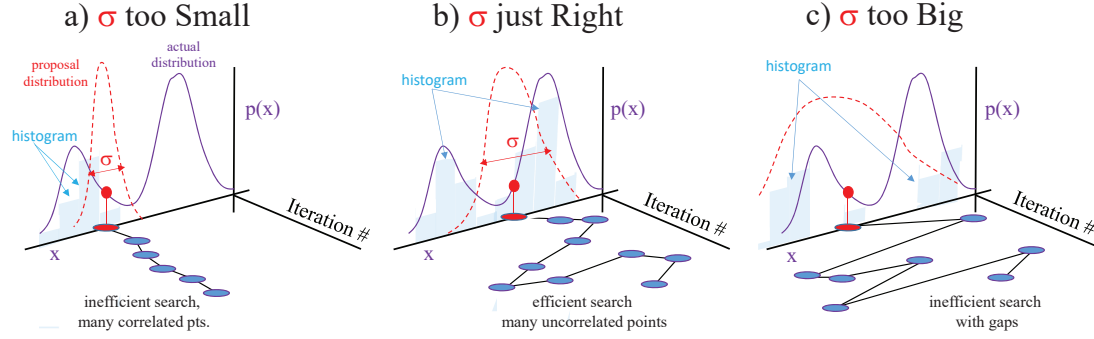


Figure 19.8: Transparent blue histograms estimated by Metropolis sampling when  $\sigma$  is a) too small, b) just right, and c) too big. Inspired by Alexander Novikov’s lecture “Bayesian Methods for Machine learning”.

be accepted or rejected. The above *detailed balance* equation (O’Hagan and Forster, 2004) says that the transition probability from  $x$  to  $x'$  is the same as the reverse route of  $x'$  to  $x$ . This is a necessary condition for a stationary Markov Chain system that does not change from iteration to iteration. Rearranging equation 19.13 gives the ratio of the  $a(x'|x)/a(x|x')$ :

$$\frac{a(x'|x)}{a(x|x')} = \frac{p(x')g(x|x')}{p(x)g(x'|x)} = A(x', x), \quad (19.14)$$

which is the acceptance-rejection ratio indicated in the Figure 19.6 workflow. If the proposal distribution is symmetric so that  $g(x'|x) = g(x|x')$  then the above equation reduces to

$$\frac{a(x'|x)}{a(x|x')} = \frac{p(x')}{p(x)} = A(x', x). \quad (19.15)$$

If the proposal distribution  $g(x'|x)$  is asymmetric then this defines the Hastings-Metropolis algorithm. Both the Metropolis and Hastings-Metropolis algorithms can converge to the target distribution  $p(x)$  (<https://similarweb.engineering/mcmc/>) if the detailed balance equation is satisfied.

### 19.5.2 Metropolis Sampling Example

The Metropolis sampling algorithm is applied to a unimodal Gaussian and a bimodal Gaussian. After 500 iterations, the results are shown in Figure 19.9a for a unimodal Gaussian and Figure 19.9b for a bimodal Gaussian. The estimated mean and standard deviation for the unimodal Gaussian in Figure 19.9a are in good agreement with the actual values of the target Gaussian colored in red. If there were an insufficient number of iterations, then the sampled points would sometimes be exclusively located under just one of the humps in Figure 19.9b. The MATLAB code for generating this figure is in the red Box in this section and the plotting functions are in Appendix 19.9.

The bimodal Gaussian is an example of a probability function associated with a non-linear misfit function, such as the traveltime tomography problem discussed in Chapter 20. The non-linear traveltime tomography problem is often characterized by local minima in the objective function, which is equivalent to multiple peaks in the target density function.

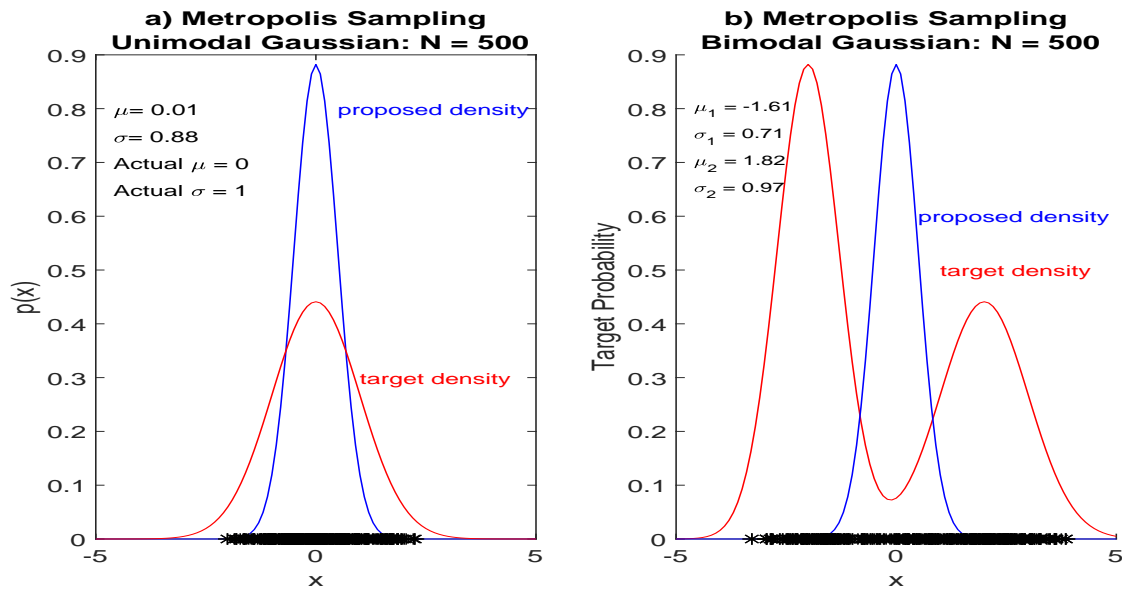


Figure 19.9: Metropolis sampling of a a) unimodal Gaussian and a b) bimodal Gaussian using  $N = 500$  iterations. The means and variances are computed for two groups of points: the points to the left of  $x = 0$  and those to the right.

#### Code 19.5.1. MATLAB 1D Metropolis Sampling

```
clear all; nit=500;
%Define stnd+means of Unimodal & Bimodal Gaussians
xhistory=zeros(nit,1); sigma=.5; signalike=sigma*2;
maxsig=max(sigma,signalike); mu=0; mu2=mu*2;

x=0.4; % Starting Sample
for i=1:nit %Unimodal Metropolis Sampling Algorithm
    u=rand(1,1);
    xp = drawxp(x,sigma);
    p = 1/sqrt(2*pi)/signalike*exp(-0.5*(mu2-x).^2/signalike^2);
    pp = 1/sqrt(2*pi)/sigma*exp(-0.5*(mu2-xp).^2/sigma^2);
    r = pp/p;
    A=min(1,r);
    if u<=A; x=xp; end
    xhistory(i)=x;
    rr(i)=r;
end

mu=0; %Define Midpoint for Range of samples x
x=[-5*maxsig-mu:.1*maxsig+mu]; %Define Sample Range of x
g=1/sqrt(2*pi)/sigma*exp(-0.5*(mu-x).^2/sigma^2); %Unimodal
g2=1/sqrt(2*pi)/signalike*exp(-0.5*(mu-x).^2/... %Bimodal
    signalike^2);

Title1 % Plot Samples & Proposed+Target Unimodal Densities

[p]=BimodalGauss(x,sigma,signalike); %Create Bimodal Density
subplot(122);
hold on; plot(x,p,'r-'); plot(x,g,'b-'); hold off
x=0; xp=x;
```

**Code 19.5.2.** *MATLAB 1D Metropolis Sampling (cont.)*

```

for i=1:nit %Bimodal Metropolis Sampling Algorithm
    u=rand(1,1);
    xp = drawxp(x,sigma);
    p = BimodalGauss(x,sigma,sigmalike);
    pp = BimodalGauss(xp,sigma,sigmalike);
    r = pp/p;
    A=min(1,r);
    if u<=A;x=xp;end
    xhistory(i)=x;
    rr(i)=r;
end

Title2 % Plot Samples, Proposed and Target Bimodal Densities

```

## 19.6 Gibbs Sampling

Gibbs sampling is a MCMC method for obtaining a sequence of points  $\mathbf{x}^n = (x_1^n, x_2^n, x_3^n)$  for  $n \in \{1, 2, \dots, N\}$  from a specified PDF that is a scaled approximation to the desired one  $\hat{p}(x_1, x_2, x_3)$ :

$$p(x_1, x_2, x_3) = \frac{\hat{p}(x_1, x_2, x_3)}{NC}, \quad (19.16)$$

where  $NC$  is the unknown normalizing constant. These samples can then be used, for example, to approximate those for a joint distribution or a marginal distribution in the parameter of interest.

The Gibbs sampling algorithm<sup>3</sup> is now given for the 3D multivariate joint distribution  $\hat{p}(x_1, x_2, x_3)$ , where we assume we know  $p(x_1, x_2, x_3)$  but do not know the normalizing constant  $NC$ .

1. At the zeroth iteration, initialize  $\mathbf{x}^n$  for  $n = 0$  at the starting point, for example,  $(x_1^0, x_2^0, x_3^0) = (0, 0, 0)$ .
2. At the 1st iteration, draw the sample of the first coordinate  $x_1^1$  from the initial sample point:

$$\begin{aligned}
 x_1^1 &\sim p(x_1 | \overbrace{x_2 = x_2^0, x_3 = x_3^0}^{\text{frozen } x_2 \text{ and } x_3 \text{ coordinates}}), \\
 &= \frac{\hat{p}(x_1, x_2^0, x_3^0)}{Z_1},
 \end{aligned} \quad (19.17)$$

where the conditional probability  $p(x_1 | x_2 = x_2^0, x_3 = x_3^0)$  is proportional to the joint distribution  $\hat{p}(x_1, x_2^0, x_3^0)$  and  $Z_1$  is the proportionality constant. The pdf  $p(x_1 | x_2 = x_2^0, x_3 = x_3^0)$  is a 1D distribution in  $x_1$  that is much easier to sample than a high-dimensional one.

We then draw the second  $x_2^1$  and third  $x_3^1$  coordinates in the same way:

$$\begin{aligned}
 x_2^1 &\sim p(x_2 | x_1^1, x_3^0), \\
 x_3^1 &\sim p(x_3 | x_1^1, x_2^1),
 \end{aligned} \quad (19.18)$$

except we use the update  $x_1^1$  or both updates  $x_1^1$  and  $x_2^1$ .

<sup>3</sup>Many of the ideas in this section are inspired by Alexander Novikov's lecture "Bayesian Methods for Machine learning" at <https://www.coursera.org/lecture/bayesian-methods-in-machine-learning/gibbs-sampling-eZBy5>.

## Gibbs Sampling

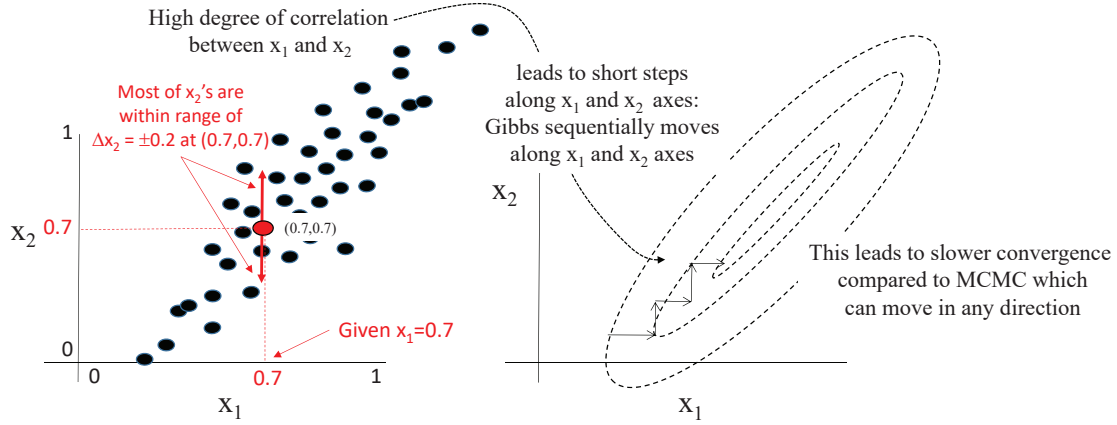


Figure 19.10: Samples of points (left plot) where  $x_1$  and  $x_2$  are highly correlated along a line slanted at  $45^\circ$ . This leads to slow convergence of Gibbs sampling on the (right plot) which sequentially samples along each of the axes. Highly correlated points for this example mean that most (i.e. highly probable) samples near  $(0.7, 0.7)$  along the vertical axis are within  $\Delta x_2 = \pm 0.2$ . Images inspired by Ben Lambert's video at <https://www.youtube.com/watch?v=ER3DDBFzH2g> viewed in November, 2021.

3. For the  $(k+1)^{th}$  iteration we have

$$\begin{aligned} x_1^{k+1} &\sim p(x_1|x_2^k, x_3^k), \\ x_2^{k+1} &\sim p(x_2|x_1^{k+1}, x_3^k), \\ x_3^{k+1} &\sim p(x_3|x_1^{k+1}, x_2^{k+1}), \end{aligned} \tag{19.19}$$

where each step depends on previous steps, so it is not suitable for a parallel computer.

The Gibbs sampling compares to the Metropolis algorithm in the following ways.

- Unlike Metropolis sampling, there is no accept or reject option in Gibbs sampling, all proposals are accepted. Not having to decide to accept or reject can make Gibbs computationally more efficient than Metropolis sampling.
- Similar to Metropolis sampling, the current choice of the sample depends on the sample in the previous iteration.
- If there is a high degree of correlation between two parameters, say  $x_1$  and  $x_2$ , then the progress of Gibbs sampling can be very slow as illustrated in Figure 19.10.
- Gibbs sampling requires conditional distributions, which are not so easy to obtain in some cases.
- Gibbs is not easily parallelized, it goes from one dimension to the next that depends on the previous point.

An example of Gibbs sampling is shown in Figure 19.11. Here, the joint distribution is given by Figure 19.11a for  $p(x, y)$ , and the conditional distributions are given in Figure 19.11b-19.11c. These

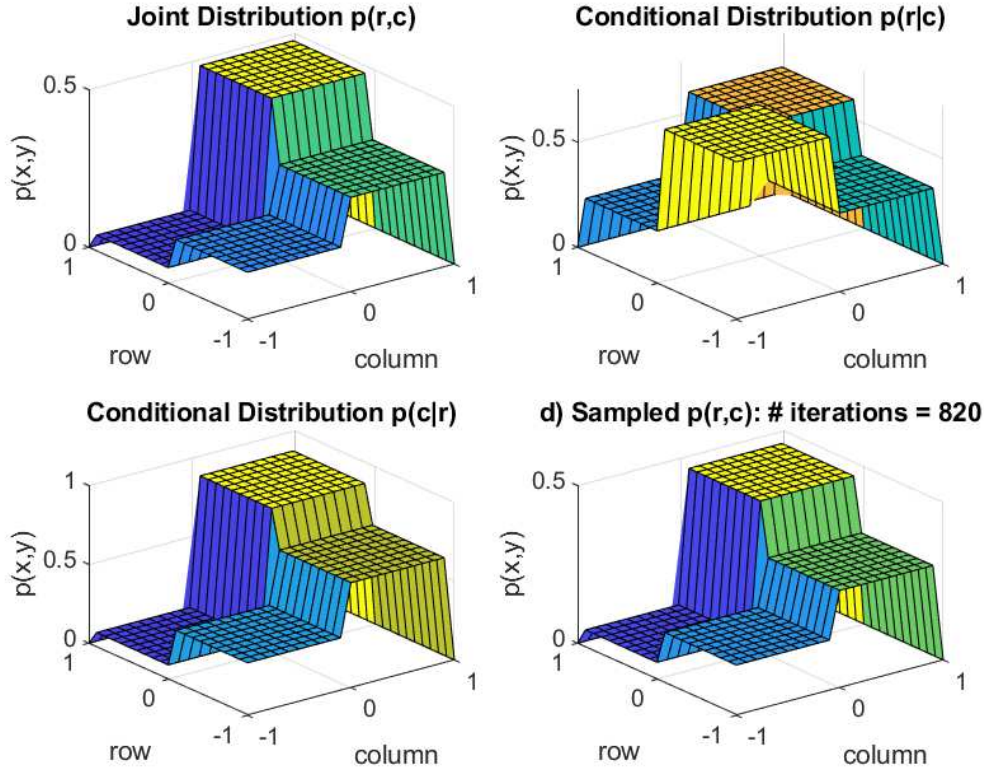


Figure 19.11: a) Actual  $p(x,y)$ , conditional b)  $p(x|y)$  and c)  $p(y|x)$ . d) is the  $p(x,y)$  estimated by Gibbs sampling after 820 iterations. Images inspired by Ben Lambert's video at <https://www.youtube.com/watch?v=ER3DDBFzH2g> viewed in November, 2021.

are then used for Gibbs sampling, and the Gibbs sampled distribution is shown in Figure 19.11d after 820 iterations. It closely matches that of the actual  $p(x,y)$  in Figure 19.11a.

We now show that the Gibbs sampling converges to points that honor the joint distribution  $\hat{p}(x_1, x_2, x_3)$ . We can define a transition probability  $q(x', y', z'|x, y, z)$  such that marginalizing over  $(x, y, z)$  gives

$$p(x', y', z') = \sum_{x,y,z} q(x', y', z'|x, y, z) p(x, y, z). \quad (19.20)$$

Here,  $q(x', y', z'|x, y, z)$  represents one iteration of Gibbs sampling in moving from  $(x, y, z)$  to  $(x', y', z')$ . As shown before, one iteration requires three transitions, each one of which freezes two of the coordinates, which can be expressed as a concatenation of conditional probabilities:

$$q(x', y', z'|x, y, z) = p(x'|y, z) p(y'|x', z) p(z'|x', y'). \quad (19.21)$$

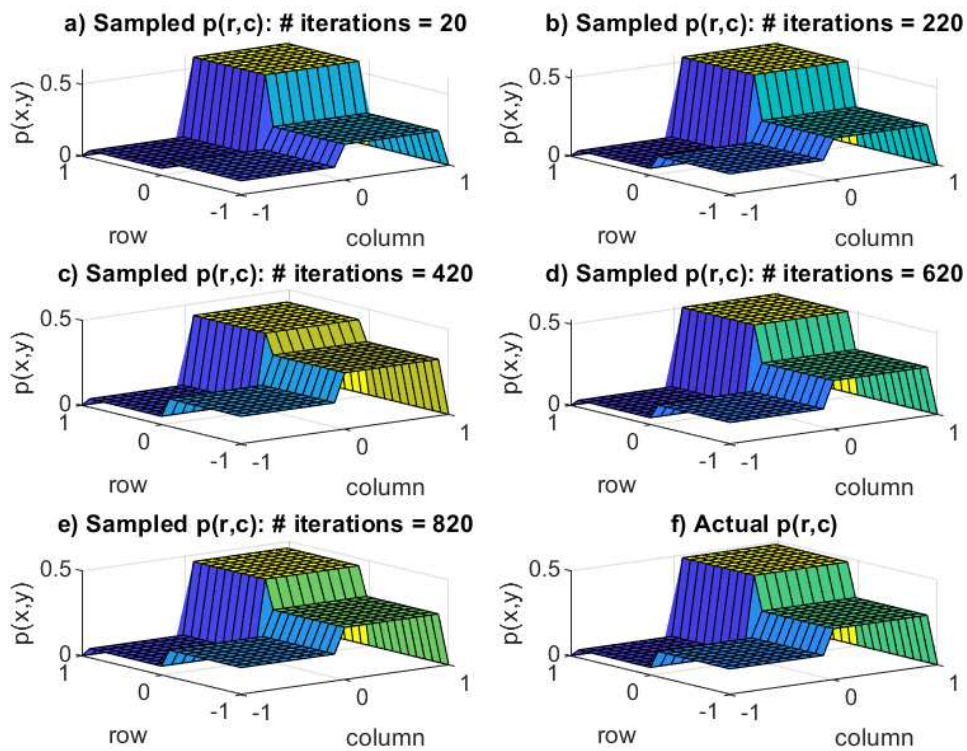


Figure 19.12: a)-e) are the joint distributions estimated by Gibbs sampling for different numbers of iterations. f) is the actual  $p(x,y)$ . Images inspired by Ben Lambert's video at <https://www.youtube.com/watch?v=ER3DDBFzH2g> viewed in November, 2021.

Plugging equation 19.21 into equation 19.20 gives

$$p(x', y', z') = \sum_{x, y, z} \overbrace{p(x'|y = y, z = z)p(y'|x = x', z = z)p(z'|x = x', y = y')}^{q(x', y', z'|x, y, z)} p(x, y, z). \quad (19.22)$$

The terms not dependent on the summation variables can be moved outside the summation to get

$$\begin{aligned} p(x', y', z') &= p(z'|x', y') \sum_{y, z} p(x'|y, z) p(y'|x', z) \overbrace{\sum_x p(x, y, z)}^{p(y, z)}, \\ &= p(z'|x', y') \sum_z \left[ \sum_y \overbrace{p(x'|y, z) p(y, z)}^{p(x', y, z)} \right] p(y'|x', z), \\ &= p(z'|x', y') \sum_z p(y'|x', z) \overbrace{\sum_y p(x', y, z)}^{p(x', z)}, \\ &= p(z'|x', y') \sum_z \overbrace{p(y'|x', z) p(x', z)}^{p(x', y', z)}, \\ &= p(z'|x', y') \overbrace{\sum_z p(y', x', z)}^{p(y', z')}, \\ &= p(z'|x', y') p(y', z') = p(x', y', z') \therefore \end{aligned} \quad (19.23)$$

Therefore, if we start out with samples  $(x, y, z)$  from the distribution  $p(x, y, z)$ , then the iterative updating steps  $x \rightarrow x'$ ,  $y \rightarrow y'$  and  $z \rightarrow z'$  in equation 19.19 will lead to the distribution  $p(x, y, z)$ .

A problem with Gibbs sampling is that it has slow convergence with coordinates that are strongly correlated with one another. At each sample point, it can only move along one, possibly inefficient, coordinate direction. It cannot move in a more reasonable direction that is available to the Metropolis algorithm. This is illustrated in Figure 19.10 where a starting point near the center axis of the ellipse did not need to move very far to get to a highly probable neighbor. This will lead to many iterations where many sampled points are clustered near to one another. Thus, many more iterations will be needed to have a more diverse sampling of points that represent the true density distribution.

## 19.7 Summary

Four different sampling techniques are presented: the inverse transform, acceptance-rejection, Gibbs sampling and the Metropolis version of MCMC. For high-dimensional distributions, Gibbs and Metropolis sampling methods are preferred because they are much more efficient than the inverse transform or AR methods. Gibbs sampling has the merit of simplicity in that it only requires a few lines of code and it only samples along a single dimension at each sub-iteration. For example, Figure 19.12 only required a few lines of MATLAB code. However, it has slow convergence when the coordinates are highly correlated, so the Metropolis or Metropolis-Hastings algorithms are preferred.

## 19.8 Exercises

1. Instead of the coin flip simulations, simulate sampling the outcomes of rolling an unfair 6-sided dice and compare the simulated PMFs with the actual one for an unfair dice. Do the same for a fair 6-sided dice. Constrain the unfair dice to have a probability of 0.05 for the unfair face. Do the two sampling exercises require the same number of dice rolls for stationarity? Explain.
2. Construct a Gibbs sampling code to sample points from a 2D Gaussian distribution. Compare the number of simulations that are needed to reach a stationary state for a round Gaussian compared to ones characterized by thin ellipsoids. Explain the different convergence rates. Plot convergence rate against the aspect ratio of the Gaussian.
3. Same as previous problem except use the Metropolis algorithm. Compare convergence rates with those of Gibbs sampling. Explain the differences in convergence rates.
4. Find the normalized eigenvectors of the transition matrix  $\mathbf{A}$  associated with the grape-lettuce-cheese transition matrix in Figure 19.5. Is one of these eigenvectors parallel with the stationary vector  $\mathbf{x} = (0.335 \ 0.3110 \ 0.3541)$ ? How can this eigenvector be use to accelerate convergence? See the *Stochastic Processes* section at <https://stephens999.github.io/fiveMinuteStats/>.

## 19.9 Appendix: MATLAB Functions for Metropolis Code

This section contains the MATLAB function calls for plotting the results from the Metropolis algorithm in section 19.5.2.

### Code 19.9.2. *Title1 Plotting Functions for MATLAB 1D Metropolis Sampling*

```
% Title1 Function: Plots densities and Samples
subplot(121);
plot(x,g,'b-',x,g2,'r-');hold on; %Plot Target+Proposed Densities
plot(xhistory,xhistory*0,'*k');hold off

mean=sum(xhistory)/nit; % Estimate Mean+stnd. Samples
std=sqrt(var(xhistory));

xlabel('x');ylabel(['p(x)'])
title(['a] Metropolis Sampling'])
[' Unimodal Gaussian: N = ',num2str(nit)]
text(-4.6,.8,['\mu= ',num2str(round(mean,2))],'fontsize',8)
text(-4.6,.75,['\sigma= ',num2str(round(std,2))],'fontsize',8)
text(-4.6,.7,['Actual \mu = ',num2str(mu2)],'fontsize',8)
text(-4.6,.65,['Actual \sigma = ',num2str(sigmalike)],'fontsize',8)
text(0.5,.8,['proposed density'],'color','blue','fontsize',8)
text(1,.3,['target density'],'color','red','fontsize',8)
```



**Code 19.9.3.** *Title2 Plotting Function for MATLAB 1D Metropolis Sampling*

```
% Title2 Plotting Code
hold on;
plot(xhistory,xhistory*0,'*k');
xlabel('x');ylabel('Target Probability')
text(1,.5,['target density'],'color','red','fontsize',8)
text(.5,.6,['proposed density'],'color','blue','fontsize',8)

jj=0;jj1=0;
for j=1:nit
    if xhistory(j)<=0;jj=jj+1;x1(jj)=xhistory(j);end
    if xhistory(j)>0;jj1=jj1+1;x2(jj1)=xhistory(j);end
end
```



## Chapter 20

# Bayes' Theorem

Bayes' formula in equation 28.28 was derived by defining the joint probability distribution  $P(X, Y)$  in terms of the conditional probabilities  $P(X|Y)$  and  $P(Y|X)$ . Now we provide the intuitive meaning of Bayes' Theorem and use examples to describe how it can be applied to geoscience data to get the most probable earth model and the uncertainty estimates of its parameters. We also show how the Bayesian maximum a posteriori (MAP) estimate of the model parameter vector  $\mathbf{m}$  is equivalent to the least squares estimate of  $\mathbf{m}$  with regularization. Finally, Bayes' Theorem is applied to satellite images of an oil spill, vertical seismic profile (VSP) data, well-log data and seismic data to estimate the most probable geophysical parameters subject to constraints.

### 20.1 Introduction

Bayes' formula

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}, \quad (20.1)$$

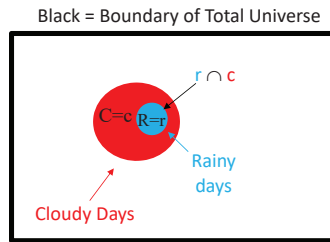
was derived in Chapter 28 where  $P(X)$  is the prior probability for  $X$ ,  $P(Y)$  is the evidence,  $P(X|Y)$  is the posterior probability and  $P(Y|X)$  is the likelihood. Now we will interpret the physical meaning of equation 20.1 using examples.

**Simple Example and 3 Key Benefits of Bayesian Inversion.** There are three significant benefits of Bayesian inversion (BI): 1) BI finds the model  $X^*$  that best explains the new data  $Y$ , 2) BI honors constraints  $P(X)$ , aka as a prior, from previously recorded information, and 3) BI gives  $P(X|Y)$  as the uncertainty estimate of  $X^*$ , which can be updated as new data becomes available.

These three benefits are illustrated with a simple example of living on the Red Sea coast near Jeddah where we often see clouds, but rarely experience rain. It usually rains about 1-3 times/year, so let's assume  $P(r) = .01$ , where  $R = r$  indicates rain and  $R = \tilde{r}$  indicates no rain. However, it is partly cloudy about 10% of the time so that  $P(c) = 0.1$ , where  $C = c$  indicates cloudiness. We also know that the few times it rains, the skies are always cloudy so the likelihood of seeing clouds if it rains is  $P(c|r) = 1$ . These facts are illustrated by the Venn diagrams in Figure 20.1.

Does reciprocity  $P(r|c) = P(c|r)$  hold true, where the probability of it raining after seeing clouds is  $P(r|c) = 1$ ? Intuitively, the answer is no. However, Bayes' Rule in equation 20.1 says that it is true if  $P(X)/P(Y) = P(r)/P(c) = 1$ , which implies that the red circle must have the same radius as the coincident blue circle. But the area ratio of rainy/cloudy days in Figure 20.1a is  $A_r/A_c = 0.1$ , so Bayes' Rule says that  $P(r|c) = P(c|r)P(r)/P(c) = 1 \times 0.1 = 0.1$ . Thus, reciprocity is invalid because the chances of it raining if you see clouds is only 10%.

## a) Venn Diagrams



## b) Bayes' Rule

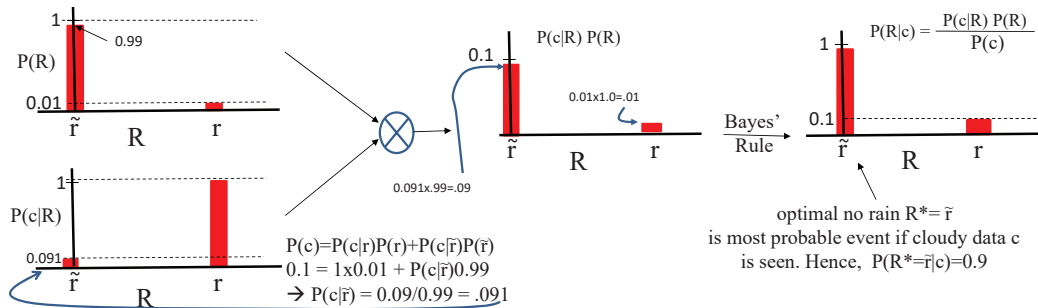


Figure 20.1: a) Venn diagrams for the prior probabilities of cloudy days  $P(c) = 0.10$  and rainy days  $P(r) = 0.01$ . b) Probability mass functions for the marginal  $P(R)$ , likelihood  $P(c|R)$ , and posterior  $P(R|c) = P(c|R)P(R)/P(c)$  functions. The Law of Total Probability  $P(c) = P(c|r)P(r) + P(c|\tilde{r})P(\tilde{r})$  is used to compute the value of  $P(c|\tilde{r}) = 0.91$  at the bottom left in b).

A histogram interpretation of Bayes' Rule is shown in Figure 20.1b. Here, the marginal PMF  $P(r) = 0.01$  is small in Figure 20.1b, but the likelihood<sup>1</sup>  $P(c|r) = 1$  says that there is 100% chance of cloudiness if it rains. Multiplying these two probabilities suggests that there is a low posterior chance  $P(r|c) \approx P(c|r)P(r) = 0.01$  for  $R = r$  of seeing rain after you see clouds. However, Bayes' Rule increases  $P(r|c) = P(c|r)P(r)/P(c) = 0.1$  by a factor of 10, which is in agreement with the analysis by Venn diagrams<sup>2</sup>. The posterior that is the highest probability  $P(R|c)$  is given by  $P(\bar{r}|c) = 0.9$ , which makes sense because if you see clouds, then it is more likely you will see no rain rather than rain.

The recorded data for cloudy and rainy days will allow us to estimate the sample variances and sample means of  $P(c|r)$ ,  $P(r)$ , and  $P(c)$ , so the probability mass function (PMF) formulas in Figure 20.1b will provide the PMF  $P(r|c)$ , which is the uncertainty function for predicting the chance for rain if you see clouds. Uncertainty analysis is needed because scientific predictions should not only provide an answer to questions, but also an uncertainty associated with that answer.

**Bayesian Analysis in Science.** Applying Bayesian analysis to geoscience data is becoming a standard procedure for many geoscientists (Tarantola and Valette, 1982; Jackson, 1979; Jackson and Matsuura, 1985; Cary and Chapman, 1988; Duijndam, 1988; Mosegaard and Tarantola, 1995; Sen and Stoffa, 1995; Malinverno and Leaney, 2000; Malinverno and Briggs, 2004; Mukerji et al., 2001; Scales and Tenorio, 2001; Ulrych et al., 2001; Buland and Omre, 2003). It can be very useful when interpreting seismic data for the presence of hydrocarbons or for reducing the risk factors in drilling expensive wells (Mukerji et al., 2001; Gonzalez et al., 2001; Ulrych et al., 2001; Gonzales et al., 2012a-b; Avseth et al., 2005). One of the reasons for its popularity, especially in the 1980s, can be attributed to the discovery of stochastic Markov chain Monte Carlo methods ([https://en.wikipedia.org/wiki/Markov\\_chain\\_Monte\\_Carlo](https://en.wikipedia.org/wiki/Markov_chain_Monte_Carlo)), which removed many of the computational problems, and an increasing interest in nonstandard, complex applications (Wolpert, 2004). Bayesian methods are strongly employed by Bishop (2006) in describing machine learning methods.

A problem with Bayes' theorem is that it assumes knowledge of the prior PDFs and often employs a normal distribution for the parameters of interest. This issue is partially addressed by Malinverno and Briggs (2004) by using both the hierarchical and empirical Bayes' approaches that allow for a range of possible prior hypotheses on the hyperparameters, and so reduces the number of assumptions.

Stochastic search methods are often used to find the Bayesian inverse to seismic data. Some examples include VSP inversion by Malinverno and Leaney (2000), inversion of Rayleigh wave data by Bodin and Sambridge (2009), and inversion of sonic acoustic data for sea-floor properties by Dettmer and Dosso (2009), Dettmer et al. (2010) and Dosso et al. (2014). In the last three examples, the number of model parameters is an unknown and the posterior probability density function includes the type of model parametrization. This procedure, denoted as transdimensional inference, partially addresses the overfitting problem of inverting for too many unknowns.

Bayesian inversion can also be applied to many other types of seismic data such as receiver functions and surface waves (Bodin et al., 2012), ambient noise (Young et al., 2013; Saygin et al., 2015), reflectivity inversion (Ray et al., 2016; Sen and Biswas, 2017; Dadi et al., 2018), and recorded reflection inverted by FWI (Hawkins and Sambridge, 2015; Ray et al., 2017). For example, Visser et al. (2019) and Gebraad et al. (2020) applied Monte-Carlo search methods to find the full-waveform

---

<sup>1</sup> $P(c|r) = 1$ , where the Bayesian literature also refers to it as a likelihood function  $\mathcal{L}(R; C) = P(C|R)$ . In the context of searching for the maximum likelihood, we can assume that the condition of cloudiness  $C = c$  is fixed and the maximum likelihood procedure looks for the value of  $R$  that most likely explains that it is cloudy. For a fixed observed value of the data  $C = c$ , the likelihood  $\mathcal{L}(R; c)$  is a function of the unknown parameter  $R$ . Integrating (or summing) it over all  $R$  is not equal to 1 so the likelihood function does honor the definition of a probability function.

<sup>2</sup>See Appendices 20.9-20.10 for more complicated examples of computing how weather conditions affect the decision to go biking.

inverse to, respectively, acoustic and elastic data. The Visser et al. (2019) strategy is to use a layer-stripping strategy to accelerate convergence in sampling, and a parsimonious dipping layer parametrization is used so that the Markov Chain Monte Carlo algorithm can efficiently search with fewer iterations. Each layer is parameterized with velocity, thickness, and lower interface dip angle. They then inverted for narrow 2D velocity models using small offset data, and used the resulting tomogram as the starting model for standard FWI.

Instead of using stochastic search methods, gradient descent methods (Liu and Wang, 2016) can be used to find the optimal model that minimizes an objective function. One such objective function is the Kullback-Leibler (KL) divergence (see Appendix 20.11 and Kullback and Leibler, (1951)). Zhang and Curtis (2021) demonstrated the success of this approach for inverting synthetic acoustic data generated from simple models.

Bayesian analysis has also been successfully used for EM data (Buland and Kolbjornsen, 2012; Hawkins et al., 2018), AVO from seismic data (Buland and Omre, 2003), fault detection (Singh et al., 2021), rock physics analysis (Bachrach, 2016; Grana, 2016), Dix inversion (Buland et al., 2011), time-lapse data (Buland and El Ouair, 2006), lithology prediction from pre-stack seismic data (Buland et al., 2008), and seismic amplitudes using the Zoeppritz equations (Buland and Omre, 2003). Inverting the rock-physics model involves establishing a linearized relation between the elastic attributes (velocities  $V_P$ ,  $V_S$  and density  $\rho$ ) and the petrophysical properties (porosity  $\phi$ , clay volume  $C$ , and water saturation  $S_w$ ). If the elastic attributes are recorded in a well then the inverse goal is to determine their associated petrophysical properties and rock types. This can be accomplished with Bayesian inversion (Gonzalez et al., 2016; Xu et al., 2016) as demonstrated by the examples in section 20.6.

More generally, Bayesian analysis (Sisson et al., 2018) is used in almost all fields of science and engineering, including astronomy (Guest, 2011; Thrane and Talbot, 2019; Lorenz, 2019; Affholder et al., 2021), biology (Yau and Campbell, 2019), medicine (Ashby, 2005; Yarnell et al., 2020), geostatistics (Gelfand and Banerjee, 2017), hydrology and groundwater management (Borsukm, 2004; Shihab, 2005; Bromley et al., 2005; Allan, 2008; Henricksen et al., 2012), and geothermal studies (Lösing et al., 2020) that use geological and heat production data as constraints to limit the number of variables to invert. All of these scientific endeavors use Bayesian analysis to form models from the data (Sheidt et al., 2018), but they also provide the uncertainty estimate  $P(\mathbf{m}|\mathbf{d})$  for the model parameters. Finally, a word of caution: Bayes' methods are widely used, but they are not accepted by all (Klir, 1994; Wang, 2004; Gelman, 2008; Horgan, 2016).

The next section provides an intuitive understanding of Bayesian analysis using the cancer problem as an example. Some properties of Bayes' analysis are then presented, including Naive Bayesian analysis and the connection between Bayes' inversion and regularized least squares inversion. This is followed by several examples of Bayesian analysis of well-log and seismic data. The final section presents the summary.

## 20.2 Intuitive Interpretation of Bayes' Theorem

Bayes' formula in equation 20.1 is trivial to derive, but like any formula we need to intuitively understand its meaning before applying it to data. This section first presents the geometrical meaning of Bayes' Theorem, and then deciphers each term in terms of the cancer probabilities in section 28.2.1. To facilitate the understanding of equation 20.1 in terms of cancer testing example,  $X$  is replaced by  $C$  which means the patient actually has cancer. Also,  $Y$  is replaced by  $T$  which means that the test for cancer is positive, with the understanding that this positive prediction is wrong a small percent of the time. Equation 20.1 in terms of the RVs for the cancer problem becomes

$$P(C|T) = P(T|C) \frac{P(C)}{P(T)}. \quad (20.2)$$

Black = Boundary of Total Universe

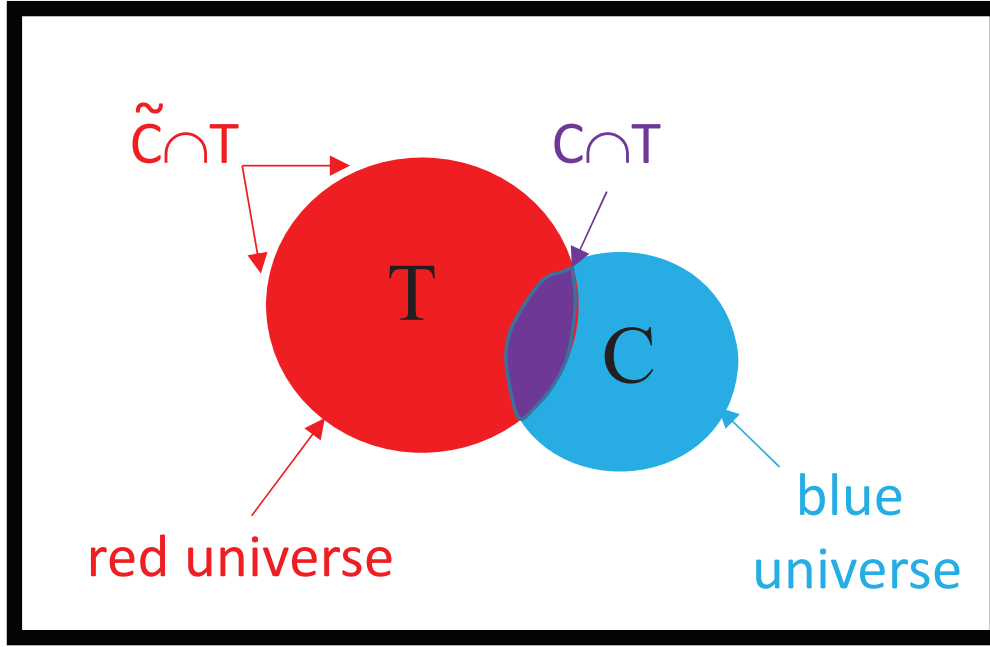


Figure 20.2: Same as Figure 28.1 except  $X$  is replaced by  $C$  where the patient actually has cancer and  $Y$  is replaced by  $T$  for a positive cancer test result. The positive test result might be mistaken so it would be labeled as a false positive (FP). Otherwise it is a true positive (TP) so that total number of positive test results is equal to *No. of TP* + *No. of FP*.

### 20.2.1 Geometrical Interpretation of $P(C)$ , $P(T)$ , $\frac{P(C)}{P(T)}$

Equation 20.2 can be geometrically interpreted in terms of the circle's areas in Figure 20.2. Here, the area  $A_C$  of the blue circle divided by the area  $A$  of the universe is  $P(C) = A_C/A$ , which represents the fraction of people who have cancer. Similarly, the conditional probability of a positive test result is  $P(T) = A_T/A$ , where  $A_T$  is the area of the red circle and represents the fraction of people who test positively for cancer. Therefore, the ratio  $\frac{P(C)}{P(T)}$  can be replaced by  $\frac{A_C}{A_T}$  to get the geometrical formulation of Bayes' Theorem:

$$\begin{aligned}
 P(C|T) &= P(T|C) \overbrace{\frac{P(C)}{P(T)}}^{\text{scaling term}}, \\
 &= \frac{A_C \cap A_T}{A_C} \frac{A_C}{A_T}.
 \end{aligned} \tag{20.3}$$

The geometrical interpretation of the scaling factor  $P(C)/P(T) = A_C/A_T$  is that it is equal to the ratio of the areas of the blue  $C$  and red  $T$  circles in Figure 20.2, which is less than 1 in this example. If the areas  $A_C = A_T$  are the same then this implies  $P(C)/P(T) = 1$  and the symmetrical

relationship  $P(C|T) = P(T|C)$ .

The small area  $A_C$  of the blue circle says that there are only a relatively small number of possible events that can occur in the huge reference universe enclosed by the black rectangle (Riley, 2020). Therefore, the purple area  $A_C \cap A_T$  divided by the smaller reference area  $A_C$  of the blue circle, i.e.  $P(T|C) = \frac{A_C \cap A_T}{A_C}$ , should be larger than the purple area  $A_C \cap A_T$  divided by the larger reference area  $A_T$  of the red circle, i.e.  $P(C|T) = \frac{A_C \cap A_T}{A_T}$ :

$$\begin{aligned} P(T|C) &= \frac{A_C \cap A_T}{A_C}, \\ &> P(C|T) = \frac{A_C \cap A_T}{A_T}, \end{aligned} \quad (20.4)$$

Therefore, someone who actually has cancer will much more likely receive a positive test result compared to actually having cancer after getting a positive test result. This is good news to the worried patient who just received a positive cancer diagnosis, he/she has good hope that this is a false-positive prediction.

### 20.2.2 Medical Interpretation of $P(C)$ , $P(T)$ , $\frac{P(C)}{P(T)}$ .

As a medical example of equation 20.4, assume that the test can detect pancreatic cancer with 100% accuracy, i.e.  $P(T|C) = 1$ , if you have pancreatic cancer. Here,  $P(T|C)$  is denoted as a *true positive* (TP). We will assume that the test is equivalent to experiencing certain symptoms suffered by all pancreatic cancer patients, e.g. allergy symptoms such as feeling weak and nausea. This test, unfortunately, can give a *false positive* (FP) result if a cancer-free person has, e.g. a special 1 out-of-10,000 person allergy, so that  $P(T|\tilde{C}) = 10^{-4}$ . Unsurprisingly, the flawed test produces many false positives (FPs) because there are ten times more people with special allergies compared to the number of pancreatic cancerous people in a large population. These allergies are time dependent so they might be prevalent during the first test but are very rarely active during a second test. Therefore, even if the test is 100% accurate  $P(T|C) = 1$  if you have a rare pancreatic cancer with  $P(C) = 10^{-5}$  (Table 20.1), there will be ten times more healthy folks ( $P(T|\tilde{C}) = 10^{-4}$ ) with allergies in a huge population of 100,000 who will falsely test. We will now explain how this skews the Bayesian estimate of  $P(C|T)$ .

Table 20.1: Empirical test statistics for pancreatic cancer ([https://en.wikipedia.org/wiki/Bayes%27\\_theorem](https://en.wikipedia.org/wiki/Bayes%27_theorem)).

$P(C) = 10^{-5}; P(\tilde{C}) = 0.99999$	Cancer: $C$	No Cancer: $\tilde{C}$
Test Positive: $T$	True Positive: $P(T C) = 1.0$	False Positive: $P(T \tilde{C}) = 10^{-4}$
Test Negative: $\tilde{T}$	False Negative: $P(\tilde{T} C) = 0.0$	True Negative: $P(\tilde{T} \tilde{C}) = 1 - 10^{-4}$

$P(T)$  in the denominator of equation 20.3 is interpreted as proportional to the number of FPs and TPs. Multiplying the fraction of the population with cancer by the area  $A$ , i.e. the total number of people, of the universe gives  $N^C = P(C) \times A$  as the number of people with cancer. Since  $P(T|C)$  is the fraction of cancer patients who test positive then the product of  $P(T|C)$  and the number of cancer patients  $N^C = P(C)A$  is

$$N^{TP} = \overbrace{P(T|C)}^{\text{frac. TP}} \overbrace{P(C)A}^{\# \text{ cancer patients}}, \quad (20.5)$$



which is the number  $N^{TP}$  of cancer patients who are issued a true-positive test result. Similarly the number  $N^{FP}$  of healthy people who receive a false-positive result is

$$N^{FP} = \overbrace{P(T|\tilde{C})}^{\text{frac. FP}} \overbrace{P(\tilde{C})A}^{\# \text{ non-cancer patients}}, \quad (20.6)$$

where  $\tilde{C}$  indicates the person does not have cancer. For a cancer such as pancreatic cancer with a very low incidence rate of  $P(C) = 10^{-5}$  ([https://en.wikipedia.org/wiki/Bayes%27\\_theorem](https://en.wikipedia.org/wiki/Bayes%27_theorem)), the number of false positives  $N^{FP} = P(T|\tilde{C})P(\tilde{C})A \gg N^{TP} = P(T|C)P(C)A$  is much more than the number of true positives because there are orders-of-magnitude more allergic people w/o cancer in the population.

Therefore, for pancreatic cancer, the total number  $N^T$  of people who test positive for cancer is the sum of the number of true positives in equation 20.5 and false positives in equation 20.6

$$\begin{aligned} N^T &= N^{TP} + N^{FP}, \\ &= \overbrace{P(T|C)P(C)A}^{\text{small}} + \overbrace{P(T|\tilde{C})P(\tilde{C})A}^{\text{big}}. \end{aligned} \quad (20.7)$$

Dividing through by  $A$  gives

$$P(T) = \frac{N^T}{A} = P(T|C)P(C) + P(T|\tilde{C})P(\tilde{C}), \quad (20.8)$$

which could have been arrived at by the more opaque Law of Probability in section 28.1.3. This formula can be substituted into the denominator of equation 20.3 to get its FP and TP interpretation:

$$P(C|T) = \frac{\overbrace{P(T|C)P(C)}^{\# \text{ true positives}}}{\underbrace{P(T|C)P(C)}_{\# \text{ true positives}} + \underbrace{P(T|\tilde{C})P(\tilde{C})}_{\# \text{ false positives}}}, \quad (20.9)$$

where we will always assume that there is an implicit multiplication of both the numerator and denominator by  $A$  so the braced terms represent the numbers of events. If the number of false positives is much more than the true positives in the denominator, then this will significantly reduce the value of  $P(C|T)$  on the left side. Thus, there will be little confidence  $0 < P(C|T) \ll 1$  in a positive test result for cancer.

Plugging the numbers in Table 20.1 into Bayes' formula gives

$$P(C|T) = \frac{1 \times 10^{-5}}{\underbrace{1 \times 10^{-5}}_{\# \text{ true positives}} + \underbrace{10^{-4} \times 0.99999}_{\# \text{ false positives}}} \approx 0.091, \quad (20.10)$$

which says that a positive test for pancreatic cancer only has a 9.1% chance of being accurate. Can we improve this test accuracy? Yes, take more data and update the statistics as the next section demonstrates.

### 20.2.3 Updating Predictions with New Information

If a second test is taken and it also indicates cancer then the updated posterior is

$$\begin{aligned} P(C|T_2, T_1) &= \frac{P(T_2, T_1|C)P(C)}{P(T_2, T_1|C)P(C) + P(T_2, T_1|\tilde{C})P(\tilde{C})}, \\ &= \frac{P(T_2|C)P(T_1|C)P(C)}{P(T_2|C)P(T_1|C)P(C) + P(T_2|\tilde{C})P(T_1|\tilde{C})P(\tilde{C})}, \end{aligned} \quad (20.11)$$

where conditional independence is assumed such that  $P(T_2, T_1|C) = P(T_2|C)P(T_1|C)$ . The probability for receiving a positive result with the second test is the same as for the first test so that  $P(T_2|C) = P(T_1|C)$ . Therefore, the above equation with the values in Table 20.1 becomes

$$\begin{aligned} P(C|T_2, T_1) &= \frac{P(T_1|C)^2 P(C)}{P(T_1|C)^2 P(C) + P(T_1|\tilde{C})^2 P(\tilde{C})}, \\ &= \frac{10^{-5}}{\underbrace{10^{-5}}_{\# \text{ true positives}} + \underbrace{(10^{-4})^2 \times 0.99999}_{\# \text{ false positives}}} = 0.999. \end{aligned} \quad (20.12)$$

Hence, the probability of having cancer after two positive tests in a row is greater than 99%. Why did the posterior value jump from 9.1% to 99%? The mathematical reason is that the number  $\approx 10^{-4}$  of false positives in the denominator of equation 20.10 is larger than the number  $\approx 10^{-5}$  of true positives. This means that the scaling term  $P(C)/P(T)$  (see equation 20.3) is much less than one so  $P(C|T) = 9.1\%$  is small. However, the second test finds the number  $(10^{-4})^2 \times 0.99999 \approx 10^{-8}$  of false positives in equation 20.12 to be squared, which now makes the denominator  $P(T)$  to be dominated by the number of true positives  $P(C|T)P(T) \approx 10^{-5}$ . Thus the scaling factor  $P(C)/P(T) \approx 0.999$ , and since  $P(T_1|C)^2 = 1$  then the posterior  $P(T_2, T_1|C) = 0.999$ .

The intuitive explanation is that for a single test, the chance of receiving a false positive is low  $\propto P(T_1|\tilde{C})P(\tilde{C}) \approx 10^{-4}$ , but it is about 10 times greater than the chance of receiving a true positive  $\propto P(T_1|C)P(C) = 10^{-5}$ . However, if you take two tests in a row, then the chance  $\propto P(T_1|\tilde{C})^2 \approx 10^{-8}$  of receiving two false positives, i.e. sampling someone with active allergies during the first and second tests, in a row is really low compared to the chance  $P(T_1|C)^2 P(C) = 10^{-5}$  of getting two true positives in a row. Thus the number of true positives dominate both the numerator and denominator, which says that you are very likely to have cancer after two positive tests in a row.

Equation 20.12 can be rearranged to give

$$P(C|T_2, T_1) = \frac{P(T_2|C) \overbrace{P(T_1|C)P(C)}^{P(C)^{new}}}{\underbrace{P(T_2|C)P(T_1|C)P(C)}_{P(C)^{new}} + \underbrace{P(T_2|\tilde{C})P(T_1|\tilde{C})P(\tilde{C})}_{P(\tilde{C})^{new}}}, \quad (20.13)$$

where  $P(C)^{new} = P(T_1|C)P(C) \propto P(C|T_1)$  and  $P(\tilde{C}) = P(T_1|\tilde{C})P(\tilde{C}) \propto P(\tilde{C}|T_1)$  are proportional to the posteriors from the first test, where the proportionality constant is  $1/P(T_1)$ . In other words, the previously computed posterior  $P(C|T_1)$  can be used to update your new prior  $P(C)^{new} \approx P(C|T_1)$ , and so compute the new posterior  $P(C|T_1, T_2)$ . This allows for real-time prediction of the posterior as new data arrives, and is much faster than recomputing all the distributions from the original data set. Many popular real-time models or online models are based on Bayesian statistics<sup>3</sup>.

<sup>3</sup><https://towardsdatascience.com/a-mathematical-explanation-of-naive-bayes-in-5-minutes-44adebcd5f8>

## 20.3 Naive Bayes' Theorem

Instead of using Bayes' Theorem in equation 20.1, we will use the reverse of it for a set  $\mathbf{X}$  of  $K$  features  $\mathbf{X} = (X_1, X_2, \dots, X_K)$  that are classified as  $Y$ :

$$\begin{aligned} P(Y|X_1, X_2, \dots, X_K) &= \frac{P(X_1, X_2, \dots, X_K|Y)P(Y)}{P(X_1, X_2, \dots, X_K)}, \\ &= \frac{P(X_1, X_2, \dots, X_K|Y)P(Y)}{\sum_{i=1}^I P(X_1, X_2, \dots, X_K|y=i)P(y=i)}, \end{aligned} \quad (20.14)$$

where there are  $I$  discrete values for  $Y \in \{1, 2, \dots, I\}$  and equation 28.25 has been used to replace  $P(X_1, X_2, \dots, X_K)$  by a weighted sum of conditional probabilities. This is known as a *discriminant* model in Chapter 21 because it predicts the class  $\mathbf{y}$  of the input data  $\mathbf{x}$ .

The rationale for using  $P(Y|X_1, X_2, \dots, X_K)$  on the left side of equation 20.14 is that we are given features  $\mathbf{X}$  in order to predict the class  $Y$ . In this case we assume that training has computed the values of the likelihood  $P(\mathbf{X}|Y)$  and prior  $P(Y)$ , as well as  $P(X_k|Y)$  for  $k \in \{1, 2, \dots, K\}$ .

However, computing the conditional probabilities is infeasible for high-dimensional data so we assume that  $X_1, X_2, \dots, X_K$  are mutually independent conditioned on  $Y$ . In this case, equation 28.22 can replace the denominator of equation 20.14 to give the formula for the Naive Bayes' classifier:

$$P(Y|X_1, X_2, \dots, X_K) = \frac{P(X_1, X_2, \dots, X_K|Y)P(Y)}{\sum_{i=1}^I P(y=i)\prod_{k=1}^K P(X_k|y=i)}; \quad (20.15)$$

$$= \frac{P(Y)\prod_{k=1}^K P(X_k|Y)}{\sum_{i=1}^I P(y=i)\prod_{k=1}^K P(X_k|y=i)}, \quad (20.16)$$

where both the numerator and denominator only require the computation of the two-term conditional probability functions  $P(X_k|Y)$  and the marginal function  $P(Y)$  for different values of  $Y = y$ . Equation 20.16 is more practical to compute than equation 20.14 because computing two-term conditional probability functions such as  $P(X_k|Y)$  requires many fewer computations and realizations than filling in a high-dimensional hypercube  $P(X_1, X_2, \dots, X_K|Y)$ . See Figure 28.2 for a 3D cube and Appendix 20.10 for a numerical example of using the Naive Bayes' Theorem.

In summary, the Naive Bayes' classifiers are a family of supervised classifiers which use Bayes' Theorem to classify a discrete response. It uses the observed features  $\{x_1, x_2, \dots, x_N\}$  with a simplifying assumption of independence. The result is the Naive Bayes' formula in equation 20.16 that is much more computational feasible than Bayes' formula in equation 20.14.

## 20.4 Motivation for Bayes' Theorem

A motivation for using equation 28.28 or equation 20.1 is that we want to find the uncertainty, i.e. probability  $P(X|Y)$ , of the occurrence of event  $X$  if  $Y$  has already happened. For example, if our cancer test  $Y$  is positive how reliable is it as a true-positive indicator of cancer  $X$ ? If the well-log indicator  $Y$  suggests a formation has oil, what is the reliability of earning profits in starting up a multi-million dollar drilling project  $X$ ? In psychology, Bayes' Theorem can give the probability of a stimulus  $X$  given a response  $Y$ . In seismology, we are given data and wish to know the uncertainty of an estimated earth model  $X$  that explains the recorded data  $Y$ . This conditional probability is used to estimate the error bars for the parameters of the earth model.

Why not just use the definition of conditional probability  $P(X|Y) = P(X, Y)/P(Y)$  to get  $P(X|Y)$ . The answer is that we often don't have direct knowledge of the numerator  $P(X, Y)$  or

denominator  $P(Y)$ . Instead we might have an analytical or counting estimate for  $P(Y|X)$  and a partial guess for  $P(X)$ ; the Law of Total Probability is used to get  $P(Y)$ . This is enough for using Bayes' Theorem to get  $P(X|Y)$ .

The generative model finds the statistical distribution  $P(X|Y)$  of the data  $X$  that best explains the observed classes  $Y$ . This type of model is classified as unsupervised because a training set of labeled data is not required.

In contrast, the discriminative model finds the statistical distribution  $P(Y|X)$  that best predicts  $Y$  from the input data  $X$ . Discriminative models often need supervised training where a large training set of training pairs (input features and output classes) is required. As an example, the supervised neural network finds the NN model that most accurately predicts the class  $Y$  of the input data  $X$  (see Chapters 4-6). This neural network model is functionally equivalent to knowing  $P(Y|X)$ .

## 20.5 Equivalence Between Bayesian Inversion and Regularized Least Squares Inversion

For geophysical inversion, the goal of the frequentist is to find the optimal model  $\mathbf{m}^*$  that predicts the observed data  $\mathbf{d}$  which minimizes a specified objective function  $\epsilon$  such as

$$\epsilon = \frac{1}{2} \overbrace{\|\mathbf{d}^{predict} - \mathbf{d}\|^2}^{\text{data penalty}} + \lambda \overbrace{g(\mathbf{m})}^{\text{model penalty}}, \quad (20.17)$$

where  $\|\mathbf{d}^{predict} - \mathbf{d}\|^2$  is the  $L_2$  norm of the data residual  $\mathbf{d}^{predict} - \mathbf{d}$  and  $g(\mathbf{m}) \geq 0$  is the scalar model penalty function, e.g.  $g(\mathbf{m}) = \frac{1}{2}\|\mathbf{m}\|^2$ . Choosing the functional form of the penalty function reflects our prior belief that the model should also minimize  $g(\mathbf{m})$ . The regularization parameter  $\lambda$  regulates how much we prefer minimizing the data residual over minimizing the model penalty function.

As shown in section 28.5, if  $\lambda = 0$  then the optimal solution of the least squares problem is equivalent to the maximum likelihood estimate when the errors belong to a normal distribution. That is, the log-likelihood function can be described as

$$\mathcal{L}(\mathbf{m}; \mathbf{d}) = \ln P(\mathbf{d}|\mathbf{m}), \quad (20.18)$$

and the maximum likelihood estimate  $\mathbf{m}^*$  is the one that maximizes  $\mathcal{L}(\mathbf{m}; \mathbf{d})$ :

$$\mathbf{m}^* = \underset{\mathbf{m}}{\operatorname{argmax}} \mathcal{L}(\mathbf{m}; \mathbf{d}) = \underset{\mathbf{m}}{\operatorname{argmin}} [-\mathcal{L}(\mathbf{m}; \mathbf{d})]. \quad (20.19)$$

Here,  $P(\mathbf{d}|\mathbf{m})$  replaces  $P(\mathbf{y}|\mathbf{x}, \mathbf{w})$  on the lefthand side of equation 28.36.

**Key Idea 20.5.1. Frequentist's Inversion**

The Maximum Likelihood Estimate, i.e. the Frequentist's inverted model, is described by equation 20.19 and is often computed by a gradient descent algorithm using the easily derived gradient formulas  $\partial L/\partial m_i$ . The likelihood function  $\mathcal{L}(\mathbf{m}; \mathbf{d}) = P(\mathbf{d}|\mathbf{m})$  cannot be interpreted as a conditional probability distribution because frequentists deny that  $\mathbf{m}$  is a RV governed by a probability distribution (Press et al., 2007). However, the uncertainty in  $\mathbf{m}$  is caused by noise in  $\mathbf{d}$ , so that the variance values of the model's parameters can be computed by the model covariance matrix in equation 28.38.

The model-parameter statistics and hyperparameters, such as the damping parameter and the number of model parameters, can be selected by a bootstrap method (Efron, 1979) or by cross-validation (Bishop, 2006). Cross-validation divides the training data into  $S$  groups of the same size, where a unique portion of the data is excluded from each group. Then,  $S$  models are inverted from these  $S$  groups for various values of the hyperparameters. The optimal hyperparameters are the ones that provide the most accurate predictions for the out-of-the-training data. The Jackboot method (Efron, 1979) creates  $S$  groups of equal size by selecting  $N$  points at random from the original data, even if the same point is selected multiple times. These groups are then inverted to get  $S$  models, which can be used to approximate the model means and variances.

In contrast, Bayesian inversion uses Bayes' Theorem to rescale  $P(\mathbf{d}|\mathbf{m})$  by  $P(\mathbf{m})/P(\mathbf{d})$  to get  $P(\mathbf{m}|\mathbf{d})$

$$P(\mathbf{m}|\mathbf{d}) = \frac{P(\mathbf{d}|\mathbf{m})P(\mathbf{m})}{P(\mathbf{d})}, \quad (20.20)$$

and then finds the optimal  $\mathbf{m}^*$  that maximizes  $P(\mathbf{m}|\mathbf{d})$ :

$$\mathbf{m}^* = \underset{\mathbf{m}}{\operatorname{argmax}} P(\mathbf{m}|\mathbf{d}), \quad (20.21)$$

$$= \underset{\mathbf{m}}{\operatorname{argmax}} \frac{P(\mathbf{d}|\mathbf{m})P(\mathbf{m})}{P(\mathbf{d})}, \quad (20.22)$$

$$= \underset{\mathbf{m}}{\operatorname{argmax}} P(\mathbf{d}|\mathbf{m})P(\mathbf{m}), \quad (20.23)$$

which is known as the maximum a posteriori (MAP) estimate of the model parameter. The denominator in equation 20.22 is eliminated to get equation 20.23 because it does not explicitly depend on  $\mathbf{m}$ . For the well-log example in the next section, a Monte Carlo search algorithm is used to find the optimal  $\mathbf{m}^*$ .

**Key Idea 20.5.2. Bayesian Inversion**

Equations 20.23 or 20.24 give the MAP estimate, which is often found by a stochastic search of earth models in which the simulated data  $\hat{\mathbf{d}}$  agree with the observations  $\mathbf{d}$  to a certain level (Xu et al., 2016). For the well-log example in section 20.6,  $\mathbf{m}^*$  from equation 20.23 is the Bayesian estimate, aka MAP estimate, of the rock type and its uncertainty is supplied by the probability distribution  $P(\mathbf{m}|\mathbf{d})$ .

The gradient descent algebra for taking derivatives is simpler if  $P(\mathbf{m}|\mathbf{d})$  in equation 20.21 is replaced

by its logarithm  $\ln P(\mathbf{m}|\mathbf{d})$  (Xu, 2020) to get

$$\begin{aligned}\mathbf{m}^* &= \underset{\mathbf{m}}{\operatorname{argmax}} \ln P(\mathbf{m}|\mathbf{d}), \\ &= \underset{\mathbf{m}}{\operatorname{argmax}} [\ln P(\mathbf{d}|\mathbf{m}) + \ln P(\mathbf{m})].\end{aligned}\quad (20.24)$$

Under certain assumptions, it is approximately equivalent to the least squares estimate of  $\mathbf{m}$  with regularization. This can be shown by equating  $P(\mathbf{y}|\mathbf{x}, \mathbf{w})$  in equation 28.36 with  $P(\mathbf{d}|\mathbf{m})$  and the prior with  $P(\mathbf{m}) = \frac{1}{\beta} e^{-\lambda \frac{1}{2} \mathbf{m}^T \mathbf{m}}$  so that the terms in equation 20.24 become

$$P(\mathbf{d}|\mathbf{m}) = \frac{1}{\alpha} e^{-\frac{1}{2} \sum_{n=1}^N (\mathbf{m}^T \mathbf{x}^{(n)} - \mathbf{y}^{(n)})^2 / \sigma_m^2}; \quad P(\mathbf{m}) = \frac{1}{\beta} e^{-\lambda \frac{1}{2} \mathbf{m}^T \mathbf{m}}, \quad (20.25)$$

where  $\mathbf{w} \rightarrow \mathbf{m}$ ,  $\alpha$  and  $\beta$  are normalization terms,  $\lambda = 1/\sigma_m^2$  is the regularization term and  $\sigma_m^2$  is the variance for the model parameters. Plugging these probability functions into equation 20.24 gives

$$\mathbf{m}^* = \underset{\mathbf{m}}{\operatorname{argmax}} \left[ -\frac{1}{2} \sum_{n=1}^N (\mathbf{w}^T \mathbf{x}^{(n)} - \mathbf{y}^{(n)})^2 - \frac{\sigma^2}{\sigma_m^2} \frac{1}{2} \mathbf{m}^T \mathbf{m} / \sigma^2 \right]. \quad (20.26)$$

In this case, the MAP estimate in equation 20.26 is the same as the regularized least squares solution  $\mathbf{m}^* = [\mathbf{L}^T \mathbf{L} + \lambda \mathbf{I}]^{-1} \mathbf{L}^T \mathbf{d}$  where the damping parameter is defined as  $\lambda = \frac{\sigma^2}{\sigma_m^2}$ . More generally, if the inverse-covariance matrix for the Bayesian model is fully populated, then the regularized least squares solution can emulate this by including an inverse-covariance weighting term in the sum of squared residuals. Sparse solutions can be promoted using an  $L^1$  norm for the model penalty function (<https://towardsdatascience.com/a-bayesian-take-on-model-regularization-9356116b6457>). An example of using a regularizer in Bayesian inversion is the Kullback-Leibler (KL) regularization described in Appendix 20.11.

## 20.6 Geophysical Examples of Bayesian Inversion

Several examples will be presented for analyzing geophysical data by Bayes' Theorem. These examples do not cover the vast number of geoscience applications with Bayes' Theorem, but they give practical insights as to how they can be implemented. See section 20.1 for a partial bibliography.

The first example is for estimating the location of an oil spill from synthetic satellite images several days after the accident. The next example inverts for the velocity model from vertical seismic profile (VSP) traveltimes. The third example is for estimating the locations of hydrofrac sources from passive seismic data. This is followed by the case history of using Bayes' theorem to predict rock types from well-log data with core calibration. The final example is for identifying different facies units from well-log and seismic data. The motivation for predicting the rock type and the uncertainty of prediction is to reduce the probability of drilling an expensive dry hole.

### 20.6.1 Prediction of an Oil-Spill Source

Oil-spill accidents in marine environments are often not reported until several days after the accident. The ocean currents carry the spill far from the location of the oil dump, and so the original location of the spill is not obvious. However, if the ocean currents and spill date are approximately known, then the image of a spill photographed by a satellite can be modeled by a current-flow program. Fluid-flow computations start with a guess of the spill's origin point at  $(x_r, y_r)$ , duration time  $\Delta t$ , release time  $d_r$  after the start time of the spill, and the quantity  $q_r$  of the spill, where it is assumed that the ocean current model is already known. Therefore different spill locations (denoted

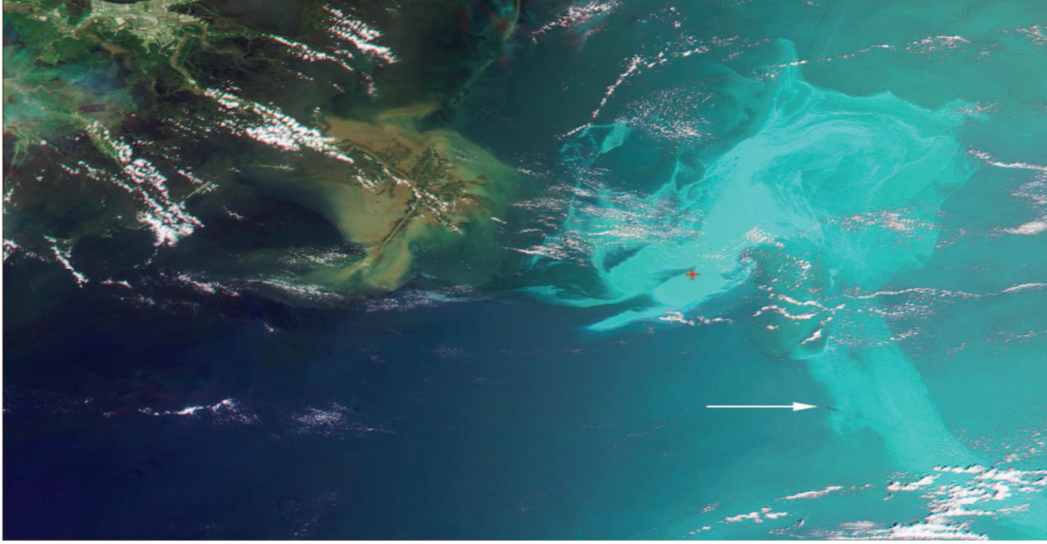


Figure 20.3: Images of the Deepwater Horizon oil slick in the Gulf of Mexico obtained by the Multi-angle Imaging SpectroRadiometer (MISR) instrument aboard NASA's Terra spacecraft on May 17, 2010. Oil-spill zone denoted by different shades of cyan color, which are related to the oil concentration on the surface. NASA/GSFC/LaRC/JPL, MISR Team.

as epicenters), oil-dump times and durations, and spill sizes can be modeled until the final boundary configuration of the model spill matches that of the satellite data (see Figure 20.3). For legal and remedial purposes, both the epicenter of the spill and the probability of the estimated epicenter should be computed as well.

To determine the source, intensity and eruption time of an oil spill from satellite images, El Mohtar et al. (2021) proposed Bayesian analysis to find the unknown model parameters  $\mathbf{m} = (x_r, y_r, \Delta t, d_r, q_r)$  that explain post-spill satellite images related to the time-varying concentration  $d(x, y, t)$  of the contaminant on the water surface. Discretizing the spatial and temporal variables transforms the function  $d(x, y, t)$  into the recorded concentration vector  $\mathbf{d}$ , where the number of elements in  $\mathbf{d}$  is equal to the number of spatial and temporal grid points. The goal is to find the model parameter  $\mathbf{m}$  that maximizes the posterior probability distribution

$$p(\mathbf{m}|\mathbf{d}) \propto p(\mathbf{d}|\mathbf{m})p(\mathbf{m}), \quad (20.27)$$

which is Bayes' theorem. Here, simulations of fluid flow are used to compute the likelihood function  $p(\mathbf{d}|\mathbf{m})$  for different samples of  $\mathbf{m}$  determined by a MCMC method. The prior  $p(\mathbf{m})$  is a uniform distribution of  $\mathbf{m}$  over pre-defined bounds.

The concentration of the contaminant can be expressed as a sum of weighted basis functions

$$d(\mathbf{x}, t) = \sum_{i=1}^{n_p} \gamma_i \phi(\mathbf{x} - \mathbf{x}_i)_\sigma \quad (20.28)$$

where  $\gamma_i$  is the mass carried by particle  $i$ ,  $\phi_\sigma$  is a radial basis function, and  $n_p$  is the total number of released particles. Here,  $\gamma_i = q/n_p$ , where  $q$  is the total mass of the released contaminant. The

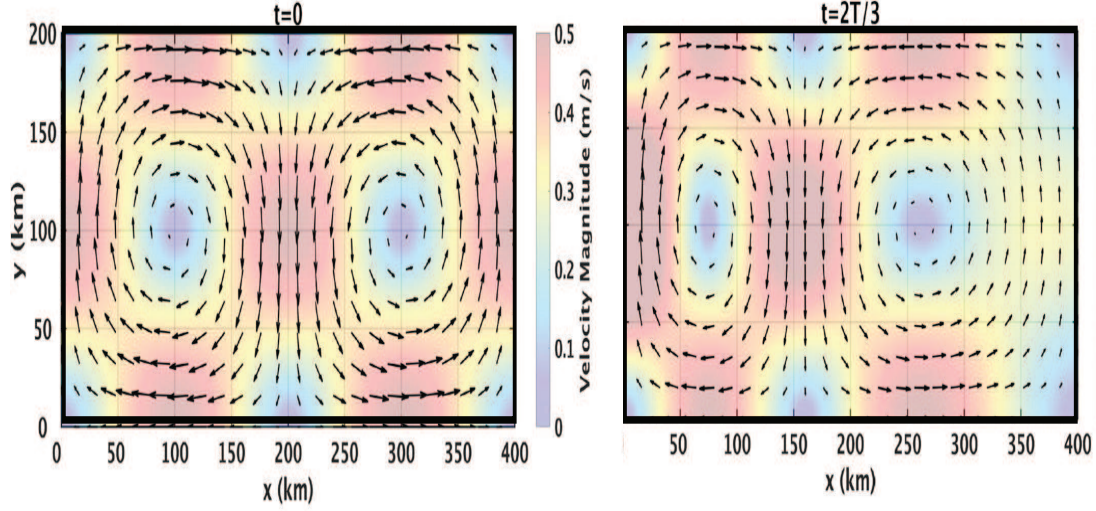


Figure 20.4: Snapshots of the flow field  $t = 0$  and  $t = T/3$ . Images extracted from El Mohtar et al. (2021).

bivariate Gaussian PDF with variance  $\sigma^2$  is the basis function:

$$\phi(\mathbf{x} - \mathbf{x}')_{\sigma} = \frac{1}{2\pi\sigma^2} \exp(-|\mathbf{x} - \mathbf{x}'|^2/2\sigma^2), \quad (20.29)$$

where

$$\sigma = 2 \times \max_i \{ \min_j (|\mathbf{x}_i - \mathbf{x}_j|) \} \quad i, j \in \{1, 2, \dots, n_p\}. \quad (20.30)$$

Here, the  $(x, y)$  coordinates of a particle at time  $t$  are embedded in a flowing ocean model. This data vector also contains information about the density of the oil on the water's surface and the duration time of the spill since its start time.

The model and data vectors are connected by the modeling operator  $\mathbf{L}$ , where  $\mathbf{L}$  is computed by a numerical solution to the 2D advection-diffusion LPT model (El Mohtar et al., 2021). An example of the flow field computed by this model is shown in Figure 20.4 for  $t = 0$  and  $t = 2T/3$ , where  $T$  is the total elapsed time of the simulation. The modeling equation is

$$\mathbf{Lm} = \mathbf{d} + \text{error}, \quad (20.31)$$

where the uncertainties in measurements and parameter estimates are accounted for by the term *error*.

### Metric for the Recorded and Predicted Data Mismatch

The contour of the contaminant concentration on a simulated image is denoted as  $\Gamma$  and corresponds to a given value of the concentration  $\tau$ . Given the density  $\eta$  of the oil, the thickness of the oil is given by  $\tau/\eta$ . To measure the dissimilarity between the observed and predicted contours, El Mohtar et al. (2021) used the global Hausdorff distance.

The typical posterior distribution is the Gaussian distribution, with  $\Delta = \Delta(\Gamma_{obs}, \Gamma_{pred})$  is the



general distance metric between the observed and predicted concentration contours of the image. However, El Mohtar et al. (2021) adopted the exponential distribution as the likelihood function:

$$p(\mathbf{d}|\mathbf{m}) = \frac{1}{\beta} \exp(-\Delta/\beta), \quad (20.32)$$

where  $\beta$  is the scale parameter of this distribution. This distribution is preferred over the Gaussian because it is a better indicator of how different shapes match each other. The Hausdorff distance is used as the argument of this exponential, where it accounts for the contour differences over the entire predicted and observed images.

### Oil Spill Example

A simulated oil spill is computed with the 2D modeling algorithm described in the previous section. The number of MCMC simulations is 10,000 for exploring the 5D parameter space of  $\mathbf{m}$ , where each simulation produced a series of snapshot images of fluid flow at different time intervals. Eight different sets of simulations were computed, each consisting of 10,000 fluid-flow simulations. The idea is to propose a theoretical location of the localized oil spill, simulate the spread of the oil along the surface over a long time interval, and then compare the distribution of the predicted oil spill on the surface to the actual distribution for different days. The predicted oil distribution that best matches the actual one will yield the highest value of the posterior function.

For burn in, the first 500 simulations were discarded. The remaining samples were then used to estimate the 5-dimensional posterior distribution associated with the predicted location of the oil spill.

The values of the Markov chain parameters  $\mathbf{m} = (x_r, y_r, \Delta t, d_r, q_r)$  plotted against the MCMC iteration number are shown in Figure 20.5. The values of these 5 parameters sample the 5D space over a restricted range of values determined by the uniform prior  $p(\mathbf{m})$  restricted to between a specified range of bounds.

Figure 20.6 depicts the snapshots of the posterior distribution (blue-red-yellow contours in the lower portion of each snapshot) obtained from the flow simulations. The flow lines are indicated by blue arrows. The groundtruth concentration is the blue kidney-bean shaped set of arrows at the upper left, the black contours indicate the predicted contour and the red dot at the bottom of each graph denotes the actual source location. The dark purple shapes in the lower portion of the graph indicate the posterior distribution at the initiation time, and the trial source locations are denoted by the black dots. Note, the highest probability values (higher probabilities indicated by hotter colors) are those in the second row of images where the trial source location is near the actual source location. Each column of images is computed for a different value of the trial total mass  $q_r$  of the spilled oil. The results indicate that the predicted contours are somewhat insensitive to the value of  $q_r$  between 50 and 200 tons. The last row images suggests that the final results are sensitive to sources mislocated by more than 50 km from the actual source.

### Corner Plots to Assess Parameter Dependency

Corner plots in Figure 20.7 are used to visualize the relationship between two different parameters. For example, the joint probability  $p(x_r, y_r)$  map in the second row shows that the probability values are somewhat insensitive to locations along the area with the yellow crooked smile. This plot was computed by marginalizing the joint probability:  $p(x_r, y_r) = \sum_{\Delta t, d_r, q_r} p(x_r, y_r, \Delta t, d_r, q_r)$ .

#### 20.6.2 VSP Traveltime Inversion

Bayesian analysis will now be applied to vertical seismic profile (VSP) traveltimes in order to invert for the velocity model as a function of depth. The inverted velocity profile is equivalent to that obtained by regularized inversion, also known as the damped least squares procedure. It is sometimes

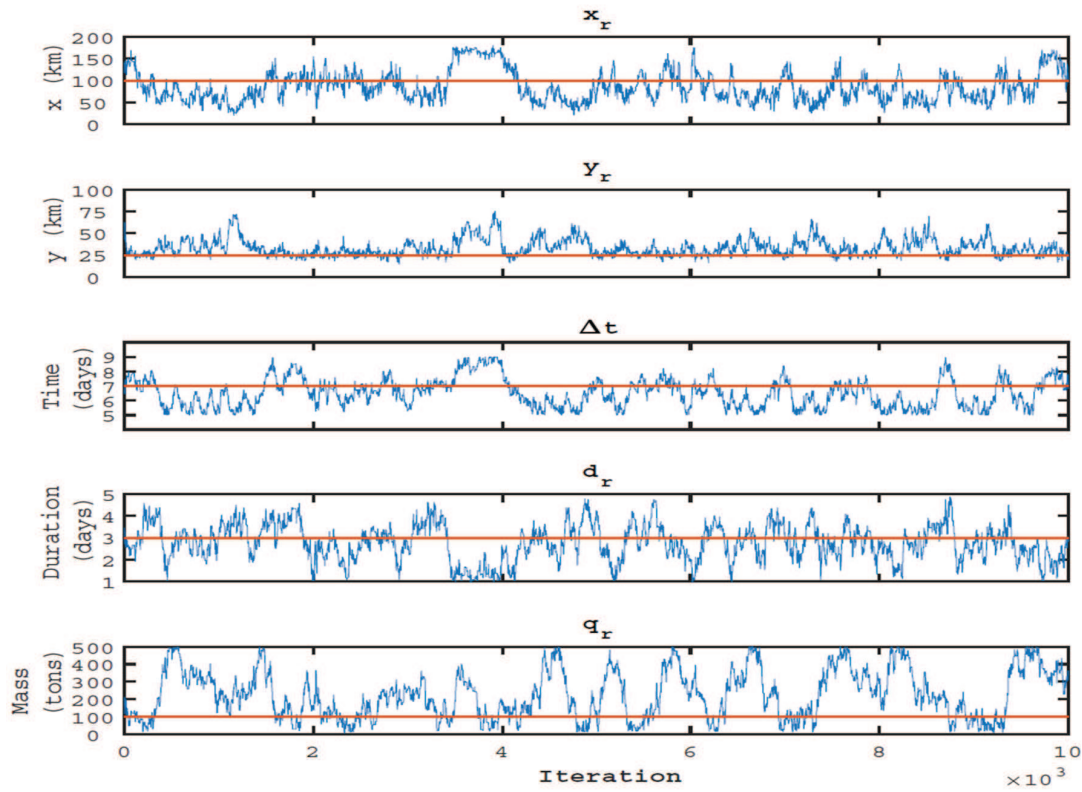


Figure 20.5: Values of Markov chain parameters computed by a MCMC simulation consisting of 10,000 iterations. The traces plot the values of the five parameters ( $x_r, y_r, \Delta t, d_r, q_r$ ). The red horizontal line in each plot represents the value of the corresponding true (reference) parameter. Images extracted from El Mohtar et al. (2021).

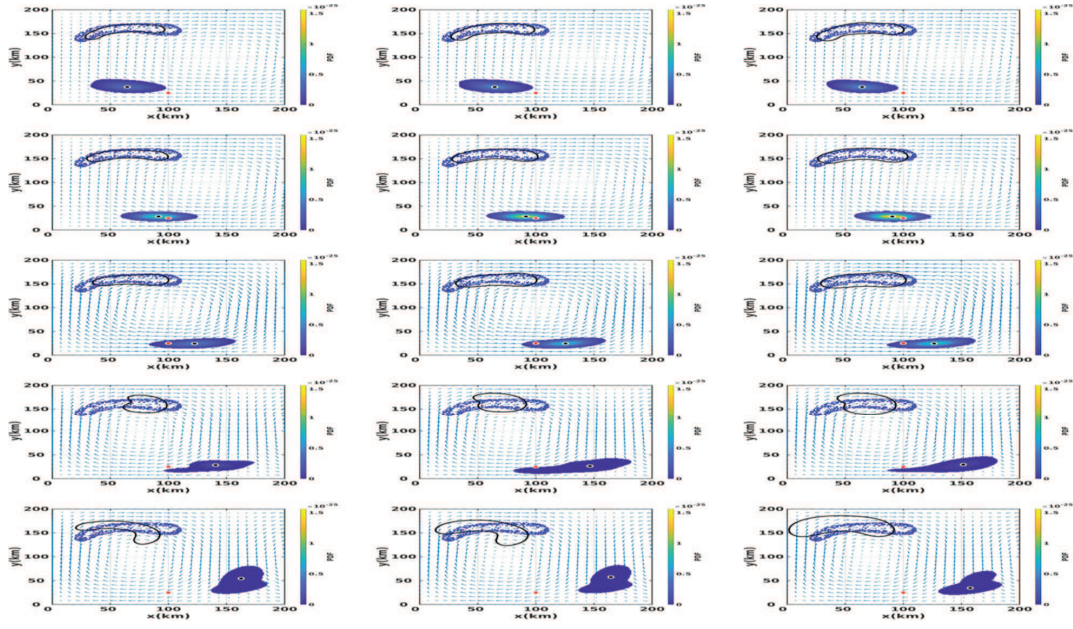


Figure 20.6:  $(x_r, y_r)$  maps (slices) of the 5D posterior probability distribution of for  $d_r = 2$  days,  $\Delta t = 5, 6, 7, 8$  and  $9$  days, and  $q_r = 50, 100$  and  $200$  tons. The red dots correspond to the true values of  $(x_r, y_r)$ . The black dot marks the  $(x_r, y_r)$  location that has the highest posterior probability in each 2D map. The blue contour represents the observation, whereas the black contour is the model output of a hypotheticalal release from the black dot, for values of  $d_r, \Delta t$  and  $q_r$  corresponding to each plot. The blue vectors show the currents at the time of release corresponding to each map. Images extracted from El Mohtar et al. (2021).

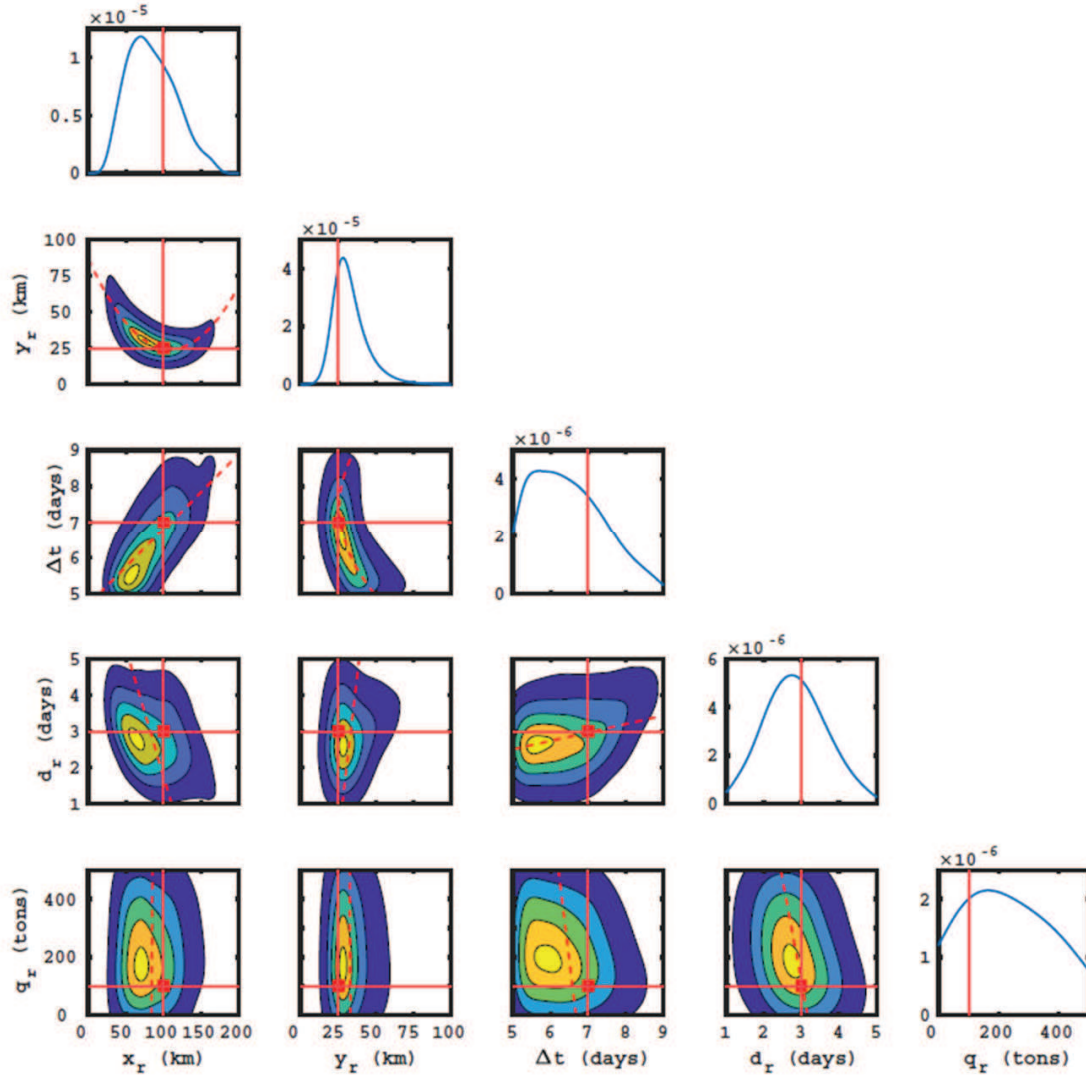


Figure 20.7: Corner plots of the five parameters  $(x_r, y_r, \Delta t, d_r, q_r)$ , formed by plots of the joint probability distributions of each pair of parameters and plots of the marginal distributions of the single parameters. The red lines represent the values of the true parameters corresponding to each plot. For example, the red dot for the  $p(x_r, y_r)$  plot is the actual location of the spill. Images extracted from El Mohtar et al. (2021).

## VSP Modeling & Traveltime Inversion

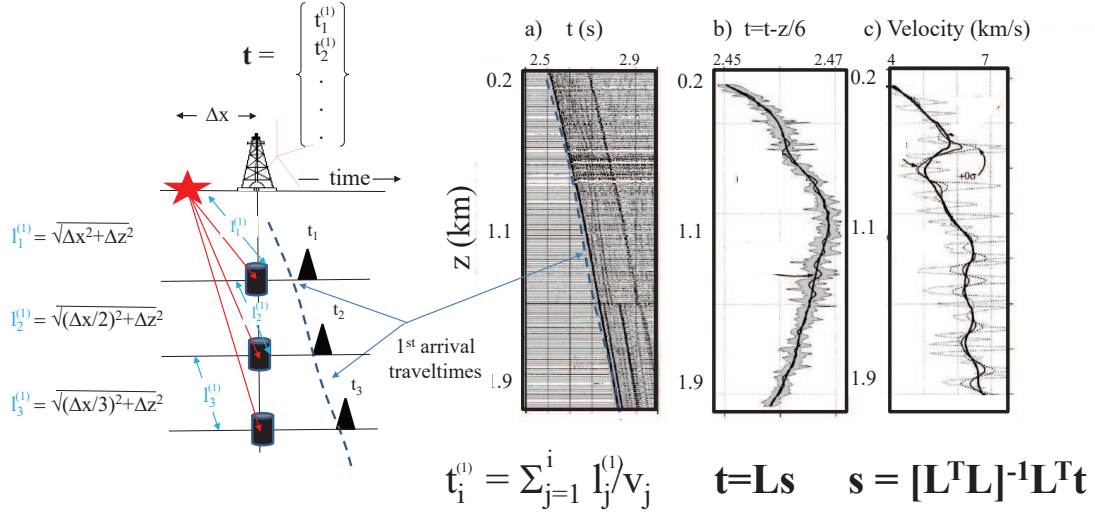


Figure 20.8: VSP configuration on the left showing red transmission rays and traces with direct waves. a) VSP common shot gather from a field experiment, b) picked traveltimes of direct arrivals after a linear moveout correction, and c) velocity profile (dark solid lines) estimated by damped least squares inversion. The width of the gray zone in b) is proportional to the standard deviation of the traveltimes errors and the wildly oscillating curve in c) is that computed by the undamped inversion. Formulas for modeling and undamped/unweighted inversion are at the bottom. The three figures on the right are from Lizarralde and Swift (1999).

necessary to get conditional probabilities from joint distributions, so Appendices 20.13-20.14 provide the corresponding formulas.

### Least Squares Inversion of VSP Traveltimes

The goal of VSP traveltime inversion by a least squares method is to invert the transmission traveltimes for the velocity model  $v(z)$  as a function of depth. These traveltimes can be modeled by the traveltime modeling equation listed under Figure 20.8a, where we assume an  $N$ -layered medium with each layer having a thickness of  $\Delta z$ , no ray bending and a VSP geophone at the interface of each layer. The modeled transmission traveltime  $t_i^{(s)}$  to the  $i^{th}$  geophone for the  $s^{th}$   $s \in \{0, 1, 2, \dots, S\}$  source is the sum of the traveltimes  $t_i^{(s)} = \sum_{j=1}^i l_j^{(s)} / v_j$  along each segment of the ray from the surface to the  $i^{th}$  geophone, and  $s_i = 1/v_i$  is the slowness in the homogeneous  $i^{th}$  layer with velocity  $v_i$ . The ray-segment length  $l_i^{(s)}$  in a layer for the  $i^{th}$  ray of the  $s^{th}$  source is equal to

$$l_i^{(s)} = \sqrt{(s\Delta x/i)^2 + \Delta z^2}; \quad s \in \{0, 1, 2, \dots, S\}, \quad i \in \{1, 2, \dots, N\}, \quad (20.33)$$

if the source is offset by  $s\Delta x$  from the vertical well.

The matrix-vector representation of the transmission traveltime equations is given as  $\mathbf{L}\mathbf{s} = \mathbf{t}$  under Figure 20.8b. For a 5-interface velocity model with one source with  $s = 0$  for a src-rec offset

$\Delta x = 0$ , the resulting traveltimes equations form a  $5 \times 5$  lower-triangular matrix:

$$\overbrace{\begin{bmatrix} l_1^{(0)} & 0 & 0 & 0 & 0 \\ l_2^{(0)} & l_2^{(0)} & 0 & 0 & 0 \\ l_3^{(0)} & l_3^{(0)} & l_3^{(0)} & 0 & 0 \\ l_4^{(0)} & l_4^{(0)} & l_4^{(0)} & l_4^{(0)} & 0 \\ l_5^{(0)} & l_5^{(0)} & l_5^{(0)} & l_5^{(0)} & l_5^{(0)} \end{bmatrix}}^{\mathbf{L}} \overbrace{\begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \end{pmatrix}}^{\mathbf{s}} = \overbrace{\begin{pmatrix} t_1^{(0)} \\ t_2^{(0)} \\ t_3^{(0)} \\ t_4^{(0)} \\ t_5^{(0)} \end{pmatrix}}^{\mathbf{t}^{obs}}. \quad (20.34)$$

The undamped solution to this  $5 \times 5$  system of equations is given by the equation below Figure 20.8c (see equation 2.25), where the inverse matrix  $[\mathbf{L}^T \mathbf{L}]^{-1}$  is given by Schuster (1988 and 1989):

$$[\mathbf{L}^T \mathbf{L}]^{-1} = \begin{bmatrix} \frac{1}{l_1^{(0)2}} & -\frac{1}{l_1^{(0)2}} & 0 & 0 & 0 \\ -\frac{1}{l_1^{(0)2}} & \frac{1}{l_1^{(0)2}} + \frac{1}{l_2^{(0)2}} & -\frac{1}{l_2^{(0)2}} & 0 & 0 \\ 0 & -\frac{1}{l_2^{(0)2}} & \frac{1}{l_2^{(0)2}} + \frac{1}{l_3^{(0)2}} & -\frac{1}{l_3^{(0)2}} & 0 \\ 0 & 0 & -\frac{1}{l_3^{(0)2}} & \frac{1}{l_3^{(0)2}} + \frac{1}{l_4^{(0)2}} & -\frac{1}{l_4^{(0)2}} \\ 0 & 0 & 0 & -\frac{1}{l_4^{(0)2}} & \frac{1}{l_4^{(0)2}} + \frac{1}{l_5^{(0)2}} \end{bmatrix}. \quad (20.35)$$

If there are multiple sources on the surface then this gives rise to an overdetermined system of equations. Appendix 20.12 provides the  $10 \times 5$  system of traveltimes equations and  $[\mathbf{L}^T \mathbf{L}]^{-1}$  for two sources.

The regularized least squares solution is equivalent to that obtained by a grid search method that finds the maximum a posterior function for  $\lambda = 1$ . There are four steps for implementing regularized, aka damped, least squares inversion.

1. Define the objective function  $\epsilon$  as the sum of the weighted squared data residuals and the model penalty function

$$\epsilon = \frac{1}{2\sigma_{\mathbf{t}|\mathbf{s}}^2} \|\mathbf{L}\mathbf{s} - \mathbf{t}^{obs}\|^2 + \frac{\lambda}{2\sigma_{\mathbf{s}}^2} \mathbf{s}^T \mathbf{s}, \quad (20.36)$$

where the traveltimes and slowness terms have been weighted by their variances  $\sigma_{\mathbf{t}|\mathbf{s}}^2$  and  $\sigma_{\mathbf{s}}^2$ , respectively. Also,  $\lambda > 0$  is the scalar weight that controls the tradeoff between having a small data residual and a solution near the origin  $\mathbf{s} = 0$ . Alternatively, this model penalty term could be  $(\mathbf{s} - \mathbf{s}_0)^T (\mathbf{s} - \mathbf{s}_0)$  where  $\mathbf{s}_0$  is known from prior information and is denoted as the background slowness model. There are other regularizers such as the norm of spatial derivatives that penalize rough velocity models (Wang and Yang, 2021).

2. Take the negative exponential  $\epsilon$  of equation 20.36 to get  $\hat{\epsilon}$ , where

$$\begin{aligned} \hat{\epsilon} &= \exp\left(-\frac{1}{2\sigma_{\mathbf{t}|\mathbf{s}}^2} \|\mathbf{L}\mathbf{s} - \mathbf{t}^{obs}\|^2 - \frac{\lambda}{2\sigma_{\mathbf{s}}^2} \mathbf{s}^T \mathbf{s}\right), \\ &= \overbrace{\exp\left(-\frac{1}{2\sigma_{\mathbf{t}|\mathbf{s}}^2} \|\mathbf{L}\mathbf{s} - \mathbf{t}^{obs}\|^2\right)}^{\propto P(\mathbf{t}|\mathbf{s})} \times \overbrace{\exp\left(-\frac{\lambda}{2\sigma_{\mathbf{s}}^2} \mathbf{s}^T \mathbf{s}\right)}^{\propto P(\mathbf{s})}, \end{aligned} \quad (20.37)$$

where we have conveniently defined the terms in the exponent as either  $P(\mathbf{t}|\mathbf{s})$  or  $P(\mathbf{s})$ . The

## Damped Least Squares Traveltime Inversion

Step 1: Define Objective  $\varepsilon = \overset{\text{data misfit}}{\frac{1}{2}\|\mathbf{t}-\mathbf{L}\mathbf{s}\|^2} + \overset{\text{model penalty}}{\frac{1}{2}\|\mathbf{s}\|^2}$

Step 2: New Objective  $\varepsilon = \exp(-\frac{1}{2}\|\mathbf{t}-\mathbf{L}\mathbf{s}\|^2 - \frac{\lambda}{2}\|\mathbf{s}\|^2)$

Step 3:  $\text{argmax}_{\mathbf{s}} \varepsilon = \mathbf{s}^* = [\mathbf{L}^T\mathbf{L} + \lambda\mathbf{I}]^{-1}\mathbf{L}^T\mathbf{t}$

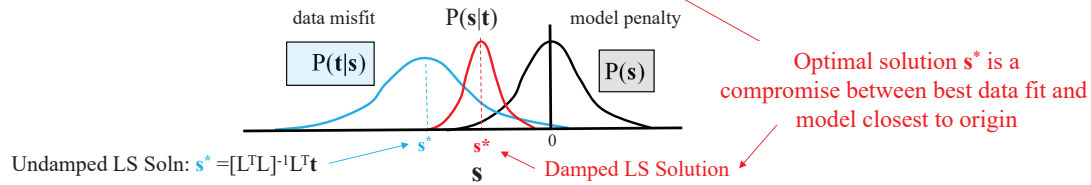


Figure 20.9: Three steps for damped least squares inversion of VSP traveltimes. The damped least squares solution  $\mathbf{s}^*$  maximizes the new objective function, plotted out as the red curve, and the undamped objective function is the blue curve. The prior belief about the model is plotted as the black curve, which is updated by the blue data misfit function to be the red curve.

stationary point  $\mathbf{s}^*$  of  $\hat{\epsilon}$  is the same as that for  $\epsilon$  in equation 20.36 because the negative exponential  $e^{-\epsilon}$  in equation 20.37 is a monotonically decreasing function for increasing values of  $\epsilon > 0$ . Therefore, the stationary point  $\mathbf{s}^*$  of equation 20.37 maximizes  $\hat{\epsilon}$  while the same point minimizes  $\epsilon$  in equation 20.36.

3. Therefore, the regularized, aka damped, least squares solution (see equation 2.19) that maximizes equation 20.37 is

$$\mathbf{s}^* = [\mathbf{L}^T \mathbf{L} + \frac{\lambda \sigma_{\mathbf{t}|\mathbf{s}}^2}{\sigma_{\mathbf{s}}^2} \mathbf{I}]^{-1} \mathbf{L}^T \mathbf{t}, \quad (20.38)$$

and plots out as the red curve at the bottom of Figure 20.9. The red curve is a compromise between maximizing the black model-penalty curve and the blue data-misfit curve. The degree of compromise is determined by the value of  $\frac{\lambda \sigma_{\mathbf{t}|\mathbf{s}}^2}{\sigma_{\mathbf{s}}^2}$ , where large values of  $\frac{\lambda \sigma_{\mathbf{t}|\mathbf{s}}^2}{\sigma_{\mathbf{s}}^2}$  emphasize honoring the model penalty term while small values favor a small data misfit.

The next section shows that the corresponding plots of the least squares objective functions are identical to the plots of the prior  $P(\mathbf{s})$ , likelihood  $P(\mathbf{t}|\mathbf{s})$  and a posterior  $P(\mathbf{s}|\mathbf{t})$  probability functions in Figure 28.4 for  $\sigma = \lambda = 1$  and certain conditions on the covariance matrices. The solid black curve in Figure 20.8c presents the velocity model using damped least squares inversion.

### Bayesian Inversion of VSP Traveltimes

The Bayesian analysis of the VSP traveltimes problem is to find the slowness model that maximizes the a posteriori probability function, i.e.

$$\begin{aligned} \mathbf{s}^* &= \underset{\mathbf{s}}{\operatorname{argmax}} P(\mathbf{s}|\mathbf{t}), \\ &= \underset{\mathbf{s}}{\operatorname{argmax}} P(\mathbf{t}|\mathbf{s})P(\mathbf{s}). \end{aligned} \quad (20.39)$$

Assuming a Gaussian distribution for the prior  $P(\mathbf{s})$  and likelihood  $P(\mathbf{t}|\mathbf{s})$  functions, equation 20.39 becomes

$$\begin{aligned} \mathbf{s}^* &= \underset{\mathbf{s}}{\operatorname{argmax}} \exp\left(-\frac{1}{2}(\mathbf{L}\mathbf{s} - \mathbf{t})^T \Sigma_{\mathbf{t}|\mathbf{s}}^{-1}(\mathbf{L}\mathbf{s} - \mathbf{t}) - \frac{1}{2}\mathbf{s}^T \Sigma_{\mathbf{s}}^{-1}\mathbf{s}\right), \\ &= \underset{\mathbf{s}}{\operatorname{argmax}} \exp\left(-\frac{1}{2\sigma_{\mathbf{t}|\mathbf{s}}^2}(\mathbf{L}\mathbf{s} - \mathbf{t})^T(\mathbf{L}\mathbf{s} - \mathbf{t}) - \frac{1}{2\sigma_{\mathbf{s}}^2}\mathbf{s}^T\mathbf{s}\right), \end{aligned} \quad (20.40)$$

where the last line assumes that the covariance matrix  $\Sigma_{\mathbf{t}|\mathbf{s}}^{-1}$  and  $\Sigma_{\mathbf{s}}^{-1}$  are diagonal matrices with the diagonal values, respectively, equal to  $1/\sigma_{\mathbf{t}|\mathbf{s}}^2$  and  $1/\sigma_{\mathbf{s}}^2$ . Therefore, the exponential argument for BI is the same as that for the objective function of regularized least squares inversion in equation 20.37.

For VSP data,  $\mathbf{d} = \mathbf{t}$  is the collection of direct-wave traveltimes picked from shot gathers excited by walkaway sources on the surface. The geophones are in the well and the 1D model  $\mathbf{m}$  is represented by an  $N \times 1$  vector, where each element represents the slowness value of a homogeneous layer. For the geometry in Figure 20.10a, the model consists of 5 layers and the data are the traveltimes from more than 20 near-offset sources. These traveltimes are inverted by unregularized inversion to give the profile represented by the dashed line in Figure 20.10c. The well log is used to compute the prior  $p(\mathbf{m})$ , where the penalty function is proportional to  $\|\mathbf{m} - \mathbf{m}_0\|^2$  and  $\mathbf{m}_0$  is the slowness profile from the sonic log.

Instead of expensive sampling, the MAP solution  $\mathbf{m}^*$  is computed by finding the regularized least squares inverse to the traveltimes equations. To reduce computational expenses, the sampling of points  $\mathbf{m}$  is only performed around the MAP points and over a restricted range of reasonable



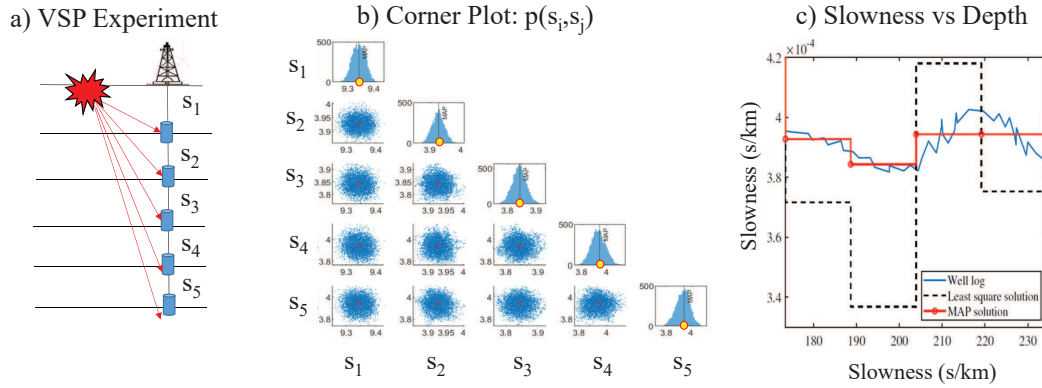


Figure 20.10: a) VSP geometry, b) Corner plot (see El Mohtar et al., 2021) of  $p(s_i, s_j)$  conditioned on fixing the other three slowness values, and c) inverted slowness profiles and sonic log. MAP points are the yellow dots along the diagonal of b). Image extracted from Qiao et al. (2022). An MCMC method with 10,000 iterations is used to compute the samples, where the 1st 1000 iterations are considered to be the burn-in samples and so were discarded.

slowness values dictated by the well log. The MAP points define the solid red line in Figure 20.10c. Appendix 20.15 describes how the statistical parameters are obtained for the prior and likelihood functions.

Once the MAP points are computed, then the sampled values are used to get the histograms around the MAP points in Figure 20.10b for different pairs of slowness values. These histograms give the mean and variance values of the MAP points denoted by the yellow points in Figure 20.10b.

Figure 20.11 illustrates the MCMC sampling efficiency when the VSP MAP points computed by least squares inversion are used as the starting points for the MCMC iterations. Only several thousand iterations, described by the red line in Figure 20.11, are required to get reliable values of the variances for the slowness model computed from VSP traveltime data. If the starting points for the MCMC iterations are perturbed from the actual MAP points by around 5%, then the number of MCMC iterations denoted by the black curve is around 20,000. Thus, the physics-informed MCMC iterations are nearly an order-of-magnitude more efficient than the standard MCMC method for this example.

Applying this physics-informed sampling strategy to all of the VSP data, five layers at a time, gives the estimate of the 40-layer velocity model shown by the blue curve in Figure 20.12. The standard deviation computed by the MCMC iterations are given by the error bars in blue, and the pseudo-code is on the left. The constraint introduced by the prior strongly pins the velocity estimate to the well-log curve shown in red.

### 20.6.3 Imaging Point Sources in a Hydrofrac Experiment

We now apply an efficient physics-informed Bayesian imaging of point sources to synthetic seismic data computed for a hydrofrac experiment. Here, the goal is to find the epicenters of point sources associated with localized rock failures induced by hydrofracturing. The efficiency of this procedure can be achieved by using either seismic migration or regularized least squares inversion to estimate the MAP points, followed by a localized grid search around the MAP points.

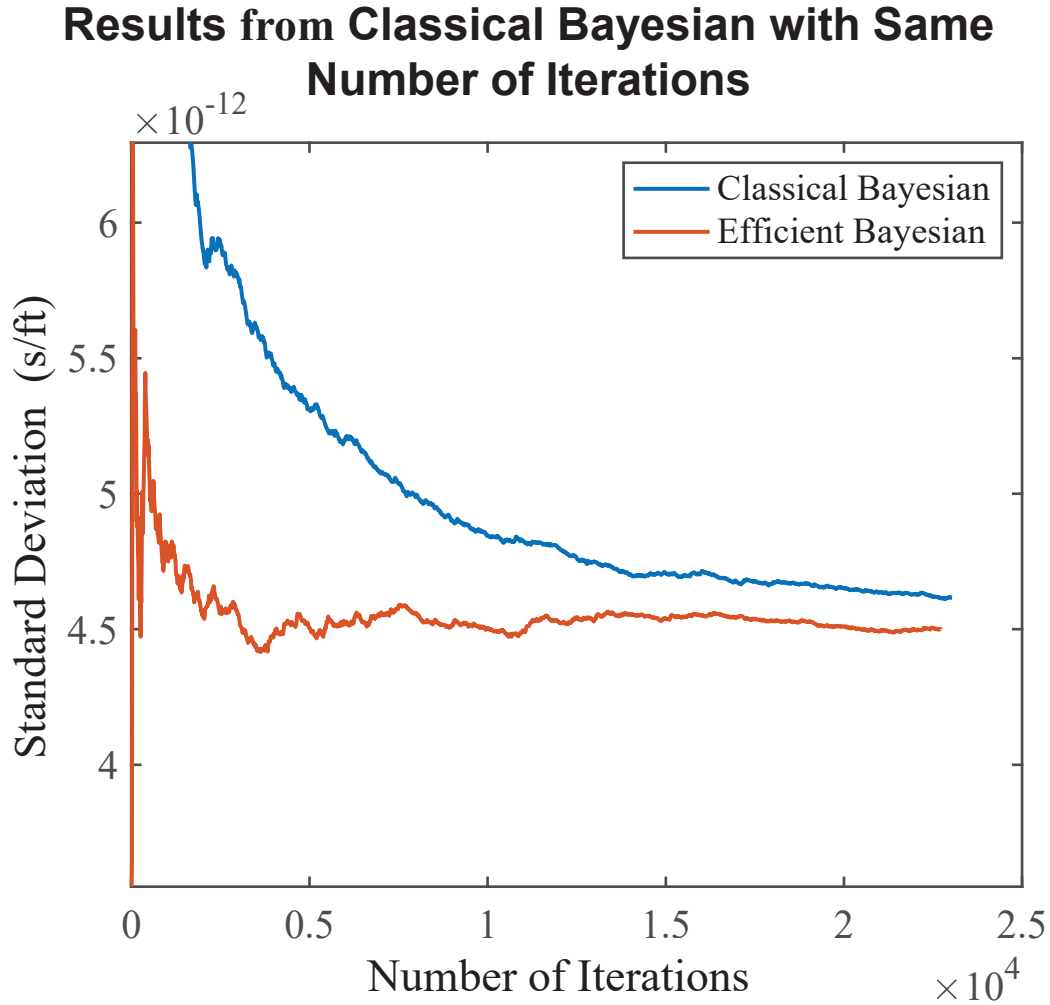


Figure 20.11: Red curve denotes the standard deviation of VSP slowness plotted against the iteration number for the physics-informed MCMC. The blue curve is the same except it describes the standard MCMC iterations where the starting slowness values are randomly selected.

**40 Layer Cases with 5 Layer-by- 5  
Layer Approach :**

1. Find the MAP solution
2. Conduct MCMC layer by layer
  - for j = 1:5: number of layers
    - for i = 1: number of iterations
      1. Sample for  $s_j$  and fixing slowness of another layers
      2. Conduct MCMC
3. Collect results from each layers

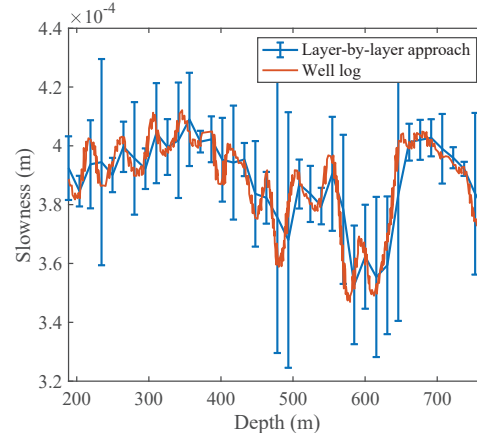


Figure 20.12: Pseudo-code for MCMC sampling by physics-informed estimate of the MAP points, and the results by MCMC sampling are shown on the right. Only several thousand MCMC iterations were required to estimate the variance of the slowness estimate for a selected slowness layer.

The following is the workflow for estimating the posterior probability function of the point-source locations.

1. Define a window in time where passive events are expected to exist. Each event is that from an assumed point source associated with a localized and impulsive rock failure due to hydrofracturing treatment. Figure 20.13a is an example of six point sources generating the passive data  $\mathbf{d}$  in Figure 20.13b. The start times of the event are unknown, but a focusing operator can be used to migrate the events to their place of origin, and therefore estimate their start time  $T_0$ . A wave-equation migration method or a wavepath migration method can be used with slant stacks (Yilmaz, 2001) to compute the migration image  $\mathbf{m}^{mig} \approx \mathbf{L}^T \mathbf{d}$  in Figure 20.13c. An inexpensive approximation, such as migration deconvolution (Hu et al., 2001; Yu et al., 2006; Aoki and Schuster, 2009), or an expensive least squares migration  $\mathbf{m}^{LSM} \approx [\mathbf{L}^T \mathbf{L}]^{-1} \mathbf{L}^T \mathbf{d}$  can be used to sharpen the estimate of the MAP points, which are the locations of the point sources. With less resolution compared to LSM, the location of MAP points can also be estimated from the migration image in Figure 20.13c. This is the physics-informed part of Bayesian inversion: samples for estimating the posterior are only taken within several wavelengths of the MAP points.
2. Each point in the subsurface is a potential source point, so an inexpensive Green's function  $G(x_g, z_g = 0, t | x_s, z_s, 0)$  for a ray-based direct wave is computed with the point source at the trial imaging point  $(x_s, z_s)$  and the receivers at  $(x_g, z_g)$ . This Green's function is inexpensively computed by a direct-wave tracing method since the direct wave is an order-of-magnitude stronger than the other events. Thus, there is no need for computing solutions to the wave equation to account for reflection events.

The Green's function for each subsurface point is stored (i.e, simply store the traveltimes to each receiver from the trial image point) for reuse with 4D monitoring.

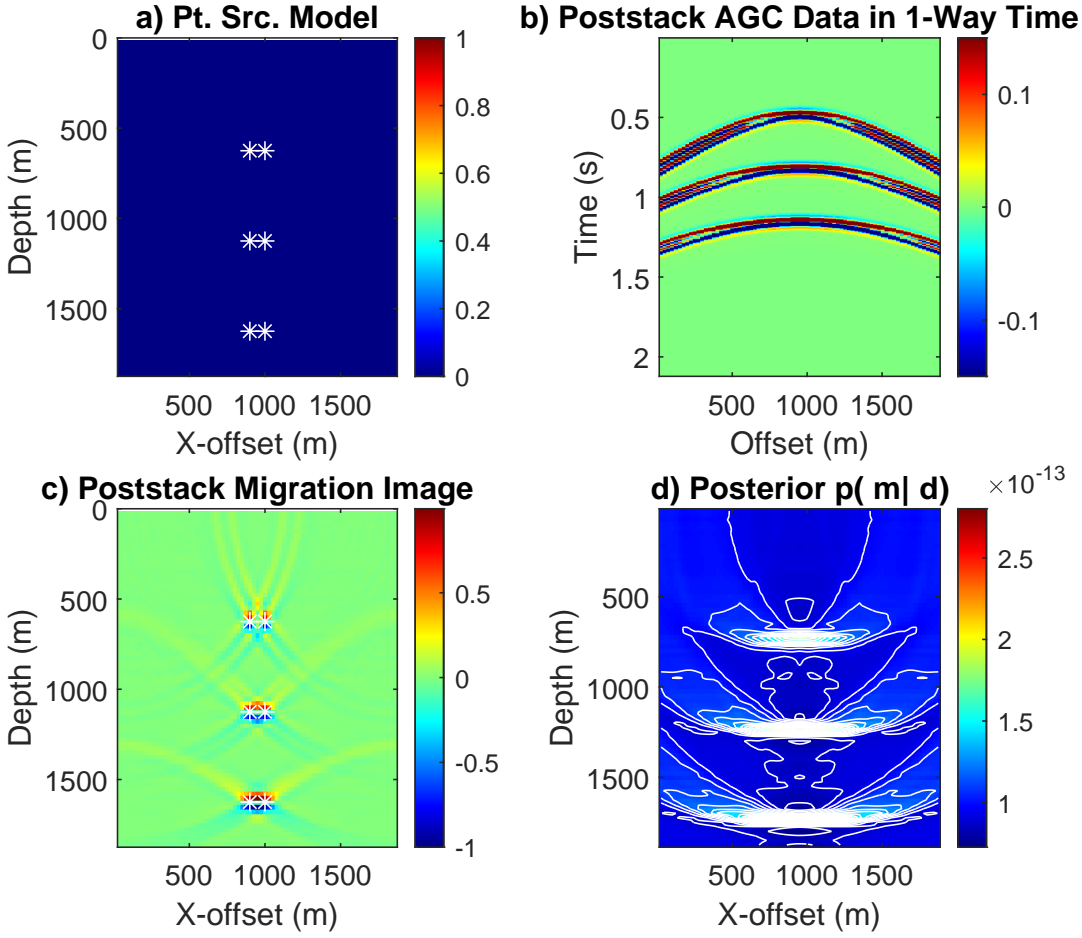


Figure 20.13: a) Point-source model with 6 point sources at white stars, b) recorded data, c) migration image, d) unnormalized posterior image. Image extracted from Qiao et al. (2022).

### 3. The sum of the squared residuals

$$r(x_s, z_s) = \sum_{(t, x_g)} \left[ \overbrace{d(x_g, t)}^{\text{recorded}} - \overbrace{G(x_g, z_g = 0, t | x_s, z_s, 0)}^{\text{predicted}} \right]^2, \quad (20.41)$$

is computed for each subsurface source point  $(x_s, z_s)$ . The sum of the squared residuals is plugged into the exponential  $\exp(-r(x_s, z_s)/\sigma^2)p(\mathbf{m})$  to give the values proportional to the posterior  $p(\mathbf{m}|\mathbf{d})$ , as plotted in Figure 20.13d. Here,  $p(\mathbf{m})$  can be a uniform distribution with restrictions for searching within a few wavelengths of the MAP points  $\mathbf{m}^* = (x_s^*, z_s^*)$ .

The final products of the above approach is an efficient method for computing both the source locations and probability estimates associated with monitoring source locations in a hydrofrac experiment. These calculations can be computed in real time with the reuse of the stored Green's functions (Qiao et al., 2022).

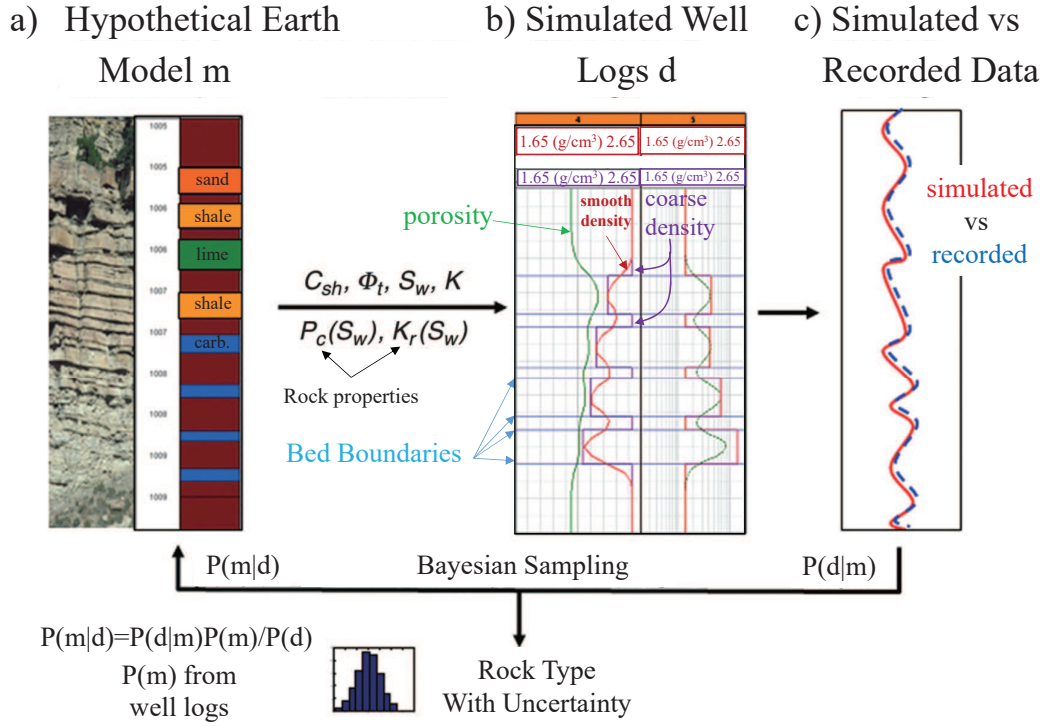


Figure 20.14: Workflow for estimating the uncertainty  $P(\mathbf{m}|\mathbf{d})$  in identifying the rock type associated with the petrophysical properties  $\mathbf{m}$  in each layer. a) Create synthetic earth model by selecting rock types and petrophysical model  $\mathbf{m}$  at each depth interval for the specified layers, b) create simulated data  $\mathbf{d}$  by forward modeling of earth model, c) Monte Carlo search of rock parameters for shales, limestones, sandstones, etc so synthetic well-log data  $\mathbf{d}$  for a specified data type matches recorded well data to give  $P(\mathbf{d}|\mathbf{m})$ . Bed boundaries are interpreted from recorded well log. Figure adapted from Xu et al. (2016).

#### 20.6.4 Predicting Rock Types and Uncertainty from Well Log Data

An example of Bayesian inversion applied to well-log data is presented in the workflow diagrams of Figure 20.14. The goal is to estimate the rock type at each depth level in the well from the well-log data  $\mathbf{d}$ . For one of the field cases to be discussed, there are 4 rock types denoted as  $rock \in \{RT1, RT2, RT3, RT4 = shale\}$ . Each rock type, such as shale, is associated with a statistical distribution of several rock properties measured by well logs and core analysis.

**Rock Properties  $\mathbf{m}$ .** The petrophysical rock properties at any point in the core are described by the model vector  $\mathbf{m}$ , which in this example is the  $7 \times 1$  vector

$$\mathbf{m} = (C_{cl}, S_w, \phi, \sigma_t, \sigma_b, \phi_N, \gamma)^T, \quad (20.42)$$

with elements denoting the values of the volumetric concentration of clay ( $C_{cl}$ ), water saturation ( $S_w$ ), total porosity ( $\phi$ ), apparent electric conductivity ( $\sigma_t$ ), bulk density ( $\rho_b$ ), neutron porosity ( $\phi_N$ ), and gamma ray ( $\gamma$ ). A rock type is associated with a range of values for these seven petrophysical properties so each petrophysical rock property is considered to be a RV with a specified mean and

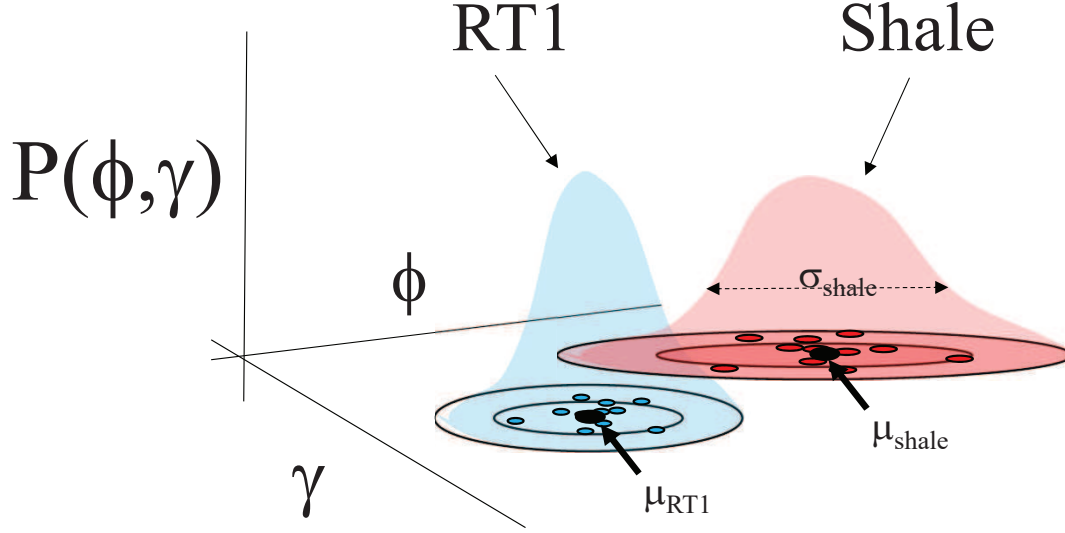


Figure 20.15: Idealized Gaussian curves  $P(\phi, \gamma)^{shale}$  and  $P(\phi, \gamma)^{RT1}$  fitted to idealized core measurements of the petrophysical properties of total porosity  $\phi$  and  $\gamma$  for the, respectively, shale and  $RT1$  rock types.

covariance matrix. For example, Figure 20.15 depicts an idealized graph of the joint probability function  $P(\phi, \gamma)$ , where a Gaussian function is fitted to the petrophysical measurements of total porosity  $\phi$  and  $\gamma$ . Note, there will be a different Gaussian for a different rock type, so the rock-type superscript  $rock$  will often be appended to the model vector  $\mathbf{m} \rightarrow \mathbf{m}^{rock}$  where, for example,  $rock \in \{RT1, RT2, RT3, RT4 = Shale\}$ ,

**Estimate of Prior  $P(\mathbf{m})^{rock}$ .** The prior distribution of the rock type is given by

$$P(\mathbf{m})^{rock} \approx \exp\left(-\frac{1}{2}(\mathbf{m} - \boldsymbol{\mu}^{rock})^T \mathbf{C}_{\mathbf{m}^{rock}}^{-1}(\mathbf{m} - \boldsymbol{\mu}^{rock})\right), \quad (20.43)$$

where  $\boldsymbol{\mu}^{rock}$  is the  $7 \times 1$  mean vector associated with the petrophysical measurements for one of the rock types  $rock \in \{RT1, RT2, RT3, RT4 = Shale\}$  and  $\mathbf{C}_{\mathbf{m}^{rock}}^{-1}$  is the model parameter covariance matrix. The mean values of petrophysical measurements can be obtained from analysis of cores extracted from regional wells. Figure 20.16 lists the mean values and standard deviations of five different rock parameters for each rock type in a field study.

**Well Log Data  $\mathbf{d}$ .** The data  $\mathbf{d}$  from which we predict the rock type correspond to different well-log measurements at each recording level along the vertical well. For the example in Xu et al. (2016), these well-log measurements at each depth level are packed into the  $1 \times 4$  data vector

$$\mathbf{d}(z) = (\sigma_t(z), \rho_b(z), \phi_N(z), \gamma(z)), \quad (20.44)$$

which records electrical conductivity  $\sigma_t(z)$ , bulk density  $\rho_b(z)$ , neutron porosity  $\phi_N(z)$ , and the gamma ray  $\gamma(z)$  value at each depth point  $z$  along the vertical well. To reduce notation complexity, the depth coordinate  $z$  will remain silent and the recorded well-log vector to be inverted is denoted

	$\phi$ (frac)	$k$ (mD)	Dean-Stark $S_w$ (frac)	Dean-Stark BVW (frac)	$C_{cl}$ (frac)
RT-1	$0.231 \pm 0.019$	$1255 \pm 455$	$0.064 \pm 0.036$	$0.014 \pm 0.006$	$0.006 \pm 0.003$
RT-2	$0.215 \pm 0.014$	$358 \pm 125$	$0.092 \pm 0.019$	$0.020 \pm 0.004$	$0.018 \pm 0.005$
RT-3	$0.140 \pm 0.023$	$52.5 \pm 27.9$	$0.337 \pm 0.105$	$0.045 \pm 0.007$	$0.051 \pm 0.018$
RT-4	$0.065 \pm 0.016$	$3.2 \pm 2$	$0.70 \pm 0.22$	$0.056 \pm 0.005$	$0.119 \pm 0.027$

Figure 20.16: Means  $\mu^{rock}$  and standard deviations  $\sigma^{rock}$  of the total porosity ( $\phi$ ), absolute permeability ( $k$ ), Dean-Stark water saturation ( $S_w$ ), bulk volume of water (BVW), and volumetric clay concentration ( $C_{cl}$ ) for each rock type ( $rock \in \{RT1, RT2, RT3, RT4\}$ ) in a field study. These data were obtained by analysis of drill cores extracted from many wells in the area of interest. Figure adapted from Xu et al. (2016).

as  $\mathbf{d}_0$ .

**Simulation of Well Log Data  $\mathbf{d}$ .** To predict the rock type associated with a recorded well log, hypothetical rock models  $\mathbf{m}$  are used to generate synthetic data  $\mathbf{d}$  until the optimal model  $\mathbf{m}_0$  closely matches the recorded well-log data  $\mathbf{d}_0$ . Well log data are simulated by the forward modeling operator  $\mathbf{G}$ , where

$$\mathbf{G}(\mathbf{m}) = \mathbf{d}, \quad (20.45)$$

and  $\mathbf{G}(\mathbf{m})$  represents fast numerical modeling codes for simulating, e.g., electrical conductivity from rock physics models. The different earth models are drawn from a stochastic ensemble of models so that the, e.g.,  $7 \times 1$  vectors  $\mathbf{m}$  and  $4 \times 1$  vectors  $\mathbf{d}$  are considered to be random vectors drawn from Gaussian-like distributions. The model and data distributions are characterized by the mean vectors  $\mu_{\mathbf{m}}$  and  $\mu_{\mathbf{d}}$ , where the covariance matrices are denoted by the  $\mathbf{C}_{\mathbf{m}}$  and matrices  $\mathbf{C}_{\mathbf{d}}$ . Note, the elliptical contours in Figure 20.15 suggest that there is a strong correlation between the RVs  $\phi$  and  $\gamma$ .

**Workflow for Bayesian Prediction of Rock Type and Uncertainty  $P(\mathbf{m}|\mathbf{d})$ .** The following steps describe the steps for Bayesian inversion of the rock type in depth and its uncertainty  $P(\mathbf{m}^{rock}|\mathbf{d})$  from a recorded well log  $\mathbf{d}_0$  (Xu et al., 2016). In this case, Xu et al. consider either 4  $\{RT1, RT2, RT3, RT4 = Shale\}$  or 5  $\{RT1, RT2, RT3, RT4, Shale\}$  rock types in this well.

1. **Construct Ensemble of Earth Models.** For an earth model with  $L$  layers and five rock types, there are  $5^L$  hypothetical vertical rock-type distributions to be tested during inference. An efficient MCMC sampling technique is used to infer the most likely rock-type distributions that honor the available well logs (Yang and Torres-Verdin, 2011). The recorded well log  $\mathbf{d}_0$  indicates the small number of facies zones and their boundaries that consist of just one of the possible rock types. For example, the blue horizontal lines in Figure 20.14b indicate the boundaries of the different facies zones; and log cores in Figure 20.14a indicate the composition of rock properties  $\mathbf{m}$  and rock type at each depth level.
2. **Zonation Strategy.** The interval of interest in the well can be divided into different zones, where each zone might correspond to one of the designated rock types. For the deltaic example in Xu et al. (2016), the reservoir unit under analysis comprised a total depth interval of 260 ft, which is a scale associated with the resolution of the seismic amplitude data. They used gamma

ray, bulk density, and resistivity logs to segment the reservoir unit into 80 petrophysical zones with an average thickness of approximately 2.9 ft. Most zones exhibit thicknesses ranging from 1.0 to 4 ft, which are mostly a mixture of more than one rock type. One method for assigning rock type to the mixture zone is to select the rock type as the one with the largest volume in that zone.

3. **Likelihood Function**  $P(\mathbf{d}_0|\mathbf{m}^{rock})$ . The likelihood function  $P(\mathbf{d}_0|\mathbf{m}^{rock})$  gives the probability of well-log data  $\mathbf{d}_0$  being recorded in the rock-type formation denoted as  $rock \in \{RT1, RT2, RT3, Shale\}$ . It can be calculated by simulating well data by using the forward modeling operation  $\mathbf{d} = \mathbf{G}(\mathbf{m}^{rock})$  and then assessing the mismatch between the measured data  $\mathbf{d}_0$  and simulations  $\mathbf{d}$ :

$$P(\mathbf{d}_0|\mathbf{m}^{rock}) \approx \exp\left(-\frac{1}{2}[\mathbf{G}(\mathbf{m}^{rock}) - \mathbf{d}_0]^T \mathbf{C}_d^{-1} [\mathbf{G}(\mathbf{m}^{rock}) - \mathbf{d}_0]\right), \quad (20.46)$$

where  $\mathbf{C}_d$  is the data covariance matrix and the data  $\mathbf{d}_0$  is a RV vector because it contains Gaussian noise<sup>4</sup>. The  $\mathbf{m}^{rock}$  model might consist of a string of different rock types over a specified depth interval such as shown by the colored rectangles in Figure 20.14a. Xu et al. (2016) indicate that they classified hybrid rock types over zones as narrow as 1 to 5 feet in thickness.

According to Xu et al. (2016): *Once a vertical rock-type distribution with associated petrophysical properties is sampled and populated into the predefined multilayered earth model, well logs are then numerically simulated and compared with field logs to test the hypothesis. A hypothesis of rock-type realization is only acceptable when all available well logs acquired from the field are reproduced to a certain fitting criteria with fast well-log forward modeling given the hypothetical rock-type realization. Following the aforementioned procedure, we obtain enough rock-type realizations to quantify the posteriori distribution. We stop the hypothesis testing when the realizations follow the same distribution.* The synthetic modeling step is illustrated in Figures 20.14a-20.14b where well-log data are modeled for rocks with different hypothetical properties.

4. **Prediction of Rock Type from  $\mathbf{d}_0$  by Bayes' Theorem.** Bayes' Theorem can then be used to determine the likely rock type of the recorded well-log data  $\mathbf{d}_0$  by using a stochastic search to get the optimal  $rock^*$  type at each depth level along the well:

$$\begin{aligned} rock^* &= \underset{rock \in \{RT1, RT2, RT3, RT4\}}{\operatorname{argmax}} P(\mathbf{m}^{rock}|\mathbf{d}_0), \\ &= \underset{rock \in \{RT1, RT2, RT3, RT4\}}{\operatorname{argmax}} P(\mathbf{d}_0|\mathbf{m}^{rock})P(\mathbf{m})^{rock}, \end{aligned} \quad (20.47)$$

where the *argmax* operation is applied to well-log data over the specified zone of a single rock type. The MAP estimate in equation 20.23 is used and the uncertainty function  $P(\mathbf{d}_0|\mathbf{m}^{rock})$  is obtained from step 3. The prior probability  $P(\mathbf{m})^{rock}$  is obtained from petrophysical measurements fitted to Gaussian curves as illustrated in Figures 20.15 and 20.17.

5. **Uncertainty of Rock Type Prediction.** The probability of predicting the correct rock type is given by Bayes' Theorem

$$P(\mathbf{m}^{rock^*}|\mathbf{d}_0) = \frac{P(\mathbf{d}_0|\mathbf{m})^{rock^*} P(\mathbf{m}^{rock^*})}{P(\mathbf{d}_0)}, \quad (20.48)$$

---

<sup>4</sup>According to Xu et al. (2016), they assume resistivity and permeability measurements to be represented by a lognormal distribution, whereas bulk density, neutron porosity, and gamma ray are represented by Gaussian distributions.



## Histogram of Gamma Ray Measurements For Different Rock Types

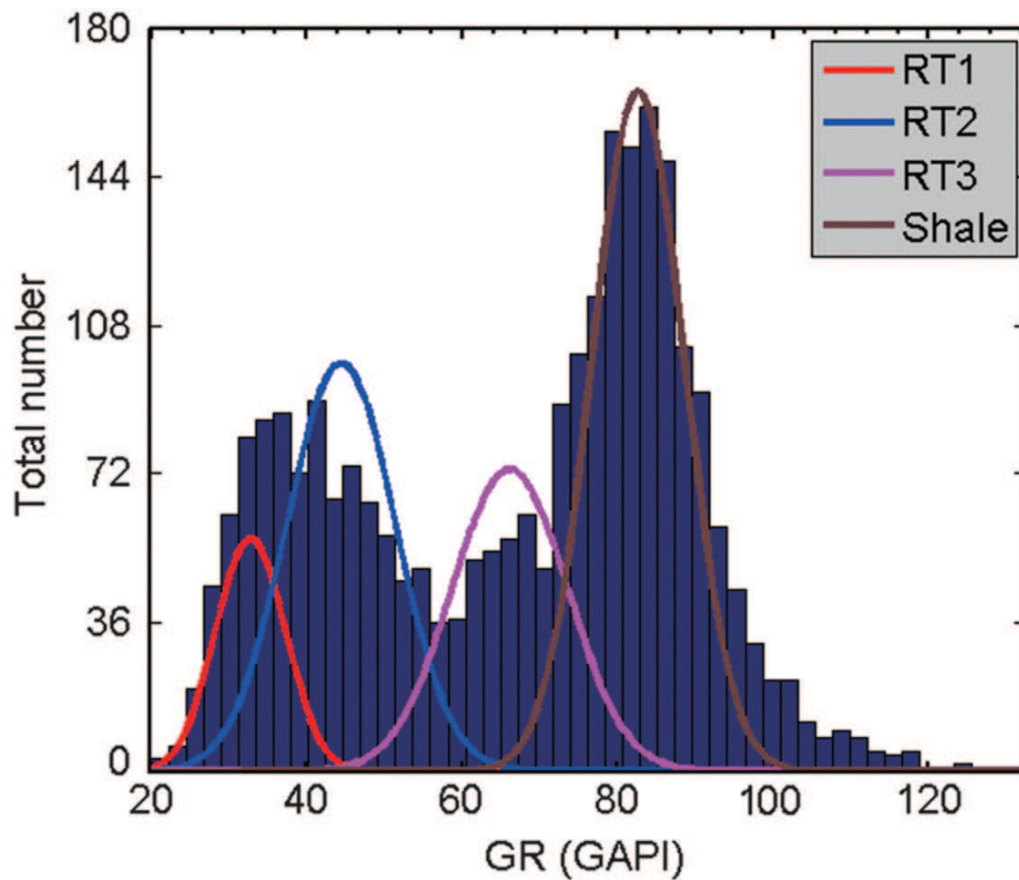


Figure 20.17: Gaussian curves fitted to Gamma Ray histograms for different rock types. Each Gaussian approximates  $P(\mathbf{d})^{rock}$  for  $rock \in \{RT1, RT2, RT3, RT4 = shale\}$  and  $\mathbf{d} = \gamma$ . Figure adapted from Xu et al. (2016).

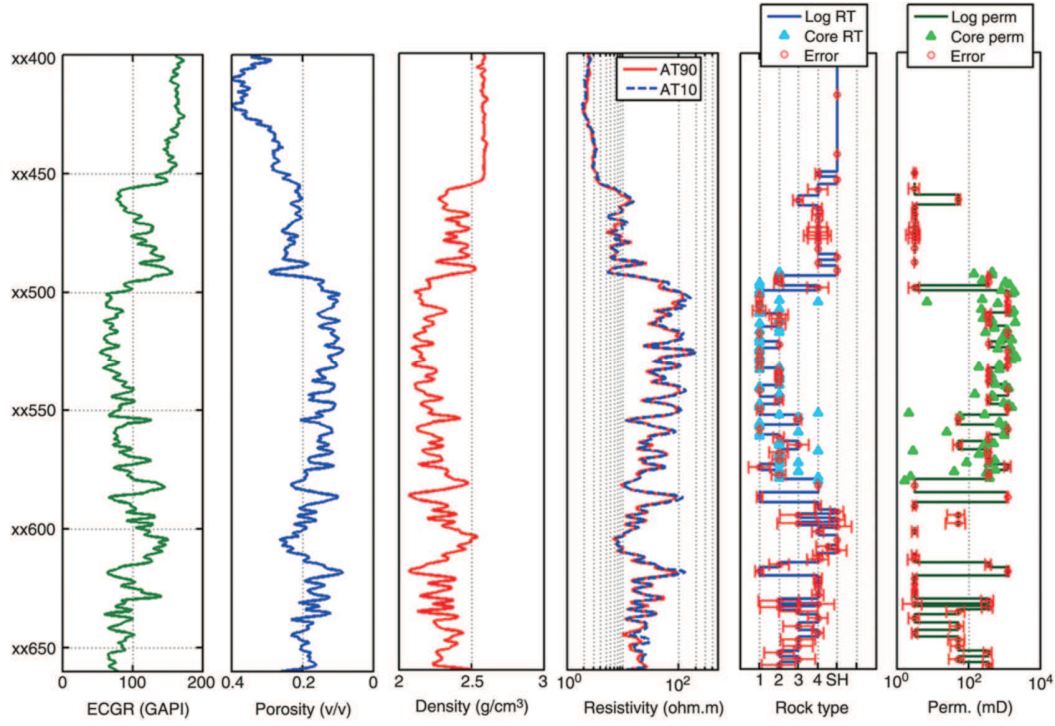


Figure 20.18: Field case of Bayesian rock typing and uncertainty quantification. From left to right, track 1: gamma ray; track 2: porosity; track 3: density; track 4: resistivity; track 5: rock type and uncertainty; track 6: permeability and uncertainty. The triangle marks in tracks 5 and 6 identify core measurements. Figure from Xu et al. (2016).

where  $P(\mathbf{d}_0)$  is the normalization term and  $\mathbf{m}^*$  is obtained from equation 20.47. The normalization term can be calculated by the product rule.

Xu et al. (2016) analyze the well logs in a sandstone unit located in a deltaic environment, and use Bayesian inversion to get the optimal model  $\mathbf{m}^{rock*}$ , rock types in each zone, and the posterior probability. This optimal model  $\mathbf{m}^{rock*}$  is plugged into the posterior probability to give the uncertainty  $P(\mathbf{m}^{rock*}|\mathbf{d}_0)$  in identifying rock types in each zone along the well. Four of the well logs are shown in Figure 20.18 alongside the rock types predicted by core analysis and Bayesian inversion.

For two thin intervals in a well they obtained the posterior probability function  $P(\mathbf{m}^{rock*}|\mathbf{d}_0) = P(\mathbf{d}_0|\mathbf{m}^{rock*})P(\mathbf{m})^{rock*}/P(\mathbf{d}_0)$  in Figure 20.19 by Bayesian inversion with a stochastic search. The comparison between the facies interpretation and the Bayesian derived rock types is shown in Figure 20.20. According to Xu et al. (2016), *The application of the new method to a field case indicated more than 77% agreement between the log- and core-derived rock types. Overall, the correlation between predicted permeability and core-measured permeability was improved by approximately 16% compared with conventional deterministic methods. In addition, the method quantified the uncertainty associated with rock-type identification and permeability estimation. The final distribution of maximum-likelihood rock types was consistent with the geologic framework and provided useful information for stratigraphic reservoir construction and modeling.*

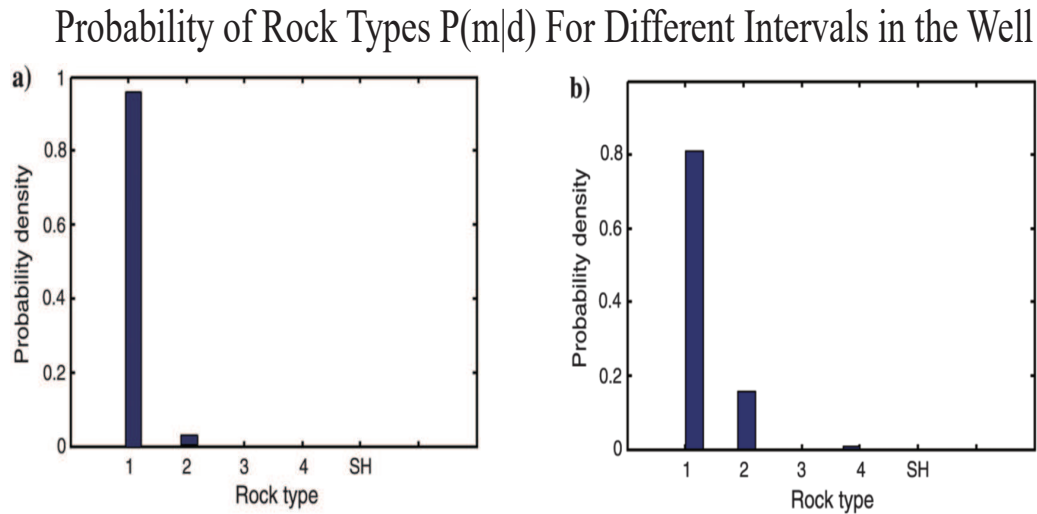


Figure 20.19: Uncertainty of rock types quantified in two petrophysical zones in the offshore deltaic field case. a) Thick bed at  $z = \text{xx}502$  ft and b) thin bed at  $z = \text{xx}508$  ft and there are 5 rock types. Figure adapted from Xu et al. (2016).

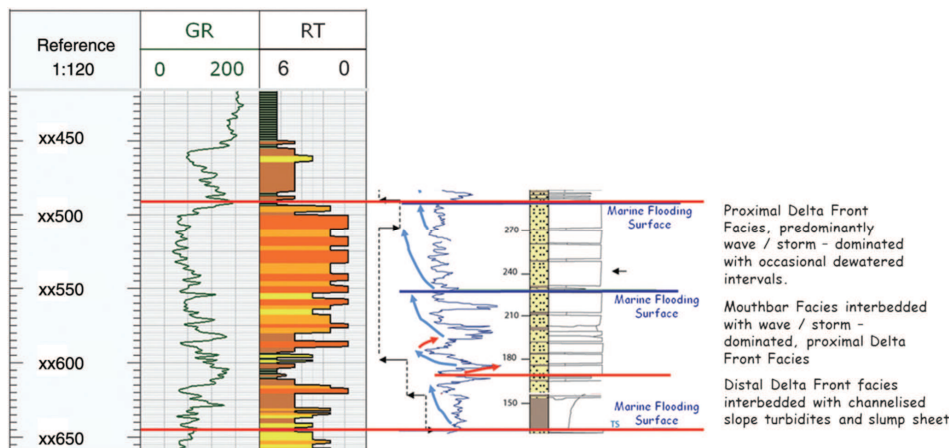


Figure 20.20: Comparison between the estimated rock types and outcrop-based facies description. Left panel: rock types (RT) inferred from the Bayesian method; right panel: facies description from Bowman (2004). Figure adapted from Xu et al. (2016).

### 20.6.5 Predicting Rock Types and Uncertainty from Well Logs and Seismic Data

Both seismic data and well-log measurements are now used to predict the facies type over a large area illuminated by seismic reflections. The goal is similar to that of the previous well-log example: determine the facies class of  $\mathbf{m}$  and its uncertainty  $P(\mathbf{m}|\mathbf{d}_0)$  from the geophysical data  $\mathbf{d}_0$ . However, Gonzalez et al. (2016) use well-log data and the seismic data in Figure 20.21a recorded over an area, and invert it to get the amplitude impedance (AI) in Figure 20.21b and the  $V_p/V_s$  section in Figure 20.21c. Now  $\mathbf{d} = (AI, V_p/V_s)$  instead of well-log data. The well data  $V_p$ ,  $V_s$ , *density* and fluid properties are used to generate realistic earth models  $\mathbf{m}$  and identify the different rock types associated with  $\mathbf{d} = (AI, V_p/V_s)$ .

Seismic forward modeling and inversion are used to compute many realizations of  $\mathbf{m}$  along the well to get histograms for  $\mathbf{d}$ . These histograms can then be used estimate the variance and mean vector for the Gaussian likelihood function  $P(\mathbf{d}|\mathbf{m})$ .

Gonzales et al. (2016) identified six distinct rock types from well core analysis: four types of shales with different silt contents, oil-saturated sandstone, and sand/shale laminar mixed with oil. Figure 20.22 shows a set of logs from the available well indicating the depth levels (colored samples) that define each of the six initial rock types. The workflow in Figure 20.22 illustrate the steps for computing the posterior  $P(\mathbf{m}|\mathbf{d})$  and the rock types from the inverted seismic data in Figure 20.21.

1. Log and core measurements from wells are used to get the marginal probability distributions  $P(\mathbf{m})^{rock}$  for different rock types.
2. The  $V_p$ ,  $V_s$ , and density logs in Figure 20.22 are used to generate a seismic model at the well, and this model is used to generate seismic data that are inverted for AI and  $V_p/V_s$  as a function of depth along the well. The range of values for AI and  $V_p/V_s$  for the shale 1 interval (dark green) are plotted as the dark green points in Figure 20.23.
3. The  $\mathbf{d} = (AI, V_p/V_s)$  generated in the previous step for a range of earth models can be used to estimate conditional probability functions  $P(\mathbf{m}|\mathbf{d})$  for each lithofluid facies. For example, the  $AI - V_p/V_s$  points for each rock type can be computed and plotted in the  $AI - V_p/V_s$  plane. A bivariate Gaussian (see Figure 20.23) is then fitted to these points to define the lithofluid facies likelihood function  $P(\mathbf{d}|\mathbf{m})$  for data along the well.
4. The  $P(\mathbf{d}|\mathbf{m})$  computed from well log data can now be applied to the inverted seismic data in Figure 20.21, and then Bayes' Theorem can be used to get the posterior probability  $P(\mathbf{m}|\mathbf{d})$  at every subsurface point in the seismic section. Similar to the Xu et al. (2016) example, both the facies type and its uncertainty can be now be computed at every subsurface point because  $\mathbf{d} = (AI, V_p/V_s)$  is known everywhere in the seismic section.

## 20.7 Summary

We reviewed important aspects of Bayes' Theorem where several examples are used to illuminate its meaning. The cancer example is used to geometrically interpret the scaling factor in Bayes' formula as the ratio of the areas in the  $X$  and  $Y$  sub-universes of events. The conditional probability  $P(X|Y)$  conditioned on the larger sub-universe  $Y$  will always be smaller than  $P(Y|X)$  for the smaller sub-universe  $X$  because there are fewer events in  $X$ . Fewer events in  $X$  means that there is a greater likelihood for the common events  $X \cup Y$  to occur. For the cancer example, Bayes' formula is also medically interpreted in terms of the number  $N^{TP}$  of people with true positives divided by the total number  $N^T$  of people who tested positive. Here,  $N^T = N^{TP} + N^{FP}$  can explain why a test with high value of  $P(T|C)$  can be an unreliable predictor  $0 < P(C|T) \ll 1$  of cancer in a largely healthy population.

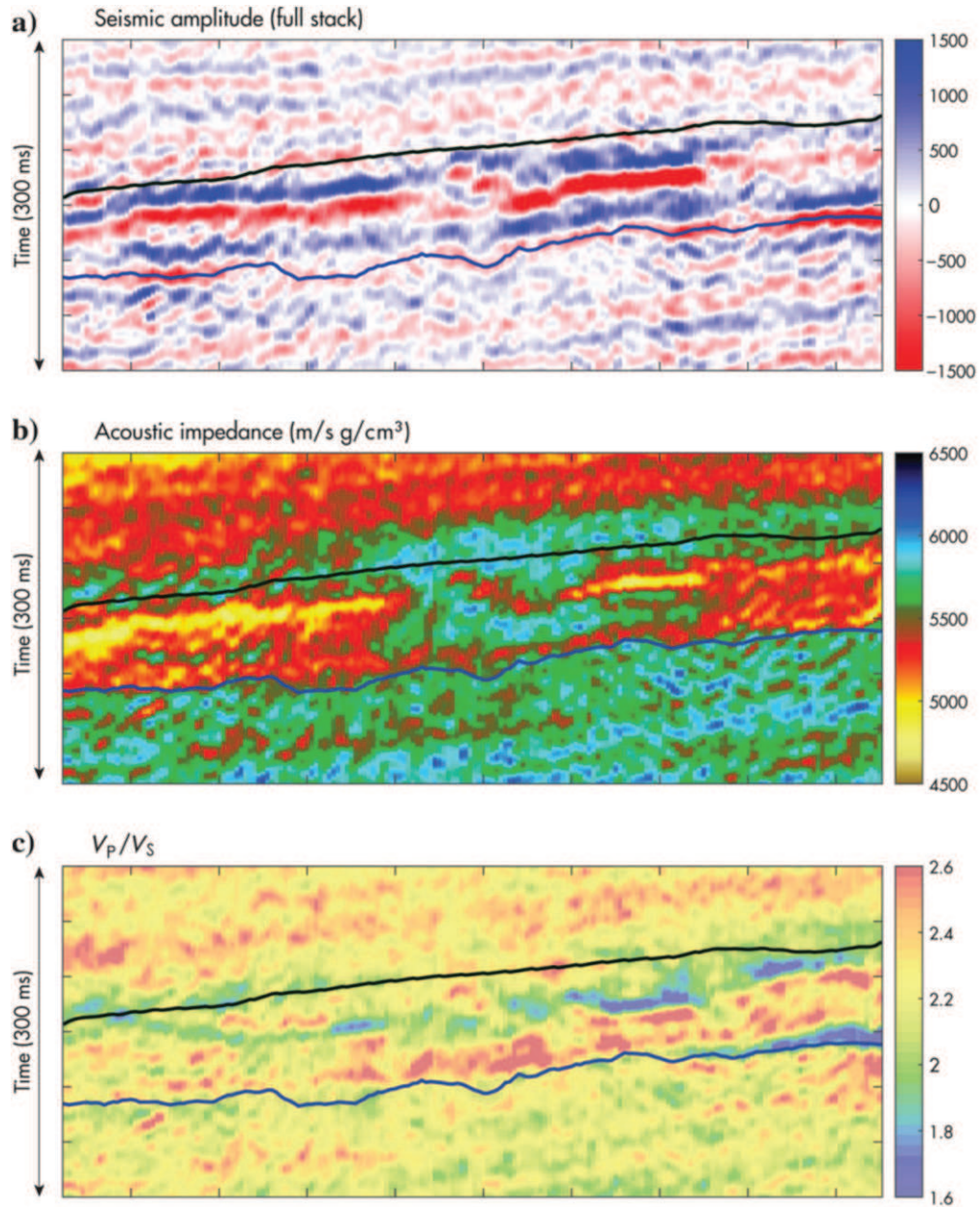


Figure 20.21: a) Stacked seismic section, b) AI section, and  $V_p/V_s$  section obtained from the seismic data. Figure adapted from Gonzales et al. (2016).



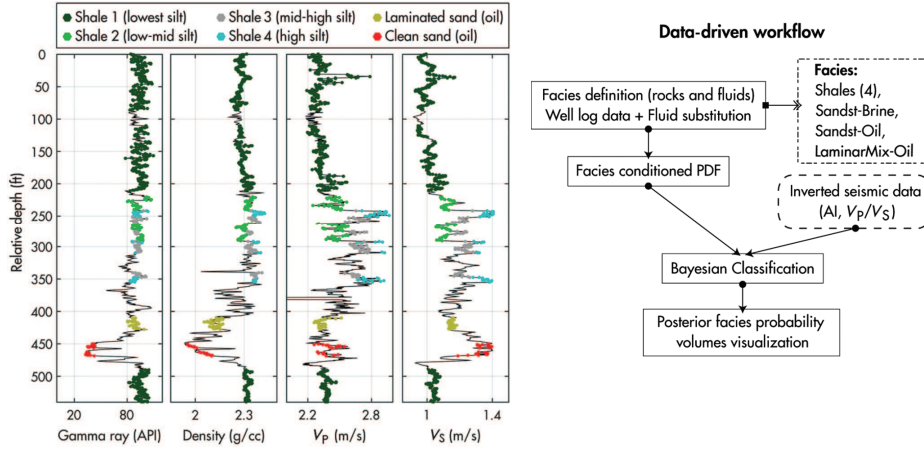


Figure 20.22: Log data on the left and workflow on the right. Workflow describes the steps for estimating the rock classes and uncertainties at each point in the subsurface from seismic data. The depth samples in the logs are color coded by rock types interpreted from petrophysical observations: four shales (increasing silt content from shale 1 to shale 4), laminated sand with hydrocarbons, and clean/blocky sand with hydrocarbons. From left to right: gamma ray (API), density ( $\text{g/cm}^3$ ), compressional velocity (m/ms), and shear velocity (m/ms). Figure from Gonzales et al. (2016).

The motivation for using Bayesian inversion or a regularizer in the least squares objective function is that we can incorporate prior beliefs or information into the final solution. This is important when there are few data points and many model parameters because it prevents overfitting. The optimal model then focuses on the really important features of the data. For example, assume you flip a coin just three times and each time it comes up heads. A classical MLE estimate of the next flip would overfit and predict a 100% chance of heads. In comparison, the Bayesian (or regularized inversion) use of the prior knowledge that, e.g.  $P(H) = 0.7$ , suggests a more reasonable estimate of the chance of flipping a head.

There are four steps to computing the posterior distribution from Bayes' Theorem:

1. Identify the independent events being investigated. For the well-log example in Xu et al. (2016), the model might consist of five types of rock  $\{RT1, RT2, RT3, RT4, Shale\}$ , and the data consist of the  $1 \times 4$  vector in equation 20.44 at each depth level.
2. Compute a frequency table for these events, and from this table compute  $P(\mathbf{m})^{rock}$ ,  $P(\mathbf{d}|\mathbf{m}^{rock})$  and  $P(\mathbf{d})$ . For geophysical analysis, the physics that connects the data with the model allows us to develop a fast forward modeling method to predict the data for each model  $\mathbf{m}$ . From these simulations, the covariance matrix and mean of the likelihood function  $P(\mathbf{d}|\mathbf{m}^{rock})$  can be estimated and then used in equation 20.46. This differs from the medical field where there often is not a rigorous modeling formula that connects  $\mathbf{m}$  with  $\mathbf{d}$ . In this case, the likelihood must often be estimated from a massive data base obtained from records of many patients.
3. Find the rock properties  $\mathbf{m}^*$  and rock type  $rock^*$  by a search method, e.g. equation 20.47, that best explains the new data  $\mathbf{d}_0$ .
4. Plug the formulas for  $P(\mathbf{d}|\mathbf{m}^{rock})$ ,  $P(\mathbf{d})$  and  $P(\mathbf{m})^{rock}$  into Bayes' formula to compute the posterior formula for  $P(\mathbf{m}|\mathbf{d}_0)$ . This can then be used to give the standard deviation for the estimate  $\mathbf{m}^*$  and rock type.

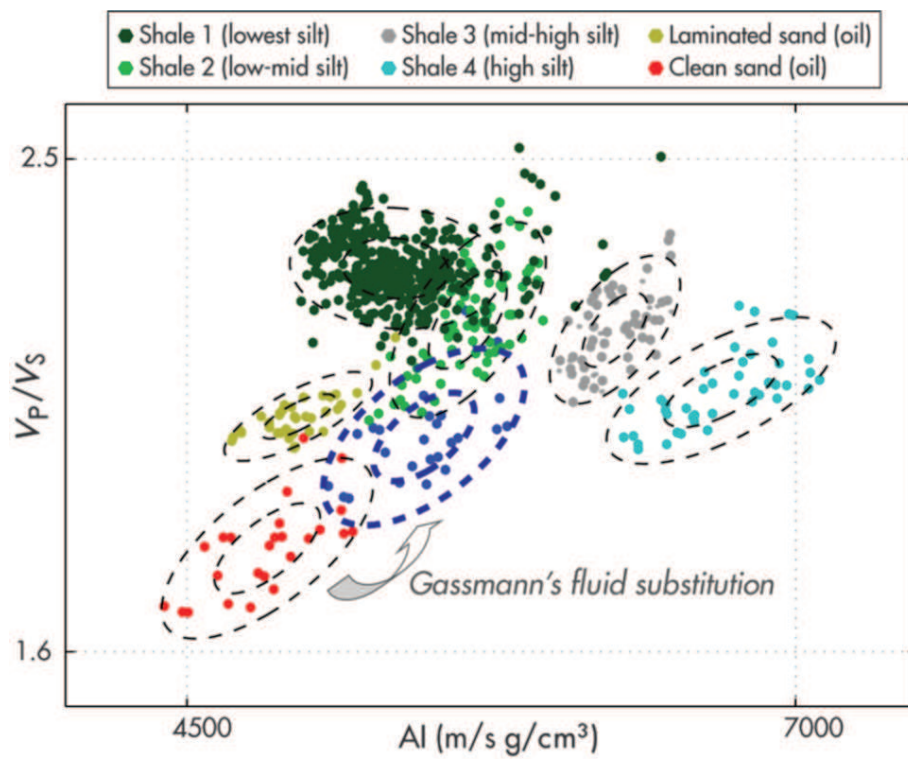


Figure 20.23: Bivariate plot of  $V_p/V_s$  plotted against AI for various points extracted from the processed synthetic data. The elliptical contours are those for the Gaussians best fitted to the data for each rock type. Figure from Gonzales et al. (2016).

The equivalence between Bayesian inversion and regularized least squares inversion is established. A penalty function, such as the KL regularizer, can be used to encourage the model's probability distribution to be the same as that for a specified one. The model penalty term for the regularized objective function encourages the model to not only fit the data but also appear similar to the characteristics of a preferred model. Finally, examples are presented that illustrate how Bayes' Theorem can be used to estimate source of an oil spill, determine the velocity model and uncertainty from VSP traveltime data, compute the location and its uncertainty of a hydrofrac source from passive seismic data, and predict rock types from geophysical data. A powerful tool for Bayesian analysis is the use of fast synthetic modeling that can accurately predict data from physical models. This can be used to generate distributions of data from distributions of physical models, and so partly alleviates the need for acquiring a large number of field data examples.

## 20.8 Exercises

1. Explain why the sum of the probabilities in the third column of Table 20.1 should be equal to 1.
2. Prove equation 20.38.
3. Show that the 1st-order finite-difference approximation to a 1st-order derivative given by

$$\bar{\mathbf{L}} = \begin{bmatrix} 1/l_1 & 0 & 0 \\ -1/l_1 & 1/l_2 & 0 \\ 0 & -1/l_2 & 1/l_3 \end{bmatrix}, \quad (20.49)$$

is the inverse to the finite-difference approximation of the integration operator

$$\mathbf{L} = \begin{bmatrix} l_1 & 0 & 0 \\ l_2 & l_2 & 0 \\ l_3 & l_3 & l_3 \end{bmatrix}. \quad (20.50)$$

Now show that  $\bar{\mathbf{L}}\bar{\mathbf{L}}^T$  is the inverse to  $\mathbf{L}^T\mathbf{L}$ . Write out the explicit matrix form for  $\bar{\mathbf{L}}\bar{\mathbf{L}}^T$  and compare it to the  $5 \times 5$  matrix  $[\mathbf{L}^T\mathbf{L}]^{-1}$  in equation 20.35.

4. Prove that  $\mathbf{L}$  in equation 20.50 can be expressed as

$$\mathbf{L} = \overbrace{\begin{bmatrix} l_1 & 0 & 0 \\ 0 & l_2 & 0 \\ 0 & 0 & l_3 \end{bmatrix}}^{\mathbf{D}_1} \overbrace{\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}}^{\mathcal{I}}, \quad (20.51)$$

where  $\mathcal{I}$  is the discrete approximation to an integration operator  $\int_0^z dz'$  with  $\Delta z' = 1$ . Now show that  $\mathcal{I}^{-1}\mathbf{D}_1^{-1}$  is the inverse of  $\mathbf{L} = \mathbf{D}_1\mathcal{I}$ . Here,  $\mathbf{D}_1^{-1}$  is the diagonal matrix with diagonal elements equal to the reciprocal of the diagonal elements of  $\mathbf{D}_1$ .

5. Show that  $\mathcal{I}^{-1}$  is a discrete approximation to a 1st-order derivative that has a lower bi-diagonal structure. Also show that  $\mathcal{I}^{-1}\mathcal{I} = \mathbf{I}$ , where  $\mathbf{I}$  is the identity matrix. Is it surprising that the inverse to an integration operator is a finite-difference approximation to a derivative?
6. Denote the top  $5 \times 5$  block of the  $\mathbf{L}$  matrix in equation 20.78 as  $\mathbf{L}_1$  and the underlying  $5 \times 5$  block as  $\mathbf{L}_2$ . Show that

$$\begin{aligned} [\mathbf{L}^T\mathbf{L}] &= [\mathbf{L}_1^T\mathbf{L}_1] + [\mathbf{L}_2^T\mathbf{L}_2], \\ &= \mathcal{I}^T[\mathbf{D}_2\mathbf{D}_2 + \mathbf{D}_1\mathbf{D}_1]\mathcal{I}. \end{aligned} \quad (20.52)$$



Now find the inverse matrix  $[\mathbf{L}^T \mathbf{L}]$  in terms of  $\mathcal{I}^{-1}$  and  $[\mathbf{D}_2 \mathbf{D}_2 + \mathbf{D}_1 \mathbf{D}_1]^{-1}$ . What is the discrete-approximation interpretation of  $\mathcal{I}^{-1} \mathcal{I}^{-T}$ ?

7. Change the probabilities in Table 20.1 so that  $P(C) = 0.7$  and  $P(\tilde{C}) = 0.3$ . Recompute the posteriors for a single cancer test and two cancer tests. Intuitively explain the results compared to the original posteriors for  $P(C) = 10^{-5}$  and  $P(\tilde{C}) = 0.99999$ .
8. **A Yunsong Huang Problem.** Assume  $N$  tests  $T_1, \dots, T_N$  that are conditionally independent given  $C$ , as well as given  $\tilde{C}$ , namely,

$$P(T_1, \dots, T_N | C) = \prod_i^N P(T_i | C), \quad (20.53)$$

$$P(T_1, \dots, T_N | \tilde{C}) = \prod_i^N P(T_i | \tilde{C}). \quad (20.54)$$

Assume

$$P(T_i | C) \equiv 1, \quad (20.55)$$

$$P(T_i | \tilde{C}) \triangleq \phi < 1, \quad \forall i. \quad (20.56)$$

We obtain

$$\begin{aligned} P(C | T_1, \dots, T_N) &= \frac{P(T_1, \dots, T_N | C) P(C)}{P(T_1, \dots, T_N | C) P(C) + P(T_1, \dots, T_N | \tilde{C}) P(\tilde{C})}, \\ &= \frac{\prod_i^N P(T_i | C) P(C)}{\prod_i^N P(T_i | C) P(C) + \prod_i^N P(T_i | \tilde{C}) P(\tilde{C})}, \end{aligned} \quad (20.57)$$

$$= \frac{P(C)}{P(C) + \phi^N P(\tilde{C})}, \quad (20.58)$$

$$\begin{aligned} &= \frac{1}{1 + \phi^N \frac{P(\tilde{C})}{P(C)}}, \\ &\approx 1 - \underbrace{\phi^N \frac{P(\tilde{C})}{P(C)}}_{P(\tilde{C} | T_1, \dots, T_N)}, \quad \text{if } N \gg 1. \end{aligned} \quad (20.59)$$

Here, equations 20.53 and 20.54 are used to yield equation 20.57; equations 20.55 and 20.56 are used to yield equation 20.58. Equation 20.59 says that, a large number of conditionally independent tests are able to strengthen the confidence of the tests in that the probability of false positive is geometrically reduced by  $\phi^N$ .

Go to the literature and extract the values of  $P(T_1 | C)$ ,  $P(T_1 | \tilde{C})$ ,  $P(C)$  and  $P(\tilde{C})$  for estimating the posteriors for various problems such as defect testing of manufactured goods, testing for diseases such as COVID, etc. Plot the graphs for  $P(C | T_1, T_2, \dots, T_N)$  versus iteration number  $N$  for these different problems. Explain their differences.

9. Go to the 22:21 clip of the video [https://www.youtube.com/watch?v=Pahyv9i\\_X2k](https://www.youtube.com/watch?v=Pahyv9i_X2k) and show that the mean and variance of the posterior are given by

$$\mu_1 = \frac{(\mu_0/\tau_0^2) + (y/\sigma^2)}{(1/\tau_0^2) + (1/\sigma^2)}; \quad \frac{1}{\tau_1^2} = \frac{1}{\tau_0^2} + \frac{1}{\sigma^2}. \quad (20.60)$$

10. We have three coins<sup>5</sup>, where the probability of getting heads for coin A is  $P(A) = 0.5$ , heads for coin B is  $P(B) = 0.6$  and heads for coin C is  $P(C) = 0.9$ . We have a drawer containing 4 coins: 2 of type A and 1 for type B and 1 for type C. I pick a coin at random from the drawer. Without showing you the coin, I flip it once and get heads. What is the probability it is type A? Type B? Type C? Create a table of posterior, likelihood and prior function values similar to that shown in Table 20.4.
11. Change the marginal probabilities in Table 20.1 so that  $P(C) = 0.4$  and  $P(\tilde{C}) = 0.6$ . Recompute  $P(C|T)$  and explain why the positive test result is much more reliable than before.
12. Let the letter  $S$  denote the event that you have been successfully vaccinated against the Covid virus. If  $Pf$  denotes the event of being vaccinated by the Pfizer vaccine, then the probability of avoiding the Covid virus is given by  $P(S|Pf) = 0.95$  according to statistical data. If  $1/2$  the population is vaccinated by Pfizer and the other half by the *Whatever* vaccine with  $P(S|W) = 0.25$ , what is the fraction  $P(S)$  of the total population that has successfully avoided the Covid virus by either the Pfizer or Whatever vaccines? What are the meanings of  $P(W)$  and  $P(Pf)$ ? From the previous problem, find the value of  $P(Pf|S)$  and describe in words what it means.
13. An ad hoc means<sup>6</sup> for updating an unknown prior  $P(M)$  is to estimate the posterior  $P(M|D) = P(D|M)P(M)/P(D)$  by assuming a non-informative prior. That is,  $P(M)$  is taken from a uniform distribution where all possible outcomes have the same probability. This is also known as the principle of insufficient reason. Each time you include a new experiment outcome, you can use the old  $P(M|D)^{old}$  to update your new prior  $P(M)^{new} \approx P(M|D)^{old}$ , which can be used to update your new posterior  $P(M|D)^{new} = P(M|D)^{new}P(M)^{new}/P(D)^{new}$  for a new flip. Here,  $M$  indicates the flip of a head,  $D$  indicates the next flip is a head, and  $\tilde{M}$  and  $\tilde{D}$  both indicate a tail, and  $P(D) = P(D|M)P(M) + P(D|\tilde{M})P(\tilde{M})$ .  
Now assume three experiments which result in three sequential head flips in a row. This suggest that  $P(D|M) = 1$ ,  $P(M) = 1$ ,  $P(D) = 1$  and  $P(M|D) = 1$ . Assume a new experiment flips a tail, so compute the new values for  $P(D)^{new}$ ,  $P(D|M)^{new}$  and  $P(M)^{new} \approx P(M|D)^{old}$ . These updated values are used to compute the new posterior  $P(M|D)$ . Repeat this process for 5 more coin flips where the outcome of each new coin flip is opposite from the previous flip. What is the final  $P(M|D)$ ?

## 20.9 Appendix: Bayes' Theorem Examples

Two examples of applying Bayes' Theorem are presented. The first one is an univariate example for predicting the chance of rain given the next day's weather: a priorcast! The second one is a multivariate example for assessing the chances of going biking as a function of the weather pattern.

### 20.9.1 Example: Rainy Weather

As an example from Wikipedia<sup>7</sup>, we define a  $P(S) = 0.40$  probability of rain on Sunday, a  $P(M) = 0.52$  probability of rain on Monday, and the conditional probability

$$P(M|S) = 0.10, \quad (20.61)$$

which are denoted by the numbers in black font seen in Figure 20.24a-20.24b. This says that there is a 10% chance that it will rain on Monday if it rains on Sunday. Here,  $S$  indicates a rain

<sup>5</sup>This problem is from the lecture notes of Orloff and Bloom's Class 11 at MIT.

<sup>6</sup>See <https://www.youtube.com/watch?v=R13BD8qKeTg>.

<sup>7</sup>[https://simple.wikipedia.org/wiki/Bayes%27\\_theorem](https://simple.wikipedia.org/wiki/Bayes%27_theorem)

event on Sunday and  $M$  indicates a rain event on Monday, and a tilde above the variable, e.g.  $\tilde{M}$  and  $\tilde{S}$ , indicates that the rain event did not happen on the day indicated by the letter. We also know that there is an  $100 \times P(M|\tilde{S}) = 80\%$  chance that it rains on Monday given that it did not rain on Sunday. This value can be obtained from the statistical records of rain. The probability  $P(M) = 0.52$  (green fonts in Figure 20.24a) of it raining on Monday can be obtained by employing the Law of Total Probability in Figure 20.24c and the given values of the conditional and marginal probabilities in Figure 20.24a-20.24b. This means that probability of it not raining on Monday is  $P(\tilde{M}) = 1 - P(M) = 1 - .52 = 0.48$ , which agrees with the value of  $P(\tilde{M}) = 0.48$  computed by the Law of Total Probability in Figure 20.24c.

## a) Marginal Probabilities

	no rain	rain
Sunday	$P(\tilde{S}) = 0.60$	$P(S) = 0.40$
Monday	$P(\tilde{M}) = 0.48$	$P(M) = 0.52$

## b) Conditional Probabilities

	Sunday no rain	Sunday rain
Monday no rain	$P(\tilde{M} \tilde{S}) = 0.20$	$P(\tilde{M} S) = 0.90$
Monday rain	$P(M \tilde{S}) = 0.80$	$P(M S) = 0.10$

## c) Total Law of Probability

$$P(M) = P(M|S)P(S) + P(M|\tilde{S})P(\tilde{S})$$

$$= 0.1 \times 0.4 + 0.8 \times 0.6 = 0.52$$

$$P(\tilde{M}) = P(\tilde{M}|S)P(S) + P(\tilde{M}|\tilde{S})P(\tilde{S})$$

$$= 0.9 \times 0.4 + 0.2 \times 0.6 = 0.48$$

## d) Bayes' Rule

$$P(S|M) = P(M|S)P(S)/P(M)$$

$$= 0.1 \times 0.4 / 0.52$$

$$= 0.0769$$

Figure 20.24: Values of the a) marginal and b) conditional probabilities for the simple rain example. The numbers in black font are given either by records or deduced from constraints such as  $P(M) + P(\tilde{M}) = 1$  or  $P(S) + P(\tilde{S}) = 1$ . The values in red or green fonts in c) are computed from the Law of Total Probability in c). Bayes' Rule in d) is used to compute the value of  $P(S|M) = 0.0769$ .

So, how can we determine the probability  $P(S|M)$  it rained on Sunday if we know that it rained the next day on Monday? The conditional probability  $P(S|M)$  can be considered as a *priorcast* which has reversed the letter ordering in equation 20.61, so we can use Bayes' Theorem

$$P(S|M) = \frac{P(M|S)P(S)}{P(M)},$$

$$= \frac{0.10 \cdot 0.40}{0.52} = 0.0769, \quad (20.62)$$

in Figure 20.24d to say there is a  $100 \times P(S|M) = 7.69\%$  chance it rained on Sunday despite the fact it rained the next day on Monday.

Wait a minute, how come  $P(S|M) \neq P(M|S)$  in equations 20.61-20.62? Shouldn't there be a symmetrical relationship? No, unless the chance  $P(M)$  for rain on Monday is the same as that  $P(S)$

for Sunday. Rearranging equation 20.61 shows that there is a reciprocal relationship in the ratios of the conditional probabilities:

$$\frac{P(M|S)}{P(S|M)} = \frac{P(M)}{P(S)}. \quad (20.63)$$

This means that  $P(M|S) = P(S|M)$  if there is an equal chance of rain on either Sunday or Monday. On the other hand, if there is a much greater chance of rain on a Monday than on a Sunday, i.e.  $P(M) \gg P(S)$ , then it makes sense that the chances of a Monday rain  $P(M|S) = 0.1$  should be greater than that of a Sunday rain  $P(S|M) = 0.0769$ .

### 20.9.2 Example: Biking and Weather

In the previous example, the RVs are binary and can take on only one of two values, e.g.  $M = 1$  (yes) or  $M = 0$  (no). Bayes' Theorem can also be applied to RVs that can take on more than 2 different values, e.g.  $W \in \{1, 2, 3\}$ . Let us now use Bayes' Theorem in an example where the decision to go biking  $B = 1$  (yes) or  $B = 0$  (no) is conditioned on the three types of weather:

$$\text{Sunny} : W = 1, \text{ Overcast} : W = 2, \text{ Rainy} : W = 3. \quad (20.64)$$

Assume that you want to go biking, but you need to decide if the weather is sunny enough. Most often, you go biking on sunny days and usually don't go biking on rainy days. The second column of Table 20.2 gives the weather forecast and the third column gives the decision of whether to go biking or not. There are three types of forecast and the input to each day is assigned to the RV  $W \in \{O, S, R\}$  associated with one of the three forecasts in equation 20.64, where  $O, S$  and  $R$  indicates overcast, sunny and rainy, respectively. The decision to go biking<sup>8</sup> is assigned the binary RV  $B \in \{B, \bar{B}\}$ , where  $B = 1$  indicates the decision to go biking and  $\bar{B} = 0$  indicates no biking.

There are two steps to implementing Bayes' Theorem for this example:

- Collect data and organize it into Table 20.2 for the training pairs  $(W, B)$ .
- Convert Table 20.2 into the frequency Table 20.3. These data can then be used to compute  $P(B|W)$  from Bayes' Theorem.

The weather frequencies  $P(O)$ ,  $P(S)$  and  $P(R)$  and biking frequency  $P(B) = 10/14$  are given in Table 20.3, and the conditional biking frequencies as a function of weather type can be obtained from Table 20.2. These are used to compute the estimates of  $P(B|W)$ , as demonstrated in equation 20.67.

The last row in Table 20.3 says that you went biking 10 out of the 14 available days, so that the probabilities of Biking and not Biking are

$$P(B) = 0.71; \text{ and not biking } P(\bar{B}) = 1 - P(B) = 0.29, \quad (20.65)$$

respectively. The last column of Table 20.3 suggests that the probabilities of Overcast, Rainy, and Sunny weather are

$$P(O) = 0.35, P(R) = 0.29; P(S) = 0.35, \quad (20.66)$$

respectively.

The last column in Table 20.2 says that on the 10 days of biking, only 3 of the days were sunny, i.e.

$$P(S|B) = 3/10 \approx 0.3. \quad (20.67)$$

---

<sup>8</sup><https://www.javatpoint.com/machine-learning-naive-bayes-classifier>

Table 20.2: Weather forecast &amp; bike decision.

Day	W: Weather Forecast	B: Biking?
0	Rainy	Yes
1	Sunny	Yes
2	Overcast	Yes
3	Overcast	Yes
4	Sunny	No
5	Rainy	Yes
6	Sunny	Yes
7	Overcast	Yes
8	Rainy	No
9	Sunny	No
10	Sunny	Yes
11	Rainy	No
12	Overcast	Yes
13	Overcast	Yes

Table 20.3: Weather Forecast Frequency.

Weather	Biking: Yes	Biking: No	Frequency
Overcast	5	0	$P(O) : 5/14=0.35$
Rainy	2	2	$P(R) : 4/14=0.29$
Sunny	3	2	$P(S) : 5/14=0.35$
Total	10	4	
Frequency	$P(B)=10/14=0.71$	$P(\bar{B})=4/14=0.29$	

What is the probability  $P(B|S)$  of it being sunny and then deciding to go biking? The answer can be obtained either by event counting in Table 20.2 or by Bayes' Theorem using the quantities in equations 20.65-20.67. Bayes' Theorem says that

$$P(B|S) = \frac{P(S|B)P(B)}{P(S)} = \frac{0.3 \times 0.71}{0.35} = 0.6, \quad (20.68)$$

where the first term on the left side expresses the conditional probability in terms of the sunny  $S$  and the biking  $B$  variables.

An alternative way for computing  $P(B|S)$  is to examine Table 20.2 and notice that there were 5 sunny days, and we decided to go biking on 3 of them. Thus,

$$P(B|S) = 3/5 = 0.6; \quad P(B|O) = 5/5 = 1; \quad P(B|R) = 2/4 = 0.5, \quad (20.69)$$

where  $P(B|S) = 3/5 = 0.6$  agrees with the estimate in equation 20.68. The other conditional probabilities were computed by counting events in Table 20.2. One of the dangers in estimating prior probabilities from a small sample size is that the sample size might be too small to get an accurate estimate of true probabilities. For example, if the bicycle was mostly in the shop for repairs for the June data in the table, then  $P(B)$  computed from June data is severely underestimated compared to a table that sampled data over a year.

Table 20.4 summarizes the values of the posterior probabilities that are used to compute  $P(W|B_1)$  for biking in different weather conditions in section 20.9.2. Instead of using the RV  $B$  to indicate biking, we will later use  $B_1$  because we will later need to use another RV, denoted as  $B_2$ , to indicate biking on another day. Note, the sum of unnormalized posterior values in the fifth column of Table 20.4 should sum to  $P(B_1)$ , i.e.,

$$P(B_1) = P(B_1|O)P(O) + P(B_1|R)P(R) + P(B_1|S)P(S), \quad (20.70)$$

because of the Law of Total Probability. Here, the events  $O$ ,  $R$ ,  $S$  are disjoint (it can't be sunny, rainy or overcast at the same time in an idealized world) and the union of these events explain any type of weather pattern. This is how the value of  $P(B_1)$  is obtained to compute the divisor for the posterior probability functions  $P(B_1|W)P(W)/P(B_1)$  in the last column.

Why did we go through the trouble of Bayes' Theorem to compute  $P(B|S)$  when we could have determined it from Table 20.2? The answer is that we might only have access to information in equations 20.65-20.67, but we do not have access to the entire contents of Table 20.2. That is, only the third column of numbers might have been given to us in order to compute  $P(B)$ ; we were not

Table 20.4: Summary of the probability mass, likelihood and posterior probability functions for the biking-weather data. Here,  $\{B_1, \tilde{B}_1\}$  and  $W \in \{O, R, S\}$  where  $B_1$  indicates the event of biking and  $\tilde{B}_1 = 0$  indicates the event of not biking. The hypothesis events for *overcast*, *sunny* and *rain* are denoted by, respectively,  $O$ ,  $S$  and  $R$ .

Evidence eq. 20.65	Hypothesis	Prior eq. 20.66	Likelihood Prior (eq. 20.69)	Unnormalized Posterior 1	Posterior 1
$P(B_1)$	W	$P(W)$	$P(B_1 W)$	$P(B_1 W)P(W)$	$P(W B_1)=P(B_1 W)P(W)/P(B_1)$
$P(B_1)=10/14$	O	$P(O)=0.35$	$P(B_1 O)=1.0$	$P(B_1 O)P(O)=0.35$	$P(B_1 O)P(O)/P(B_1)=0.49$
$P(B_1)=10/14$	R	$P(R)=0.29$	$P(B_1 R)=0.5$	$P(B_1 R)P(R)=0.145$	$P(B_1 R)P(R)/P(B_1)=0.203$
$P(B_1)=10/14$	S	$P(S)=0.35$	$P(B_1 S)=0.6$	$P(B_1 S)P(S)=0.21$	$P(B_1 S)P(S)/P(B_1)=0.294$
Total		$\Sigma=0.99$		$\Sigma=0.705$	$\Sigma=0.99$

given each weather forecast paired with the decision to go biking. Separately, the numbers in the 2nd column of Table 20.2 are given to us to get  $P(S)$ , but again the numbers are not paired up with a particular day or a biking decision. For the case of continuous variables, the posterior function is fitted to a Gaussian after suitable training, which can then be used to accurately predict the posterior for new input data.

## 20.10 Appendix: Naive Bayes' Theorem, Weather and Biking

Assume that the decision to go biking also depends on how windy it is: if it is too windy then one tends to not go biking. Table 20.5 is the same as Table 20.2, except it now lists the wind conditions as either windy  $X_2 = 1$  or not windy  $X_2 = 0$ . In this case  $X_2$  is a binary RV  $\in \{0, 1\}$ . Therefore, if  $X_1$  is the RV associated with the weather forecast and  $X_2$  is the one for the wind conditions then the multinomial conditional PMF is  $P(X_1, X_2|Y)$  rather than  $P(X|Y)$  for Table 20.2. Therefore, the Naive Bayes' equation 20.16 becomes

$$P(Y|X_1, X_2) = \frac{P(X_1|Y)P(X_2|Y)P(Y)}{P(X_1, X_2)}. \quad (20.71)$$

The numbers in Table 20.5 can be used to get  $P(X_1|Y)$  and  $P(X_2|Y)$ , similar to the procedure for getting the conditional probabilities in Table 20.2. The marginal PMFs  $P(X_1)$  and  $P(X_2)$  can be obtained from a table similar to that of Table 20.3, except it would contain the wind conditions as well for each day.

## 20.11 Appendix: Kullback-Leibler Regularizer

An objective function  $\epsilon$  is usually composed of a data loss function such as  $\sum_n (d^{(n)} - \tilde{d}(\mathbf{w})^{(n)})^2$  and a model penalty function  $R(\mathbf{w})$ :

$$\epsilon = \frac{1}{2} \sum_n (d^{(n)} - \tilde{d}(\mathbf{w})^{(n)})^2 + \lambda R(\mathbf{w}), \quad (20.72)$$

where  $d(\mathbf{w})^{(n)}$  is the predicted data for the  $n^{th}$  training example,  $\mathbf{w}$  represents the model parameters and the summation in  $n$  is over training examples. The damping parameter is  $\lambda$ , where large values

Table 20.5: Weather forecast, wind &amp; bike decision.

Day	$X_1$ : Weather Forecast	$X_2$ : Windy	Y: Biking?
0	Rainy	Not Windy	Yes
1	Sunny	Not Windy	Yes
2	Overcast	Not Windy	Yes
3	Overcast	Not Windy	Yes
4	Sunny	Windy	No
5	Rainy	Not Windy	Yes
6	Sunny	Not Windy	Yes
7	Overcast	Not Windy	Yes
8	Rainy	Windy	No
9	Sunny	Windy	No
10	Sunny	Not Windy	Yes
11	Rainy	Not Windy	No
12	Overcast	Not Windy	Yes
13	Overcast	Not Windy	Yes

of  $\lambda$  prioritize models that minimize the model penalty term and smaller  $\lambda$  values prefer models that minimize the data loss function. The rationale for this regularization is that ill-posed optimization problems are often characterized by many models that almost fit the same data. Of all these possibilities we might prefer a model that has certain smoothness properties or one that is close to the property of a preferred model. An example of such a property might be that the mean of the model parameters should be small, or perhaps the  $j^{\text{th}}$  activation output  $a_j(\mathbf{x}^{(i)})$  of a neural network layer should have a mean that is small; here, the superscript for the layer index is silent and  $\mathbf{x}^{(i)}$  represents the input data into that layer for the  $i^{\text{th}}$  training example.

The average value of the output of an activation layer is defined as

$$\hat{\rho} = \frac{1}{M} \sum_{j=1}^M [a_j(\mathbf{x}^{(i)})], \quad (20.73)$$

for the  $i^{\text{th}}$  training example. The motivation for using a regularizer that promotes a small value of  $\hat{\rho}$  is that it acts as a sparsity constraint for the NN model. This means less complex models and encourages the trained network to focus on the really important features, highly reducing the risk of overfitting (Hinton, 2013; Berthelot et al., 2018). One such regularizer is the Kullback-Leibler (KL) regularization term described below<sup>9</sup>.

The goal of the KL regularizer is to (approximately) enforce the constraint

$$\hat{\rho}_j = \rho, \quad (20.74)$$

where  $\rho$  is a 'sparsity parameter' that is close to zero (say  $\rho = 0.5$ ). This means that the KL penalty term will prefer models in which the average activation of each hidden neuron  $j$  to be close to 0.5 (say). The consequence of enforcing equation 20.74 is that most of the hidden unit's activations will be near 0, and only a few, i.e. sparse, number of nodes will be strongly activated.

The KL regularizer  $R(\mathbf{w})$  that enforces sparsity is

$$R(\mathbf{w}) = \sum_{j=1}^M \overbrace{\rho \log \frac{\rho}{\hat{\rho}_j} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_j}}^{\text{KL}(\rho || \hat{\rho}_j)}, \quad (20.75)$$

<sup>9</sup>[view-source:http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/](http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/)

where the summation in  $j$  is over the number of neurons  $M$  in the hidden layer, and this sparsity regularizer can be applied to any specified number of layers. A plot of  $\text{KL}(x||x_0) = |\ln(x/x_0)|$  versus  $x$  is in Figure 20.25, and shows the minimum at  $x_0 = 0.5$ , with steep sides on either side of this minimum point  $x_0$ .

Here,  $\text{KL}(\rho||\hat{\rho}_j)$  is the Kullback-Leibler (KL) divergence between a binary, i.e. Bernoulli, random variable with mean  $\rho$  and a binary random variable with mean  $\hat{\rho}_j$ . KL divergence is a standard function for measuring the difference between two different distributions.

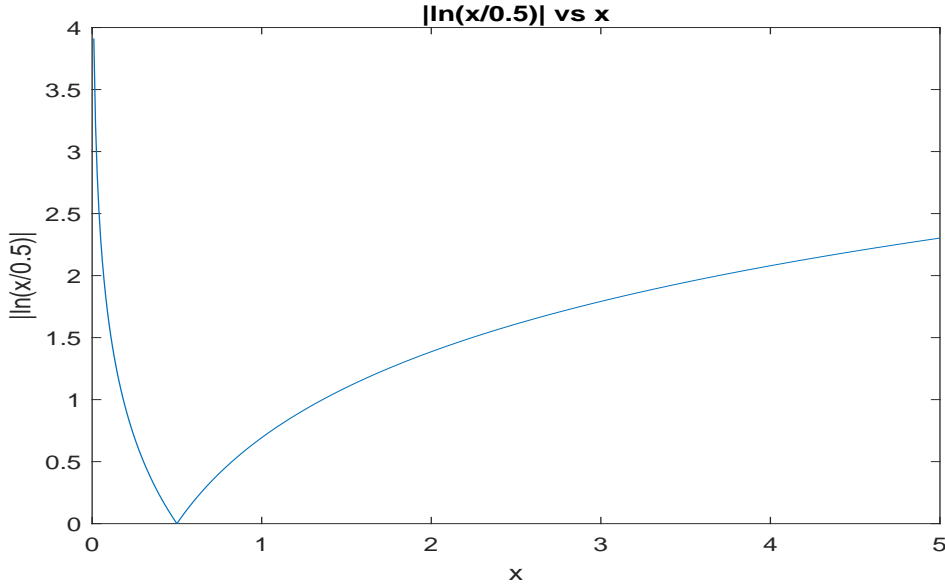


Figure 20.25:  $|\ln(x/x_0)|$  penalty function plotted against  $x$  for  $x_0 = 0.5$ .

The KL penalty function has the property that  $\text{KL}(\rho||\hat{\rho}_j) = 0$  if  $\hat{\rho}_j = \rho$ , and otherwise it increases monotonically as  $\hat{\rho}_j$  diverges from  $\rho$ . For example, in Figure 20.25,  $x_0 = \rho = 0.5$  and  $\text{KL}(\rho||\hat{\rho}_j) = |\ln(x/x_0)|$  is plotted for a range of values of  $\hat{\rho}_j = x$ : We see that the KL-divergence reaches its minimum of 0 at  $\hat{\rho}_j = \rho = x_0 = 0.5$ , and blows up (approaches  $\infty$ ) as  $\hat{\rho}_j$  approaches 0 or 1. Thus, minimizing this penalty term has the effect of causing  $\hat{\rho}_j = x$  to be close to the specified  $\rho = x_0 = 0.5$ .

The KL regularizer in equation 20.75 can be inserted into equation 20.72 to give the cost function with the KL regularizer. The term  $\hat{\rho}_j$  implicitly depends on  $\mathbf{w}$  because it is the average activation of the  $j^{\text{th}}$  hidden unit, which explicitly depends on these model parameters. The gradient of this term for back-propagation is

$$\delta_i^{[2]} = \left( \sum_{j=1}^{s_2} w_{ji}^{[2]} \delta_j^{[3]} \right) f'(z_i^{[2]}), \quad (20.76)$$

where we have inserted the layer superscript number [2] to terms in the second layer and the superscript [3] for the input from the next deeper layer. Here,  $\delta_i^{[2]}$  and  $\delta_i^{[3]}$  are, respectively, the  $i^{\text{th}}$  components of the back-propagation gradients in layers 2 and 3 (see equation 4.23 for the term



$f'(z_i^{[2]})$ ). If the regularizer term is included then the gradient becomes

$$\delta_i^{[2]} = \left( \left( \sum_{j=1}^M w_{ji}^{[2]} \delta_j^{[3]} \right) + \gamma \left( -\frac{\rho}{\hat{\rho}_i} + \frac{1-\rho}{1-\hat{\rho}_i} \right) \right) f'(z_i^{[2]}). \quad (20.77)$$

According to Ng et al.<sup>10</sup> we have the following comments about implementation of the KL gradient.

*One subtlety is that you'll need to know  $\hat{\rho}_i$  to compute equation 20.77. Thus, you'll need to compute a forward pass on all the training examples first to compute the average activations on the training set, before computing back-propagation on any example. If your training set is small enough to fit comfortably in computer memory (this will be the case for the programming assignment), you can compute forward passes on all your examples and keep the resulting activations in memory and compute the  $\hat{\rho}_i$ 's. Then you can use your precomputed activations to perform back-propagation on all your examples. If your data is too large to fit in memory, you may have to scan through your examples computing a forward pass on each to accumulate (sum up) the activations and compute  $\hat{\rho}_i$  (discarding the result of each forward pass after you have taken its activations  $a_i^{[2]}$  into account for computing  $\hat{\rho}_i$ ). Then after having computed  $\hat{\rho}_i$ , you'd have to redo the forward pass for each example so that you can do back-propagation on that example. In this latter case, you would end up computing a forward pass twice on each example in your training set, making it computationally less efficient.*

## 20.12 Appendix: Multisource VSP and $[\mathbf{L}^T \mathbf{L}]^{-1}$

For a 5-interface velocity model with two sources, the first source with  $s = 0$  has a src-rec offset  $\Delta x = 0$  and the other with  $s = 1$  has offset  $\Delta x$ . The resulting traveltime equations form a  $10 \times 5$  matrix  $\mathbf{L}$  that is a concatenation of two  $5 \times 5$  blocks of lower-triangular matrices:

$$\overbrace{\begin{bmatrix} l_1^{(0)} & 0 & 0 & 0 & 0 \\ l_2^{(0)} & l_2^{(0)} & 0 & 0 & 0 \\ l_3^{(0)} & l_3^{(0)} & l_3^{(0)} & 0 & 0 \\ l_4^{(0)} & l_4^{(0)} & l_4^{(0)} & l_4^{(0)} & 0 \\ l_5^{(0)} & l_5^{(0)} & l_5^{(0)} & l_5^{(0)} & l_5^{(0)} \\ l_1^{(1)} & 0 & 0 & 0 & 0 \\ l_2^{(1)} & l_2^{(1)} & 0 & 0 & 0 \\ l_3^{(1)} & l_3^{(1)} & l_3^{(1)} & 0 & 0 \\ l_4^{(1)} & l_4^{(1)} & l_4^{(1)} & l_4^{(1)} & 0 \\ l_5^{(1)} & l_5^{(1)} & l_5^{(1)} & l_5^{(1)} & l_5^{(1)} \end{bmatrix}}^{\mathbf{L}} \overbrace{\begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \end{pmatrix}}^{\mathbf{s}} = \overbrace{\begin{pmatrix} t_1^{(0)} \\ t_2^{(0)} \\ t_3^{(0)} \\ t_4^{(0)} \\ t_5^{(0)} \\ t_1^{(1)} \\ t_2^{(1)} \\ t_3^{(1)} \\ t_4^{(1)} \\ t_5^{(1)} \end{pmatrix}}^{\mathbf{t}^{obs}}. \quad (20.78)$$

The above  $10 \times 5$  matrix  $\mathbf{L}$  can be expressed as  $\mathbf{L} = [\mathbf{L}_0, \mathbf{L}_1]$ , where  $\mathbf{L}_0 = \mathbf{D}_0 \mathbf{T}$  and  $\mathbf{L}_1 = \mathbf{D}_1 \mathbf{T}$  are the, respectively, top and bottom  $5 \times 5$  sub-blocks of  $\mathbf{L}$ . Here,  $\mathbf{T}$  is the  $5 \times 5$  lower-triangular matrix with lower-triangular values of 1 and  $\mathbf{D}_0$  and  $\mathbf{D}_1$  are  $5 \times 5$  diagonal matrices with diagonal elements  $[\mathbf{D}_0]_{ii} = l_i^{(0)}$  and  $[\mathbf{D}_1]_{ii} = l_i^{(1)}$ . The inverse  $\mathbf{T}^{-1}$  is the 1st-order backward-difference operator (Schuster, 1989) and  $[\mathbf{T}^T \mathbf{T}]^{-1} = \mathbf{T}^{-1} \mathbf{T}^{-T}$  is the second-order backward-difference operator.

<sup>10</sup><http://deeplearning.stanford.edu/tutorial/> and material at <http://ufldl.stanford.edu/tutorial/>.

Therefore, the normal matrix is given by

$$\begin{aligned}\mathbf{L}^T \mathbf{L} &= [\mathbf{L}_0^T \mathbf{L}_0 + \mathbf{L}_1^T \mathbf{L}_1], \\ &= \mathbf{T}^T [\mathbf{D}_0^2 + \mathbf{D}_1^2] \mathbf{T}, \\ &= \mathbf{T}^T \mathbf{D}^2 \mathbf{T},\end{aligned}\tag{20.79}$$

where the sum of the squared diagonal matrices can be expressed as the diagonal matrix  $\mathbf{D}^2 = \mathbf{D}_0^2 + \mathbf{D}_1^2$ , with diagonal components  $[\mathbf{D}]_{ii} = \sum_{s=0}^1 l_i^{(s)2}$ . The inverse (Schuster, 1988 and 1989) to the normal matrix  $[\mathbf{L}^T \mathbf{L}]^{-1} = \mathbf{T}^{-1} \mathbf{D}^{-2} \mathbf{T}^{-T}$  is

$$[\mathbf{L}^T \mathbf{L}]^{-1} = \begin{bmatrix} \frac{1}{\sum_{s=0}^1 l_1^{(s)2}} & -\frac{1}{\sum_{s=0}^1 l_1^{(s)2}} & 0 & 0 & 0 \\ -\frac{1}{\sum_{s=0}^1 l_1^{(s)2}} & \frac{1}{\sum_{s=0}^1 l_1^{(s)2}} + \frac{1}{\sum_{s=0}^1 l_2^{(s)2}} & -\frac{1}{\sum_{s=0}^1 l_2^{(s)2}} & 0 & 0 \\ 0 & -\frac{1}{\sum_{s=0}^1 l_2^{(s)2}} & \frac{1}{\sum_{s=0}^1 l_2^{(s)2}} + \frac{1}{\sum_{s=0}^1 l_3^{(s)2}} & -\frac{1}{\sum_{s=0}^1 l_3^{(s)2}} & 0 \\ 0 & 0 & -\frac{1}{\sum_{s=0}^1 l_3^{(s)2}} & \frac{1}{\sum_{s=0}^1 l_3^{(s)2}} + \frac{1}{\sum_{s=0}^1 l_4^{(s)2}} & -\frac{1}{\sum_{s=0}^1 l_4^{(s)2}} \\ 0 & 0 & 0 & -\frac{1}{\sum_{s=0}^1 l_4^{(s)2}} & \frac{1}{\sum_{s=0}^1 l_4^{(s)2}} + \frac{1}{\sum_{s=0}^1 l_5^{(s)2}} \end{bmatrix},$$

which reduces to the 2nd-order backward-difference operator if  $\mathbf{D}$  is the identity matrix. As the number of sources increase the variance  $[\mathbf{L}^T \mathbf{L}]_{ii}^{-1}$  associated with the  $i^{th}$  slowness parameter decreases. The inverse matrix in equation 20.80 has the same tridiagonal structure as that in the equation 20.35, except now the squared segment lengths  $l_n^{(s)2}$  in the denominators have been replaced by their summation  $\sum_{s=1}^2 l_n^{(s)2}$  over the number of sources.

## 20.13 Appendix: Conditional Covariance Matrix

The formulas for the mean and covariance matrix will be derived for the conditional distribution  $p(\mathbf{d}|\mathbf{m})$ , which will be referred to as the likelihood function in the context of Bayes' Theorem. We will make use of the fact that exponent of a general Gaussian function can be written as

$$-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}) = \underbrace{-\frac{1}{2}\mathbf{x}^T \boldsymbol{\Sigma}^{-1}\mathbf{x}}_{\text{quadratic in } \mathbf{x}} + \underbrace{\mathbf{x}^T \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}}_{\text{linear in } \mathbf{x}} + cst,\tag{20.80}$$

where  $\boldsymbol{\Sigma}^{-1}$  is the inverse covariance matrix and  $\boldsymbol{\mu}$  is the mean. In determining the mean and covariance matrix of target distributions, the *key step* is to beat their exponentials into forms that are identical to the one of the right, which is a sum of weighted quadratic and linear terms in  $\mathbf{x}$ . Once this is accomplished then their weights can be identified as the covariance and mean of the target distribution. This procedure is called *completing the square* (Bishop, 2006).

The goal is to derive formulas for the mean and covariance matrix of the likelihood function  $p(\mathbf{d}|\mathbf{m})$  in terms of the mean and covariance matrix of the joint distribution  $p(\mathbf{m}, \mathbf{d})$ . We begin by assuming the joint probability distribution  $p(\mathbf{z}) = p(\mathbf{d}, \mathbf{m})$  with the vector  $\mathbf{z}$  and its mean defined as

$$\mathbf{z} = \begin{Bmatrix} \mathbf{d} \\ \mathbf{m} \end{Bmatrix} \text{ and } \boldsymbol{\mu} = \begin{Bmatrix} \boldsymbol{\mu}_d \\ \boldsymbol{\mu}_m \end{Bmatrix}.\tag{20.81}$$

Therefore, the joint covariance matrix is

$$\begin{aligned}\Sigma &= \langle (\mathbf{z} - \boldsymbol{\mu}_z)(\mathbf{z} - \boldsymbol{\mu}_z)^T \rangle = \begin{bmatrix} \langle (\mathbf{d} - \boldsymbol{\mu}_d)(\mathbf{d} - \boldsymbol{\mu}_d)^T \rangle & \langle (\mathbf{d} - \boldsymbol{\mu}_d)(\mathbf{m} - \boldsymbol{\mu}_m)^T \rangle \\ \langle (\mathbf{m} - \boldsymbol{\mu}_m)(\mathbf{d} - \boldsymbol{\mu}_d)^T \rangle & \langle (\mathbf{m} - \boldsymbol{\mu}_m)(\mathbf{m} - \boldsymbol{\mu}_m)^T \rangle \end{bmatrix}, \\ &= \begin{bmatrix} \Sigma_{dd} & \Sigma_{dm} \\ \Sigma_{md} & \Sigma_{mm} \end{bmatrix}.\end{aligned}\quad (20.82)$$

Symmetry of the covariance matrix demands  $\langle (\mathbf{m} - \boldsymbol{\mu}_m)(\mathbf{d} - \boldsymbol{\mu}_d)^T \rangle = \langle (\mathbf{d} - \boldsymbol{\mu}_d)(\mathbf{m} - \boldsymbol{\mu}_m)^T \rangle^T$ . That is,  $\Sigma_{md} = \Sigma_{dm}^T$ .

The inverse of this joint covariance matrix is given by a system of sub-block matrices:

$$\Sigma^{-1} = \begin{bmatrix} \Sigma_{dd} & \Sigma_{dm} \\ \Sigma_{md} & \Sigma_{mm} \end{bmatrix}^{-1} = \begin{bmatrix} \Lambda_{dd} & \Lambda_{dm} \\ \Lambda_{md} & \Lambda_{mm} \end{bmatrix}, \quad (20.83)$$

where the sub-blocks of the matrix  $\Sigma^{-1}$  are denoted by terms such as  $\Lambda_{dm}$  with subscripts associated with the type of variable that appears in the exponent of the Gaussian  $p(\mathbf{d}, \mathbf{m})$ . These sub-blocks  $\Lambda_{dd}$ ,  $\Lambda_{dm}$ ,  $\Lambda_{md}$ , and  $\Lambda_{mm}$  are those for the inverse covariance matrix  $\Sigma^{-1} = \Lambda$  of the joint distribution. Surprisingly, the expression for the conditional covariance matrix  $\Sigma_{d|m}$  will be the same as  $\Lambda_{dd}^{-1}$ , i.e.  $\Lambda_{dd}^{-1} = \Sigma_{d|m}$  as we will now show by comparing the arguments of the exponentials for the joint and conditional distributions.

The argument of the exponential in the joint Gaussian distribution is

$$\begin{aligned}\text{exp. argument of joint distribution } p(\mathbf{m}, \mathbf{d}) &= \overbrace{-\frac{1}{2}(\mathbf{z} - \boldsymbol{\mu}_z)^T \Sigma^{-1}(\mathbf{z} - \boldsymbol{\mu}_z)}^{\text{quadratic in } \mathbf{d}} = -\frac{1}{2}(\mathbf{d} - \boldsymbol{\mu}_d)^T \Lambda_{dd}(\mathbf{d} - \boldsymbol{\mu}_d) - \frac{1}{2}(\mathbf{d} - \boldsymbol{\mu}_d)^T \Lambda_{dm}(\mathbf{m} - \boldsymbol{\mu}_m) \\ &\quad - \frac{1}{2}(\mathbf{m} - \boldsymbol{\mu}_m)^T \Lambda_{md}(\mathbf{d} - \boldsymbol{\mu}_d) - \overbrace{\frac{1}{2}(\mathbf{m} - \boldsymbol{\mu}_m)^T \Lambda_{mm}(\mathbf{m} - \boldsymbol{\mu}_m)}^{\text{quadratic in } \mathbf{m}}.\end{aligned}\quad (20.84)$$

The goal is to now identify the argument of this joint distribution  $p(\mathbf{m}, \mathbf{d})$  with that of the likelihood distribution  $p(\mathbf{d}|\mathbf{m}) = p(\mathbf{m}, \mathbf{d})/p(\mathbf{m})$ , and then determine the formulas for the mean covariance matrix of  $p(\mathbf{d}|\mathbf{m})$  from those in  $p(\mathbf{m}, \mathbf{d})$ . How do we do this? The answer is to recognize that  $p(\mathbf{d}|\mathbf{m}) = p(\mathbf{m}, \mathbf{d})/p(\mathbf{m})$  considers  $\mathbf{m}$  to be a constant, so the argument of its exponential will contain the following quadratic and linear terms in  $\mathbf{d}$ :

$$\begin{aligned}\text{exp. argument of likelihood distribution } p(\mathbf{d}|\mathbf{m}) &= \overbrace{-\frac{1}{2}(\mathbf{d} - \boldsymbol{\mu}_{d|m})^T \Sigma_{d|m}^{-1}(\mathbf{d} - \boldsymbol{\mu}_{d|m})}^{\text{quadratic in } \mathbf{d}} = -\frac{1}{2} \overbrace{\mathbf{d}^T \Sigma_{d|m}^{-1} \mathbf{d}}^{\text{quadratic in } \mathbf{d}} + \overbrace{\mathbf{d}^T \Sigma_{d|m}^{-1} \boldsymbol{\mu}_{d|m}}^{\text{linear in } \mathbf{d}} + \text{const},\end{aligned}\quad (20.85)$$

where the likelihood's quadratic  $\mathbf{d}$  terms here should be matched to the quadratic  $\mathbf{d}$  terms in equation 20.84 for the joint distribution. Matching the quadratic  $\mathbf{d}$  terms in equations 20.84-20.85 allows us to conclude

$$\Sigma_{d|m}^{-1} = \Lambda_{dd}. \quad (20.86)$$

To derive the equation for the likelihood's mean  $\boldsymbol{\mu}_{d|m}$ , we equate the linear-in- $\mathbf{d}$  term in equation 20.84

$$\text{linear term in joint } \ln p(\mathbf{m}, \mathbf{d}) = \mathbf{d}^T (\Lambda_{dd} \boldsymbol{\mu}_d - \Lambda_{dm}(\mathbf{m} - \boldsymbol{\mu}_m)), \quad (20.87)$$

to the likelihood's linear-in- $\mathbf{d}$  term of equation 20.85 to get

$$\Sigma_{d|m}^{-1} \boldsymbol{\mu}_{d|m} = \{\Lambda_{dd} \boldsymbol{\mu}_d - \Lambda_{dm}(\mathbf{m} - \boldsymbol{\mu}_m)\}. \quad (20.88)$$

Multiplying both sides by  $\Sigma_{d|m}$  gives

$$\boldsymbol{\mu}_{d|m} = \Sigma_{d|m} \{\Lambda_{dd} \boldsymbol{\mu}_d - \Lambda_{dm}(\mathbf{m} - \boldsymbol{\mu}_m)\}, \quad (20.89)$$

$$= \boldsymbol{\mu}_d - \Lambda_{dd}^{-1} \Lambda_{dm}(\mathbf{m} - \boldsymbol{\mu}_m), \quad (20.90)$$

where equation 20.86 is used to go from equation 20.89 to equation 20.90.

The analytic formula for the inverse matrix in equation 20.83 can be obtained (see equation 20.100 or p. 87 in Bishop, 2006) so that  $\Lambda_{dd}$  and  $\Lambda_{dm}$  of the precision matrix  $\Lambda$  can be expressed as

$$\Lambda_{dd} = (\Sigma_{dd} - \Sigma_{dm} \Sigma_{mm}^{-1} \Sigma_{md})^{-1}, \quad (20.91)$$

$$\Lambda_{dm} = -(\Sigma_{dd} - \Sigma_{dm} \Sigma_{mm}^{-1} \Sigma_{md})^{-1} \Sigma_{dm} \Sigma_{mm}^{-1}. \quad (20.92)$$

Manipulation of these equations (Bishop, 2006) provides the following formulas for the conditional means and covariance terms:

$$\boldsymbol{\mu}_{d|m} = \boldsymbol{\mu}_d + \Sigma_{dm} \Sigma_{mm}^{-1} (\mathbf{m} - \boldsymbol{\mu}_m), \quad (20.93)$$

$$\Sigma_{d|m} = \Sigma_{dd} - \Sigma_{dm} \Sigma_{mm}^{-1} \Sigma_{md}. \quad (20.94)$$

Therefore, the conditional covariance matrix  $\Sigma_{d|m}$  and mean  $\boldsymbol{\mu}_{d|m}$  can be obtained from those in the joint distribution.

## 20.14 Appendix: Bayes' Theorem and Conditional Covariance Matrix

In Appendix 20.13 the conditional covariance matrix  $\Sigma_{d|m}$  and mean  $\boldsymbol{\mu}_{d|m}$  of the likelihood  $p(\mathbf{d}|\mathbf{m})$  were expressed in terms of those for the joint distribution  $p(\mathbf{m}, \mathbf{d})$ . In the context of Bayes' theorem, these parameters are those for the likelihood. Now we employ Bayes' theorem with the predictor equation  $\mathbf{Lm} = \mathbf{d} + \text{error}$  and compute the covariance matrix  $\Sigma_{m|d}$  and mean  $\boldsymbol{\mu}_{m|d}$  of the posterior distribution  $p(\mathbf{m}|\mathbf{d})$  in terms of those for the joint distribution.

The starting point is to assume

$$\begin{aligned} p(\mathbf{m}) &= \mathcal{N}(\mathbf{m}|\boldsymbol{\mu}_m, \Lambda^{-1}), \\ p(\mathbf{d}|\mathbf{m}) &= \mathcal{N}(\mathbf{d}|\mathbf{Lm}, \mathcal{L}^{-1}), \end{aligned} \quad (20.95)$$

where  $\Lambda^{-1}$  is the covariance matrix for  $p(\mathbf{m})$  and  $\mathcal{L}^{-1}$  is that for the conditional distribution  $p(\mathbf{d}|\mathbf{m})$ .

The natural log of the joint distribution  $p(\mathbf{m}, \mathbf{d}) = p(\mathbf{d}|\mathbf{m})p(\mathbf{m})$  is

$$\begin{aligned}
 \ln p(\mathbf{m}, \mathbf{d}) &= \ln p(\mathbf{m}) + \ln p(\mathbf{d}|\mathbf{m}), \\
 &= -\frac{1}{2} [(\mathbf{m} - \boldsymbol{\mu}_m)^T \boldsymbol{\Lambda} (\mathbf{m} - \boldsymbol{\mu}_m) + (\mathbf{Lm} - \mathbf{d})^T \boldsymbol{\mathcal{L}} (\mathbf{Lm} - \mathbf{d}) + \text{cnst}], \\
 &\quad \text{2nd-order terms in joint } \ln p(\mathbf{m}, \mathbf{d}) \\
 &= -\frac{1}{2} [\mathbf{m}^T (\boldsymbol{\Lambda} + \mathbf{L}^T \boldsymbol{\mathcal{L}} \mathbf{L}) \mathbf{m} + \mathbf{d}^T \boldsymbol{\mathcal{L}} \mathbf{d}] + \mathbf{m}^T \mathbf{L}^T \boldsymbol{\mathcal{L}} \mathbf{d} \\
 &\quad \text{1st-order term in } \mathbf{m} \text{ for } \ln p(\mathbf{m}, \mathbf{d}) \\
 &\quad + \underbrace{\boldsymbol{\mu}_m^T \boldsymbol{\Lambda} \mathbf{m}}_{\text{1st-order term in } \mathbf{m} \text{ for } \ln p(\mathbf{m})} - \frac{1}{2} \boldsymbol{\mu}_m^T \boldsymbol{\Lambda} \boldsymbol{\mu}_m + \text{cnst}
 \end{aligned} \tag{20.96}$$

where  $\text{cnst}$  is independent of  $\mathbf{m}$  and  $\mathbf{d}$ . The second-order terms are not all quadratic in  $\mathbf{m}$  because the aim is to transform these second-order terms into a vector-matrix-vector product similar to the partitioned matrix in equation 20.84, as will be demonstrated below.

The second-order terms in  $\mathbf{m}$  and  $\mathbf{d}$  can be expressed as a vector-matrix-vector product:

$$\overbrace{-\frac{1}{2} [\mathbf{m}^T (\boldsymbol{\Lambda} + \mathbf{L}^T \boldsymbol{\mathcal{L}} \mathbf{L}) \mathbf{m} + \mathbf{d}^T \boldsymbol{\mathcal{L}} \mathbf{d} - \mathbf{d}^T \boldsymbol{\mathcal{L}} \mathbf{L} \mathbf{m} - \mathbf{m}^T \mathbf{L}^T \boldsymbol{\mathcal{L}} \mathbf{d}]}^{\text{2nd-order terms in joint } \ln p(\mathbf{m}, \mathbf{d})} = \tag{20.97}$$

$$= -\frac{1}{2} \begin{Bmatrix} \mathbf{m} \\ \mathbf{d} \end{Bmatrix}^T \overbrace{\begin{bmatrix} \boldsymbol{\Lambda} + \mathbf{L}^T \boldsymbol{\mathcal{L}} \mathbf{L} & -\mathbf{L}^T \boldsymbol{\mathcal{L}} \\ -\boldsymbol{\mathcal{L}} \mathbf{L} & \boldsymbol{\mathcal{L}} \end{bmatrix}}^{\mathbf{R}} \begin{Bmatrix} \mathbf{m} \\ \mathbf{d} \end{Bmatrix} = -\frac{1}{2} \mathbf{z}^T \mathbf{R} \mathbf{z}, \tag{20.98}$$

where  $\mathbf{z} = (\mathbf{m}^T \ \mathbf{d}^T)^T$  and

$$\mathbf{R} = \begin{bmatrix} \boldsymbol{\Lambda} + \mathbf{L}^T \boldsymbol{\mathcal{L}} \mathbf{L} & -\mathbf{L}^T \boldsymbol{\mathcal{L}} \\ -\boldsymbol{\mathcal{L}} \mathbf{L} & \boldsymbol{\mathcal{L}} \end{bmatrix}. \tag{20.99}$$

Here,  $\mathbf{R}$  is the inverse covariance matrix, aka as the precision matrix, for the joint distribution  $p(\mathbf{m}, \mathbf{d})$ .

We know that the analytic inverse (see p. 87 in Bishop, 2006) to a  $2 \times 2$  block matrix is

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{M} & -\mathbf{M} \mathbf{B} \mathbf{D}^{-1} \\ -\mathbf{D}^{-1} \mathbf{C} \mathbf{M} & \mathbf{D}^{-1} + \mathbf{D}^{-1} \mathbf{C} \mathbf{M} \mathbf{B} \mathbf{D}^{-1} \end{bmatrix}, \tag{20.100}$$

where  $\mathbf{M} = (\mathbf{A} - \mathbf{B} \mathbf{D}^{-1} \mathbf{C})^{-1}$ . Therefore, equation 20.100 can be used to get the analytical inverse of  $\mathbf{R}$  in equation 20.98 to give the formula for the joint covariance matrix:

$$\text{cov}(\mathbf{z}) = \mathbf{R}^{-1} = \begin{bmatrix} \boldsymbol{\Lambda}^{-1} & \boldsymbol{\Lambda}^{-1} \mathbf{L}^T \\ \mathbf{L} \boldsymbol{\Lambda}^{-1} & \boldsymbol{\mathcal{L}}^{-1} + \mathbf{L} \boldsymbol{\Lambda}^{-1} \mathbf{L}^T \end{bmatrix}. \tag{20.101}$$

Similar to the derivation of the likelihood's mean in equation 20.88, we can compare the joint distribution's first-order linear-in- $\mathbf{m}$  term in equation 20.96

$$\text{linear } \mathbf{z} \text{ term in joint } \ln p(\mathbf{m}, \mathbf{d}) = \begin{Bmatrix} \mathbf{m} \\ \mathbf{d} \end{Bmatrix}^T \begin{bmatrix} \boldsymbol{\Lambda} \boldsymbol{\mu}_m \\ 0 \end{bmatrix} = \mathbf{m}^T \boldsymbol{\Lambda} \boldsymbol{\mu}_m. \tag{20.102}$$

to that in the natural log of the posterior function  $\ln p(\mathbf{m}|\mathbf{d})$ , i.e.

$$\overbrace{-\frac{1}{2}(\mathbf{z} - \boldsymbol{\mu}_z)^T \mathbf{R}(\mathbf{z} - \boldsymbol{\mu}_z)}^{\ln \text{ posterior } p(\mathbf{m}|\mathbf{d})} = -\frac{1}{2} \overbrace{\mathbf{z}^T \mathbf{R} \mathbf{z}}^{\text{quadratic in } \mathbf{z}} + \overbrace{\mathbf{z}^T \mathbf{R} \boldsymbol{\mu}_z}^{\text{linear in } \mathbf{z}} + \text{cnst.} \quad (20.103)$$

The linear  $\mathbf{z}$ -multiplicative factors of  $\mathbf{z}^T$  in equations 20.103 and 20.102 can be equated to give

$$\mathbf{R} \boldsymbol{\mu}_z = \begin{bmatrix} \boldsymbol{\Lambda} \boldsymbol{\mu}_m \\ 0 \end{bmatrix}. \quad (20.104)$$

Multiplying both sides by  $\mathbf{R}^{-1}$  gives the formula for the mean of  $\mathbf{z}$ :

$$\boldsymbol{\mu}_z = \langle \mathbf{z} \rangle = \langle \begin{Bmatrix} \mathbf{m} \\ \mathbf{d} \end{Bmatrix} \rangle = \mathbf{R}^{-1} \begin{Bmatrix} \boldsymbol{\Lambda} \boldsymbol{\mu}_m \\ 0 \end{Bmatrix} = \begin{Bmatrix} \boldsymbol{\mu}_m \\ \mathbf{L} \boldsymbol{\mu}_m \end{Bmatrix}. \quad (20.105)$$

which implies

$$\langle \mathbf{d} \rangle = \mathbf{L} \boldsymbol{\mu}_m. \quad (20.106)$$

This is not surprising because  $\mathbf{L} \mathbf{m}$  predicts  $\mathbf{d}$ , so the average over  $\mathbf{d}$  should be the same as the average over  $\mathbf{m}$  multiplied by  $\mathbf{L}$ .

The covariance matrix of the marginal distribution for  $\mathbf{d}$  is

$$\boldsymbol{\Sigma}_{dd} = \boldsymbol{\mathcal{L}}^{-1} + \mathbf{L} \boldsymbol{\Lambda}^{-1} \mathbf{L}^T, \quad (20.107)$$

which is obtained from the lower-right sub-block in equation 20.101. A special case is that  $\boldsymbol{\Sigma}_{dd} = \mathbf{L} \mathbf{L}^T$  becomes the expression for the unregularized data covariance matrix when  $\boldsymbol{\Lambda} = \mathbf{I}$  and  $\boldsymbol{\mathcal{L}}^{-1} = 0$ . Therefore, equations 20.106-20.107 represent, respectively, the mean and covariance matrix of the variable  $\mathbf{d}$  in the joint distribution  $p(\mathbf{m}, \mathbf{d})$ .

The covariance matrix of the posterior  $p(\mathbf{m}|\mathbf{d})$  can be expressed using the same idea in the derivation of the covariance of the partitioned covariance  $\boldsymbol{\Sigma}_{d|m}^{-1} = \boldsymbol{\Lambda}_{dd}$  in equation 20.86. In that example, the quadratic-in- $\mathbf{d}$  term in the upper-left sub-block  $\boldsymbol{\Lambda}_{dd}$  of equation 20.84 is matched with the quadratic-in- $\mathbf{d}$  term  $\boldsymbol{\Sigma}_{d|m}^{-1}$  of equation 20.85 to get equation 20.86. Now, we match the quadratic-in- $\mathbf{m}$  term in equation 20.98 with that in equation 20.80 where  $\mathbf{x} \rightarrow \mathbf{m}$  and  $\boldsymbol{\Sigma}^{-1} \rightarrow \boldsymbol{\Sigma}_{m|d}^{-1}$  to get

$$\boldsymbol{\Sigma}_{m|d} = (\boldsymbol{\Lambda} + \mathbf{L}^T \boldsymbol{\mathcal{L}} \mathbf{L})^{-1}. \quad (20.108)$$

To compute the mean  $\boldsymbol{\mu}_{m|d}$  we collect the terms linear in  $\mathbf{m}$  in equation 20.96

$$\text{linear-in-}\mathbf{m} \text{ terms} = \mathbf{m}^T \mathbf{L}^T \boldsymbol{\mathcal{L}} \mathbf{d} + \mathbf{m}^T \boldsymbol{\Lambda} \boldsymbol{\mu}_m, \quad (20.109)$$

and match them to the linear-in- $\mathbf{m}$  term in equation 20.80 where  $\mathbf{x} \rightarrow \mathbf{m}$  and  $\boldsymbol{\Sigma}^{-1} \rightarrow \boldsymbol{\Sigma}_{m|d}^{-1}$  to get

$$\mathbf{m}^T \boldsymbol{\Sigma}_{m|d}^{-1} \boldsymbol{\mu}_{m|d} = \mathbf{m}^T \mathbf{L}^T \boldsymbol{\mathcal{L}} \mathbf{d} + \mathbf{m}^T \boldsymbol{\Lambda} \boldsymbol{\mu}_m. \quad (20.110)$$

Replacing  $\boldsymbol{\Sigma}_{m|d}^{-1}$  with the expression on the right-side of equation 20.108, we conclude that

$$\boldsymbol{\mu}_{m|d} = (\boldsymbol{\Lambda} + \mathbf{L}^T \boldsymbol{\mathcal{L}} \mathbf{L})^{-1} (\mathbf{L}^T \boldsymbol{\mathcal{L}} \mathbf{d} + \boldsymbol{\Lambda} \boldsymbol{\mu}_m). \quad (20.111)$$

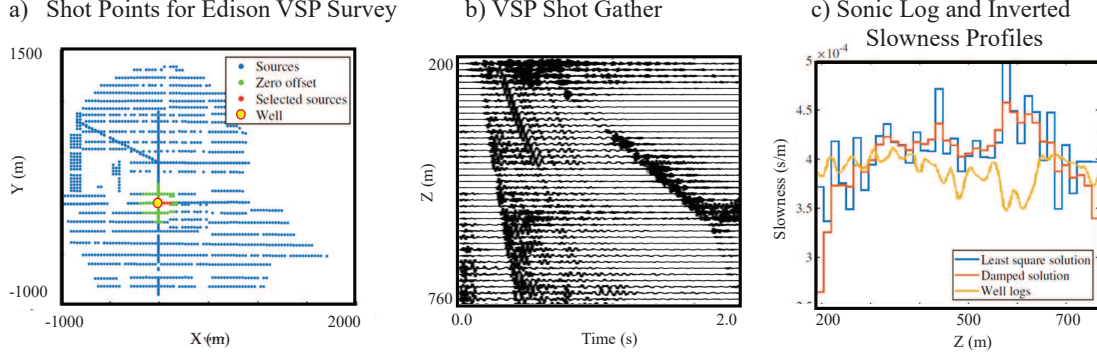


Figure 20.26: a) Shot points for the VSP survey where the well is located at the large yellow dot, b) VSP shot gather and c) sonic log (yellow lines) and slowness profiles inverted by undamped (blue lines) and damped (brown lines) least squares inversion. The green dots in a) denote the near-offset sources that generated the near-offset shot gathers, where the direct arrivals are picked and inverted for the velocity model.

## 20.15 Appendix: Estimation of Statistical Parameters for VSP Data

The VSP traveltimes are picked from direct arrivals recorded by the Edison 3D VSP survey (Paulsson and Kelley, 2021). Figure 20.26a depicts the shot points for this survey, but only the sources (green points) near the well (yellow dot) were used for inverting the traveltimes of direct waves. A typical shot gather is shown in Figure 20.26b, and the damped and undamped least squares solutions are compared to the slowness profile from the sonic log in Figure 20.26c.

Bayes' formula requires the values of the statistical parameters for both  $p(\mathbf{d})$  and  $p(\mathbf{s})$ . The next two sections describe how the means and covariance matrices are computed for  $p(\mathbf{d})$ ,  $p(\mathbf{m})$  and  $p(\mathbf{d}|\mathbf{m})$ .

### 20.15.1 Estimating $p(\mathbf{d})$ and $p(\mathbf{m})$

To determine the standard deviation of the traveltimes values  $t_i$  at the  $i^{th}$  depth level, arrival times of direct waves recorded at the  $i^{th}$  geophone<sup>11</sup> are plotted against the straight-line distance between the source and  $i^{th}$  geophone in Figure 20.27a. A straight line is fitted to these points and the traveltimes are approximately flattened by subtracting them from the time along the best-fit straight line. The standard deviation  $\sigma_i^d$  of these flattened traveltimes is computed for  $i = \{1, 2, \dots, N\}$ , where  $N$  is the number of geophones. It is assumed that the picking errors at the  $i^{th}$  geophone are zero-mean Gaussian distributed with the standard deviation given by  $\sigma_i^d$ , where Figure 20.27b depicts  $\sigma_i^d$  plotted against the depth coordinate  $z_i$ . The mean slownesses associated with the five shallowest geophones are plotted in Figure 20.27c, where the standard deviations are denoted by bars at each depth level.

The picking errors at different geophones are independent of one another. Therefore, the marginal probability  $p(\mathbf{d})$  is defined by the Gaussian  $p(\mathbf{d}) = p(t_1, t_2, \dots, t_N) = p(t_1)p(t_2)\dots p(t_N)$ , where  $\mathbf{d} = (t_1, t_2, \dots, t_N)$ ,  $N$  is the number of geophones and  $t_i$  is the picked traveltimes at the  $i^{th}$  geophone.

<sup>11</sup>The  $i^{th}$  depth level is the depth of the  $i^{th}$  geophone along the vertical well.

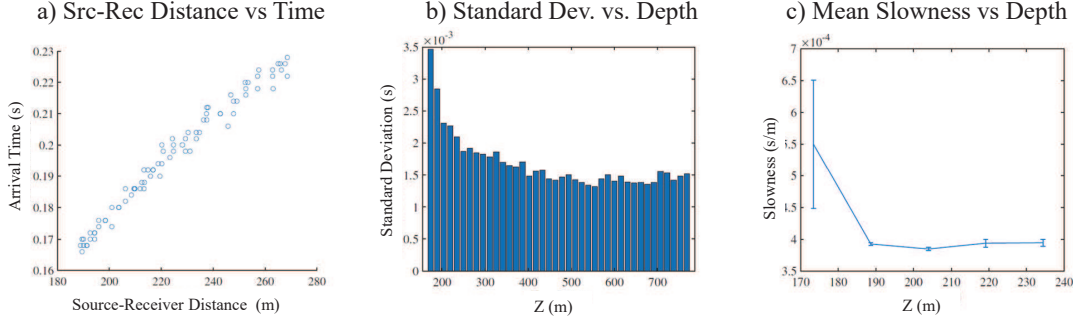


Figure 20.27: a) Arrival times plotted against straight-line distance between the source and downhole receiver, b) standard deviation  $\sigma_i^d$  of traveltimes from a) for each receiver, and c) mean slowness  $\bar{s}_i$  computed by dividing the distance  $d_{ij}$  from the  $j^{th}$  source to the  $i^{th}$  receiver to get the average slowness  $\bar{s}_i = \frac{1}{N_i} \sum_{j=1}^{N_i} t_{ij}/d_{ij}$  down to the  $i^{th}$  interface. Here,  $N_i$  is the number of direct arrivals recorded by the  $i^{th}$  receiver and  $t_{ij}$  is the arrival time of the direct wave excited by the  $j^{th}$  source and recorded at the  $i^{th}$  receiver.

The statistical parameters for  $p(\mathbf{m})$  are determined by assuming ergodicity in the slowness variable so that the average slowness  $\mu_i^{slow}$  and standard deviation  $\sigma_i^{slow}$  at the  $i^{th}$  depth level are computed over a small window in the sonic log. The window is centered at the  $i^{th}$  depth level, and both  $\mu_i^{slow}$  and  $\sigma_i^{slow}$  for  $i \in \{1, 2, \dots, N\}$  are computed by centering this window at each geophone level. The covariance values  $\langle s_i s_j \rangle$  can be computed as well, but for simplicity we assumed uncorrelated slowness values  $\langle s_i s_j \rangle = 0$  if  $i \neq j$ .

### 20.15.2 Estimating $p(\mathbf{d}|\mathbf{m})$

We assume that the data have independent random errors that are normally distributed with mean-value zero. The probability density function for the  $i^{th}$  observation  $d_i$  is taken to be

$$p_i(d_i|\mathbf{m}) = \frac{1}{(2\pi)^{\frac{1}{2}}\sigma_i} e^{-\frac{1}{2}\frac{(d_i - (\mathbf{Lm})_i)^2}{\sigma_i^2}}, \quad (20.112)$$

where the standard deviation of the  $i^{th}$  observation  $d_i$  is  $\sigma_i$ , which is approximated by the computed error  $\sigma_i^d$  in this case.

The likelihood function for the complete dataset is now given by

$$p(\mathbf{d}|\mathbf{m}) = p_1(d_1|\mathbf{m})p_2(d_2|\mathbf{m}) \dots p_m(d_m|\mathbf{m}) = \frac{1}{(2\pi)^{\frac{M}{2}} \prod_{i=1}^m \sigma_i} \prod_{i=1}^m e^{-\frac{1}{2}\frac{(d_i - (\mathbf{Lm})_i)^2}{\sigma_i^2}}. \quad (20.113)$$

We can write the likelihood function  $p(\mathbf{d}|\mathbf{m})$  in the more compact form as

$$p(\mathbf{d}|\mathbf{m}) = \frac{1}{(2\pi)^{\frac{M}{2}} |\boldsymbol{\Sigma}_{d|m}|^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{d} - \mathbf{Lm})^T |\boldsymbol{\Sigma}_{d|m}|^{-1} (\mathbf{d} - \mathbf{Lm})}, \quad (20.114)$$

where the data variances form the diagonal elements of the conditional covariance matrix  $\boldsymbol{\Sigma}_{d|m}$ .



A more general form of  $\Sigma_{d|m}$  starts with the definition

$$\begin{aligned}\Sigma_{d|m} &= \langle \mathbf{d}\mathbf{d}^T \rangle, \\ &= \langle \mathbf{L}\mathbf{m}\mathbf{m}^T\mathbf{L}^T \rangle, \\ &= \mathbf{L} \overbrace{\langle \mathbf{m}\mathbf{m}^T \rangle}^{\Sigma_m} \mathbf{L}^T,\end{aligned}\tag{20.115}$$

where  $\Sigma_m = \langle \mathbf{m}\mathbf{m}^T \rangle$  can be estimated from the sonic log.



## Chapter 21

# Discriminant Analysis and Gaussian Mixture Model

The theories of Discriminant Analysis and the Gaussian Mixtures Model (GMM) are presented. A discriminant function takes an input vector  $\mathbf{x}$  and assigns it to one of  $K$  classes (Bishop, 2006). For linear discriminant analysis and binary classification, a decision hyperplane is computed that separates the points classified as  $y = 0$  from the points classified as  $y = 1$ . For Gaussian Discriminant Analysis (GDA) and the GMM, each method fits Gaussian density functions to a group of  $N$  data points  $\mathbf{x}^{(n)}$   $n \in \{1, 2, \dots, N\}$ . However, the training data for GDA consists of both the data  $\mathbf{x}^{(n)}$  and the specified class  $y^{(n)}$  for each point, i.e.  $(\mathbf{x}^{(n)}, y^{(n)})$ , which means it is a supervised learning method. In contrast, the GMM is an unsupervised learning method because the training data are only the unclassified data  $\mathbf{x}$ . The output is the optimal mean and variance values of the Gaussian density functions that best explain the data by dividing them into  $K$  clusters. The number of Gaussian fitting functions is specified, i.e. guessed at, by the user. Similar to the iterative K-means cluster method, the number  $K$  of Gaussian clusters is user specified. Unlike the K-means hard-clustering method where each point belongs to only one cluster, the GMM is a soft-clustering method where all points have a non-zero probability of belonging to any one cluster. GDA is also a hard-clustering method because each point only belongs to one Gaussian.

### 21.1 Introduction

Figure 21.1a-21.1b depicts a cluster of Yellowstone points (see Figure 17.1) fitted to a) a single Gaussian density function and b) two Gaussian density functions. Each cluster in b) is tightly enclosed by Gaussian-like contours centered at the mean point and shaped by the variance of the correlated points. Fitting the unclassified data to  $K$  clusters governed by  $K$  Gaussian density functions is the goal of the Gaussian Mixture Model, where the fitted function  $f(\mathbf{x})$  is simply described as

$$f(\mathbf{x}) = \sum_{k=1}^K w_k \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \quad (21.1)$$

where  $\sum_{k=1}^K w_k = 1$ ,  $(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$  describes the mean and covariance matrix, respectively, for the  $k^{th}$  Gaussian,  $0 \leq w_k \leq 1$  is the mixture weight for the  $k^{th}$  Gaussian,  $\mathbf{x}$  is a  $N \times 1$  point and there are  $K$  Gaussians. Unlike the hard-clustering method of K-means (see Chapter 17), each point  $\mathbf{x}$  in the training data is shared by all  $K$  Gaussians. Therefore, the GMM is a soft-clustering method similar to fuzzy clustering (see Chapter 17), except the GMM uses the Gaussian probability functions to determine the degree, i.e. probability, of membership (Károly et al., 2018). This allows

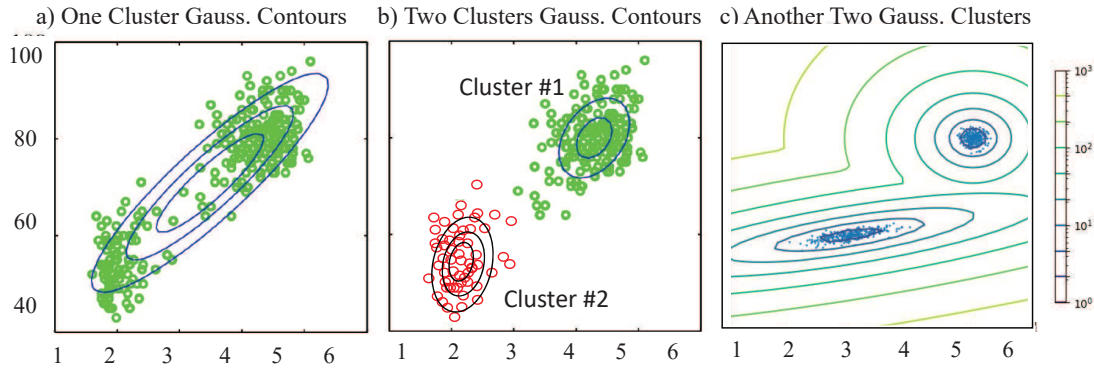


Figure 21.1: Data points fitted to a) one Gaussian density function, b) contours fitted to two Gaussian density functions that define the red and green clusters of points, and c) fully contoured GMM contours from another data set demonstrating that all points are shared by the two Gaussians. Points with a sufficiently low probability of membership might be considered to be too noisy for further consideration. Figures a and b modified from Bishop (2006), c) modified from [https://scikit-learn.org/stable/auto\\_examples/mixture/plot\\_gmm\\_pdf.html](https://scikit-learn.org/stable/auto_examples/mixture/plot_gmm_pdf.html).

for the construction of irregular and multimode probability functions as illustrated in Figure 21.1c. Moreover,  $f(\mathbf{x})$  provides a reliability estimate of the class membership for any point.

In contrast, GDA fits only one Gaussian to each point, class, to give  $p(\mathbf{x}|y)$ . Each class of training points is fitted to its own unique Gaussian to yield the associated covariance matrix and mean vector. Bayes' theorem can be used to generate  $p(y|\mathbf{x})$ , which allows a new data point  $\mathbf{x}_{new}$  to be classified as  $p(y|\mathbf{x}_{new})$ . This also gives an uncertainty estimate of this prediction.

A practical industrial application of the GMM is to fit arbitrary probability distributions to multidimensional data  $\mathbf{x}^{(n)}$ ;  $n \in (1, 2, \dots, N)$ . According to Reynolds et al. (2009), GMMs are commonly used as parametric models that predict the probability distribution of continuous biometric measurements, such as recognizing the speaker identity from the spectral features of voice recordings. Another application is in assessing the quality of fabricated machine parts<sup>1</sup>. For assessing quality control of manufactured products,  $\mathbf{x}^{(n)} = (x_1^{(n)}, x_2^{(n)}) = (strength, temperature)$  might be the test result for a constructed engine component. The components might have been constructed at two different manufacturing plants, and so the test results might plot as the bi-modal contours in Figure 21.1c. The points near the peaks of these contours might be designated as meeting the design criteria, but those far from the peaks, i.e. means of the two Gaussians, can be flagged as engine components with insufficient strength under harsh temperature conditions.

GMMs can also be applied to geophysical data  $\mathbf{x}^{(n)}$  in order to classify different types of signals and to distinguish signal from outlier noise. As an example of signal classification, Lubo et al. (2014) applied a GMM to well-log data from different wells in a Texas oil field. They were able to distinguish between lithologies from six different wells to produce a probabilistic map of the seismic lithologies over the area covered by the wells. In a related study, Bradley and Hardisty (2019) extracted classes of seismic facies by applying the GMM to data computed by a self-organizing map (SOM) of seismic data. The GMM clusters were then used to delineate features in the turbidite system with more accuracy than the SOM map. Moreover, they produced an uncertainty map that indicated highly reliable area of their interpretation.

To account for the multi-modal nature of describing rock properties with probability density

<sup>1</sup>See lecture by Andrew Ng at <https://www.youtube.com/watch?v=rVfZHWtwXSA>.

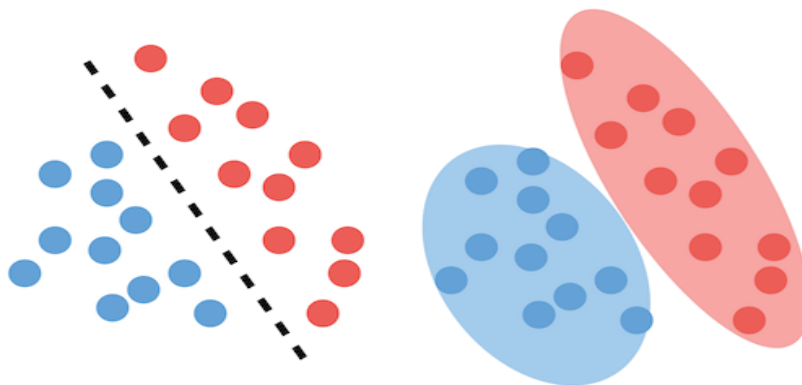
a) Discriminative  $\rightarrow P(Y|X)$    b) Generative  $\rightarrow P(X|Y)$ 

Figure 21.2: Illustrations of a) discriminative and b) generative models. Figure modified from Bishop (2006).

functions, Grana and del Monte (2010) used a two-step approach: 1) estimate the probability distributions of elastic parameters (velocities and density) and 2) estimate the multi-modal probability distributions of the underlying reservoir properties and conditions by assuming that the reservoir properties obey a Gaussian mixture distribution. This assumption reflects the realistic properties of these rocks.

Instead of just classifying the data by a GMM, it can be used to assist in inverting the seismic traces for lithology information. As an example, de Figueiredo et al. (2021) combined seismic inversion with the probability density function from a GMM for different facies units. The result is joint inversion of seismic data for the prediction of facies and elastic properties with both synthetic data and field records. In a related work, Astic and Oldenburg (2018) used the density function obtained from the GMM to replace the Gaussian prior in Tikhonov regularized inversion of seismic data.

The following sections describe the theories of linear discriminant analysis (LDA), GDA and GMM. This is followed by the description of the iterative Expectation-Maximization (EM) algorithm, which is used to find the parametric parameters for the GMM. The EM algorithm consists of alternating estimates of the statistical parameters ( $\mu$ ,  $\Sigma$ ) and the Gaussian mixture weights for each point. The next section provides numerical examples where the GMM is applied to seismic data for both lithology classification and seismic inversion.

## 21.2 Generative and Discriminative Models

There are two types of functions used for the GMM: discriminative and generative functions. The discriminative function  $p(Y|X)$  predicts the class  $Y$  from the input data  $X$ . For example, in a supervised neural network the NN model, a proxy for  $p(Y|X)$ , is computed from training data to get the optimal decision boundaries (see Figure 21.2a) that separate one class of points from another. This is done by finding the NN parameters  $\mathbf{w}$  that minimize a loss function. The parameters that define the optimal NN model are given by  $\mathbf{w}$  and the training pairs are multivariate random vectors with the discriminative model described the conditional probability function  $p(\mathbf{Y}|\mathbf{X}, \mathbf{w})$ .

In contrast, the generative function  $p(X|Y)$  is the reverse of the discriminative function in that it *generates* the data  $X$  from the class  $Y$ , as illustrated in Figure 21.2b. For the GMM, the  $k^{th}$

class of data is characterized by the weighted Gaussian distribution with the statistical parameters  $\mathbf{Y} = \{w_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}$ . The GMM in equation 21.28 predicts the probability of the data  $\mathbf{X}$ . This is equivalent to the decoder portion of an autoencoder in Chapter 14 where the latent variables  $\mathbf{z}$  in the autoencoder are used to predict the data  $\mathbf{X}$ .

### Key Idea 21.2.1. Key Idea: Example of a Generative Function

An example of a generative model with a low-dimensional class variable  $k = \{0, 1\}$  is illustrated in Figure 21.3a. Here, high-dimensional vectors flattened from the images of faces are mapped to the real line with values  $x$ . A face that is of the male class ( $k = 2$ ) honors the red Gaussian curve, where the probability of finding a Chinese face is highest with probability  $p(\text{Chinese}|k = 2) \approx 0.18$ ; more than 1.4 billion people live in China out of the 2021 world population of more than 7.7 billion people.

The Gaussian can be sampled so that it answers the question of which nationality  $x$  is associated with a probability of, say,  $p(x|k = 2) \approx 0.03$ . In this case it might be a white American male whose average picture can be *generated* by following the reverse path of the arrows that start with the tip at the green arrow (faces from <https://leadingpersonality.wordpress.com/2013/09/30/average-faces-of-men-and-women-around-the-world/>). Using a lookup table, the probability function  $p(x|k = 2) = \alpha$  can be sampled to generate faces of males that are coded along the real axis  $x$  and represent the fraction  $\alpha$  of the world's male population. The female faces can be generated in the same way, except the blue Gaussian curve can be used to assess the frequency of occurrence and generate their images. We will denote this as sampling the probability density function.

If a probability value  $p(x|k = 1) = \beta$  is between two values with known faces, then the new face is an interpolation of the neighboring faces. A general interpolator is that of an autoencoder in Figure 21.3b. After training the autoencoder, the generator (a fancy version of a lookup table) takes a low-dimensional vector  $\mathbf{k}$  and generates the image on the right. If the latent vector  $\mathbf{k}$  is generated by sampling a probability distribution hidden in the encoder, then this type of autoencoder is known as a variational autoencoder. See Chapter 14 for details.

## 21.3 Linear and Gaussian Discriminant Analysis

A discriminant is a classifier function that learns from training data  $(\mathbf{x}^{(n)}, y^{(n)})$  to accurately predict the class  $y^{(n)}$  of the input vector  $\mathbf{x}^{(n)}$ . The multinomial output can be assigned integer values from one to  $K$  classes, where  $y \in \{1, 2, \dots, K\}$ . In the context of Gaussian analysis, each class index is associated with a different Gaussian. If the class index  $y^{(n)}$  is given for each training data point, then this is denoted as supervised GDA. If the class is not given in the training set, then this is known as a Gaussian Mixture Model.

We now present the theories associated with two supervised discriminant analysis methods: LDA and GDA. To simplify the discussion, we will confine the discriminants to be binary classifiers, with the understanding that they can be extended to multinomial classifiers (Bishop, 2006). In contrast, the Gaussian Mixture Model is a soft clustering method that is unsupervised where any point belongs to every Gaussian, which is a generalization of hard-clustering GDA where any point belongs to just one Gaussian.

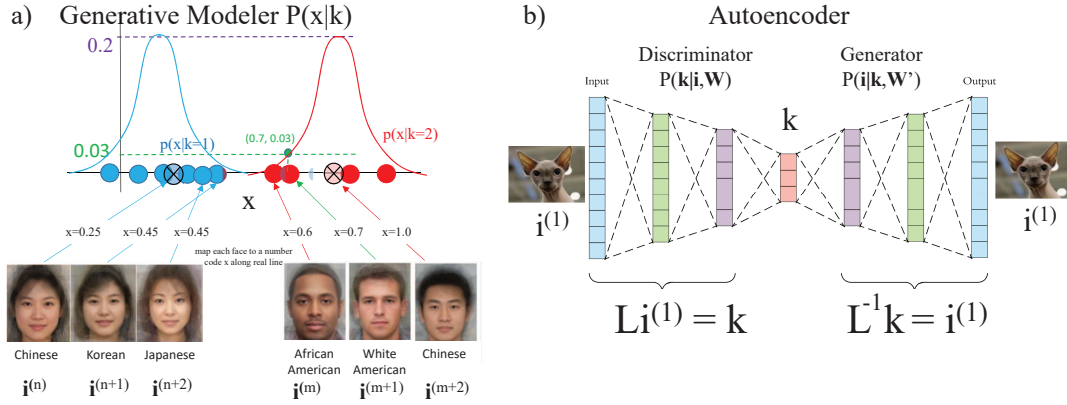


Figure 21.3: a) Average faces  $i^{(m)}$  mapped to numbers  $x$  along the real line, where  $x$  is the coded value of the  $10^6 \times 1$  vector  $i^{(m)}$  that is the flattened image of a face with  $10^6$  pixels. The faces are ordered along the real line so that the probability of a face is equal to the fraction of that person's nationality in the world's population. These faces belong to the class of either female ( $k=1$ ) or male ( $k=2$ ). The Chinese faces are the most numerous in the world so they appear with the highest probability. b) Architecture of an autoencoder, where the decoder represented by  $L^{-1}k$  takes the low-dimensional vector  $k$  and maps it into the high-dimensional image  $i^{(1)}$ .

### 21.3.1 Linear Discriminant Analysis

LDA for binary classification linearly models the input vector  $D \times 1$  vector  $\mathbf{x}$  and output class label  $y$  for  $y \in \{0, 1\}$  as

$$\overbrace{\mathbf{w}^T \mathbf{x} + w_0}^{f(\mathbf{x})} = y, \quad (21.2)$$

where  $\mathbf{w}$  is the  $D \times 1$  weight vector,  $f(\mathbf{x})$  is the discriminant, and  $w_0$  is the bias term. The LDA goal is to find  $(w_0, \mathbf{w}^T)$  that accurately predicts the class  $y^{(n)}$  from the training input data  $\mathbf{x}^{(n)}$  for  $n \in \{1, 2, \dots, N\}$ . The optimal  $(w_0, \mathbf{w}^T)$  defines a hyperplane, aka as the decision surface, that separates the  $y = 1$  points from the  $y = 0$  points.

For comparison, Chapter 4 presented the non-linear NN model which also finds the optimal weights  $(w_0, \mathbf{w}^T)$  that define the optimal decision surface separating the two classes of points. In this case, the discriminant function  $f(\mathbf{w}^T \mathbf{x} + w_0)$  is a non-linear function that consists of squashing-type functions such as the sigmoid and softmax functions. However, the decision surface plotted in  $(x_1, x_2, \dots, x_D)$  coordinates is described by

$$y(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0 = \text{constant}, \quad (21.3)$$

which is a linear function in  $(x_1, x_2, \dots, x_D)$  even though the NN function  $f(\mathbf{x})$

$$y(\mathbf{x}) = f(\mathbf{w}^T \mathbf{x} + w_0) = \text{constant}, \quad (21.4)$$

is a non-linear function. Therefore, equation 21.4 describes a generalized linear model (p. 138 in Bishop, 2006).

There are several LDA methods besides the NN model, such as the support vector machine in

Chapter 7, least squares discriminants and Fisher discriminant analysis (Bishop, 2006). The least squares classification problem is defined in Box 21.3.1 for a 1-out-of-K classifier scheme, where an element of 1 is appended to  $\mathbf{x}$  to give the  $(D+1) \times 1$  vector  $\tilde{\mathbf{x}} = (1, \mathbf{x}^T)^T$ , and  $w_0$  is appended to  $\mathbf{w}$  to give the  $(D+1) \times 1$  vector  $\tilde{\mathbf{w}} = (w_0, \mathbf{w}^T)^T$ .

To account for the multinomial 1-of-K classification scheme, we can use the compact variables  $\tilde{\mathbf{w}}$  and  $\tilde{\mathbf{x}}$  to define

$$y_k(\mathbf{x}) = \tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}}; \text{ for } k \in \{1, 2, \dots, K\}, \quad (21.5)$$

where the integer subscript  $k$  denotes the  $k^{th}$  class. If there are  $K$  classes, then the 1-out-of-K class  $K \times 1$  vector is  $\mathbf{y} = (0, 0, \dots, 0, 1, 0, \dots, 0)^T$ , where all of its element values are 0 except for the  $k^{th}$  one with class label  $y_k = 1$ . According to equation 21.5, there will be  $K$  row vectors  $\tilde{\mathbf{w}}_k^T$   $k \in \{1, 2, \dots, K\}$  that can be used to form the  $K \times (D+1)$  matrix  $\tilde{\mathbf{W}}^T$

To form a matrix  $\tilde{\mathbf{W}}$  from  $\tilde{\mathbf{w}}_k$ , define the  $K \times 1$  label vector  $\mathbf{y} = (y_1, y_2, \dots, y_K)^T$ . with the  $i^{th}$  component  $y_i = \delta_{ik}$  for  $i \in \{1, 2, \dots, K\}$ . This says that the elements of  $\mathbf{y}$  have value zero except for the  $k^{th}$  element  $y_k = 1$ . Therefore, the  $K \times (D+1)$  matrix  $\tilde{\mathbf{W}}^T$  can be formed from the  $k \in \{1, 2, \dots, K\}$  equations in equation 21.5 to give

$$\tilde{\mathbf{W}}^T \tilde{\mathbf{x}} = \mathbf{y}; \text{ or } \tilde{\mathbf{x}}^T \tilde{\mathbf{W}} = \mathbf{y}^T, \quad (21.6)$$

where the  $1 \times (D+1)$  vector  $(w_{k0}, \mathbf{w}_k^T)$  is the  $k^{th}$  row vector of  $\tilde{\mathbf{W}}^T$ ,  $\tilde{\mathbf{x}} = (1, \mathbf{x}^T)^T$  is the  $(D+1) \times 1$  data vector and  $\mathbf{y}^T$  is the  $1 \times K$  vector.

The training data  $(1, \mathbf{x}^{(n)})^T$  for  $n \in \{1, 2, \dots, N\}$  can be assembled into the  $N \times (D+1)$  matrix  $\tilde{\mathbf{X}}$ , such that

$$\tilde{\mathbf{X}} \tilde{\mathbf{W}} = \mathbf{Y}. \quad (21.7)$$

The  $n^{th}$  row of  $\tilde{\mathbf{X}}$  is the  $1 \times (D+1)$  training vector  $(1, \mathbf{x}^{(n)T})$  and the corresponding row in the  $N \times K$  matrix  $\mathbf{Y}$  consists of the associated  $1 \times (D+1)$  class vector  $\mathbf{y}^{(n)T}$ . The least squares solution to the above equations is given below, which is then used to compute the least squares discriminant that determines the class of a new input vector  $\mathbf{x}$ .



**Example 21.3.1. Least Squares Discriminant Analysis**

*Given :* Training Data  $(1, \mathbf{x}^{(n)})^T, \mathbf{y}^{(n)}$  for  $n \in \{1, 2, \dots, N\}$

and  $\mathbf{x}^{(n)} \in \mathbb{R}^D, \mathbf{y}^{(n)} \in \mathbb{R}^K, y_i^{(n)} \in \{0, 1\} \forall i, n.$

$$\tilde{\mathbf{X}}\tilde{\mathbf{W}} = \mathbf{Y}; \tilde{\mathbf{W}} \in \mathbb{R}^{(D+1) \times K}, \tilde{\mathbf{X}} \in \mathbb{R}^{N \times (D+1)}, \mathbf{Y} \in \mathbb{R}^{N \times K}$$

$$\text{Find : } \tilde{\mathbf{W}}^* = \arg \min_{\tilde{\mathbf{W}}} \text{Tr}[(\tilde{\mathbf{X}}\tilde{\mathbf{W}} - \mathbf{Y})^T(\tilde{\mathbf{X}}\tilde{\mathbf{W}} - \mathbf{Y})] \quad (21.8)$$

$$\text{Solution : } \tilde{\mathbf{W}}^* = (\tilde{\mathbf{X}}^T \tilde{\mathbf{X}})^{-1} \tilde{\mathbf{X}}^T \mathbf{Y}. \quad (21.9)$$

Here,  $\text{Tr}$  is the trace of the matrix where the diagonal elements are summed to get the sum of the squared errors (see Exercise 21.7.2). The class label of a new point  $\mathbf{x}^{new}$  is obtained by applying  $\tilde{\mathbf{w}}_k^{T*} \mathbf{x}^{new} = y_k$  and selecting the class index  $k^*$  such that

$$k^* = \underset{k \in \{1, 2, \dots, K\}}{\operatorname{argmax}} \overbrace{\tilde{\mathbf{w}}_k^{T*} \mathbf{x}^{new}}^{y_k}, \quad (21.10)$$

which is the one associated with the maximum value of  $y_k$  for  $k \in \{1, 2, \dots, K\}$ . For the two-class problem, the least squares solution is equivalent to the Fisher solution (Duda and Hart, 1973; Bishop, 2006). The Fisher solution for binary classification finds the optimal  $\mathbf{w}$ , which gives a large class separation between the means of two clusters projected along  $\mathbf{w}$ , while also giving a small variance within each class.

A problem with least squares discriminant analysis is that the quadratic misfit function in equation 21.8 is strongly influenced by outlier points that are far from the decision surface. This is illustrated in Figure 21.4a, which shows that both logistic regression and LSDA give similar decision surfaces, but the LSDA decision surface fails compared to logistic regression when there are a large number of outliers in Figure 21.4b. This problem can be mitigated by regularization, where a weighted diagonal matrix is added to equation 21.8 as a penalty term, with  $\|\tilde{\mathbf{w}}_k\|^2$  for the  $k^{th}$  diagonal element. Another possibility is replace the quadratic norm in equation 21.8 with the  $L_1$  norm of the data residual. The resulting solution algorithm is known as reweighted least squares (Bishop, 2006).

**21.3.2 Gaussian Discriminant Analysis**

Gaussian Discriminant Analysis (GDA) is a special case of the GMM in that each training point  $\mathbf{x}^{(n)} \in \mathbb{R}^D$  belongs to only one Gaussian, and the cluster membership, indicated by the scalar RV  $y^{(n)}$ , is known for each training point. Therefore GDA is a supervised learning method where the training pairs are  $(\mathbf{x}^{(n)}, y^{(n)})$ . If we know that the labels  $y^{(n)}$  and the points only belong to one of the Gaussians, then we can use Maximum Likelihood Estimation (MLE) to find the optimal values of the parameters  $(\boldsymbol{\mu}, \boldsymbol{\Sigma}, \phi)$ . This assumes that the probability distribution of the data is that of a Gaussian.

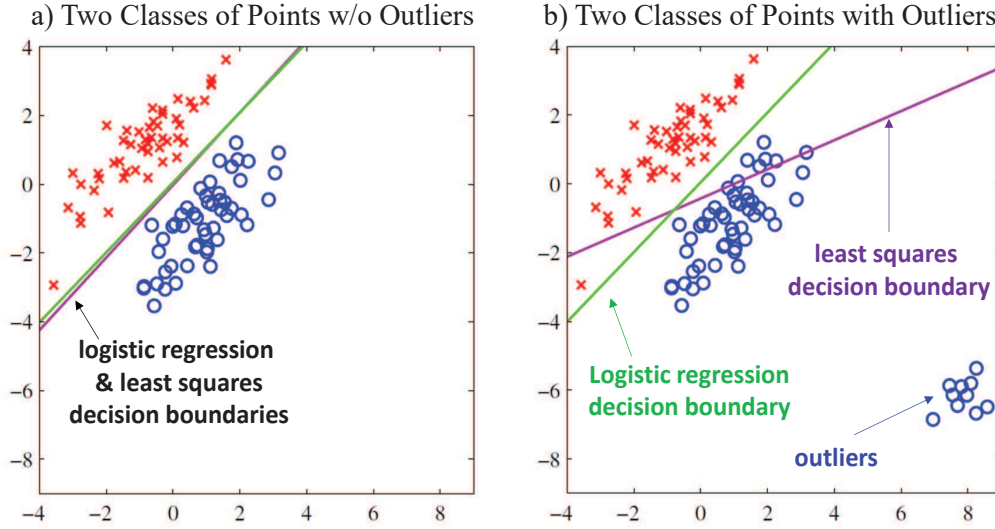


Figure 21.4: Comparison of decision boundaries computed by logistic regression and least squares LDA for points a) without and b) with outliers. Figure modified from Bishop (2006).

For example, joint probabilities for the data pairs  $(\mathbf{x}^{(n)}, y^{(n)})$  with  $y \in \{0, 1\}$  are given as

$$\begin{aligned} p(\mathbf{x}, y = 0) &= p(y = 0)p(\mathbf{x}|y = 0) \\ &= (1 - \phi) \frac{1}{\sqrt{(2\pi)^D |\boldsymbol{\Sigma}_0|}} e^{-\frac{1}{2}(\mathbf{x}^{(n)} - \boldsymbol{\mu}_0)^T \boldsymbol{\Sigma}_0^{-1} (\mathbf{x}^{(n)} - \boldsymbol{\mu}_0)}, \end{aligned} \quad (21.11)$$

$$\begin{aligned} p(\mathbf{x}, y = 1) &= p(y = 1)p(\mathbf{x}|y = 1) \\ &= \phi \frac{1}{\sqrt{(2\pi)^D |\boldsymbol{\Sigma}_1|}} e^{-\frac{1}{2}(\mathbf{x}^{(n)} - \boldsymbol{\mu}_1)^T \boldsymbol{\Sigma}_1^{-1} (\mathbf{x}^{(n)} - \boldsymbol{\mu}_1)}, \end{aligned} \quad (21.12)$$

where  $p(\mathbf{x}|y)$  is a generative probability function,  $\phi = p(y = 1)$ , and the subscripts in the  $\boldsymbol{\mu}$ 's and  $\boldsymbol{\Sigma}$ 's indicate the binary class of either  $y = 0$  or  $y = 1$ . We also know that  $y$  is a Bernoulli RV so its probability mass function is

$$p(y) = \phi^y (1 - \phi)^{1-y}, \quad (21.13)$$

where  $\phi$  is the probability that  $y = 1$  and  $1 - \phi$  is the probability that  $y = 0$ . Here,  $\boldsymbol{\mu}_k = (E(x_1)_k, E(x_2)_k, \dots, E(x_D)_k)^T$  for  $k \in \{0, 1\}$  is the  $D$ -dimensional mean vector,  $E[\cdot]$  represents the expectation or averaging operation over the random variables, and  $\boldsymbol{\Sigma}_k$  is the  $D \times D$  covariance matrix for the  $k^{th}$  cluster. For example, if the data dimension  $D = 3$ , then  $\mathbf{x}$  is a  $3 \times 1$  matrix,  $\boldsymbol{\mu}$  is a  $3 \times 1$  vector, and  $\boldsymbol{\Sigma}$  is a  $3 \times 3$  matrix where

$$\boldsymbol{\Sigma}_{ij} = E[(x_i - \mu_i)(x_j - \mu_j)]. \quad (21.14)$$

In practice, the covariance matrix is sometimes assumed to be a diagonal matrix<sup>2</sup>.

<sup>2</sup>Reynolds (2009) states: "...the component Gaussian are acting together to model the overall feature density, full covariance matrices are not necessary even if the features are not statistically independent. The

The optimal fit of a Gaussian to the points  $\mathbf{x}^{(n)}$  is equivalent to finding the statistical parameters  $(\boldsymbol{\mu}, \boldsymbol{\Sigma})$  that maximize the log-likelihood function  $\mathcal{L}$  defined as

$$\begin{aligned}\mathcal{L} &= \ln \prod_{n=1}^N p(\mathbf{x}^{(n)}, y^{(n)} = 1; \phi, \boldsymbol{\mu}, \boldsymbol{\Sigma}), \\ &= \ln \prod_{n=1}^N p(\mathbf{x}^{(n)} | y^{(n)} = 1) p(y^{(n)} = 1).\end{aligned}\quad (21.15)$$

If only one Gaussian is fitted to all the points then the closed form formulas for the parameters are given in Box 21.3.2. Box 21.2.3 gives the formulas for fitting two Gaussians, and the derivations are in Appendix 21.9.

**Example 21.3.2. Gaussian Discriminant Analysis for a Single Gaussian**

*Given :* Training Data  $(\mathbf{x}^{(n)}, y^{(n)} = 1)$  for  $n \in \{1, 2, \dots, N\}$   
 where  $\mathbf{x}^{(n)} \in \mathbb{R}^D$ ,  $y^{(n)} = 1$ ;  $p(y = 1) = \phi^y$ ;  

$$p(\mathbf{x}, y) = p(\mathbf{x}|y)p(y) = \frac{1}{\sqrt{(2\pi)^D |\boldsymbol{\Sigma}|}} e^{-\frac{1}{2}(\mathbf{x}^{(n)} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}^{(n)} - \boldsymbol{\mu})} p(y); \quad (21.16)$$

*Find :*  $(\phi^*, \boldsymbol{\mu}^*, \boldsymbol{\Sigma}^*) = \underset{(\phi, \boldsymbol{\mu}, \boldsymbol{\Sigma})}{\operatorname{argmax}} \underbrace{\mathcal{L}(\mathbf{x}^{(n)}, y^{(n)} = 1; \phi, \boldsymbol{\mu}, \boldsymbol{\Sigma})}_{\text{equation 21.15}}. \quad (21.17)$

where  $\mathcal{L}$  is the log-likelihood function. Defining  $y^{(n)} = 1 \forall n$ , defining the number of Gaussians  $K = 1$ , setting the gradients  $\nabla_\phi, \nabla_{\boldsymbol{\mu}}, \nabla_{\boldsymbol{\Sigma}}$  of  $\mathcal{L}$  to zero, and solving for  $\phi, \boldsymbol{\mu}$  and  $\boldsymbol{\Sigma}$ , gives (see Appendix 21.9)

$$\text{Solution : } \phi^* = \frac{1}{N} \sum_{n=1}^N \mathcal{I}\{y^{(n)} = 1\} = 1; \quad (21.18)$$

$$\boldsymbol{\mu}^* = \frac{\sum_{n=1}^N \mathcal{I}\{y^{(n)} = 1\} \mathbf{x}^{(n)}}{\sum_{n=1}^N \mathcal{I}\{y^{(n)} = 1\}} = \frac{\sum_{n=1}^N \mathbf{x}^{(n)}}{N}; \quad (21.19)$$

$$\begin{aligned}\boldsymbol{\Sigma}^* &= \frac{\sum_{n=1}^N \mathcal{I}\{y^{(n)} = 1\} (\mathbf{x}^{(n)} - \boldsymbol{\mu})^T (\mathbf{x}^{(n)} - \boldsymbol{\mu})}{N}, \\ &= \frac{\sum_{n=1}^N (\mathbf{x}^{(n)} - \boldsymbol{\mu})^T (\mathbf{x}^{(n)} - \boldsymbol{\mu})}{N},\end{aligned}\quad (21.20)$$

where  $\mathcal{I}\{y^{(n)} = k\}$  is an indicator function that is equal to 1 when the argument is true, otherwise it is equal to the value 0. The indicator function is equal to the Kronecker delta function  $\delta_{y^{(n)}k}$ .

---

linear combination of diagonal covariance basis Gaussians is capable of modeling the correlations between feature vector elements. The effect of using a set of  $M$  full covariance matrix Gaussians can be equally obtained by using a larger set of diagonal covariance Gaussians”.

**Example 21.3.3. Gaussian Discriminant Analysis for Two Gaussians**

*Given :* Training Data  $(\mathbf{x}^{(n)}, y^{(n)})$  for  $n \in \{1, 2, \dots, N\}$   
 where  $\mathbf{x}^{(n)} \in \mathbb{R}^D$ ,  $y^{(n)} \in \{0, 1\}$ ;  $p(y) = \phi^y(1 - \phi)^{1-y}$ ;  
 $p(\mathbf{x}, y) = p(\mathbf{x}|y)p(y)$ ,  

$$p(\mathbf{x}|y) = \frac{1}{\sqrt{(2\pi)^D |\Sigma_y|}} e^{-\frac{1}{2}(\mathbf{x}^{(n)} - \mu_y)^T \Sigma_y^{-1} (\mathbf{x}^{(n)} - \mu_y)}; \quad (21.21)$$

*Find :*  $(\phi^*, \mu_0^*, \mu_1^*, \Sigma_0^*, \Sigma_1^*) = \underset{(\phi, \mu_1, \Sigma_1, \mu_0, \Sigma_0)}{\operatorname{argmax}} \mathcal{L}(\mathbf{x}^{(n)}, y^{(n)}; \phi, \mu_0, \mu_1, \Sigma_0, \Sigma_1) \quad (21.22)$

where  $\mathcal{L} = \ln \Pi_{n=1}^N p(\mathbf{x}^{(n)}, y^{(n)}; \phi, \mu, \Sigma)$ , (21.23)

$= \ln \Pi_{n=1}^N (p(\mathbf{x}^{(n)}|y^{(n)})p(y^{(n)}))^{y^{(n)}} (p(\mathbf{x}^{(n)}|y^{(n)})p(y^{(n)}))^{1-y^{(n)}}. \quad (21.24)$

Setting the gradients  $\nabla_\phi, \nabla_\mu, \nabla_\Sigma$  of  $\mathcal{L}$  to zero, defining the number of Gaussians  $K = 2$ , and solving for  $\phi, \mu_k$  and  $\Sigma_k$ ,  $k \in \{0, 1\}$  gives (see Appendix 21.9)

*Solution :*  $\phi_k^* = \frac{1}{N} \sum_{n=1}^N \mathcal{I}\{y^{(n)} = k\}; \quad (21.25)$

$\mu_k^* = \frac{\sum_{n=1}^N \mathcal{I}\{y^{(n)} = k\} \mathbf{x}^{(n)}}{\sum_{n=1}^N \mathcal{I}\{y^{(n)} = k\}}; \quad (21.26)$

$\Sigma_k^* = \frac{\sum_{n=1}^N \mathcal{I}\{y^{(n)} = k\} (\mathbf{x}^{(n)} - \mu_k)^T (\mathbf{x}^{(n)} - \mu_k)}{N}. \quad (21.27)$

See Exercise 21.7.3.

Note that equation 21.25 counts the number of points that belong to the  $k^{th}$  Gaussian and divides by the total number of points in the training set. Therefore,  $\phi$  is the estimated chance that a point belongs to the  $k^{th}$  Gaussian<sup>3</sup>. The mean vector associated with the  $k^{th}$  Gaussian in equation 21.26 is estimated by summing up the points  $\mathbf{x}^{(n)}$  belonging to the  $k^{th}$  Gaussian cluster and dividing by the number of such points. The covariance terms are estimated in a similar way except the covariance values of points belonging to the  $k^{th}$  Gaussian cluster are summed and divided by the number of points in that cluster<sup>4</sup>.

In summary, GDA is a *supervised* learning and hard clustering method that requires assigning every point to just one of the  $K$  Gaussians. GDA provides closed form formulas for the statistical parameters and it is easily understood. This will help us understand the steps for the unsupervised and soft clustering GMM, where each point partially belongs to every weighted Gaussian. Unfortunately, not knowing the class labels of the input points prevents the derivation of closed-form formulas for  $(\mu, \Sigma)$ . Instead, an iterative method known as Expectation-Maximization is used to estimate these parameters.

<sup>3</sup>See lecture by Andrew Ng at <https://www.youtube.com/watch?v=rVfZHWTwXSA>.

<sup>4</sup>Estimated statistical parameters such as the mean are often denoted by the mean symbol  $\mu$  with a hat, i.e.  $\hat{\mu}$ . We will not use this notational distinction between the analytical and estimated formulas for the statistical parameters.

### Probability Functions

- 1).  $P(\mathbf{x}^{(n)})$ : Probability density function for finding a data point at  $\mathbf{x}^{(n)}$ .
- 2).  $P(z_k)$ : Probability a data point chosen at random is in  $k^{\text{th}}$  Gaussian. Denotes fraction of data points in  $k^{\text{th}}$  Gaussian.
- 3).  $P(z_k|\mathbf{x}^{(n)})$ : Probability that given data point at  $\mathbf{x}^{(n)}$  came from  $k^{\text{th}}$  Gaussian.  
 $P(z_k|\mathbf{x}^{(n)}) = P_{nk}$  responsibility matrix of dimension  $N \times K$ .
- 4).  $P(\mathbf{x}^{(n)}|z_k)$ : Probability that selected  $k^{\text{th}}$  Gaussian contains data point at  $\mathbf{x}^{(n)}$ .  
 $P(\mathbf{x}^{(n)}|z_k) := P(\mathbf{x}^{(n)}|\mathbf{w}_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$
- 5).  $\mathcal{L} := \prod_n P(\mathbf{x}^{(n)}|\mathbf{w}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ : Likelihood function. Probability finding all  $N$  points at  $(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(n)})$ :  
 $\mathcal{L} := P(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(N)}|\mathbf{w}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$  where  $(\mathbf{w}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = (w_1, w_2, \dots, w_K, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \dots, \boldsymbol{\mu}_K, \boldsymbol{\Sigma}_1, \boldsymbol{\Sigma}_2, \dots, \boldsymbol{\Sigma}_K)$

Figure 21.5: Interpretation of probability functions (Press et al., 2007) used for the GMM and EM algorithms.

## 21.4 Theory of the Gaussian Mixture Model

The Gaussian Mixture model (Reynolds et al., 2009; Bishop, 2006) for the random  $D \times 1$  vector  $\mathbf{x} = (x_1, x_2, \dots, x_D)^T$  is a sum of  $K$  weighted Gaussian densities given by

$$p(\mathbf{x}) = \sum_{k=1}^K w_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \quad (21.28)$$

where

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{1}{\sqrt{(2\pi)^D |\boldsymbol{\Sigma}_k|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x} - \boldsymbol{\mu}_k)\right), \quad (21.29)$$

and  $\mathbf{X} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(N)}\}$  is drawn independently from a Gaussian distribution<sup>5</sup>. The interpretation of  $p(\mathbf{x})$  is that it is the probability of finding a point at the position  $\mathbf{x}$ . For example, if the point  $\mathbf{x}$  is far from the mean of the fitted Gaussian then this point has a low probability of occurrence. See item 1 in Figure 21.5, where a list of probability functions are interpreted to help deconfuse the reader about their meaning.

The mixture weights are  $\mathbf{w} = (w_1, w_2, \dots, w_K)$  and, just like a discrete probability function, must be greater than or equal to zero and sum to the value 1:

$$\sum_{k=1}^K w_k = 1; \quad w_k \geq 0. \quad (21.30)$$

This can be proven by integrating equation 21.28 over  $\mathbf{x}$ , noting that  $1 \geq p(\mathbf{x}) \geq 0$  and remembering that the Gaussian is normalized. See Exercise 21.7.6.

The association of the rightside of equation 21.28 with a probability expression is given by the definition of the marginal probability  $p(\mathbf{x})$  expressed as a sum of weighted conditional probabilities:

$$p(\mathbf{x}) = \sum_{k=1}^K \overbrace{p(k)}^{w_k} \overbrace{p(\mathbf{x}|k)}^{\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}, \quad (21.31)$$

<sup>5</sup>Reynolds et al. (2009) uses the notation  $p(\mathbf{x}|\mathbf{w}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$  for the leftside of equation 21.28 while Bishop (2006) and Press et al. (2007) 'uses the traditional notation  $p(\mathbf{x})$  for a marginal probability function.

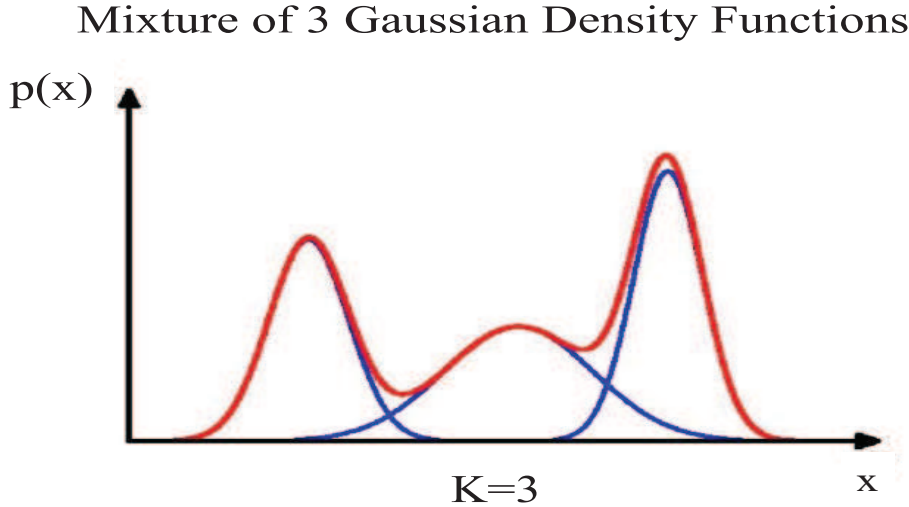


Figure 21.6: The red curve is the result of a weighted sum  $p(\mathbf{x}) = \sum_{k=1}^3 w_k \mathcal{N}(\mathbf{x}|\mu_k, \Sigma_k)$  of hidden causes, where each cause is governed by a Gaussian density function. Therefore, the GMM can form smooth approximations to an arbitrary density function. Figure modified from Bishop (2006).

where the mixture coefficient is defined as  $w_k = p(k)$ . Here,  $p(k)$  is interpreted as the fraction of the points shared by the  $k^{th}$  Gaussian<sup>6</sup>. See item 2 in Figure 21.5 and the red curve  $p(\mathbf{x})$  in Figure 21.6.

The data points are assumed to be independent so the GMM log-likelihood function is

$$\begin{aligned} \mathcal{L} &= \ln \prod_{n=1}^N p(\mathbf{x}^{(n)}), \\ &= \sum_{n=1}^N \ln \sum_{k=1}^K \overbrace{w_k \mathcal{N}(\mathbf{x}^{(n)}|\mu_k, \Sigma_k)}^{p(k)p(\mathbf{x}|k) \text{ in equation 21.31}}. \end{aligned} \quad (21.32)$$

The gradients of  $\mathcal{L}$  can be set to zero, similar to the GDA procedures in Boxes 21.3.2-21.2.3, in order to estimate the mean vectors and covariance matrices for each Gaussian (see Box 21.4.1). However, the labels and weights  $w_k = p(k)$  are unknown as well and so obtaining closed form formulas for the parameters  $(\mathbf{w}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$  is not possible. Therefore, an iterative solution method is used for computing these parameters, and one of the most popular methods is known as the Expectation-Maximization (EM) method in Box 21.4.2.

Before discussing the EM method, we first define the responsibility function  $p(k|\mathbf{x}^{(n)})$ . This is also denoted as a discriminative function which is required because the generative function  $p(\mathbf{x}^{(n)}|k) = \mathcal{N}(\mathbf{x}^{(n)}|\mu_k, \Sigma_k)$  in equation 21.32 will be computed using Bayes' theorem (see equation 21.38), which requires knowing the so-called responsibility  $p(k|\mathbf{x}^{(n)})$ .

---

<sup>6</sup>Any single point is shared by all the Gaussians, but the sharing is not always equal for each Gaussian. If there are two Gaussians and one of them has a fractional ownership of  $w_1 = 1/3$  for all the points then the other Gaussian has  $w_2 = 2/3$  ownership.

**Example 21.4.1. Gaussian Mixture Model**

Given : Training Data  $(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(N)})$ ,

$$\mathcal{L} = \sum_{n=1}^N \ln \sum_{k=1}^K \overbrace{w_k \mathcal{N}(\mathbf{x}^{(n)} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}^{\text{equation 21.28}}. \quad (21.33)$$

$$p(\mathbf{x}) = \sum_{k=1}^K w_k \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \quad (21.34)$$

$$\text{Find : } (\mathbf{w}^*, \boldsymbol{\mu}^*, \boldsymbol{\Sigma}^*) = \underset{(\mathbf{w}, \boldsymbol{\mu}, \boldsymbol{\Sigma})}{\operatorname{argmax}} \mathcal{L}(\mathbf{w}, \boldsymbol{\mu}, \boldsymbol{\Sigma}), \quad (21.35)$$

$$\text{where } 0 \leq w_k \leq 1 \text{ and } \sum_{k=1}^K w_k = 1. \quad (21.36)$$

$$\text{Solution : } \text{Expectation} - \text{Maximization}. \quad (21.37)$$

The condition  $\sum_{k=1}^K w_k = 1$  follows from the fact that  $p(\mathbf{x})$  and the Gaussian components are normalized so that integrating equation 21.34 over  $\mathbf{x}$  is equal to 1. Also, the Gaussian has the property  $\mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \geq 0$  so that a sufficient condition for  $p(\mathbf{x}) \geq 0$  is that  $w_k \geq 0$  for all  $k$ .

The maximum likelihood estimator finds  $(\mathbf{w}^*, \boldsymbol{\mu}^*, \boldsymbol{\Sigma}^*)$  that maximizes the log-likelihood function  $\mathcal{L}$ . These parameters are computed by the expectation-maximization algorithm (Dempster et al., 1977) described in section 21.4.2 and Box 21.3.2.

**21.4.1 Responsibility Function  $p(k|\mathbf{x}^{(n)})$** 

The Gaussians in the GMM all share the same points and so the sharing responsibility function  $p(k|\mathbf{x}^{(n)})$  of the  $k^{th}$  Gaussian needs to be estimated by the EM algorithm for a given point  $\mathbf{x}^{(n)}$ . But how do we get the responsibility function  $p(k|\mathbf{x}^{(n)})$  if we only know the function  $p(\mathbf{x}^{(n)}|k) = \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$  from equation 21.31 and the marginal probability  $p(k)$ ? The answer is Bayes' theorem:

$$p_{nk} = p(k|\mathbf{x}^{(n)}) = \frac{p(\mathbf{x}^{(n)}|k)p(k)}{p(\mathbf{x}^{(n)})}, \quad (21.38)$$

where  $p_{nk} = p(k|\mathbf{x}^{(n)})$  is interpreted as the probability that  $\mathbf{x}^{(n)}$  is in the  $k^{th}$  Gaussian. Since each point belongs to every Gaussian, then  $p_{nk}$  is also interpreted as the fraction of the point  $\mathbf{x}^{(n)}$  whose *responsibility* is shared by the  $k^{th}$  Gaussian. See items 2-4 in Figure 21.5.

The intuitive understanding of the responsibility function is shown in Figure 21.7. Here, Figure 21.7a depicts a group of unlabeled points that will be GMM fitted to two Gaussians. After an approximate fitting, the two Gaussians in Figure 21.7b share each point, and the fraction of sharing is denoted by the multicolored circles. The ratio of blue and red colors in a circle is proportional to the ratio of weights  $p(k=1|\mathbf{x}^{(n)})/p(k=2|\mathbf{x}^{(n)})$ , where  $k=1$  indicates the blue Gaussian and  $k=2$  indicates the red one.

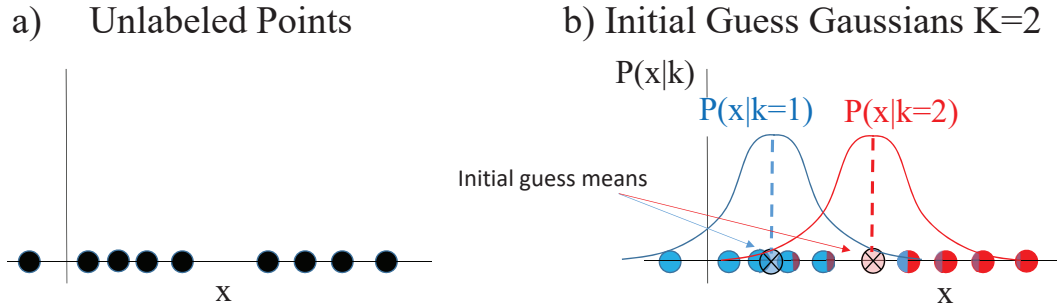


Figure 21.7: a) Unlabeled points and b) points fitted to blue and red Gaussians, as indicated by the latent variables  $k = 1$  and  $k = 2$ , respectively. The latent variable  $k$  is a compressed code indicating the  $k^{th}$  Gaussian associated with  $(w_k, \mu_k, \Sigma_k)$ . Each point is shared by both the blue and red Gaussians, where the ratio of blue/red colors in each point is proportional to the ratio  $p(k = 1|x)/p(k = 2|x)$ .

### 21.4.2 Expectation-Maximization Method

The EM method is similar to the alternating direction method in that the original set of unknown parameters is broken up into two sets. As described in Box 21.4.2, the parameter values for the first set are guessed at, and the second set is solved for. Then, the updated second-set parameters are used to update the first set. This procedure is repeated until convergence.



**Example 21.4.2.** *Expectation-Maximization Algorithm*

1. **Initial Values of Parameters.** Guess the number  $K$  of Gaussians and the initial values for the first set of parameters  $\Phi_1 = (p(k), \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$  for  $k \in \{1, 2, \dots, K\}$ . The initial guess parameters give rise to the red and blue Gaussians in Figure 21.8b, and these guessed parameters can be provided by a K-means method. For example, the value of  $p(k) \approx N_k/N$  can be estimated by counting the number of points  $N_k$  in the  $k^{\text{th}}$  K-means cluster and dividing by the total number  $N$  of points. The means and covariances of each K-means cluster can easily be computed to get  $(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \forall k \in \{1, 2\}$  for the red and blue Gaussians in Figure 21.8b.
2. **Expectation Step.** Knowing  $\Phi_1 = (p(k), \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ , the second set of unknowns  $\Phi_2 = p(k|\mathbf{x}^{(n)}) = p_{nk}$ , aka the responsibility functions, are computed using equation 21.38 from Bayes' theorem. Here,  $p(\mathbf{x})$  is obtained from equation 21.31 and  $p(\mathbf{x}^{(n)}|k)$  is obtained from the Gaussian  $\mathcal{N}(\mathbf{x}^{(n)}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ .
3. **Maximization Step.** The updated values of  $\Phi_2 = p(k|\mathbf{x}^{(n)})$  can then be used to form a new  $\mathcal{L}$  in equation 21.28, and the gradients are computed w/r to the variables in  $\Phi_1 = (p(k), \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ . These gradients are set to zero to get updates to  $\Phi_1$ . This is the *Maximization-* or M-step and the formulas for  $(p(k), (\mathbf{w}, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k))$  are arrived at in a manner similar to the procedure in Appendix 21.9 except every Gaussian shares any point  $\mathbf{x}^{(n)}$ . That is, the hard-clustering Kronecker delta functions in Appendix 21.9 are replaced by the soft-clustering responsibility functions. These formulas (Bishop, 2006; Press et al., 2007) are given by

$$\boldsymbol{\mu}_k = \frac{\sum_{n=1}^N p_{nk} \mathbf{x}^{(n)}}{\sum_{n=1}^N p_{nk}}; \quad (21.39)$$

$$\boldsymbol{\Sigma}_k = \frac{\sum_{n=1}^N p_{nk} (\mathbf{x}^{(n)} - \boldsymbol{\mu}_k)(\mathbf{x}^{(n)} - \boldsymbol{\mu}_k)^T}{\sum_{n=1}^N p_{nk}}; \quad (21.40)$$

$$N_k = \sum_{n=1}^N p_{nk}; \quad p(k) = w_k = N_k/N. \quad (21.41)$$

4. **Iteration Steps.** Steps 2-3 are repeated until there is little change in  $\mathcal{L}$ . Figure 21.8c-21.8d illustrate the red and blue Gaussians at the intermediate and final iterations. See Exercises 21.7.4.

**21.4.3 1D Example of a Responsibility Function**

As a 1D example of a responsibility function, the black curves in Figure 21.9 depict the  $p(k=1|x)$  and  $p(k=2|x)$  conditional probabilities for the Bernoulli variable  $k$ . In contrast, the colored Gaussian curves depict the conditional probabilities  $p(x|k=1)$  and  $p(x|k=2)$ . Here, the  $p(k|x)$  probabilities are those for the Bernoulli distribution.

The solid black curve in Figure 21.9 represents the sigmoid probability  $p(k=2|x) = 1/(1 + e^{-\mathbf{w}_2 x - w_{20}})$  and says that the red points on the right are highly likely to belong to the red  $k=2$  Gaussian because they are closer to the red mean point marked by the red x. This is confirmed by

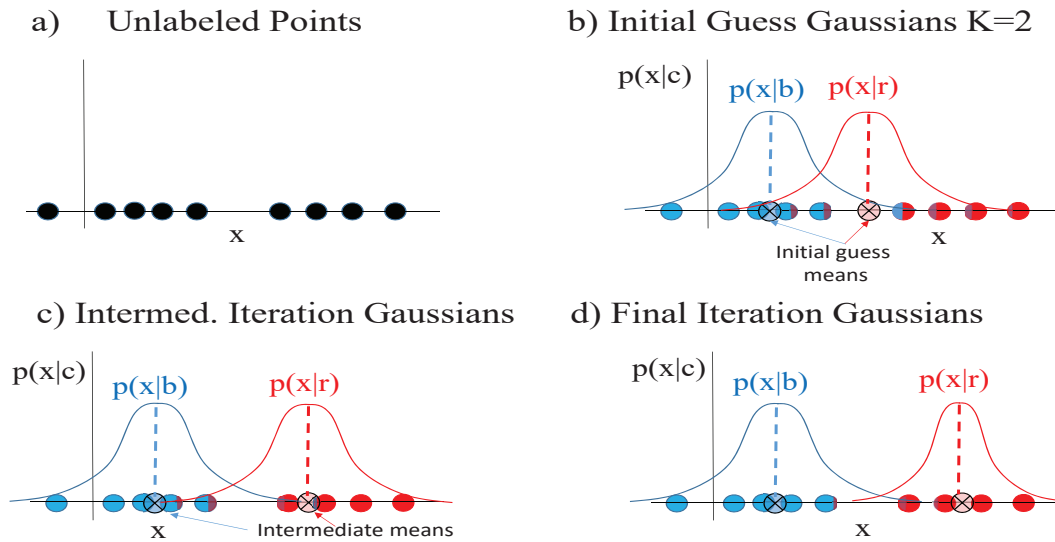


Figure 21.8: a) Unlabeled data points along the real line. The conditional probabilities  $p(x|r)$  and  $p(x|b)$  are plotted against  $x$  for b) the initial guessed means (indicated by the colored symbols  $\otimes$ ) and variances, c) the means and variances determined by an intermediate iteration of the GMM, and the d) final iteration of the GMM. Here, the percent membership of each point with either a blue or red Gaussian is colored by a blend of blue and red colors, where the red and blue color proportions are equal to the values of the responsibility functions  $p(r|x)$  and  $p(b|x)$ .

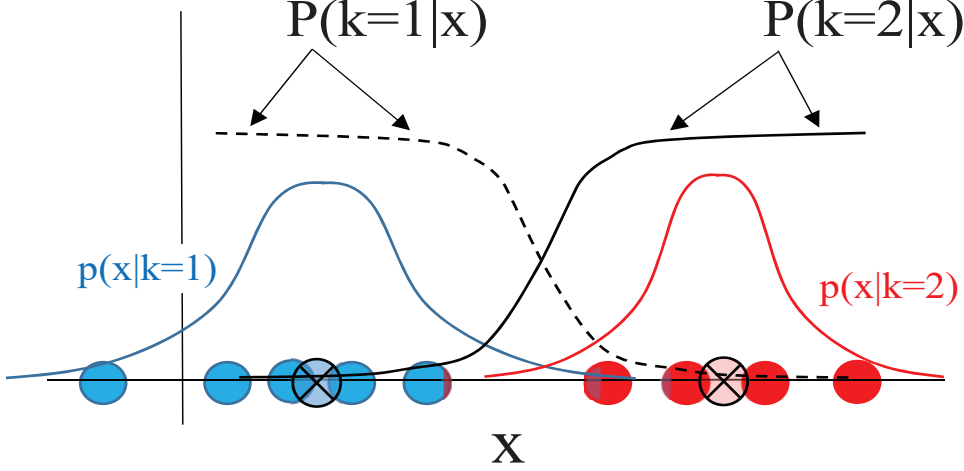


Figure 21.9: Black-solid and black-dashed lines depict the  $p(k = 2|x)$  and  $p(k = 1|x)$  posterior conditional probabilities. The reverse class-conditional probabilities  $p(x|k = 1)$  and  $p(x|k = 2)$  are plotted as the colored Gaussian density curves.

setting  $p(x|k = 2) \gg p(x|k = 1)$  for  $x \in \text{red}$  so that

$$\begin{aligned} p(k = 2|x) &= \frac{p(x|k = 2)p(k = 2)}{p(x|k = 1)p(k = 1) + p(x|k = 2)p(k = 2)}, \\ &\approx p(x|k = 2)/p(x|k = 2) = 1, \quad \mathbf{x} = \text{red points} \end{aligned} \quad (21.42)$$

for the red points on the right. In contrast, the points on the left have a very small probability  $p(k = 2|x \in \text{left}) \approx 0$  of belonging to the red Gaussian.

#### 21.4.4 Geophysical Inversion

The goal of geophysical inversion is to find the Earth model  $\mathbf{m}$  that best explains the recorded data  $\mathbf{d}$ . Here, the forward modeling operator  $\mathbf{L}$  computes the physics-based prediction  $\mathbf{Lm} = \mathbf{d}$  of the data  $\mathbf{d}$  from the model  $\mathbf{m}$ . The recorded data have random amplitude errors so the data amplitudes are RVs with the conditional probability given by  $p(\mathbf{d}|\mathbf{m})$ .

In contrast to forward modeling as the deterministic data generator, the  $L_2$  estimate of the model  $\hat{\mathbf{m}} = [\mathbf{L}^T \mathbf{L}]^{-1} \mathbf{L}^T \mathbf{d}$  plays the role of the *statistical discriminator*  $p(\mathbf{m}|\mathbf{d})$ . If prior information about the model  $\mathbf{m}$  is to be used then the regularized solution is given by  $\hat{\mathbf{m}} = [\mathbf{L}^T \mathbf{L} + \lambda \mathbf{I}]^{-1} \mathbf{L}^T \mathbf{d}$ , which accounts for the priori information that  $\|\mathbf{m}\|^2 \approx 0$  in this example. This is equivalent to the prior knowledge of  $p(\mathbf{m})$  used in the formula for Bayes' rule  $p(\mathbf{m}|\mathbf{d}) = \frac{p(\mathbf{d}|\mathbf{m})p(\mathbf{m})}{p(\mathbf{d})}$ .

#### 21.4.5 Example of Generative and Discriminative Modeling

One of the advantages of using an adaptive generative approach to computing the decision boundary is that it requires less data compared to the discriminant approach. For example, the NN is a discriminative method using the discriminant modeler  $p(Y|X)$  that requires a huge number of training examples to obtain an accurate NN model. In comparison, the Gaussian generative

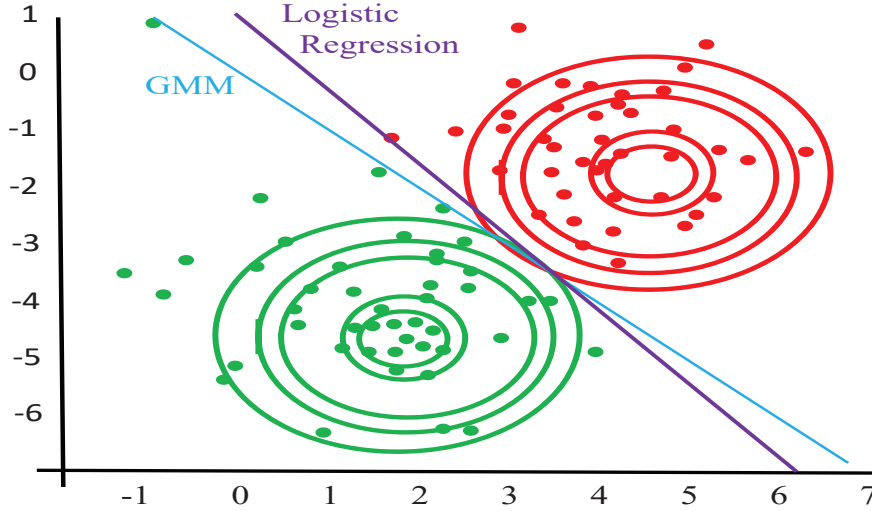


Figure 21.10: Best fit Gaussians to the binary points provide the blue decision boundary between the green and red points while logistic regression provides the purple decision boundary. Figure adapted from Lecture 5 at Stanford University (<https://betterprogramming.pub/generative-vs-discriminative-models-d26def8fd64a>).

model, represented by  $p(X|Y)$ , often requires much less training data because it fits an analytic function to the training data. Empirical results suggest that this fitting procedure does not require so many training examples to get accurate models. Then, Bayes' theorem can be used to estimate the discriminant model, represented by  $p(Y|X)$ , that will accurately predict the class of untrained data<sup>7</sup>.

As a comparison, Figure 21.10 depicts the results for estimating the decision boundary using a supervised Gaussian modeling method and a supervised logistic regression method. Both methods give adequate decision boundaries, but the Gaussian method requires less training data. For the Gaussian method, the maximum likelihood estimates are obtained by maximizing a joint log-likelihood function for all of the training data  $(\mathbf{x}^{(n)}, y^{(n)})$

$$\begin{aligned} \text{joint log likelihood: } \mathcal{L}(\phi, \boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_0, \boldsymbol{\Sigma}_1) &= \ln \prod_{n=1}^N p(\mathbf{x}^{(n)}, y^{(n)}), \\ &= \ln \prod_{n=1}^N p(\mathbf{x}^{(n)} | y^{(n)}) p(y^{(n)}). \end{aligned} \quad (21.43)$$

In comparison, the log-likelihood function for a neural network discriminant model is

$$\text{conditional log likelihood: } \hat{l}(\mathbf{w}) = \ln \prod_{n=1}^N p(y^{(n)} | \mathbf{x}^{(n)}, \mathbf{w}), \quad (21.44)$$

where  $\hat{l}(\mathbf{w})$  is the conditional likelihood compared to the joint likelihood  $\mathcal{L}(\phi, \boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_0, \boldsymbol{\Sigma}_1)$  in equation 21.43.

<sup>7</sup><https://betterprogramming.pub/generative-vs-discriminative-models-d26def8fd64a>.

	<b>GMM</b>	<b>Fuzzy Clustering</b>
Objective function	$\mathcal{L} = \sum_{n=1} \ln \sum_{k=1}^K w_k \mathcal{N}(\mathbf{x}^{(n)}   \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$	$\varepsilon = \frac{1}{2} \sum_{n=1} \sum_{k=1}^K w_{kn} \ \mathbf{x}^{(n)} - \mathbf{C}_k\ ^2$
Weight constraint	$\sum_{k=1}^K w_k = 1$	$\sum_{k=1}^K w_{kn} = 1$
Weight formula	$p(\mathbf{x}^{(n)}   k) = \mathcal{N}(\mathbf{x}^{(n)}   \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k);$ $p_{nk} = \frac{p(\mathbf{x}^{(n)}   k) w_k}{p(\mathbf{x}^{(n)})}$ $w_k = \frac{\sum_{n=1}^N p_{nk}}{N}$	$w_{kn} = \frac{\ \mathbf{x}^{(n)} - \mathbf{C}_k\ ^{-1}}{\sum_{k'=1}^K \ \mathbf{x}^{(n)} - \mathbf{C}_{k'}\ ^{-1}}$
Mean	$\boldsymbol{\mu}_k = \frac{\sum_{n=1}^N p_{nk} \mathbf{x}^{(n)}}{\sum_{n=1}^N p_{nk}}$	$\mathbf{C}_k = \frac{\sum_{n=1}^N w_{kn} \mathbf{x}^{(n)}}{\sum_{n=1}^N w_{kn}}$
Covariance	$\boldsymbol{\Sigma}_k = \frac{\sum_{n=1}^N p_{nk} (\mathbf{x}^{(n)} - \boldsymbol{\mu}_k) (\mathbf{x}^{(n)} - \boldsymbol{\mu}_k)^T}{\sum_{n=1}^N p_{nk}}$	

Figure 21.11: Formulas for the soft unsupervised clustering methods of GMM and the FCM. See Chapter 17 for the details of fuzzy clustering.

#### 21.4.6 Fuzzy Clustering and GMM

The Fuzzy C-means Clustering method (FCM) in Chapter 17 is an unsupervised clustering method where any point is shared by every cluster. Therefore, it is a soft clustering method similar to that of GMM. However, unlike fuzzy clustering, GMM produces a PDF  $p(\mathbf{x})$  that provides a probabilistic estimate of its membership to any cluster, which is important if uncertainty estimates are required for the features of interest. For example, FCM was used in section 17.5 to pick arrival times of P and S waves in seismic data. These traveltimes can be used to estimate hypocenter locations or be inverted to give the velocity tomogram. In these cases, it is important to provide the standard deviation of traveltime picks so they can be used to estimate errors in the inverted model. Only GMM provides reliable statistics in this example. A comparison of the formulas for GMM and the FCM are displayed in Figure 21.11. The key advantage of GMM compared to fuzzy clustering is that it produces a posterior PDF for the model, but the disadvantage is that FCM is theoretically simpler and does not have to abide by the constraints of probability theory. Therefore it is more easily adapted to different problems at the expense of less rigor.

### 21.5 Numerical Examples of GMM in Geoscience

The GMM can be used to cluster many types of geophysical data. We now give several examples associated with analysis of turbidite models from seismic data and seismic inversion.

#### 21.5.1 Turbidite Identification from Seismic Data

Bradley and Hardisty (2019) extracted classes of seismic facies by applying GMM to data computed by a self-organizing map (SOM) of seismic data. Figure 21.12b depicts the SOM associated with the seismic horizon slice in Figure 21.12a.

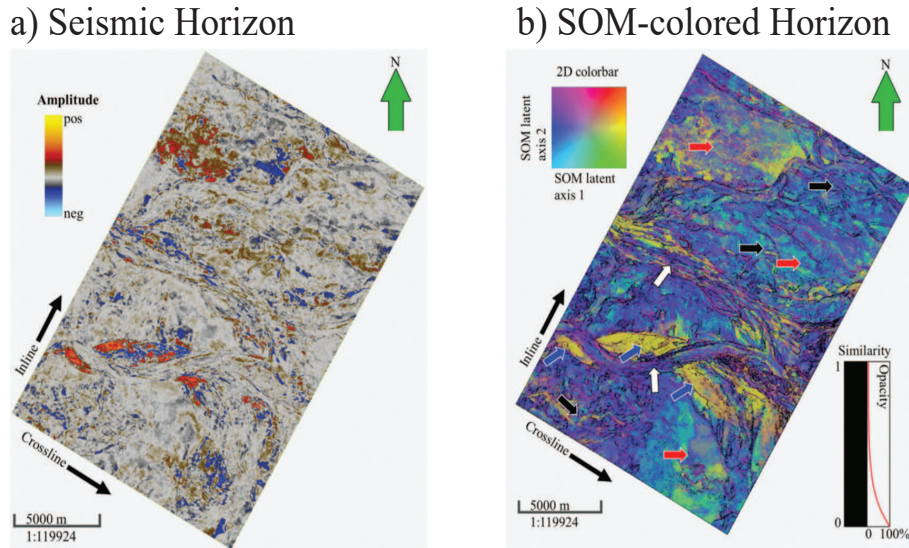


Figure 21.12: Slices of the a) seismic horizon from a 3D seismic data cube and b) SOM coloration associated with the seismic data slice. The red arrows indicate possible slope fans, the blue arrows indicate high-amplitude deposits, the white arrows indicate stacked channels, and the black arrows indicate weak sinuous channels (Zhao et al., 2016). A 2D color bar is used to represent two SOM latent axes.

The attributes associated with the input to the SOM procedure were peak spectral frequency, peak spectral magnitude, coherent energy, and curvedness of a seismic horizon slice tied to a picked continuous reflector below a turbidite system. The output of the SOM was projected onto a 2D manifold in Figure 21.12b. The 2D points in the SOM manifold were then used as input training data into a GMM with the goal of generating maps of different (though still unsupervised) seismic facies. Bradley and Hardisty (2019) used Bayes' Theorem to obtain uncertainty estimates that were corendered with their facies classification map. The number of clusters was obtained using an elbow-like method where the Bayesian Information Criterion<sup>8</sup> (BIC) is plotted against the number of different clusters (Schwarz, 1978), as shown in Figure 21.13. The number of Gaussian clusters on the elbow of the BIC curve was selected as the optimal number, which was selected as 7 clusters.

Figure 21.14 depicts the colors associated with the seven GMM clusters<sup>9</sup>, which reveals a more quantitative identification of the turbidite channels compared to the SOM image in Figure 21.12b. The uncertainty map is a byproduct of the GMM, which produces statistical information from  $p(\mathbf{m}|\mathbf{d})$  at every point of the horizon. Bradley and Hardisty (2019) use these maps to make the following interpretation of colors 1-7 in Figure 21.14a. For a non-geologist, these final results do

<sup>8</sup>The BIC is defined as  $BIC = k \ln(N) - 2 \ln L^*$ , where  $N$  = number of data points,  $L^* = p(\mathbf{x}|\boldsymbol{\mu}^*, \boldsymbol{\Sigma}^*, \mathbf{w}^*)$  is the maximized likelihood function,  $k$  is the number of unknown parameters estimated by the model, and the star superscript indicates the optimal parameters. See [https://en.wikipedia.org/wiki/Bayesian\\_information\\_criterion](https://en.wikipedia.org/wiki/Bayesian_information_criterion).

<sup>9</sup>Instead of the EM algorithm, the neighborhood EM algorithm is used. Bradley and Hardisty state the following: 'The neighborhood EM (NEM) algorithm is implemented to consider spatial correlations using a small subset of the horizon to highlight a channel feature. NEM adds a component to the objective function being optimized that rewards neighboring observations having similar cluster posterior probabilities. In such a way, NEM constructs models that have the data well-explained and spatially constrained to be smooth to such an extent as the data support.'

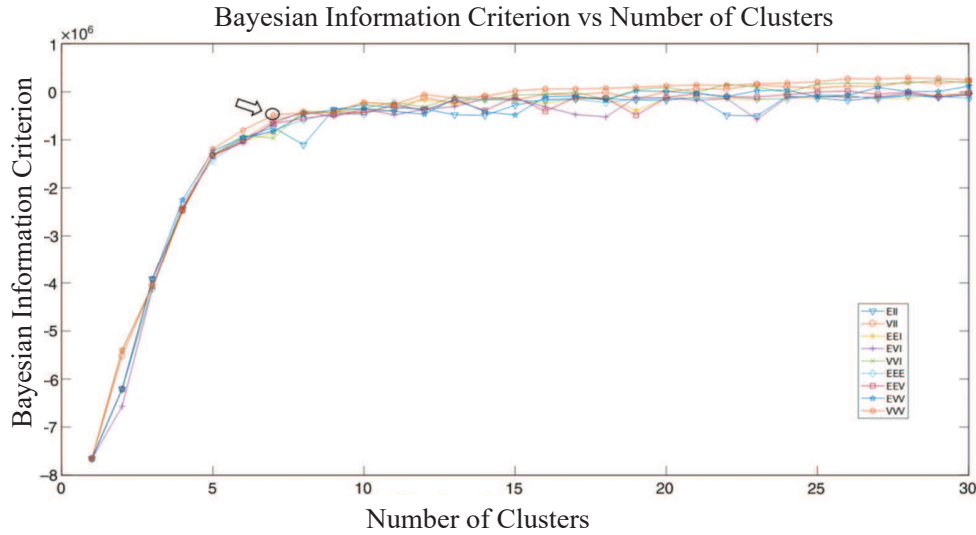


Figure 21.13: Plot of BIC vs number of clusters. The module symbols, EII, VII, EEI, EVI, etc., represent the different parameterizations of the covariance matrix. Because the BIC does not decrease with the number of clusters, the model on the shoulder of the curve is picked. Figure from Bradley and Hardisty (2019).

not appear too different from the original SOG image, but the subtle changes appear to make a difference in the final interpretation.

1. Discontinuous; widespread.
2. Discontinuous; some constraint in distribution.
3. Clear lineaments along channels.
4. Continuous; possibly slope fan and lobe deposits.
5. Discontinuous; some constraint in distribution.
6. Continuous; associated with slope fan and lobe deposits.
7. Continuous; constrained to channels, slope fan, and lobe deposits;
8. lobe deposits; sandy

At the end of their paper they state the following: 'Ambiguity and uncertainty measurements highlight the strengths of GMMs as an interpretation tool.'

### 21.5.2 Gravity and Magnetic Inversion with GMM Constraints

Giraud et al. (2017) jointly inverted synthetic gravity and magnetic data using a GMM density function as a penalty function in the following objective function:

$$\epsilon = \underbrace{\|\mathbf{d}(\mathbf{m}) - \mathbf{d}^{data}\|_2^2}_{\text{data misfit}} + \underbrace{\alpha \|\mathbf{m} - \mathbf{m}^{prior}\|_2^2}_{\text{model penalty function}} + \underbrace{\beta \|p(\mathbf{m}_\mu) - p(\mathbf{m})\|_2^2}_{\text{petrophysical PDF misfit}}, \quad (21.45)$$



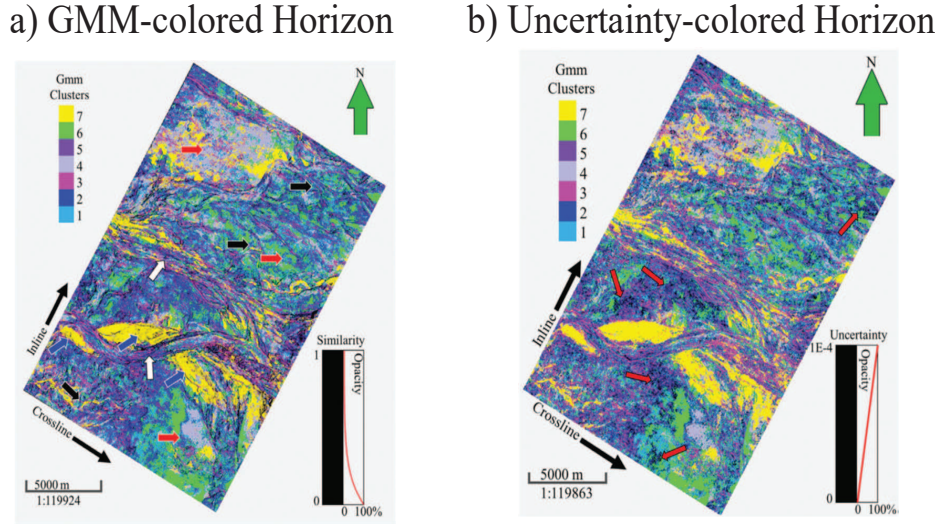


Figure 21.14: Plots of a) classified turbidite system using a GMM and b) ambiguity plot overlain on classification image. Figure from Bradley and Hardisty (2019).

where  $\alpha, \beta > 0$  are damping coefficients. The predicted data vector  $\mathbf{d}(\mathbf{m}) = (\mathbf{d}(\mathbf{m})^{grav}, \mathbf{d}(\mathbf{m})^{mag})$  consists of the gravity data  $\mathbf{d}(\mathbf{m})^{grav}$  and magnetic data  $\mathbf{d}(\mathbf{m})^{mag}$  simulated along the surface in the Figure 21.15a models. The term  $p(\mathbf{m}_\mu)$  is a Gaussian PDF with the mean value of the Gaussians associated with the petrophysical data. Recorded data are denoted as  $\mathbf{d}^{data}$  and an example of the slowly-varying gravity data and rapidly-varying magnetic data is in the second row of graphs in Figure 21.15a. The data misfit and model penalty terms in equation 21.45 silently contain the data and model covariance matrices  $\mathbf{C}^{data}$  and  $\mathbf{C}^{model}$ , respectively. Giraud et al. (2017) describes them as spherical covariance matrices corresponding to data and model noise.

The petrophysical PDF term  $p(\mathbf{m})$  is a weighted sum of Gaussian density functions that account for the prior estimates of both the gravity and magnetic models. It describes the GMM model of crossplot values shown in Figure 21.15b. In practice, this PDF is obtained by fitting a sum of weighted Gaussians, e.g. equation 21.31, to well-log and rock measurements. An idealized crossplot of 4 Gaussian functions fitted to petrophysical properties is depicted in Figure 21.16c. The properties for the  $i^{th}$  cell of the model might be assigned to the red peak in Figure 21.16c, but can change for different cells associated with different lithologies.

The simulated gravity and magnetic data are shown in Figure 21.15a and the crossplot of the physical property values is shown in Figure 21.15b; here, there are 5 different lithologies denoted by numbers at the peaks of the colored contours. The means and variances of the simulated Gaussians are given in Figure 21.15c and represent real-world estimates from different rock types.

The problem with separately inverting gravity data without taking into account constraints is illustrated in Figure 21.15b. The horizontal dashed green line intersects the peaks labeled as +2 and +4. This means that there are two different rock types with the same magnetic susceptibility but dramatically different densities. This non-unique mapping between susceptibility and two different rock types cannot be resolved by solely inverting the magnetic data in Figure 21.15a. However, if the gravity and magnetic data are jointly inverted using the GMM constraint  $\|p(\boldsymbol{\mu}) - p(\mathbf{m})\|^2$  in equation 21.45, then the inverted density and susceptibility models shown in Figure 21.17c are much more accurate than the Figure 21.17b models reconstructed by unconstrained inversion. This highlights the value of GMM as a means for constraining joint inversion of different types of data.



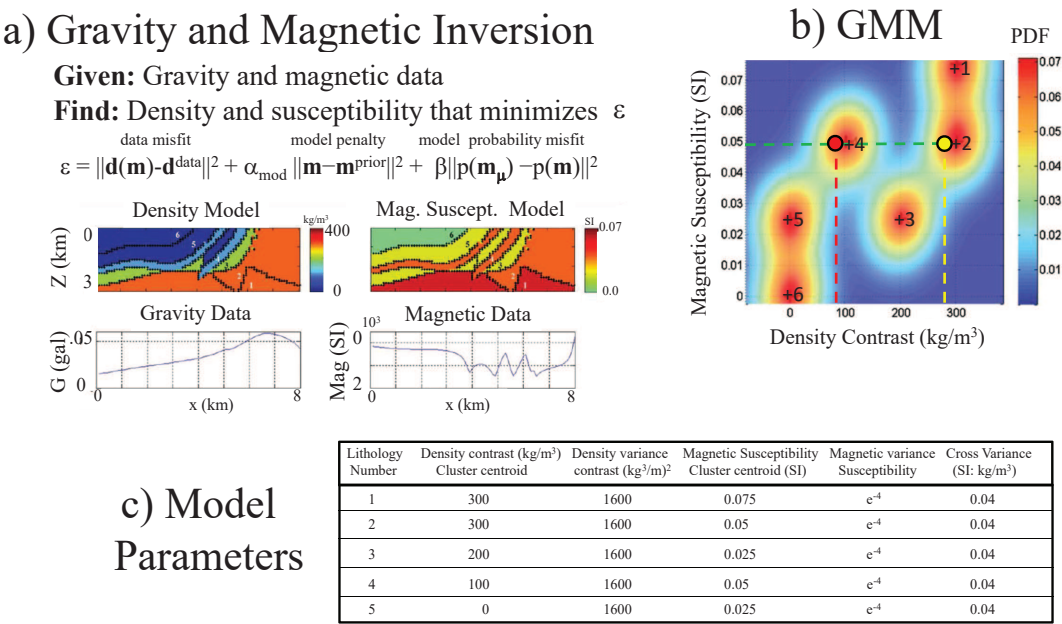


Figure 21.15: a) Definition of joint inversion problem for gravity and magnetic data. b) crossplot of the GMM density function  $P(\mathbf{m})$ , where  $\mathbf{m} = (\Delta\rho, S)$ , and c) table of mean and variance values of susceptibility and density for different lithologies. Here, density contrast and magnetic susceptibility are represented by  $\Delta\rho$  and  $S$ , respectively. Figure adapted from Giraud et al. (2017).

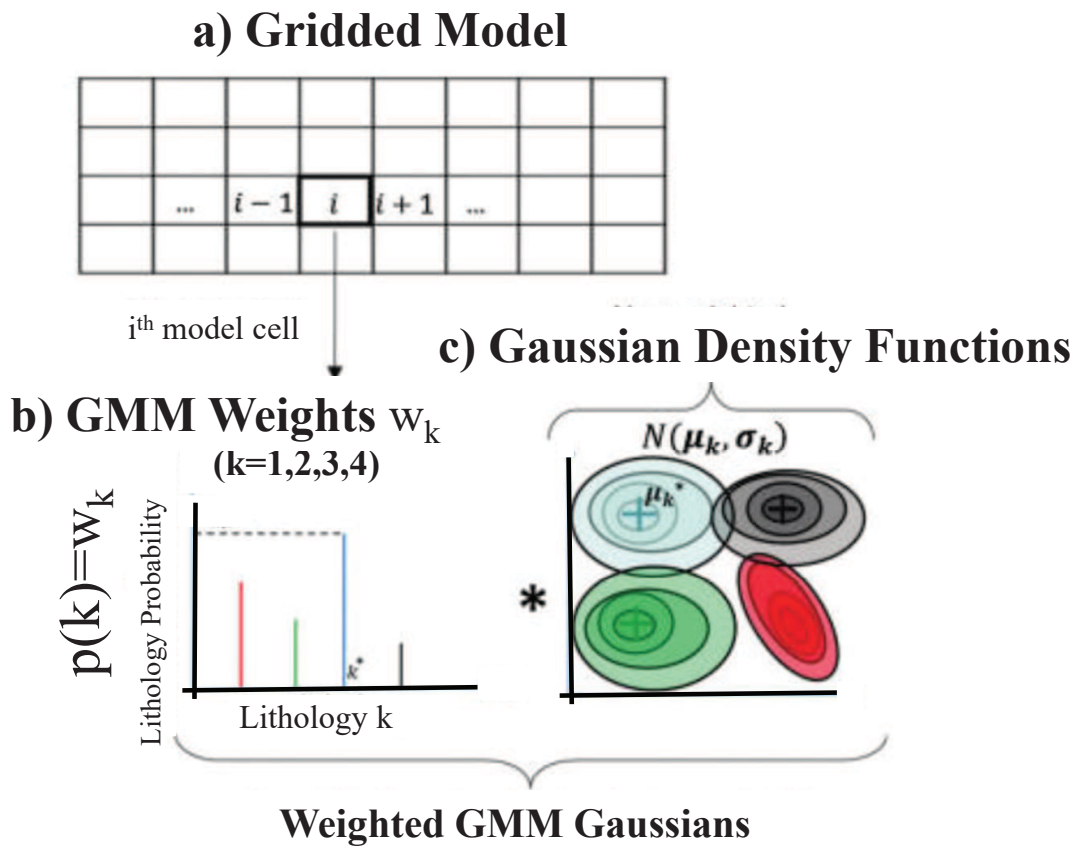


Figure 21.16: a) Gridded model, b)  $p(k)$  for  $k = 1, 2, 3, 4$  and c) Gaussians for equation 21.31. Figure adapted from Giraud et al. (2017).

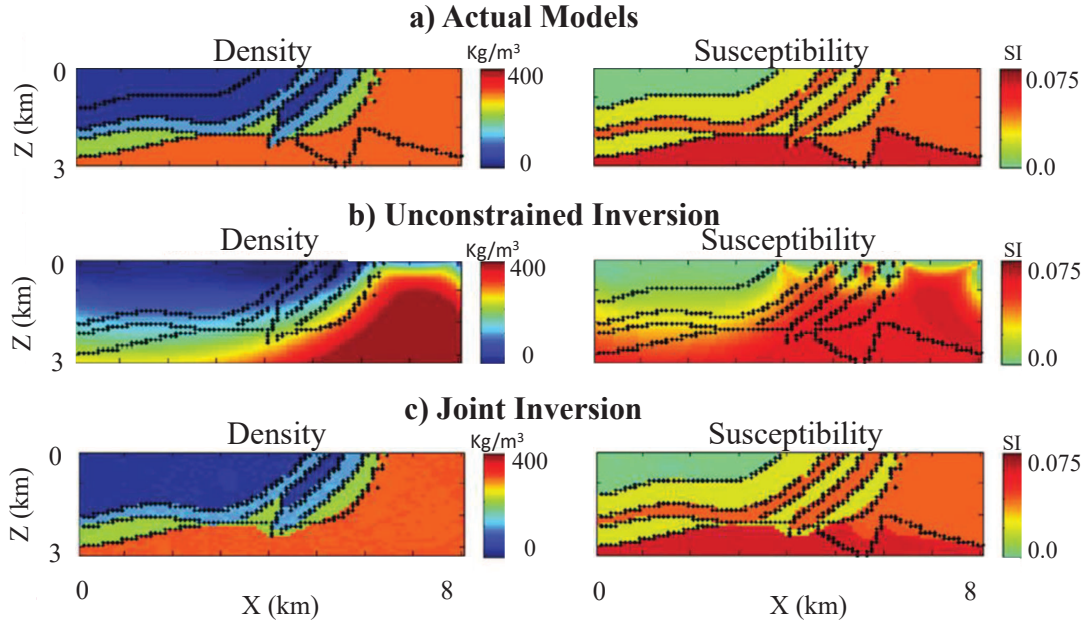


Figure 21.17: a) Actual model, b) unconstrained inversion model and c) constrained inversion model. Figure adapted from Giraud et al. (2017).

In this case, the gravity data largely provided long-wavelength information about the subsurface structure and the magnetic data mainly provided the short-wavelength variations.

### 21.5.3 Joint Inversion with Updated GMM Penalty Terms

Astic and Oldenburg (2018) used a GMM to replace the Gaussian prior in Tikhonov regularized inversion of gravity and magnetic data. They defined the objective function  $\epsilon$  of Tikhonov regularization as a combination of data misfit and model penalty terms:

$$\epsilon = \underbrace{\|\mathbf{d}(\mathbf{m}) - \mathbf{d}^{data}\|_2^2}_{\text{data misfit}} + \underbrace{\alpha_{small} \|\mathbf{m} - \mathbf{m}^{prior}\|_2^2}_{\text{prior model penalty}} + \underbrace{\beta_{smooth} \|D\mathbf{m}\|_2^2}_{\text{rough model penalty}}, \quad (21.46)$$

where  $\mathbf{d}(\mathbf{m})$  is the predicted data computed from the model  $\mathbf{m}$ ,  $\mathbf{d}^{data}$  represents the recorded data,  $D$  is a spatial differentiation-like operator,  $\mathbf{m}^{prior}$  is a prior estimate of the expected model, and  $(\alpha_{small}, \beta_{smooth})$  are the damping coefficients greater than zero that determine the degree of (*smallness*, *smoothness*) in the inverted model  $\mathbf{m}^*$  that minimizes  $\epsilon$ . If the model is spatially rough and has sharp gradients, then  $\|D\mathbf{m}\|_2^2$  will penalize the objective function with a large value. Thus, the iterative optimization procedure will iterate to a smoother model. Similarly, the term  $\|\mathbf{m} - \mathbf{m}^{prior}\|_2^2$  will encourage the iterative model to resemble the prior model  $\mathbf{m}^{prior}$ . The innovation here compared to the approach employed by Giraud et al. (2017) is that the mixture parameters are iteratively recomputed with the updated model.

The optimal model  $\mathbf{m}^*$  that minimizes  $\epsilon$  in equation 21.46 is equivalent to the one that maximizes the posterior distribution  $p(\mathbf{m}|\mathbf{d})$ , which can be expressed by Bayes' Theorem (see section 20.5) as

$$p(\mathbf{m}|\mathbf{d}) \propto p(\mathbf{d}|\mathbf{m})p(\mathbf{m})_{small} p(\mathbf{m})_{smooth}. \quad (21.47)$$

Here, the priors  $p(\mathbf{m})_{small}$  and  $p(\mathbf{m})_{smooth}$  are typically expressed as multivariate Gaussian distributions. For example,

$$p(\mathbf{m})_{small} \sim \mathcal{N}(\mathbf{m}|\mathbf{m}_{prior}, [\mathbf{W}_{small}^T \mathbf{W}_{small}]^{-1})^{\beta\alpha_{small}}, \quad (21.48)$$

where  $\mathbf{W}_{small}$  is the matrix that expresses confidence in the prior model  $\mathbf{m}_{prior}$  and  $[\mathbf{W}_{small}^T \mathbf{W}_{small}]^{-1}$  is akin to a model covariance matrix. Astic and Oldenburg (2018) replaced the unimodal Gaussian in equation 21.48, which can be obtained by GMM analysis of petrophysical data from outcrops, well logs, and lab measurements. The result is

$$p(\mathbf{m})_{small} \sim \mathcal{M}(\mathbf{m}|\Phi)^{\beta\alpha_{small}}, \quad (21.49)$$

where  $\mathcal{M}$  is the weighted sum of Gaussians from a GMM (see equation 21.28) and  $\Phi$  represents the petrophysical mixture weights, means and covariance information for each Gaussian associated with a different rock unit. The optimal model is found by applying the EM method that maximizes the logarithm of equation 21.47. After each iteration, the mixture parameters are recomputed from the geophysical model and the lithological characteristics are updated using the EM algorithm. This two-step process of geophysical inversion followed by a GMM update produces the expected petrophysical and geological characteristics.

To test the effectiveness of this approach, Astic and Oldenburg (2019) generated gravity and magnetic data for synthetic gravity and susceptibility models of a kimberlite deposit. There are 375,442 cells with unknown density and magnetic susceptibility values in the 3D model. The smallest square cells are 10 m along a side and the initial model is the background half-space. There is a depth weighting applied to the model parameters because the data are less sensitive to parameters in the deeper parts of the model. Each inversion required ten to twenty iterations before being halted.

The models inverted from the gravity and magnetic data are shown in Figure 21.18. Here, a) and b) are the inverted susceptibility and density models using Tikhonov inversion, where the data are inverted separate from one another. The second row is the same as the 1st row except the data are separately inverted from one another and the petrophysical model is employed as a constraint. The third row of results are obtained by joint inversion of both the gravity and magnetic data, as well as using the GMM as a constraint from the petrophysical data. The GMM prior is updated after each iteration. Individual petrophysically guided inversions are better at recovering clusters but only the joint inversion accurately recovered the yellow regions in the model.

## 21.6 Summary

The unsupervised clustering method of the GMM is presented, which assigns clusters to points that are best fitted by a specified number of Gaussians. It is similar to the K-means clustering method in that it consists of an iterative procedure that first finds the statistical parameters of the Gaussians (means, covariances, and mixture weights), and then determines the new cluster membership of the points. These new members are then used to update the statistical parameters, and this procedure is repeated until convergence. The benefit of this method over clustering methods such as K-means and Fuzzy C-means Clustering method (FCM) is that the final point distributions are governed by probability distributions, which can be used to assign uncertainty values in their membership class. Figure 21.11 shows the close relationship between the formulas for updating the parameters in the FCM and GMM.

The discriminative models compute the decision boundary between the classes, and so learns the conditional probability  $p(Y|X)$ . This learning is usually done by a supervised learning procedure. In contrast, the generative model learns the probability distribution  $p(X|Y)$  of each class. It can be used in an unsupervised manner such as the GMM, or a supervised manner as in supervised GDA. An advantage of supervised GDA over supervised NN is that it does not require as much training

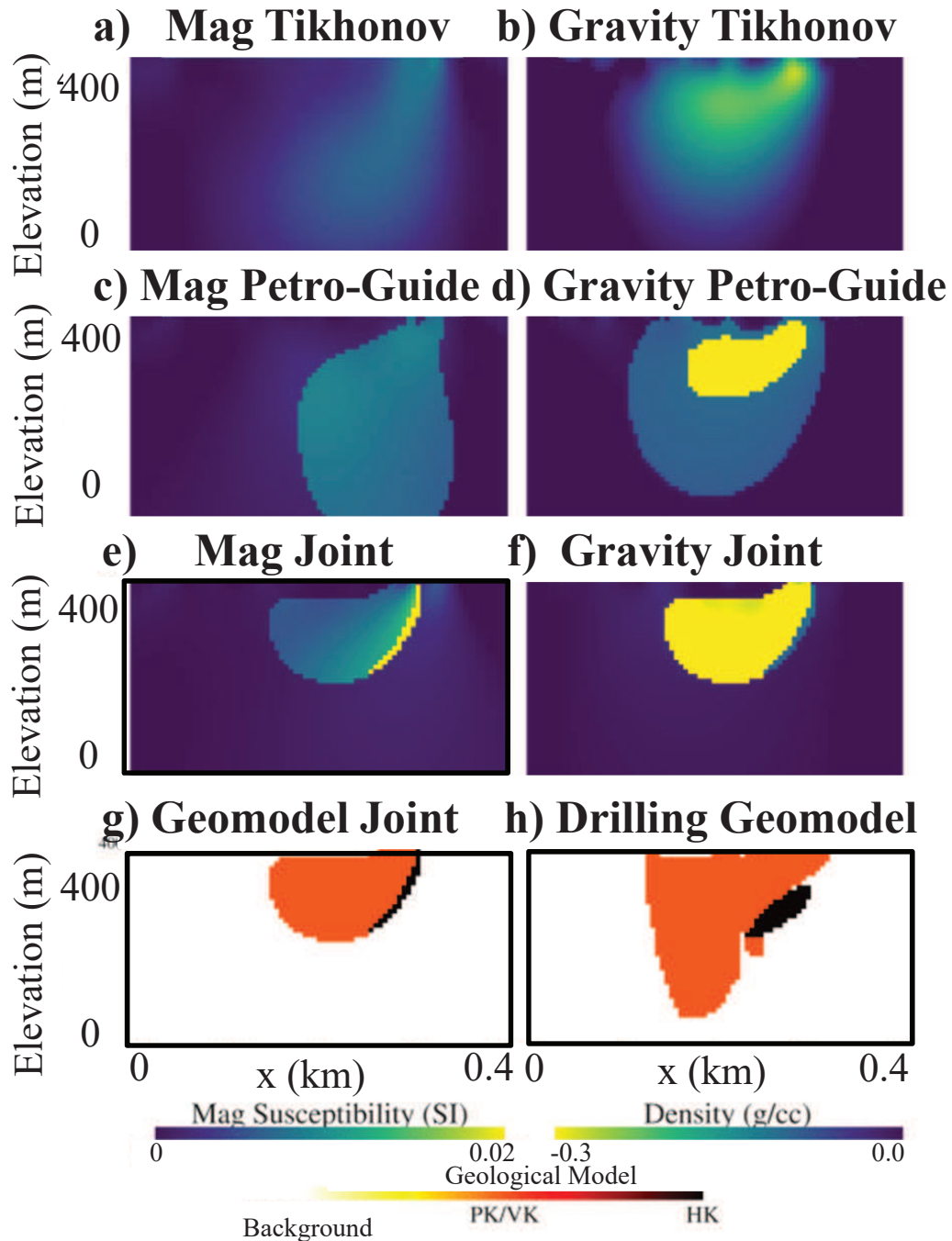


Figure 21.18: First column consists of magnetic susceptibility models and the second column are the density models recovered by inverting the gravity data. Figure adapted from Astic and Oldenburg (2019).

data.

## 21.7 Exercises

1. Carry out Exercises 4.7.10-4.7.11.
2. Assume that the data residual matrix  $\epsilon = \hat{\mathbf{X}}\hat{\mathbf{W}} - \hat{\mathbf{Y}}$  in equation 21.8 is  $2 \times 2$ , with the first column vector given by  $(\epsilon_1, \epsilon_2)^T$  and the second is  $(\tilde{\epsilon}_1, \tilde{\epsilon}_2)^T$ . Show that  $Tr[\epsilon^T \epsilon]$  is the sum of the squared errors  $\epsilon_1^2 + \epsilon_2^2 + \tilde{\epsilon}_1^2 + \tilde{\epsilon}_2^2$ .
3. Derive equations similar to equations 21.25-21.27 for fitting three Gaussians.
4. Derive equations 21.39-21.41. See section 21.9.4.
5. Show that the optimal mean  $\mu$  and variance  $\sigma^2$  formulas for maximizing the Gaussian  $\frac{1}{\sigma} \exp(-\frac{1}{2N} \sum_{n=1}^N (x^{(n)} - \mu)^2)$  for a group of points  $x^{(n)}$   $n \in \{1, 2, \dots, N\}$  are

$$\mu = \frac{1}{N} \sum_{n=1}^N x^{(n)}; \quad \sigma^2 = \frac{1}{N} \sum_{n=1}^N (x^{(n)} - \mu)^2. \quad (21.50)$$

6. Integrate equation 21.28 over  $\mathbf{x}$  and show that the summation of mixture weights  $w_k$  is equal to 1. Note that both  $p(\mathbf{x})$  and the Gaussian are normalized.
7. Prove the identity  $\nabla_{\mathbf{X}} \mathbf{a}^T \mathbf{X}^{-1} \mathbf{b} = -\mathbf{X}^{-T} \mathbf{a} \mathbf{b}^T \mathbf{X}^{-T}$ .
8. Create a triangle like PDF and create a MATLAB code that samples this PDF by the inverse-transform method. As your starting point, use the MATLAB code listed below.

```
clear all;hold off
x=[0:.1:3];u=x.*x/9;
subplot(131);
plot(x,2/9*x);xlabel('x');
ylabel('f_X(x)');
title({'a} PDF: f_X(x)= 2/9*x', 'for 0\leq x \leq 3'})

for ix=2:3
N=15;
if ix==2;ur=rand(N,1);end
if ix==2;subplot(132);end
if ix==3;subplot(133);end
hold on;
plot(ur*0,ur,'r*')
plot(x,u,'b-');
xlabel('x');
if ix==2;ylabel('F_X(x\geq X)');
title({'b} CDF: F_X(x)', '\int_0^x f_X(x_0)dx_0 = x^2/9'})
text(.2,.85,{'\color{red}red\color{black} pts sampled from', 'uniform distribution'}, 'fontsize',7)
end

if ix==3;
ylabel('F_X(\color{green}x\color{black}\geq X)=\color{red}u\color{black}');

title({'c} x = F_x^{-1}(u)')
xlabel('x', 'color', 'green')
end

if ix==3;subplot(133);
plot(3*sqrt(ur),ur*0,'g*');end
if ix==3;
for i=1:N
```

```

        plot([3*sqrt(ur(i)),3*sqrt(ur(i))],[0,ur(i)], '--g')
        plot([0,3*sqrt(ur(i))],[ur(i),ur(i)], '--r')
    end
end
hold off
end

subplot(131)
hold on
i=3;
x=3*sqrt(ur(i));y=2*x/9;
plot([3*sqrt(ur(i)),3*sqrt(ur(i))],[0,y], '--g')
plot(3*sqrt(ur(i)),0, '*g')
x=3*sqrt(ur(i));y=2*x/9;
plot([0,x],[y,y], '--r')
plot(0,y, 'r*')
plot(x,y, 'k0')
plot(3*sqrt(ur),ur*0, 'g*')
    hold off

```

9. Extend the acceptance-rejection code below to 2D.

```

clear all;subplot(111);plot(1);N=300;dx=1/N;sigma=.1;
ii=zeros(N,1);f=ii;
for i=1:N
    ii(i)=i*dx;
    f(i)=1/(sqrt(2)*sigma)*exp(-0.5*(i*dx-0.5)^2/sigma^2);
end
f=f/max(f(:));
subplot(221);plot(ii,f);
% Sampling
b=1;a=0;N1=N;
r = a + (b-a).*rand(N1,1);%Random Sampling along x axis
r1 = a + (b-a).*rand(N1,1);%Probability of acceptance
r1=r1*max(f);
subplot(222);
hold on
plot(r,r1,'*g',ii,f,'b-')
plot([0 1 1 0 0],[0 0 1 1 0], 'r-')
xlabel('x');ylabel('z')
title('b) Random Points and f_X(x)')
ia=0;N1=N;xa=zeros(N1,1);ff=xa;
for i=1:N1
    ir1=round(r(i)*N);if ir1<1;ir1=1;end
    if f(ir1)>=r1(i);ia=ia+1;xa(ia)=r(i);izz(ia)=r1(i);
        ff(ia)=f(i);end
end
hold off
subplot(223)
hold on;
plot([0 1 1 0 0],[0 0 1 1 0], 'r-')
plot(ii,f,'b-');
xlabel('x');ylabel('z')
text(.03,.9,'\color{black}Accepted pts = \color{green}', 'fontsize',7)
text(0.37,.87,'\color{green}*', 'fontsize',15)
title('c) Accepted Points (x_{acc},z_{acc})')
for i=1:ia;
    plot(xa(i),izz(i), '*g')
end
hold off

subplot(221);
hold on;
plot([0 1 1 0 0],[0 0 1 1 0], 'r-')
xlabel('x');ylabel('z')

```

```

title('a) PDF: f_X(x) vs x')
for i=1:ia;
    plot(xa(i),0,'*r')
end
text(.046,.08,['\color{black}Accepted x pts = '],'fontsize',8)
text(0.465,.05,'\color{red}*','fontsize',15)
hold off

subplot(224)
hold on
for i=1:ia;
    plot(xa(i),0,'*r')
end
histogram(xa(1:ia),10)
mx=max(histcounts(xa(1:ia),10));
plot(0,mx*1.2,'.')
plot([0 1 1 0 0],[0 0 mx*1.2 mx*1.2 0],'r-')
text(.05,mx*1.1,['\color{black}Accepted x pts = \color{red}*'],'fontsize',9)
text(0.536,mx*1.07,'\color{red}*','fontsize',15)
plot(ii,f/max(f)*mx,'b-');xlabel('x');ylabel('Count');title('d) Histogram vs x')
hold off

```

## 21.8 Computational Labs

1. Download the Yellowstone data from *LAB1/Chapter.YellowstoneData/YellowstoneData.docx*. Compute the GMM contours for these data using the lab at <https://colab.research.google.com/drive/1KScCs6jwUt5WQkz0lvqfDltOjHaZBYQ2?usp=sharing>.

Select the number of clusters to be two, and use the GMM responsibility function  $p_{nk}$  to decide which cluster a point belongs to. That is, for any point assign it to the cluster associated with the highest probability. This transforms the GMM into a hard clustering method. How does it compare to the K-means results shown in Figure 17.1.

2. Write the MATLAB or Python code for the expectation-maximization algorithm in Box 21.4.2. Apply it to the Yellowstone data in *LAB1/Chapter.YellowstoneData/YellowstoneData.docx* for two clusters. Compare your results to the K-means results shown in Figure 17.1.

Select the number of clusters to be two, and use the GMM responsibility function  $p_{nk}$  to decide which cluster a point belongs to. That is, for any point assign it to the cluster associated with the highest probability. This transforms the GMM into a hard clustering method. How does it compare to the K-means results shown in Figure 17.1.

## 21.9 Appendix: Gradients of the GDA Log-likelihood Function

If the points  $\mathbf{x}^{(n)}$  belong to just two Gaussians then there are two means ( $\boldsymbol{\mu}_0, \boldsymbol{\mu}_1$ ) and two covariance matrices ( $\boldsymbol{\Sigma}_0, \boldsymbol{\Sigma}_1$ ), two possible values for the class label,  $y \in \{0, 1\}$ , and  $p(y^{(n)}) = \phi^{y^{(n)}}(1 - \phi)^{1-y^{(n)}}$  is the probability of the  $n^{(th)}$  point belonging to either Gaussian. We now derive the analytic formulas for the means, covariance matrices and  $\phi$  for the case of fitting two Gaussians to the binary training data  $(\mathbf{x}^{(n)}, y^{(n)})$ , where we assume  $\boldsymbol{\Sigma}_0 = \boldsymbol{\Sigma}_1 = \boldsymbol{\Sigma}$ . If only one Gaussian is fitted to all of the training data then it will be shown that  $p(y) = \phi = 1$  and the mean and covariance formulas are easily derived.



The log-likelihood function for GDA in equation 21.24 is defined as

$$\begin{aligned}
\mathcal{L} &= \ln \Pi_{n=1}^N p(\mathbf{x}^{(n)} | y^{(n)}; \underbrace{\phi, \boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \boldsymbol{\Sigma}}_{\Phi}) p(y^{(n)}), \\
&= \ln \Pi_{n=1}^N \left[ \frac{1}{(2\pi)^{D/2}} |\boldsymbol{\Sigma}|^{-1/2} \{ \delta_{y^{(n)}0} (1 - \phi) e^{-0.5(\mathbf{x}^{(n)} - \boldsymbol{\mu}_0)^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}^{(n)} - \boldsymbol{\mu}_0)} + \delta_{y^{(n)}1} \phi e^{-0.5(\mathbf{x}^{(n)} - \boldsymbol{\mu}_1)^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}^{(n)} - \boldsymbol{\mu}_1)} \}, \right. \\
&= \sum_{n=1}^N \left[ -0.5D \ln 2\pi - 0.5 \ln |\boldsymbol{\Sigma}| - \frac{\delta_{y^{(n)}0}}{2} (\mathbf{x}^{(n)} - \boldsymbol{\mu}_0)^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}^{(n)} - \boldsymbol{\mu}_0) - \frac{\delta_{y^{(n)}1}}{2} (\mathbf{x}^{(n)} - \boldsymbol{\mu}_1)^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}^{(n)} - \boldsymbol{\mu}_1) \right. \\
&\quad \left. \left. + y^{(n)} \ln \phi + (1 - y^{(n)}) \ln(1 - \phi) \right] \right],
\end{aligned} \tag{21.51}$$

where  $\Phi$  represents the unknown statistical parameters and, for convenience,  $\boldsymbol{\Sigma}_0 = \boldsymbol{\Sigma}_1 = \boldsymbol{\Sigma}$ . Note, the Kronecker delta function indicates which conditional probability function is activated. For example, if the  $n^{th}$  sample is classified as  $y^{(n)} = 1$  then the term with  $\delta_{y^{(n)}1}$  is activated and the term with  $\delta_{y^{(n)}0} = 0$  is eliminated.

### 21.9.1 Solving for $\phi$

Setting  $\partial \mathcal{L} / \partial \phi = 0$  gives

$$\frac{\partial \mathcal{L}}{\partial \phi} = \sum_{n=1}^N \left[ \frac{y^{(n)}}{\phi} - \frac{1 - y^{(n)}}{1 - \phi} \right] = 0, \tag{21.52}$$

which can be rearranged to give

$$\begin{aligned}
\sum_{n=1}^N [y^{(n)} - y^{(n)}\phi - \phi + y^{(n)}\phi] &= \sum_{n=1}^N [y^{(n)} - \phi], \\
&= \sum_{n=1}^N y^{(n)} - N\phi = 0.
\end{aligned} \tag{21.53}$$

Solving for  $\phi$  gives

$$\begin{aligned}
\phi &= \frac{1}{N} \sum_{n=1}^N y^{(n)}, \\
&= \frac{1}{N} \sum_{n=1}^N \mathcal{I}\{y^{(n)} = 1\},
\end{aligned} \tag{21.54}$$

where  $\mathcal{I}\{y^{(n)} = 1\}$  is the indicator function in equation 21.25. This expression gives the probability  $p(y = 1) = \phi = N_1/N$  for  $y = 1$ , where  $N_1$  is the number of points that belong to the Gaussian labeled as  $y = 1$ . If the points all belong to a single Gaussian where  $y^{(n)} = 1 \forall n \in \{1, 2, \dots, N\}$ , then  $\phi = 1$  in equation 21.54.

### 21.9.2 Solving for $\mu$

Setting  $\nabla_{\mu_0} \mathcal{L} = 0$  gives

$$\begin{aligned} \nabla \mathcal{L}_{\mu_0} &= -\frac{1}{2} \sum_{n=1}^N \nabla_{\mu_0} [\delta_{y^{(n)}0} (\mathbf{x}^{(n)} - \mu_0)^T \Sigma^{-1} (\mathbf{x}^{(n)} - \mu_0) + \delta_{y^{(n)}1} (\mathbf{x}^{(n)} - \mu_1)^T \Sigma^{-1} (\mathbf{x}^{(n)} - \mu_1)], \\ &= -\frac{1}{2} \sum_{n=1}^N \nabla_{\mu_0} [\delta_{y^{(n)}0} \overbrace{(\mathbf{x}^{(n)} - \mu_0)^T \Sigma^{-1} (\mathbf{x}^{(n)} - \mu_0)}^{\mathbf{y}^T \mathbf{H} \mathbf{y}}], \end{aligned} \quad (21.55)$$

where the term with  $\delta_{y^{(n)}1}$  does not contain  $\mu_0$ , so its derivatives with respect to components of  $\mu_0$  are zero. The term under the overbrace is composed of the  $D \times D$  symmetric matrix  $\mathbf{H} = \Sigma^{-1}$  that is left- and right-multiplied by the  $D \times 1$  vectors  $\mathbf{y} = (\mathbf{x}^{(n)} - \mu_0)$  and  $\mathbf{y} = (\mathbf{x}^{(n)} - \mu_0)$ , respectively. From the least squares section in Chapter 2, we know that

$$\frac{1}{2} \nabla_{\mathbf{y}} \mathbf{y}^T \mathbf{H} \mathbf{y} = \mathbf{H} \mathbf{y}, \quad (21.56)$$

so that equation 21.55 becomes

$$\begin{aligned} \nabla \mathcal{L}_{\mu_0} &= -\sum_{n=1}^N \delta_{y^{(n)}0} \mathbf{H} (\mathbf{x}^{(n)} - \mu_0), \\ &= -\sum_{n=1}^N \delta_{y^{(n)}0} [\Sigma^{-1} \mathbf{x}^{(n)} - \Sigma^{-1} \mu_0], \\ &= \Sigma^{-1} \sum_{n=1}^N \delta_{y^{(n)}0} \mathbf{x}^{(n)} - \Sigma^{-1} \mu_0 \sum_{n=1}^N \delta_{y^{(n)}0}, \\ &= 0. \end{aligned} \quad (21.57)$$

Left-multiplying both sides of this equation by  $\Sigma$  and solving for  $\mu_0$  yields

$$\mu_0 = \frac{\sum_{n=1}^N \delta_{y^{(n)}0} \mathbf{x}^{(n)}}{\sum_{n=1}^N \delta_{y^{(n)}0}}, \quad (21.58)$$

where  $\delta_{y^{(n)}0}$  is the indicator function  $\mathcal{I}\{y^{(n)} = 0\}$ . Similarly, the derivation for the gradient with respect to  $\mu_1$  yields

$$\mu_1 = \frac{\sum_{n=1}^N \delta_{y^{(n)}1} \mathbf{x}^{(n)}}{\sum_{n=1}^N \delta_{y^{(n)}1}}. \quad (21.59)$$

These two equations are consistent with equation 21.26. If the points all belong to a single Gaussian where  $y^{(n)} = 1 \forall n \in \{1, 2, \dots, N\}$ , then equation 21.54 reduces to

$$\begin{aligned} \mu_1 &= \frac{\sum_{n=1}^N \delta_{y^{(n)}=1} \mathbf{x}^{(n)}}{\sum_{n=1}^N \delta_{y^{(n)}=1}}, \\ &= \frac{\sum_{n=1}^N \mathbf{x}^{(n)}}{N}, \end{aligned} \quad (21.60)$$

and  $\boldsymbol{\mu}_0 = 0$ .

### 21.9.3 Solving for $\boldsymbol{\Sigma}$

Setting  $\nabla_{\boldsymbol{\Sigma}} \mathcal{L} = 0$  gives

$$\begin{aligned} \nabla_{\boldsymbol{\Sigma}} \mathcal{L} &= \sum_{n=1}^N -0.5 \nabla_{\boldsymbol{\Sigma}} \ln |\boldsymbol{\Sigma}| - 0.5 \nabla_{\boldsymbol{\Sigma}} [\delta_{y^{(n)}0} (\mathbf{x}^{(n)} - \boldsymbol{\mu}_0)^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}^{(n)} - \boldsymbol{\mu}_0) \\ &\quad + \delta_{y^{(n)}1} (\mathbf{x}^{(n)} - \boldsymbol{\mu}_1)^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}^{(n)} - \boldsymbol{\mu}_1)], \end{aligned} \quad (21.61)$$

$$= \sum_{n=1}^N \left\{ -\frac{1}{2} \boldsymbol{\Sigma}^{-T} - \frac{1}{2} \sum_{k=0}^1 \delta_{y^{(n)}k} [-\boldsymbol{\Sigma}^{-T} (\mathbf{x}^{(n)} - \boldsymbol{\mu}_k) (\mathbf{x}^{(n)} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-T}] \right\}, \quad (21.62)$$

$$= -\frac{1}{2} \sum_{n=1}^N \left\{ \boldsymbol{\Sigma}^{-T} - \sum_{k=0}^1 \delta_{y^{(n)}k} [\boldsymbol{\Sigma}^{-T} (\mathbf{x}^{(n)} - \boldsymbol{\mu}_k) (\mathbf{x}^{(n)} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-T}] \right\} = 0, \quad (21.63)$$

where the identity  $\nabla_{\mathbf{X}} \mathbf{a}^T \mathbf{X}^{-1} \mathbf{b} = -\mathbf{X}^{-T} \mathbf{a} \mathbf{b}^T \mathbf{X}^{-T}$  is used in transforming equation 21.61 to equation 21.62. The term  $\boldsymbol{\Sigma}^{-T}$  is independent of the summation index  $n$ , so the above equation can be right multiplied by  $\boldsymbol{\Sigma}$  to get

$$\nabla_{\boldsymbol{\Sigma}} \mathcal{L} = \sum_{n=1}^N \left\{ 1 - \sum_{k=0}^1 \delta_{y^{(n)}k} [\boldsymbol{\Sigma}^{-T} (\mathbf{x}^{(n)} - \boldsymbol{\mu}_k) (\mathbf{x}^{(n)} - \boldsymbol{\mu}_k)^T] \right\} = 0. \quad (21.64)$$

We know that  $\sum_{n=1}^N 1 = N$  so we have

$$\nabla_{\boldsymbol{\Sigma}} \mathcal{L} = N - \sum_{n=1}^N \sum_{k=0}^1 \delta_{y^{(n)}k} [\boldsymbol{\Sigma}^{-T} (\mathbf{x}^{(n)} - \boldsymbol{\mu}_k) (\mathbf{x}^{(n)} - \boldsymbol{\mu}_k)^T] = 0. \quad (21.65)$$

Multiplying the above equation by  $\boldsymbol{\Sigma}$ , dividing by  $N$  and solving for  $\boldsymbol{\Sigma}$  gives

$$\boldsymbol{\Sigma} = \frac{1}{N} \sum_{n=1}^N \sum_{k=0}^1 \delta_{y^{(n)}k} (\mathbf{x}^{(n)} - \boldsymbol{\mu}_k) (\mathbf{x}^{(n)} - \boldsymbol{\mu}_k)^T. \quad (21.66)$$

If the  $\boldsymbol{\Sigma}$  is taken to be different for each Gaussian, then the covariance gradient is with respect to the  $k^{th}$  covariance matrix for  $k \in 0, 1$ . That is,  $\nabla_{\boldsymbol{\Sigma}}$  is replaced by  $\nabla_{\boldsymbol{\Sigma}_k}$  in equation 21.61. Therefore, only the terms that are a function of  $\nabla_{\boldsymbol{\Sigma}_k}$  survive in equation 21.63, the summation  $\sum_{k=0}^1$  is replaced by the value 1 and  $k = 1$ . In this case, equation 21.66 becomes

$$\boldsymbol{\Sigma} = \frac{1}{N} \sum_{n=1}^N \delta_{y^{(n)}1} (\mathbf{x}^{(n)} - \boldsymbol{\mu}_1) (\mathbf{x}^{(n)} - \boldsymbol{\mu}_1)^T. \quad (21.67)$$

If the points all belong to a single Gaussian where  $y^{(n)} = 1 \forall n \in \{1, 2, \dots, N\}$  and  $k = 1$ , then  $\delta_{y^{(n)}k} = \delta_{y^{(n)}=11} = 1$  becomes unity in the above equation.

### 21.9.4 Gradient of the GMM Log-likelihood Function

Setting the gradients of the GMM log-likelihood in equation 21.33 to zero and solving for the parameters gives expressions similar to the GDA formulas, except now the gradient w/r to the

mixture coefficients  $w_k \forall k$  needs to be estimated instead of the gradient w/r to  $\phi$  in equation 21.52. The constraint  $\sum_{k=1}^K w_k = 1$  must be enforced, which can be done by using a Lagrange multiplier and maximizing the adjusted log-likelihood function

$$\hat{\mathcal{L}} = \mathcal{L} + \lambda \left( \sum_{k=1}^K w_k - 1 \right). \quad (21.68)$$

Taking the gradient of  $\hat{\mathcal{L}}$  w/r to  $w_k$  gives the formula (Bishop, 2006)

$$0 = \sum_{n=1}^N \frac{\mathcal{N}(\mathbf{x}^{(n)} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^K w_j \mathcal{N}(\mathbf{x}^{(n)} | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} + \lambda. \quad (21.69)$$

Multiplying both sides by  $w_k$ , summing over  $k$  and using the constraint  $\sum_{k=1}^K w_k = 1$  we find  $\lambda = -N$ . This expression for  $\lambda = -N$  can then be used to find the final expression for

$$w_k = \frac{N_k}{N}, \quad (21.70)$$

where  $N_k = \sum_k p_{nk}$  in equation 21.41 is the total number of points effectively shared by the  $k^{th}$  Gaussian.

## Part VI

# Seismic Inversion and Machine Learning



## Chapter 22

# Physics-Informed Machine Learning Inversion of Seismic Data

*The second-best way to solve a problem is sometimes by machine learning. The best way is to understand the physics and use a physics-based inversion strategy.*

This chapter overviews the general procedures for deterministic and stochastic inverse algorithms that find the optimal geophysical model  $\mathbf{m}$  that best explains the data  $\mathbf{d}$ . The inverse solutions by machine learning and a physics-based algorithm can be combined to give a hybrid ML+physics algorithm denoted as physics-informed machine learning. The benefit of combining both strategies in parallel is that the ML algorithm is trained to recognize a familiar data pattern, and so encourages a physics-based gradient algorithm to search in the neighboring model space. The hope is that this search avoids many local minima that surround the global minima. Another hybrid strategy is sequential where ML is first used to reduce the dimension of the data, and then a physics-based inversion is used to estimate the optimal model. This chapter is followed by three more that present examples of physics-informed ML inversion of seismic data.

### 22.1 Deterministic and Stochastic Inversion

Chapter 20 presented several examples of geophysical inversion, including the deterministic inversion of the velocity model from VSP data  $\mathbf{d}^{obs}$  and locating the point source model  $\mathbf{m}$  from passive seismic data. This is denoted as a deterministic search because the optimal velocity model used a physics-based ray theory to compute the gradients in a steepest-descent algorithm. The optimal model  $\mathbf{m}^*$  minimizes the regularized sum of the squared residuals, such as

$$\epsilon = \frac{1}{2} \overbrace{\|\mathbf{W}(\mathbf{L}\mathbf{m} - \mathbf{d}^{obs})\|^2}^{\text{sum of squared residuals}} + \frac{\lambda}{2} \|\mathbf{W}_m \mathbf{m}\|^2, \quad (22.1)$$

to get the regularized least squares formula for  $\mathbf{m}^*$ :

$$\mathbf{m}^* = [\mathbf{L}^T \mathbf{W}^T \mathbf{W} \mathbf{L} + \lambda \mathbf{W}_m^T \mathbf{W}_m]^{-1} \mathbf{L}^T \mathbf{W}^T \mathbf{W} \mathbf{d}^{obs}, \quad (22.2)$$

where  $\mathbf{W}$  ( $\mathbf{W}_m$ ) is the data weighting matrix that downweights unreliable parts of the data (model). For non-linear problems such as FWI, the forward modeling operator  $\mathbf{L}$  is an implicit function of  $\mathbf{m}$ . In contrast, migrating seismic data is a linear problem because the background velocity is fixed and  $\mathbf{L}$  is independent of the reflectivity model  $\mathbf{m}$ .

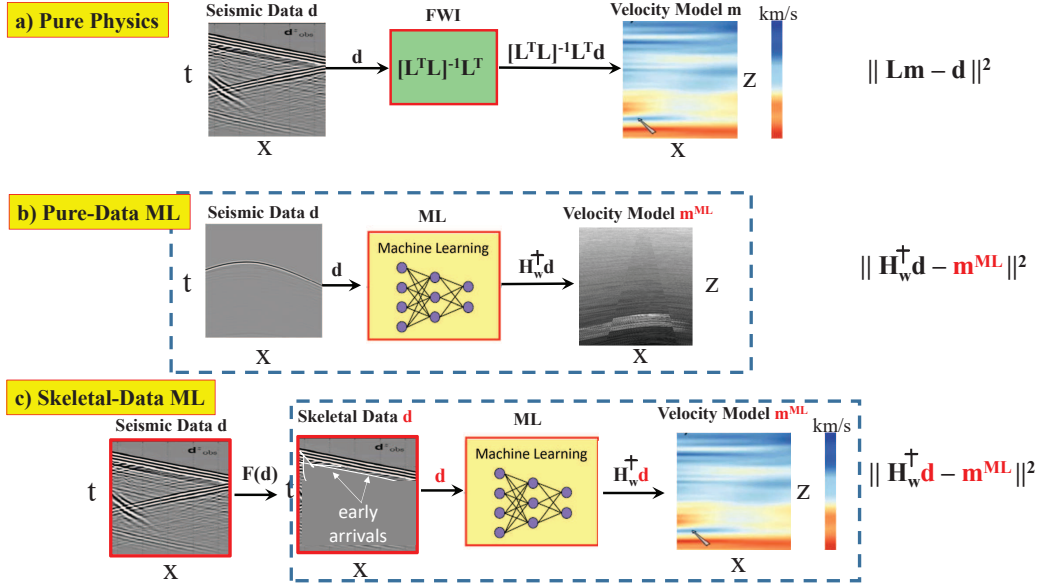


Figure 22.1: Workflows for a) Physics-only inversion, b) pure-data ML (Yang, 2021), and c) skeletal-pure-data ML. The dashed rectangles indicate that the enclosed system trains on many examples to get the optimal weights  $\mathbf{w}$  that minimize  $\|\mathbf{H}_{\mathbf{w}}^\dagger \mathbf{d} - \mathbf{m}^{ML}\|^2$ .

The computational workflow for the general FWI inversion algorithm or least squares migration algorithm is illustrated in Figure 22.1a where, for pedagogical simplicity, we set  $\mathbf{W} = \mathbf{I}$  and  $\mathbf{W}_m = \mathbf{0}$ . These deterministic algorithms are completely physics based because they compute the gradient descent updates using solutions to the wave equation.

The prediction of a model by a trained stochastic neural network is completely devoid of physics. It is an inversion in the sense that the weights  $\mathbf{w}$  are obtained by training to predict the given output  $\mathbf{y}^{train} = \mathbf{H}_{\mathbf{w}}^\dagger \mathbf{d}^{train}$  from the input data  $\mathbf{d}^{train}$ . In this case, the  $L^2$  objective function for the neural network is

$$\epsilon^{ML} = \|\mathbf{H}_{\mathbf{w}}^\dagger \mathbf{d}^{train} - \mathbf{y}^{train}\|^2, \quad (22.3)$$

where  $(\mathbf{d}^{train}, \mathbf{y}^{train}) = (input, output)$  represents the training pair for the neural network and  $\mathbf{H}_{\mathbf{w}}^\dagger$  represents the non-linear forward modeling operator of the NN, which is a function of the layer's weights denoted by the vector  $\mathbf{w}$ . The goal is to find the optimal weights  $\mathbf{w}^*$  of the neural network s.t.

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} \|\mathbf{H}_{\mathbf{w}}^\dagger \mathbf{d}^{train} - \mathbf{y}^{train}\|^2, \quad (22.4)$$

which is typically achieved by a gradient-descent algorithm. For example, the input data  $\mathbf{d}^{train}$  might be land resistivity measurements along an array of electrodes for many different resistivity models  $\mathbf{y}^{train}$ , which is the output. The elements of the resistivity vector  $\mathbf{y}^{train}$  are the resistivity values in a 2D or 3D grid in the subsurface.

Once the optimal weights  $\mathbf{w}^*$  are computed for a sufficiently comprehensive set of training pairs, then the resistivity model  $\mathbf{y}$  predicted from the recorded field data  $\mathbf{d}^{field}$  is given by

$$\mathbf{y} = \mathbf{H}_{\mathbf{w}^*}^\dagger \mathbf{d}^{field}. \quad (22.5)$$



To be consistent with the notation in equation 22.2, the symbol for, e.g. the resistivity model,  $\mathbf{y}$  in equation 22.3 is replaced by the model parameter  $\mathbf{m}^{ML}$  to give the pure-data ML objective function for training:

$$\epsilon^{ML} = \|\mathbf{H}_w^\dagger \mathbf{d} - \mathbf{m}^{ML}\|^2, \quad (22.6)$$

whose computational workflow is depicted in Figure 22.1b. If the input data are skeletalized and then trained by a NN algorithm, then its workflow is depicted in Figure 22.1c.

Some successful examples that only use raw data pairs for training include recognizing different bird types in section 10.13 or rock cracks images in Chapter 11, filtering migration images in Chapter 5, or estimating the optimal stacking velocity from the stacks of common midpoint gathers in Chapter 17. A seismic example of a pure-data ML inversion is illustrated in Figure 22.1 (Yang, 2021), who synthetically trained a neural network to predict the locations of point sources from passive seismic data computed along a string of geophones. The point sources emulated localized rock failures associated with a synthetic hydrofrac experiment.

Using a data-pure neural network for solving geophysical problems does not always meet with success. For example, Xi et al. (2020) trained a neural network purely on synthetic data to predict faults in seismic images, but the NN failed when applied to field data as illustrated in Figure 10.38c. Other examples include the failure of DL-predicted resistivity models for simulating EM wavefields in logging while drilling or characterization of transient electromagnetics (Jin et al., 2019; Shariari et al., 2020; Colombo et al., 2021).

## 22.2 Incomplete Training and Physics-Informed ML

The failure of a pure-data ML algorithm is typically caused by incomplete training of the network. Incomplete training, where the training set is limited in its diversity and number of examples, is inevitable for high-dimensional data spaces because increasing the dimension of the data space leads to a shrinkage of the distance between the nearest and farthest data points (Durrant and Kában, 2009). This shrinkage can be fatal for inversion methods that use a limited number of diverse training examples and employ a distance metric to measure the difference between the predicted and observed data. Thus, achieving a marginal improvement in generalizing a NN requires an exponential increase in the number of training examples. As Bishop (1995b) states: *"increasing the input space dimension without enhancing the quantity of available information reduces the model's power and may give rise to the curse of dimensionality."*

### 22.2.1 Data Skeletonization and ML

Can one reduce the dimension of the data space to improve the generalization of a neural network? Yes, the input data should be skeletonized into smaller dimensional data which still retains essential characteristics sensitive to model changes. This well-known principle is used in many areas of geophysics where, for example, only the traveltimes of selected events are inverted by the wave equation to get the velocity model (Luo and Schuster, 1991a and 1991b; Yilmaz, 2001). Other dimension reduction methods are PCA and SVD in Chapter 16. The cost, however, is often a loss of model resolution. Resolution can be improved by gradually including more types of events in iterative traveltime inversion or higher frequencies in the data for multiscale inversion (Bunk et al., 1995).

The workflow for skeletonizing data and then feeding it into a NN is depicted in Figure 22.1c. An example of skeletonized inversion is discussed in section 5.3 where complicated surface waves are simplified by a FK transform. In this case, the dominant Rayleigh modes are coherently localized along a prominent dispersion curve for the fundamental mode. The local windows that define these, hopefully, simple dispersion curves are obtained by a trained NN algorithm. Chapter 23 uses a

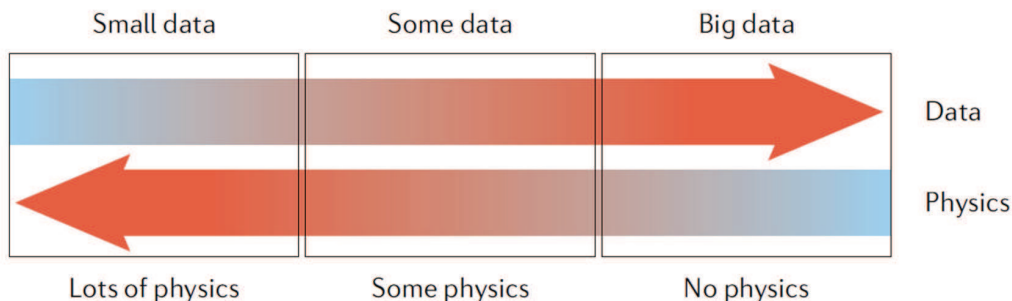


Figure 22.2: Diagram from illustrating physics-related constraints as a function of data size (Karniadakis et al., 2021).

physics-based traveltime tomography algorithm to invert synthetic traveltime data for the velocity tomogram. These blurry tomograms are used as input to a CNN algorithm to denoise the tomogram and get higher resolution images of point slowness anomalies. The CNN is previously trained on synthetic data to reduce the artifacts in the tomogram.

An example of skeletonizing the data by a physics-based algorithm is in Chapter 24. Here, the complexity of the seismic traces is reduced by a physics-based migration algorithm that focuses the reflection energy in the data domain to the reflectors  $\mathbf{m}^{mig}$  in the image domain. The role of the CNN algorithm is to denoise the input migration image and estimate the pseudo-reflectivity sections.

## 22.3 Physics-Informed Machine Learning

In the pure-data ML case, the weights for the ML algorithm are obtained by extensive training to minimize the ML objective function. This type of algorithm requires an enormous number<sup>1</sup> of training examples for high-dimensional data (Durrant and Kában, 2009). On the other hand, a deterministic algorithm avoids training, but suffers from the problem of getting stuck in local minima because the input data are so complex. The more wiggles in the trace, the more complexity and local minima in the objective function (Bunks et al., 1995; Schuster, 2017). Combining both a ML and a physics-based inversion strategy can provide the best of both worlds where, ideally, the constraints of the ML model encourage the gradient-descent algorithm to search near the global minimum.

Constraining the data to honor the physics during or after ML training is a class of deep learning algorithms known as physics-informed ML (PIML). It can seamlessly integrate data and abstract mathematical operators, including PDEs with or without missing physics (Karniadakis et al., 2021). The sparser the data set, the more the need for physics constraints as illustrated in Figure 22.2. A significant benefit of PIML is that it can provide accurate solutions in the presence of noisy or incomplete training data, including the extrapolation of training data to unseen models.

For example, if the input field data resemble some of the trained data, then the iterative gradient-descent algorithm might be steered to be near a trained model. This model is presumably near the desired model, so the gradient descent can descend to it without, ideally, getting stuck in a local minima.

<sup>1</sup>Some practitioners use 5-10 training examples per dimension as an empirical rule-of-thumb for training. If the input image contains  $10^6$  pixels, this means that  $O(10^7)$  training pairs are needed for training!

### 22.3.1 Sequential ML+Physics Inversion

To reduce both the dimension and complexity of the input to a wave-equation inversion algorithm, Chen and Schuster (2020) introduced a hybrid ML and physics-based inversion strategy. Their strategy is to train an autoencoder to condense important features of input seismic traces into a small-dimensional latent vector  $\mathbf{z}$ . The elements of this latent vector  $\mathbf{z}$  are then inverted for the velocity model by a gradient-descent method that uses solutions to the wave equation for both forward modeling and backprojection.

Figure 22.3a illustrates the workflow of this physics-informed ML algorithm. Chapters 14 and 25 provide the mathematical details and examples for this procedure applied to crosswell and refraction data.

### 22.3.2 Simultaneous ML+Physics Inversion

Instead of a sequential use of ML training followed by a physics-based inversion algorithm, Figure 22.3 illustrates the parallel minimization (Columbo et al., 2021a and 2021b) of the ML and physics-based objective functions:

$$\epsilon^{ML+Physics} = \underbrace{\alpha \|\mathbf{W}^{physics}(\mathbf{L}\mathbf{m} - \mathbf{d})\|^2}_{physics\ inversion} + \underbrace{\beta \|bfW^{ML}(\mathbf{H}^\dagger \mathbf{d}^{obs} - \mathbf{m}^{ML})\|^2}_{ML\ inversion} + \underbrace{\gamma \|\mathbf{W}^{joint}(\mathbf{m} - \mathbf{m}^{ML})\|^2}_{joint}. \quad (22.7)$$

Here, the weights ( $\mathbf{W}^{physics}$ ,  $\mathbf{W}^{ML}$ ,  $\mathbf{W}^{joint}$ ) are used to downweight unreliable parts of the data or model parameters, and the joint misfit term is used to encourage the ML and physics-driven models to be the same. The positive scalar variables ( $\alpha, \beta, \gamma$ ) moderate the relative importance of the three misfit functions.

A gradient-descent strategy with alternating updates of  $\mathbf{w}$  and  $\mathbf{m}$  is used to find the optimal model  $\mathbf{m}$ . First, the model  $\mathbf{m}$  is fixed and the weights  $\mathbf{w}$  in the NN are computed to minimize the objective function  $\epsilon^{ML+Physics}$ . Then the updated weights  $\mathbf{w}$  are fixed and the model  $\mathbf{m}$  is found by gradient descent that minimizes  $\epsilon^{ML+Physics}$ . The term  $\|\mathbf{W}^{joint}(\mathbf{m} - \mathbf{m}^{ML})\|^2$  encourages agreement between the deterministic  $\mathbf{m}$  and stochastic  $\mathbf{m}^{ML}$  models. This alternating use of the different objective functions is repeated until acceptable convergence. Examples of this procedure for geophysical inversion are presented in Columbo et al. (2021a, 2021b).

## 22.4 Automatic Differentiation

Finally, Chapter 26 presents the concept of automatic differentiation, which computes the exact values of the partial derivatives associated with a complicated function. These derivatives are used to update the weights of a neural network during training by a gradient-descent algorithm. A benefit of AD is that it can be used to compute the gradients associated with CNNs where the input consists of skeletonized data. This avoids the derivation of complicated gradient formulas for such problems. The penalty, however, is an increase in computational cost compared to using formulas to update the model parameters.

## 22.5 Summary

The concept of physics-informed ML inversion is presented where a physics-based inversion algorithm is combined with a ML algorithm. Once the ML algorithm is trained, then the ML prediction of a model from new field data is typically much less costly than that from a physics-based inversion algorithm. Unfortunately, the ML training is often incomplete so that the ML model

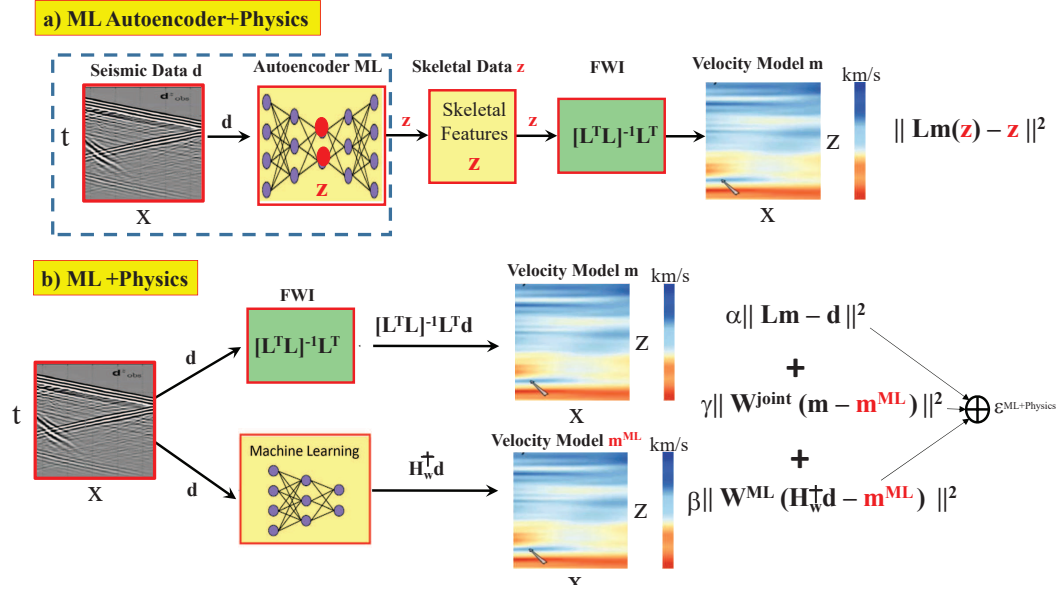


Figure 22.3: Workflow for a) sequential and b) simultaneous estimation of the optimal model  $\mathbf{m}$  that minimizes both the ML and physics-based objective functions.

$\mathbf{m}^{ML}$  obtained from the field data is insufficiently accurate. However, it is hoped that  $\mathbf{m}^{ML}$  is close enough to the global minima so that its neighborhood can be explored by the physics-based gradient algorithm to quickly find the global minimum.

Another hybrid strategy is sequential, where ML is first used to reduce the dimension of the data, and then a physics-based inversion is used to estimate the optimal model. Dimensional reduction of the data often reduces the chances of getting stuck in a local minimum, but at the possible cost of losing some resolution. Searching for improvements to these hybrid strategies is an active area of research.

## Chapter 23

# Tomographic Deconvolution

A tomographic deconvolution procedure is presented for high-resolution imaging of velocity anomalies between reflecting interfaces. The key idea is to first use standard tomography to invert reflection, refraction or transmission traveltimes for the background slowness tomogram. A convolutional neural network (CNN) model is then trained to estimate the inverse to the blurred tomogram  $s(\mathbf{x})^{tomo}$  to get the actual slowness model  $s(\mathbf{x})$ , where  $s(\mathbf{x})$  consists of small scatterers in the background velocity model. The synthetic training pairs consist of  $(s(\mathbf{x})^{tomo}, s(\mathbf{x}))$ , where  $s(\mathbf{x})$  represents the actual slowness model associated with the scatterers. We call this CNN a tomographic deconvolution (TD) operator because it deconvolves the blurring artifacts in traveltime slowness tomograms. This procedure is similar to that of migration deconvolution which deconvolves the migration butterfly artifacts in migration images. Results with a limited number of synthetic examples show the effectiveness of this procedure in significantly sharpening the tomographic images of small scatterers. These results suggest that the task of computing the TD operator is theoretically much simpler than that of general ML training of FWI operators. However, the effectiveness of TD with general slowness models and field data is still an open question for research.

### 23.1 Introduction

The seismic forward modeling operator  $\mathbf{L}$  computes the reflection traces  $\mathbf{d}$  from the slowness model  $\mathbf{m}$  by

$$\mathbf{L}\mathbf{m} = \mathbf{d}, \quad (23.1)$$

where  $\mathbf{m} = \mathbf{m}^{smooth} + \mathbf{m}^{scatt}$  is composed of the low-wavenumber smooth velocity model  $\mathbf{m}^{smooth}$  and the high-wavenumber velocity variations  $\mathbf{m}^{scatt}$ . In the context of seismic migration or traveltime tomography, equation 23.1 is often an overdetermined, ill-posed and inconsistent set of equations.

To reduce the artifacts associated with inconsistent and overdetermined equations, we often compute the least squares solution given by (Yilmaz, 2001)

$$\begin{aligned} \mathbf{m}^{tomo} &= \underbrace{[\mathbf{L}^T \mathbf{L}]^{-1} \mathbf{L}^T}_{\text{tomographic PSR}} \overbrace{\mathbf{d}}^{\mathcal{L}\mathbf{m}}, \\ &= \underbrace{[\mathbf{L}^T \mathbf{L}]^{-1} \mathbf{L}^T \mathcal{L}}_{\text{tomographic PSR}} \underbrace{\sum_i \mathbf{m}_i}_{\mathbf{m}}, \end{aligned} \quad (23.2)$$

where a regularization term is often used to stabilize the solution and  $\mathcal{L}$  is the Earth's actual modeling operator. Here, we assume that  $\mathbf{d}$  can be decomposed into a summation of modeled

slowness anomalies

$$\mathbf{d} = \sum_i \mathbf{L} \mathbf{m}_i, \quad (23.3)$$

where  $\mathbf{m}_i$  is the  $i^{th}$  localized slowness anomaly. These anomalies can be combined together to form a larger slowness anomaly  $\mathbf{m} = \sum_i \mathbf{m}_i$  such as a large scattering body.

The Hessian inverse  $[\mathbf{L}^T \mathbf{L}]^{-1}$  is often too expensive to compute so an iterative regularized gradient method (Lailly, 1983; Tarantola, 1987) is used to estimate the solution:

$$\mathbf{m}^{(k+1)} = \mathbf{m}^{(k)} - \alpha [\mathbf{L}^T \overbrace{\{\mathbf{L} \mathbf{m}^{(k)} - \mathbf{d}^{obs}\}}^{residual = \mathbf{r}^{(k)}} + \lambda \mathbf{m}^{(k)}], \quad (23.4)$$

where  $k$  is the iteration index,  $\lambda > 0$  is the regularization term (Tikhonov and Arsenin, 1977; Scales et al., 1990), the step length is  $\alpha > 0$ , and a preconditioning operator is sometimes chosen to be a diagonal matrix where  $\mathbf{C}_{ii} = [\mathbf{L}^T \mathbf{L}]_{ii}^{-1}$ . In practice, either an efficient preconditioned conjugate gradient (Scales, 1987) or a quasi-Newton method (Nocedal and Wright, 1999) can be used instead of the steepest decent procedure in equation 23.4.

In the context of traveltimes tomography,  $\mathbf{m}$  is the slowness model and  $\mathbf{d}$  defines the picked traveltimes of transmission, refraction and/or reflection traveltimes (Bishop et al., 1985; Langan et al., 1985; White, 1989; Zhu and McMechan, 1989; Delprat-Jannaud and Lailly, 1993; Hole and Zelt, 1995; Stefani, 1995; Langan et al., 1997; Bube and Langan, 1997; Zhang and Toksöz, 1998; Yilmaz, 2001; Bube and Langan, 2008; Kotsi et al., 2019). The operator  $\mathbf{L}$  represents the forward modeling of traveltimes by either ray tracing or a numerical solution to the wave equation (Woodward, 1989, 1992; Luo and Schuster, 1991). Despite the efforts to stabilize and precondition the gradient methodology, the final tomogram is always filled with artifacts associated with either incomplete data and/or physics. Such artifacts can be caused by, for example, a limited source-receiver aperture, uneven density and/or angular diversity of rays in each slowness cell, strong velocity contrasts that prevent many rays from entering areas of interest, and an irregular and coarse distribution of sources and receivers (Schuster, 2017).

Gautam et al. (2021) proposed a non-linear deconvolution filter  $F$  that can suppress some of these artifacts in the slowness tomogram  $\mathbf{m}^{tomo}$  in equation 23.2, i.e.

$$\mathbf{m} \approx F(\mathbf{m}^{tomo}), \quad (23.5)$$

where  $\mathbf{m}$  is the actual slowness model and the deconvolved tomogram  $F(\mathbf{m}^{tomo})$  is a more accurate rendering of  $\mathbf{m}$  than given by the standard tomogram  $\mathbf{m}^{tomo}$ . According to equation 23.2, the deconvolution filter in equation 23.5 is given by

$$F = ([\mathbf{L}^T \mathbf{L}]^{-1} \mathbf{L}^T \mathcal{L})^{-1}, \quad (23.6)$$

where  $\mathcal{L}$  can be roughly approximated by the computer-based operator  $\mathbf{L}$ . The deconvolution filter  $F(\cdot)$  is also denoted as the inverse to the tomographic point-scatterer response (PSR). This filter is computed by training a convolution neural network (CNN), where the input is the tomogram  $\mathbf{m}^{tomo}$  and the output is the sum of localized slownesses  $\mathbf{m} \approx \sum_i \mathbf{m}_i$ . Here, the index  $i$  indicates different locations of the localized slowness anomalies  $\mathbf{m}_i$ .

### 23.1.1 Previous Work

Equation 23.2 implies that the deconvolution operator  $F$  can be efficiently found by only finding the inverse operator to localized tomographic PSRs of individual anomalies. If the background velocity variations are not strong, then the localized PSRs can be assumed to be spatially invariant

over a small region centered about the anomaly. These anomalies only differ from one another in their location and strengths. In this way training a CNN to find  $F(\cdot)$  can be significantly more efficient than training on a huge number of randomly selected slowness models. This is the same strategy used to develop efficient and effective migration deconvolution filters for migration (Hu et al., 1998 and 2001; ; Yu et al., 2006; Aoki and Schuster, 2009). Related migration deconvolution filters are developed by Fletcher et al. (2016), Cavalca et al. (2018), and Guitton (2004). They are all inexpensive alternatives to iterative least squares migration (LSM) (Lailly, 1983; Tarantola, 1987; Schuster, 1993; Nemeth et al., 1999; Chavent and Plessix, 1999; Duquet et al., 2000; Köhl and Sacchi, 2003; Wang et al., 2016 and many others) where at least 4 iterations are usually required to provide a reasonable estimate of the actual reflectivity distribution.

The migration deconvolution efforts are related to those for waveform inversion by machine learning methods. For example, Araya-Polo (2018) and others (Yang and Ma, 2019; Park et al., 2020) train a neural network to invert the input traces for the velocity model. This is an ambitious goal because a successfully trained network would require millions of training examples that account for the huge number of possible data sets and geology models (Karpatsch et al., 2017). A means for reducing this computational cost is by processing the input into velocity semblance panels (Sun et al. 2021), which are then used as part of the input training data. The semblance panels distill important information about the velocity into a more compact form, and so can possibly lead to more accurate and efficient results. Nevertheless, this *brute-force* approach is often challenged by high computational costs, uncertain effectiveness and the inability to generate a diverse range of velocity models that represents the wide spectrum of possible earth models.

A strategy for generating a wide variety of velocity models is to use GANs. Sun et al. (2021) propose velocity inversion by U-Net-like architectures trained with pseudo-randomly-generated velocity models and then, with transfer learning, re-trained with GAN-generated synthetic velocity models. Results demonstrate that the prediction accuracies for both salt geometries and background velocities can be largely improved with the proposed hybrid workflow. However, the final models are not in acceptable agreement with the desired ones if the training models are far from the true models.

In comparison, the approach of TD uses a physics-based tomographic operator to compute the tomogram, which computes the traveltime tomographic response of the subsurface velocity model. Only the point-source artifacts of the tomographic operator and their elimination are used to train the neural network. Similar to MD, TD assumes that the point-scatterer responses are spatially invariant over a localized region of space. This reduces the huge number of training models required for proper training. The TD procedure is a sequential use of physics-based and machine learning operations that can be iteratively repeated to give new starting models. This hybrid approach of using inverse operators based on physics and ML is related to the approaches of Columbo et al. (2021), Alyousuf et al. (2021) and others who use objective functions that are a composite of ML and physics-based misfit functions.

The next section presents the theory of TD. The fundamental concept of the tomographic point-scatterer response (PSR) is defined, and then its inverse is obtained using a neural network model. A convolutional neural network is used and the workflow for its implementation is described. Then the numerical results are presented for tomographic deconvolution of tomograms computed by traveltime inversion of reflection data and refraction data.

## 23.2 Theory of Tomographic Deconvolution

The first Fresnel zones for transmission and reflection arrivals are illustrated in Figure 23.1b for a single source-receiver pair. These zones are defined as the points visited by rays that can be traced from the source to the receiver with a traveltime greater than or equal to the central ray traveltime at time  $t$  and less than  $t + T_0/2$ . Here,  $T_0$  is the dominant period of the source (Schuster, 2017).

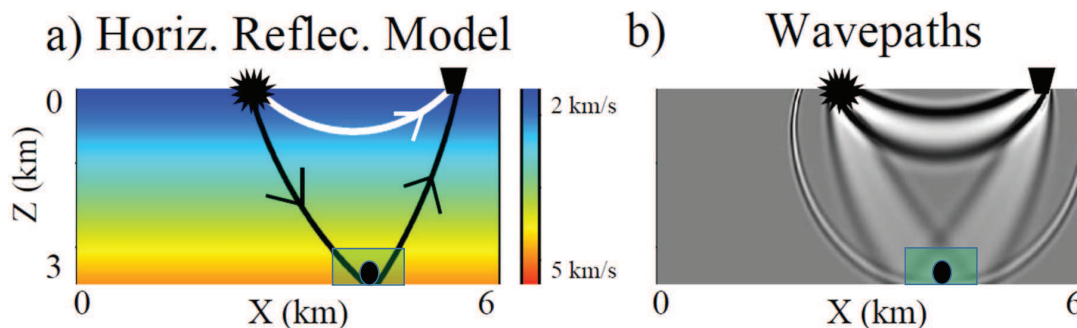


Figure 23.1: a) Horizontal reflector model along with the white direct and black reflection raypaths. b) Specular reflection and transmission wavepaths obtained by reverse time migration of a trace recorded at the geophone (quadrilateral) and excited by a bandlimited source (star). The gray-shaded migration ellipse has foci at the source and receiver. Figures courtesy of Ge Zhan.

These first Fresnel zones are also denoted as wavepaths (Woodward, 1989, 1992; Luo, 1991).

For the TD problem, we assume that the location of the reflector and the background velocity is known, and the unknown is the location and extent of the green velocity anomaly in Figure 23.1b. This unknown anomaly is the only part of the model that generates a traveltime residual in the data. For a single source-receiver pair, this means that the velocity is updated all along the rabbit-ear wavepath in Figure 23.1b because this is the zone which largely influences the phase of the reflection arrival (Huang and Schuster, 2014). The green velocity anomaly is at the intersection of the rabbit ears on the reflector so that the resulting slowness tomogram  $\mathbf{m}^{(k+1)} = \mathbf{m}^{(k)} - \alpha \mathbf{L}^T \Delta \mathbf{r}^{(k)}$  will take the appearance of a pair of rabbit ears. This is what we call the *point-scatterer response* (PSR) of the tomographic operator for a single traveltime residual. The shape of this response will change depending on the location of the scatterer relative to the reflector boundary and the number of traces, as illustrated in Figures 23.2a-23.2b. Figure 23.2c is the ideal deconvolved tomogram  $F(\mathbf{m}^{tomo})$  in equation 23.5, where the rabbit-ear artifacts are removed by the deconvolution operator  $F$  to obtain the actual shape of the scatterer.

### 23.2.1 Migration Deconvolution versus Tomographic Deconvolution

The PSR of the migration operator takes the shape of a butterfly centered at the location of the localized scatterer (Schuster and Hu, 2000). For a sufficiently dense source-receiver spacing and wide acquisition aperture, these artifacts are mostly localized to be within several wavelengths of the anomaly. This means that the cross-talk noise from distant butterfly artifacts is minimal, so a local linear migration deconvolution operator can effectively suppress artifacts (Hu et al., 2001; Yu et al., 2006). This is not the case with the rabbit-ear artifacts of the tomographic PSR, where the tomographic artifacts associated with a single scatterer can occupy a large portion of the model. These artifacts can vary dramatically for different locations of the slowness anomaly. Therefore, TD filters require a non-linear inverse filter, such as provided by a CNN.

### 23.2.2 Tomographic Deconvolution Workflow

The tomographic deconvolution workflow consists of 4 steps.



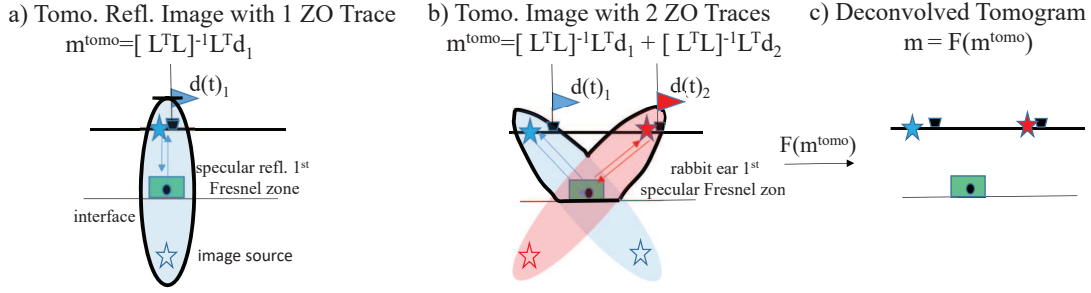


Figure 23.2: Tomographic models  $\mathbf{m}^{tomo} = [\mathbf{L}^T \mathbf{L}]^{-1} \mathbf{L}^T \mathbf{d}$  inverted from specular reflection traveltimes picked from a) one and b) two ZO traces. The slowness model is updated by smearing the traveltimes residuals to be within the Fresnel zone bounded by the thick black line in b) that forms the rabbit ears. c) is the deconvolved tomogram  $F(\mathbf{m}^{tomo})$ .

- Collect traveltimes reflection or refraction data and invert by traveltimes tomography to get the smooth background slowness model  $\mathbf{m}^{tomo}$ .
- Train the CNN model with synthetic reflection or refraction data to find  $F(\mathbf{m}^{tomo})$  in Figure 23.2c. The background slowness model can be obtained by smoothing  $\mathbf{m}^{tomo}$ , or the original tomogram can be used. The input to the training consist of small 'point' slowness anomalies located at different positions in the model. The size of these point anomalies is not less than one to two wavelengths in size if ray tracing is used for forward modeling.
- Once the CNN model is properly trained and validated, it is applied to the tomogram  $\mathbf{m}^{tomo}$  to get  $\mathbf{m}^{decon} = F(\mathbf{m}^{tomo})$ , where the tomographic artifacts are suppressed and  $\mathbf{m} \approx \mathbf{m}^{decon}$ .
- Predicted data are computed  $\mathbf{d}^{pred} = \mathbf{L} \mathbf{m}^{decon}$  and a sanity test is used to validate whether  $\mathbf{m}^{decon}$  accurately predicts the recorded traveltimes data  $\mathbf{d}$ .

### 23.2.3 CNN Architecture

The CNN model in Figure 23.3 consists of a contracting path and an expansive path (which gives it the U-shaped architecture). The contracting path consists of six "Down Blocks", where each block is made up of two repeated convolutional operations, each followed by a ReLU and a max-pooling operation. During the contraction, the spatial information is reduced while feature information is increased. The expansive pathway is made up of six "Up Blocks", where each block combines the features and spatial information through a sequence of transpose convolutions and concatenations with high-resolution features from the contracting path, followed by two convolutional operations. At the bottom-most part (just where the Down Blocks end), there's a 'bottleneck block' which is made up of two back-to-back convolutional operations. The thresholding operators are ReLU.

The training of the CNN model uses synthetic reflection traveltimes data computed by a reflection ray-tracing code, and then inverted by an iterative gradient method that minimizes the  $L^1$  data misfit function. The slowness models, for example, are localized slowness anomalies in Figures 23.4a and 23.4d, and the tomograms  $\mathbf{m}^{tomo}$  are shown in Figures 23.4b and 23.4e. The supervised output are the desired slowness models in Figures 23.4a and 23.4d. More than a thousand training pairs ( $\mathbf{m}^{tomo}, \mathbf{m}$ ) are used to train the CNN model in Figure 23.3. Each training model is a localized slowness anomaly with a different location of the slowness anomaly. This model is validated using about 200 out-of-the-training set pairs.

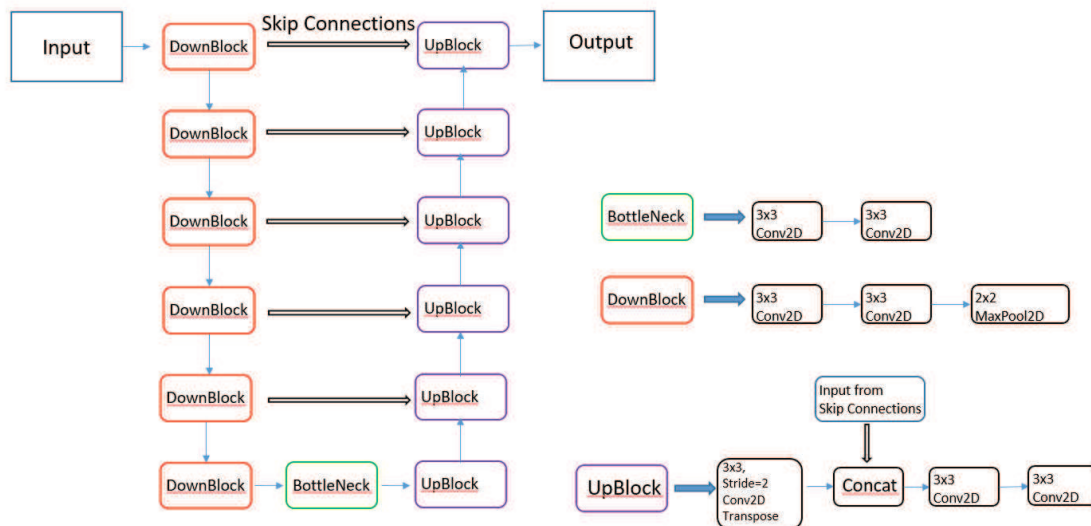


Figure 23.3: U-Net architecture. Each red, green and blue box consists of an output in the form of a multi-channel feature map with the number of channels increasing to 512 at the 'bottleneck' block. The blue arrows denote the input and outputs to/from different blocks/layers, and the black arrows define the skip connections.

## 23.3 Numerical Results

Numerical results are now presented for deconvolving tomograms by the TD operator  $F$  computed by a CNN. The reflection traveltime data are computed by reflection ray tracing in a two-layer model with embedded slowness anomalies. A source and receiver spacing of 10 m is used, and there are 199 sources located on the surface. Each source shoots into 200 receivers so 39,800 traveltimes are generated for each simulation. It is assumed that the background medium is a two-layer model with a horizontal density contrast and having the same homogeneous velocity in each layer; the depth of the horizontal reflector is known. The localized slowness anomalies have a small contrast in velocity so ray bending is negligible, and the training is computed for localized slowness anomalies located randomly between the reflecting interface and the recording surface.

Similar tests are also conducted for refraction tomography. A source and receiver spacing of 10 m is used, and 50 sources (40 m apart) are placed on the surface with 200 receivers associated with each source. Apart from this, ray bending is used during ray tracing for refraction results.

### 23.3.1 Reflections: Single Anomaly Models

Several tests are conducted using models with a single slowness anomaly. The CNN is trained on 5000 pairs of slowness tomograms and the actual slowness models similar to the ones shown in Figure 23.4a and Figure 23.4d.

Figure 23.4a shows a reflector model with a yellow velocity anomaly centered above the horizontal reflector at  $z = 1000$  m. These traveltimes are inverted using a preconditioned  $L^1$  traveltime tomography algorithm to give the slowness tomogram in Figure 23.4b. Notice the rabbit-ear artifacts associated with the slowness anomaly. Applying the TD filter  $F(\cdot)$  in equation 23.6 to this tomogram gives the result shown in Figure 23.4c. The second row in Figure 23.4 is similar to the first row except the slowness anomaly in Figure 23.4d is smaller sized and lowered from about 450 m to 750 meters in depth.

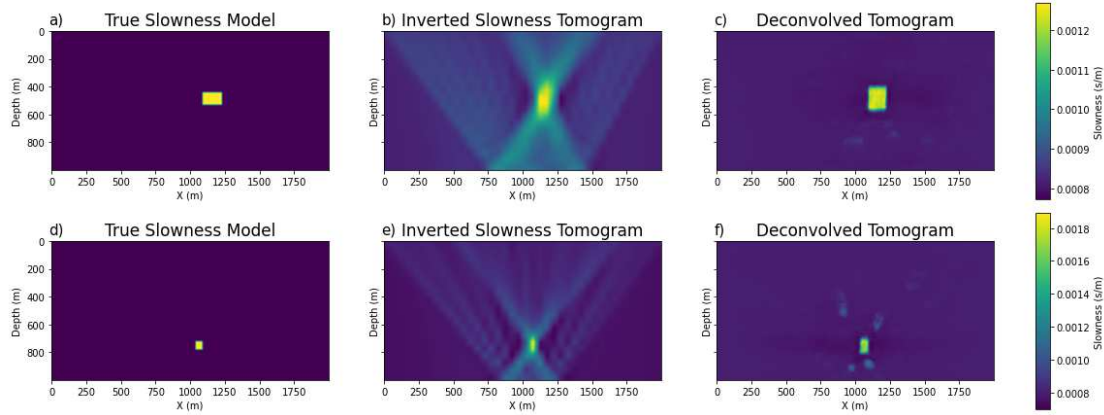


Figure 23.4: a) Slowness model, b) slowness tomogram after inverting 30,900 reflection traveltimes and c) the deconvolved tomogram. The second row of images are similar to the first row except the yellow slowness anomaly in d) is lowered in depth.

The same CNN was also tested on velocity models with multiple anomalies. For the case where, the two anomalies were very close, the results were not satisfactory. The main reason for this was the presence of overlapping rabbit-ear artifacts, which was resolved by training the CNN on additional velocity models which had multiple anomalies. The results for this test are discussed in the next section.

### 23.3.2 Reflections: Multiple Anomaly Models

To get good results on models with multiple anomalies, the CNN trained on an additional 10,000 pairs of slowness tomograms and actual slowness models. The training dataset contains multiple anomalies at random locations (Figure 23.5). The similarity (Figure 23.6) between the loss functions for the training and validation sets suggests that the model is well trained without overfitting. In Figures 23.7 and 23.8, the first column of images represents the out-of-the-training set models in which the CNN was tested. The deconvolved tomograms in the third column suggest that TD can significantly improve the resolution of the slowness tomograms shown in the middle column of images. Many other examples (not shown) were tested and the vast majority of results suggested that the deconvolved tomograms provide significant improvement in resolution compared to the original tomograms. The cases where there was no improvement are discussed in the next section.

### 23.3.3 Models that Break the Deconvolution Filter

It is well known that the best resolution of a raypath is perpendicular to the ray (Schuster, 2017). Therefore slowness variations in depth are not well resolved for reflection data with narrow incident angles. In fact, the 2D Fourier transform of a horizontal slowness layer that is  $\lambda_0$  thick is dominated by the spectral energy at  $k_z = 2\pi/\lambda_0$ . In contrast, the reflection rays are devoid of horizontal or nearly horizontal raypaths, so they are unable to accurately recover the  $k_z = 2\pi/\lambda_0$  component of the model spectrum (Schuster, 2017). This is illustrated in Figure 23.9 where two oscillatory interfaces at different depths in Figures 23.9a and 23.9d give about the same traveltimes data. This means that there are many models that satisfy nearly the same data so that there is not a unique solution to the traveltimes equations, as illustrated by the tomograms in Figures 23.9b and 23.9e. Therefore, it is not surprising that the deconvolution operator gives poor results in Figures 23.9c and 23.9f. The null space of  $\mathbf{L}^T \mathbf{L}$  is non-empty.

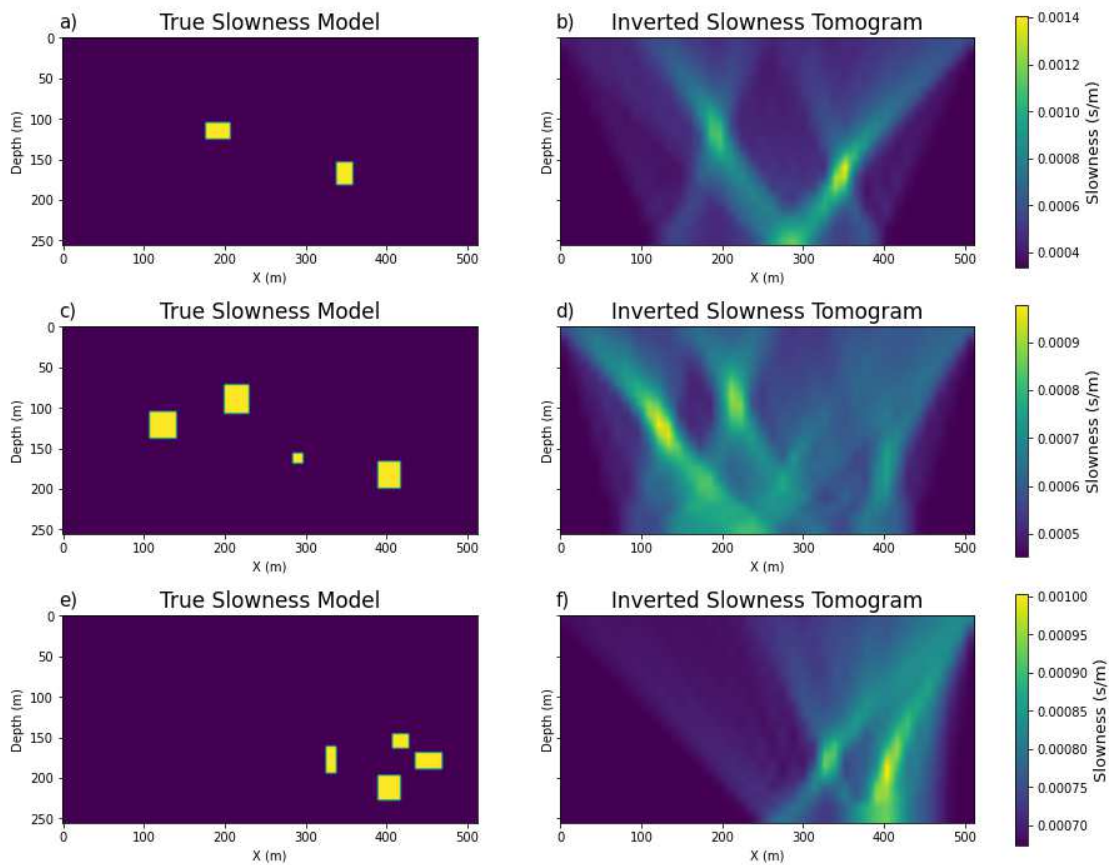


Figure 23.5: a) True slowness model with two anomalies and b) is the corresponding inverted slowness tomogram. The second and third row of images are similar to the first row except there are four yellow slowness anomalies in c) and e).

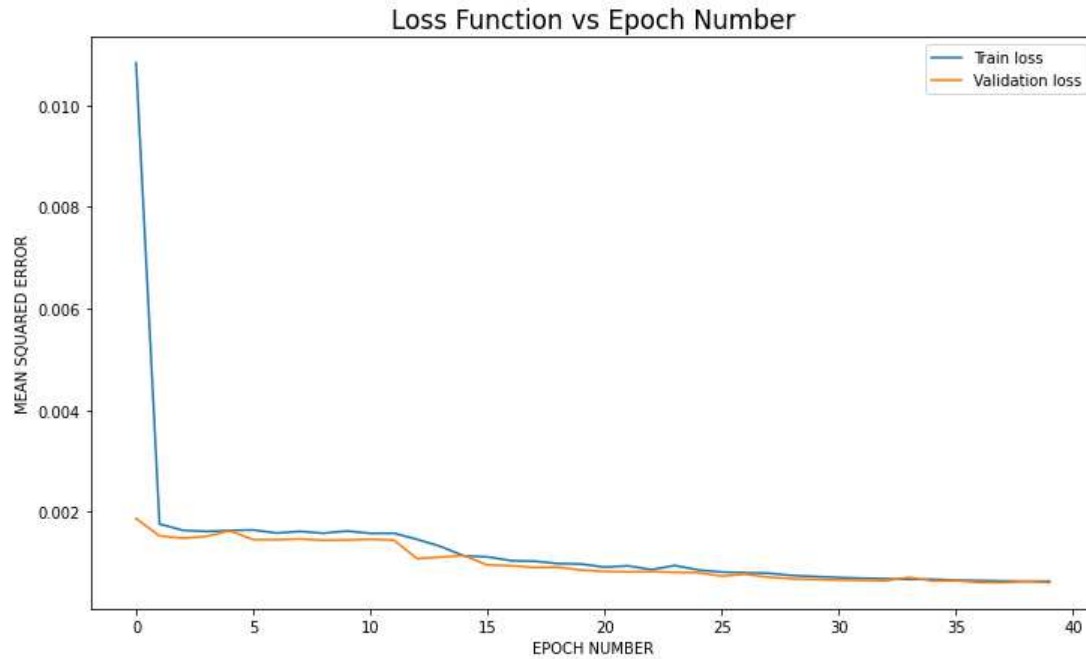


Figure 23.6: The training and validation loss functions plotted against iteration number. These data were computed for the training set consisting of 10,000 pairs of slowness tomograms and actual slowness models.

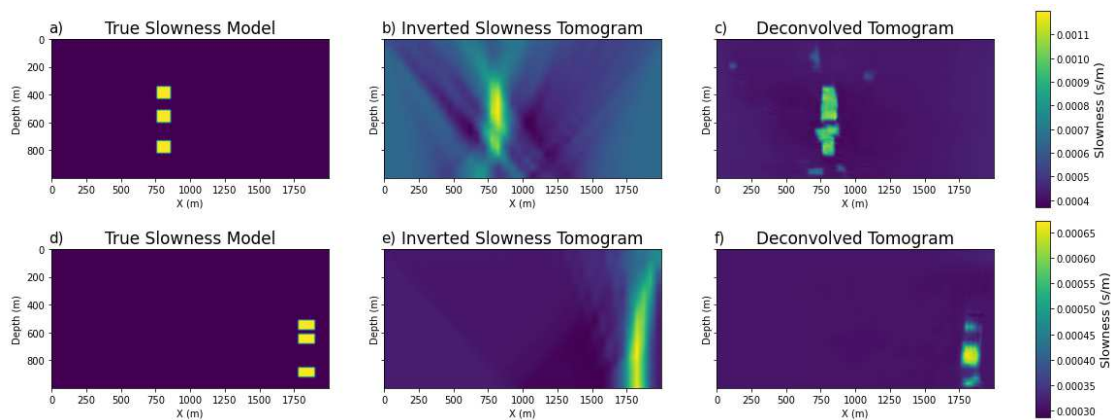


Figure 23.7: Same as Figure 23.4 except the input slowness models consisted of multiple anomalies stacked vertically one on top of another.

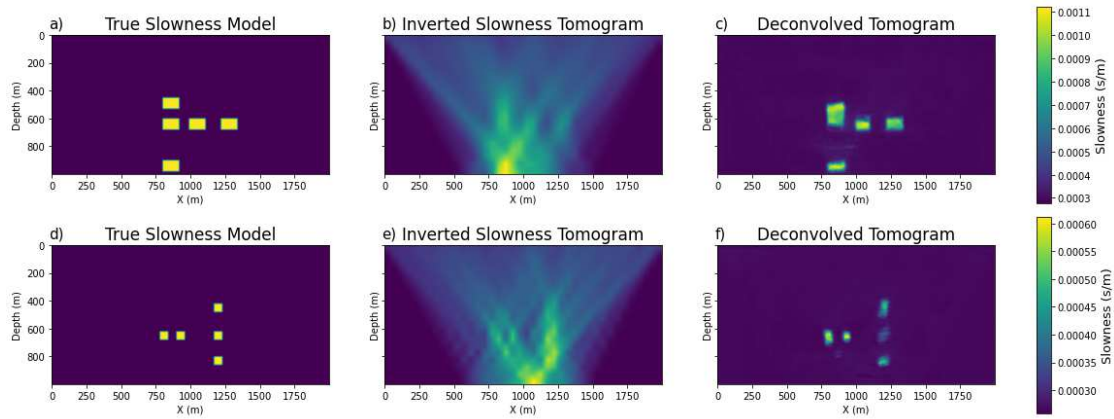


Figure 23.8: First column of images depicts the actual slowness models, the second column shows the traditional slowness tomograms and the third column presents the deconvolved tomograms.

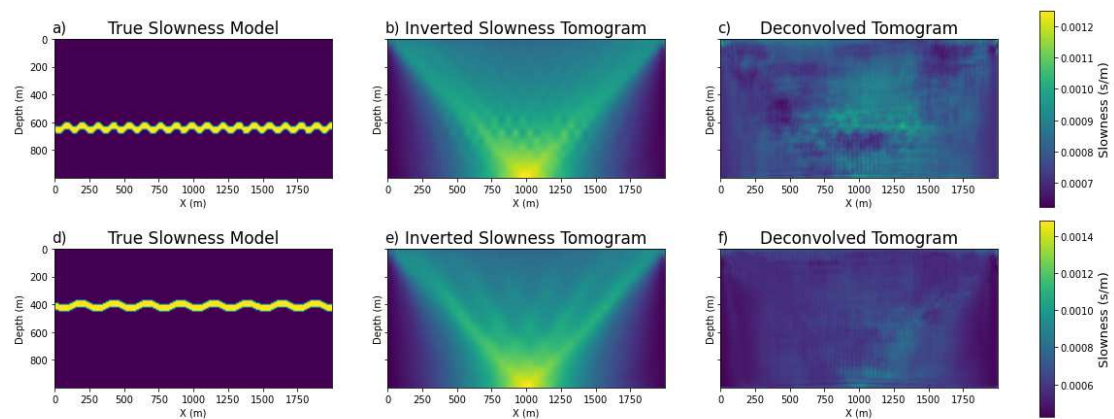


Figure 23.9: Same as the previous figure except the input slowness models consist of an oscillatory low-velocity layer at different depths.



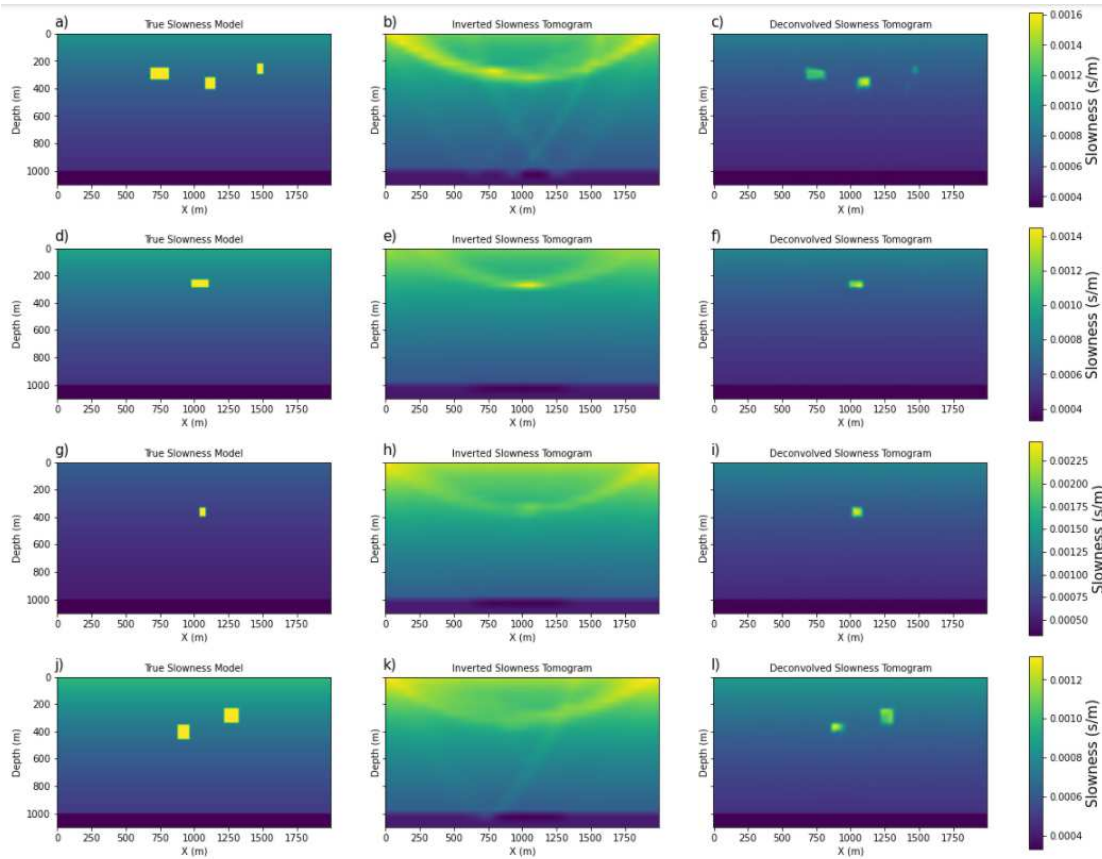


Figure 23.10: First column depicts the actual slowness models, the second column shows the refraction tomograms and the third column displays the deconvolved tomograms for refraction data.

### 23.3.4 Refractions: Multiple Anomaly Models

The CNN is now trained on refraction events generated in models with velocity increasing in depth. Similar to the reflections models, the refraction models also contain small velocity anomalies between the deep refractor interface and the free surface as shown in Figure 23.10. Unfortunately, the range of incidence angles of refractions is not as diverse as those from reflections, so the image resolution from refraction tomography in the middle column of Figure 23.10 is not as accurate as that from reflection tomography. Nevertheless, the deconvolved tomograms showed reasonable resolution as shown by the results in the third column of Figure 23.10.

### 23.3.5 Refractions: Fault Anomaly Models

The CNN is also extensively tested on models with faults. The training dataset consists of velocity models with varying faults lengths and locations. Figure 23.11 shows the test results for faults with random lengths, orientations and locations. The third row of tomograms demonstrates that the CNN can deconvolve complex subsurface structures.

To make the test even more complex, the CNN was tested on velocity models where faults start at the surface and extend deeply into the subsurface. The deconvolved tomograms in Figure 23.12

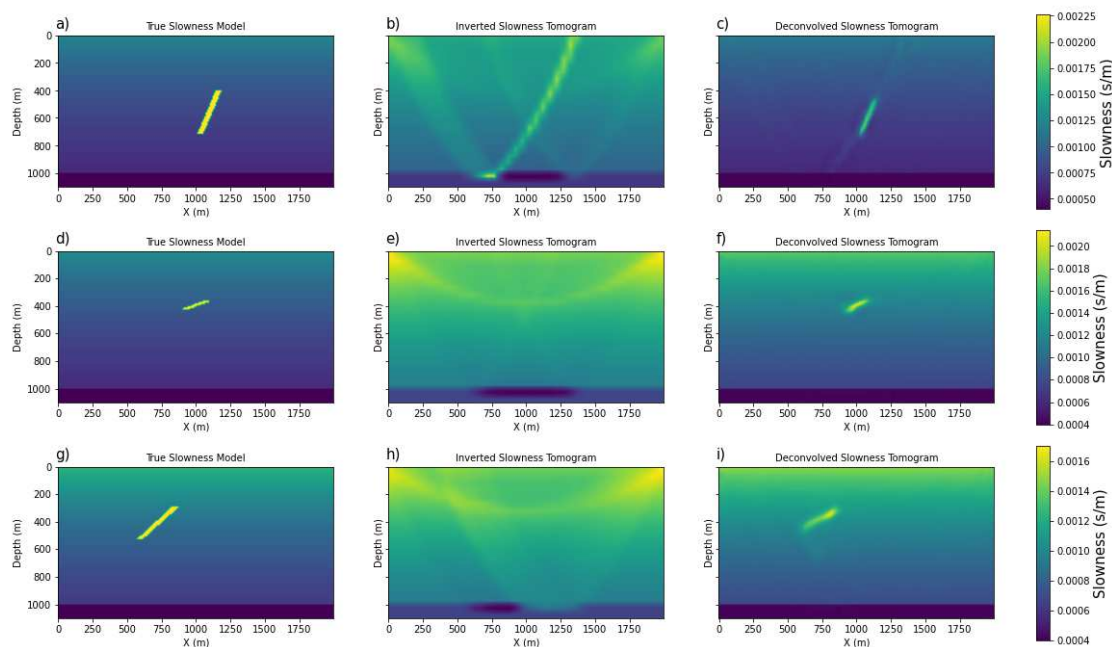


Figure 23.11: First column of images depicts the actual fault models, the second column shows the refraction tomograms and the third column provides the deconvolved refraction tomograms.

suggest that the trained CNN is equally effective with data recorded over these types of models.

## 23.4 Summary

The theory of TD is presented where the tomogram is deconvolved of artifacts to give a high-resolution estimate of the localized velocity anomalies in a reflecting layer. The assumptions are that ray bending can be neglected in the slowness anomalies and that the anomalies are no smaller than about a wavelength in order to satisfy the high-frequency approximation of ray tracing. To go beyond this approximation we should use a bending-ray tomography code or a wave-equation tomography method. If the model with the anomalies is partly in the null space of  $[\mathbf{L}^T \mathbf{L}]$  then this method will not be able to recover the null space components.

Tomographic deconvolution is similar to more general procedures that train a network with (input, output) = (synthetic seismic data, velocity model), except the input for TD is the standard tomogram which is assumed to consist of point-scatterer artifacts of the tomographic operator. Thus, the task of computing the TD operator should be theoretically much simpler than that of general ML training of FWI operators.

To ensure that the CNN does not overfit the training data, a diverse suite of training models should be used and the convergence curves for both the training and validation data should be similar. In addition, a sanity test is recommended after computing the deconvolved tomogram. That is, the final deconvolved tomogram should be used to generate synthetic traveltimes. These traveltimes can be compared to the recorded ones to ensure that the  $L^2$  residual of the deconvolved tomogram is equal to or less than that from the input tomogram.

The computational requirements for training the CNN are high in this feasibility study, but it is straightforward to reduce costs by reusing the forward modeling Green's functions as described



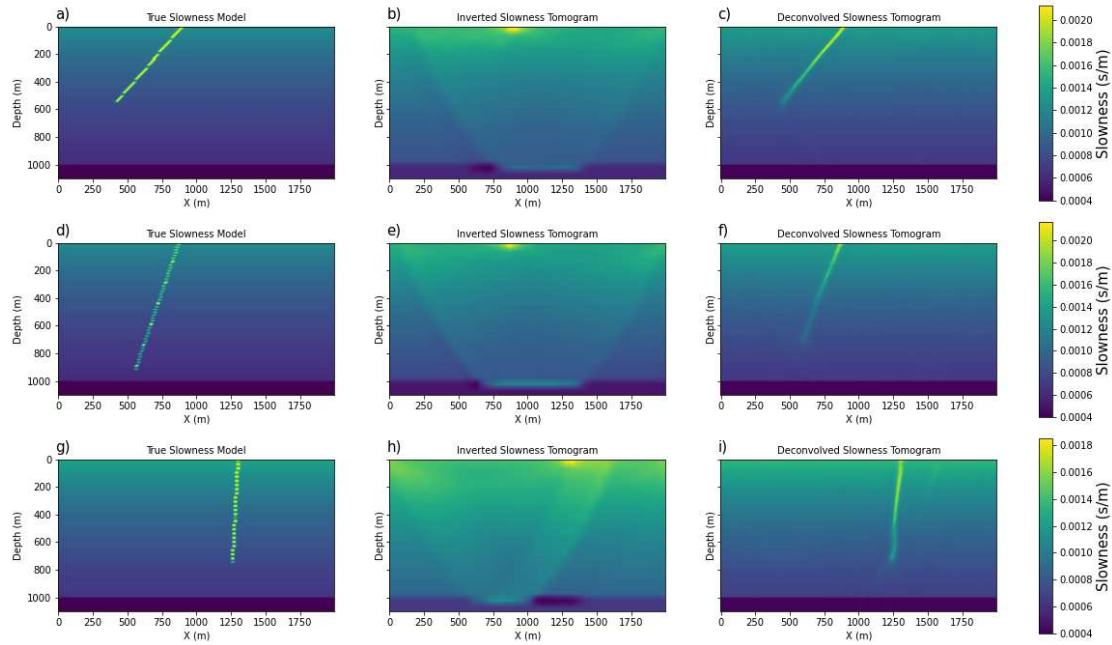


Figure 23.12: Same as previous figure except a different model model.

by the generalized diffraction migration procedure in Schuster (2017). Other short cuts can be used as well such as transfer training. Tomographic deconvolution can also be used for transmission tomography and refraction tomography.

There are two significant difficulties with the TD: high computational cost in training the CNN and lack of testing on field data. Forward modeling with thousands of forward computations for training is not feasible for large 3D velocity models and limited computational resources. Also, the results here were for the idealized case and did not account for the noisy nature and the complexities of finite-frequency wave propagation in complex models. Both of this issues must be addressed before TD can be successfully implemented on real data.



## Chapter 24

# Neural Network Least Squares Migration

The forward pass of a multilayered convolutional neural network can be recast as the solution to the problem of sparse least squares migration (LSM). The CNN filters and feature maps are shown to be analogous, but not equivalent, to the migration Green's functions and the quasi-reflectivity distribution, respectively. This provides a physical interpretation of the filters and feature maps in deep CNN in terms of the operators for seismic imaging. Unlike the standard LSM method that finds the optimal reflectivity image, neural network LSM (NNLSM) finds both the optimal quasi-reflectivity image and the quasi-migration Green's functions. These quasi-migration-Green's functions are also denoted as the convolutional filters in a CNN and are similar to migration Green's functions. The advantage of NNLSM over standard LSM is that, if the training efforts are ignored, its computational cost is significantly less and it can be used for denoising coherent and incoherent noise in migration images. Its disadvantage is that the NNLSM quasi-reflectivity image is only an approximation to the actual reflectivity distribution. However, the quasi-reflectivity image can be used as a super-resolution attribute image for high-resolution delineation of geologic bodies.

### 24.1 Introduction

Deep convolutional neural networks (CNNs) have been recently used for solving geophysical problems, such as seismic first-arrival picking (Yuan et al., 2018; Hu et al., 2019), seismic interpretation (Wu et al., 2019a, 2019b; Shi et al., 2019), and seismic imaging (Xu et al., 2019; Sun et al., 2020; Kaur et al., 2020). Interpretation examples include salt classification (Waldeland et al., 2019; Shi et al., 2019), fault detection (Huang et al., 2017; Xiong et al., 2018; Wu et al., 2019a, 2019b; Zheng et al., 2019), reservoir characterization (Karimpouli et al., 2010; Cao and Roy, 2017; Zhu et al., 2017), and seismic lithofacies classification (Ross and Cole, 2017; Liu et al., 2019) with semi-supervised learning (Di et al., 2020). Other applications include the use of neural networks for well-log interpolation (Saggaf, and Nebrija, 2003; Salehi et al., 2017; Pham et al., 2020), seismic data interpolation (Mandelli et al., 2018; Mandelli et al., 2019; Wang et al., 2019), velocity model building (Araya-Polo et al., 2018; Richardson, 2018), well-log ties (Bader et al., 2019), synthetic well-data generation (Rolon et al., 2009), autoencoders for unsupervised facies analysis (Qian et al., 2018), and supervised horizon tracking (Peters et al., 2019a, 2019b). Recently, an unsupervised autoencoder method with regularization was developed by (Shi et al., 2020) to track target horizons.

There are many types of neural network or machine learning methods, selections ranging from generative adversarial networks for seismic interpolation (Siahkoobi et al., 2018), residual networks for traces missing at regular intervals (Wang et al., 2020), Monte Carlo and support vector regression

(Jia et al., 2018) for data interpolation, autoencoders (Wang et al., 2020; Shi et al., 2020) for target horizons tracking, and recurrent neural networks for well-log interpolation (Pham et al., 2019, 2020). The design of geophysical CNN architectures has largely been based on empirical evidence from computer vision research, insights from the principles of artificial intelligence and heuristic experimentation. Heuristic experimentation is most often used to decide the parameter design<sup>1</sup> for the CNN architecture, which has both merits and liabilities. The merit is that trial-and-error testing with different architecture parameters is likely to give excellent results for a particular data set, but it might not be the best one for a completely different data set. This shortcoming in using empirical tests for parameter selection partly results from the absence of a rigorous mathematical foundation (Pappas et al., 2016; Pappas et al., 2017a, 2017b) for neural networks in general, and CNN in particular.

To mitigate this problem for CNN-based imaging algorithms, we now present a physical interpretation of the CNN filters and feature maps in terms of the physics-based operators for seismic imaging. With such an understanding, we can use a physics-based rationale for the better design of CNN architectures in seismic migration and inversion.

Donoho (2019) points out that "machine learning has a troubled relationship with understanding the foundation of its achievements well and its literature is admittedly corrupted by anti-intellectual and anti-scholarly tendencies". Progress in advancing the capabilities of deep neural networks will be severely stymied unless its mathematical foundations are established. As a first step in this direction, Pappas et al. (2016) proposed that the forward modeling operation of CNN could be recast as finding the sparsest model  $\mathbf{m}$  under the  $L^1$  norm subject to honoring the data misfit constraint  $\|\mathbf{\Gamma}\mathbf{m} - \mathbf{d}\|_2^2 \leq \beta$ :

$$\begin{aligned} \text{Given : } & \quad \mathbf{\Gamma}, \mathbf{d} \quad \text{and} \quad \mathbf{\Gamma}\mathbf{m} = \mathbf{d} + \textit{noise}, \\ \text{Find : } & \quad \mathbf{m}^* = \arg \min_{\mathbf{m}} \|\mathbf{m}\|_1, \\ & \quad \text{subject to } \|\mathbf{\Gamma}\mathbf{m} - \mathbf{d}\|_2^2 \leq \beta \end{aligned} \tag{24.1}$$

where  $\mathbf{m}^*$  is the optimal solution of equation 24.1,  $\mathbf{\Gamma}$  is the dictionary matrix,  $\mathbf{d}$  represents the signal, and the scalar  $\beta$  is the specified noise tolerance. The iterative solution to this problem is a series of forward-modeling operations of a neural network, where the mathematical operations of each layer consist of two steps: a weighted summation of input values to give the vector  $\mathbf{z}$  followed by a two-sided soft thresholding operation denoted as  $\sigma(\mathbf{z})$  (Pappas et al., 2016).

The sparse constraint problem defined in equation 24.1 is commonly seen in geophysics, for example, the least square migration (LSM) with a sparse constraint. LSM is an important seismic-imaging technique to produce images with better balanced amplitudes, fewer artifacts and better resolution than standard migration (Lailly, 1983; Tarantola, 1987; Schuster, 1993; Nemeth et al., 1999; Chavent et al., 1999; Duquet et al., 1999; Feng and Schuster, 2017; Schuster, 2017). The sparse constraint is one of the important regularization terms used in solving the ill-conditioned least-squares problem (Sacchi and Ulrych, 1995; De Roeck, 2002; Köhl and Sacchi, 2003; Wang and Sacchi, 2005) and sparse LSM (SLSM) has been demonstrated to be effective for mitigation of aliasing artifacts and crosstalk in LSM (Wang and Sacchi, 2005; Herrmann et al., 2009; Dutta, 2017; Witte et al., 2017; Li et al., 2018). The image-domain SLSM finds the sparsest reflectivity  $\mathbf{m}^*$  with sparse constraints to minimize the objective function  $\|\mathbf{\Gamma}\mathbf{m} - \mathbf{m}^{mig}\|_2^2$ , where  $\mathbf{\Gamma}$  is the Hessian matrix and  $\mathbf{m}^{mig}$  is the migration image. Here, we can see SLSM shares the same problem as that defined in equation 24.1. Following the work of (Pappas et al., 2016), we show that the sparse solution to the LSM problem reduces to the forward modeling operations of a multilayered neural network. The CNN filters and feature maps are shown to be analogous, but not equivalent, to the migration

---

<sup>1</sup>The design of a CNN architecture selects, for example, the number of CNN layers, the number of filters/layer, the size of the filters, the type of activation functions, the number of skip layers, the stride length and whether a layer acts as a decoding or encoding operation.

Green's functions (Hessian) and the reflectivity distribution.

The standard SLSM algorithm requires the numerical solutions to the wave equation, which is computationally costly. Motivated by the connection between sparse LSM and CNN, Liu et al. (2020) propose the neural network version of sparse LSM, which does not require the numerical solutions to the wave equation and is faster than standard SLSM. Instead of just finding the optimal reflectivity  $\mathbf{m}^*$ , we optimize for both the quasi-reflectivity  $\mathbf{m}$  and the quasi-migration-Green's functions  $\mathbf{\Gamma}$ . These quasi-migration-Green's functions approximate the role of migration Green's function (Schuster and Hu, 2000) and are denoted as the convolutional filters in a convolutional neural network. As discussed in Appendix 24.6, the migration Green's function is the point-scatterer response of the migration operator. The final image is denoted as the neural network least squares migration (NNLSM) estimate of the quasi-reflectivity distribution that honors the  $L^1$  sparsity condition. The next section shows the connection between the multilayer neural network and the solution to the multilayer NNLSM problem. This is followed by the numerical examples with the synthetic models and field data from the North Sea.

## 24.2 Theory of Neural Network Least Squares Migration

The theory of standard image-domain LSM is first presented to establish the benchmark solution where the optimal reflectivity function minimizes the image misfit under the  $L_2$  norm. This is then followed by the derivation of the sparse least squares migration (SLSM) solution for a single-layer network. The final two subsections derive the NNLSM solution for single-layer and multilayer networks, respectively.

### 24.2.1 Least Squares Migration

The least squares migration (LSM) problem can be defined (Nemeth et al., 2009; Schuster, 2017) as finding the reflectivity coefficients  $m_i$  in the  $N \times 1$  vector  $\mathbf{m}$  that minimize the  $L_2$  objective function  $\epsilon = \frac{1}{2} \|\mathbf{\Gamma}\mathbf{m} - \mathbf{m}^{mig}\|_2^2$ ,

$$\mathbf{m}^* = \arg \min_{\mathbf{m}} \left\{ \frac{1}{2} \|\mathbf{\Gamma}\mathbf{m} - \mathbf{m}^{mig}\|_2^2 \right\}, \quad (24.2)$$

where  $\mathbf{\Gamma} = \mathbf{L}^T \mathbf{L}$  is the symmetric  $N \times N$  Hessian matrix,  $\mathbf{L}$  is the forward modeling operator, and  $\mathbf{L}^T$  is the migration operator. Here,  $\mathbf{m}^{mig} = \mathbf{L}^T \mathbf{d}$  is the migration image computed by migrating the recorded data  $\mathbf{d}$  with the migration operator  $\mathbf{L}^T$ . Alternatively, the image-domain LSM problem can also be defined as finding  $\mathbf{m}$  that minimizes  $\epsilon = \frac{1}{2} (\mathbf{m}^T \mathbf{\Gamma} \mathbf{m} - \mathbf{m}^T \mathbf{m}^{mig})$ , which has a more well-conditioned solution than the one in equation 2. However, we will use equation 24.2 as the definition of the LSM problem in order to be consistent with the notation from Papyan et al. (2017a, 2017b). The kernel associated with the Hessian matrix  $\mathbf{L}^T \mathbf{L}$  is also known as the point scatterer response of the migration operator or the migration Green's function (Schuster and Hu, 2000). It is a square matrix that is assumed to be invertible, otherwise a regularization term is incorporated into the objective function.

A formal solution to equation 24.2 is

$$\mathbf{m}^* = \mathbf{\Gamma}^{-1} \mathbf{m}^{mig}, \quad (24.3)$$

where it is too expensive to directly compute the inverse Hessian  $\mathbf{\Gamma}^{-1}$ . Instead, a gradient method gives the iterative solution

$$\mathbf{m}^{(k+1)} = \mathbf{m}^{(k)} - \alpha \mathbf{\Gamma}^T (\mathbf{\Gamma} \mathbf{m}^{(k)} - \mathbf{m}^{mig}), \quad (24.4)$$

where  $\alpha$  is the step length,  $\mathbf{\Gamma}$  is symmetric, and  $\mathbf{m}^{(k)}$  is the solution at the  $k^{th}$  iteration. Typically,

a regularization term is used to stabilize the solution; for example, the sparse constraint which will be introduced in the next subsection.

### 24.2.2 Sparse Least Squares Migration

The sparse least squares migration (SLSM) in the image domain is defined as finding the reflectivity coefficients  $m_i$  in the  $N \times 1$  vector  $\mathbf{m}$  that minimize the objective function  $\epsilon$  (Perez et al., 2013):

$$\epsilon = \frac{1}{2} \|\mathbf{\Gamma} \mathbf{m} - \mathbf{m}^{mig}\|_2^2 + \lambda S(\mathbf{m}), \quad (24.5)$$

where  $\mathbf{\Gamma} = \mathbf{L}^T \mathbf{L}$  represents the migration Green's function (Schuster and Hu, 2000),  $\lambda > 0$  is a positive scalar,  $\mathbf{m}^{mig} = \mathbf{L}^T \mathbf{d}$  is the migration image, and  $S(\mathbf{m})$  is a sparseness function. For example, the sparseness function might be  $S(\mathbf{m}) = \|\mathbf{m}\|_1$  or  $S(\mathbf{m}) = \log(1 + \|\mathbf{m}\|_2^2)$ .

The solution to equation 24.5 is

$$\mathbf{m}^* = \arg \min_{\mathbf{m}} \left[ \frac{1}{2} \|\mathbf{\Gamma} \mathbf{m} - \mathbf{m}^{mig}\|_2^2 + \lambda S(\mathbf{m}) \right], \quad (24.6)$$

which can be approximated by an iterative gradient descent method:

$$\begin{aligned} m_i^{(k+1)} &= m_i^{(k)} - \alpha \left[ \mathbf{\Gamma}^T \overbrace{(\mathbf{\Gamma} \mathbf{m} - \mathbf{m}^{mig})}^{\mathbf{r}=\text{residual}} + \lambda S(\mathbf{m})' \right]_i, \\ &= m_i^{(k)} - \alpha [\mathbf{\Gamma}^T \mathbf{r} + \lambda S(\mathbf{m})']_i. \end{aligned} \quad (24.7)$$

Here,  $S(\mathbf{m})'_i$  is the derivative of the sparseness function with respect to the model parameter  $m_i$  and the step length is  $\alpha$ . Vectors and matrices are denoted by boldface lowercase and uppercase letters, respectively. If  $S(\mathbf{m}) = \|\mathbf{m}\|_1$ , the iterative solution in equation 24.7 can be recast as

$$m_i^{(k+1)} = \text{soft} \left( \left[ \mathbf{m}^{(k)} - \frac{1}{\alpha} \mathbf{\Gamma}^T (\mathbf{\Gamma} \mathbf{m}^{(k)} - \mathbf{m}^{mig}) \right]_i, \frac{\lambda}{\alpha} \right), \quad (24.8)$$

where, *soft* is the two-sided soft thresholding function (Pappyan et al., 2016) derived in Appendix 24.7 (see equation 24.22). Here,  $\mathbf{\Gamma} = \mathbf{L}^T \mathbf{L}$  is computed by solving the wave equation to get the forward modeled field and back-propagating the data by a numerical solution to the adjoint wave equation.

Equation 24.8 is similar to the forward modeling operation associated with the first layer of the neural network in Figure 24.1. That is, set  $k = 0$ ,  $\mathbf{m}^{(0)} = 0$ ,  $\alpha = 1$ , and let the input vector be the scaled residual vector  $\mathbf{r} = -(\mathbf{\Gamma} \mathbf{m}^{(0)} - \mathbf{m}^{mig}) = \mathbf{m}^{mig}$  so that the first-iterate solution can be compactly represented by

$$\mathbf{m}^{(1)} = \text{soft}(\mathbf{\Gamma}^T \mathbf{m}^{mig}, \lambda). \quad (24.9)$$

Here, the input vector  $\mathbf{r} = \mathbf{m}^{mig}$  is multiplied by the matrix  $\mathbf{\Gamma}^T$  to give  $\mathbf{z} = \mathbf{\Gamma}^T \mathbf{r}$ , and the elements of  $\mathbf{z}$  are then thresholded and shrunk to give the output  $\mathbf{m} = \text{soft}(\mathbf{z}, \lambda)$ . If we impose a positivity constraint for  $\mathbf{z}$  and a shrinkage constraint so  $\lambda$  is small, then the soft thresholding function becomes that of a one-sided threshold function, also known as the Rectified Linear Unit or ReLU function. To simplify the notation, the  $\text{soft}(\mathbf{z}, \lambda)$  function or  $\text{ReLU}(\mathbf{z})$  function is replaced by  $\sigma_\lambda(\mathbf{z})$  so that equation 24.9 is given by

$$\mathbf{m}^{(1)} = \sigma_\lambda(\mathbf{\Gamma}^T \mathbf{m}^{mig}). \quad (24.10)$$

For the ReLU function there is no shrinkage so  $\lambda = 0$ . However,  $\mathbf{\Gamma}$  in equation 24.10 is computed

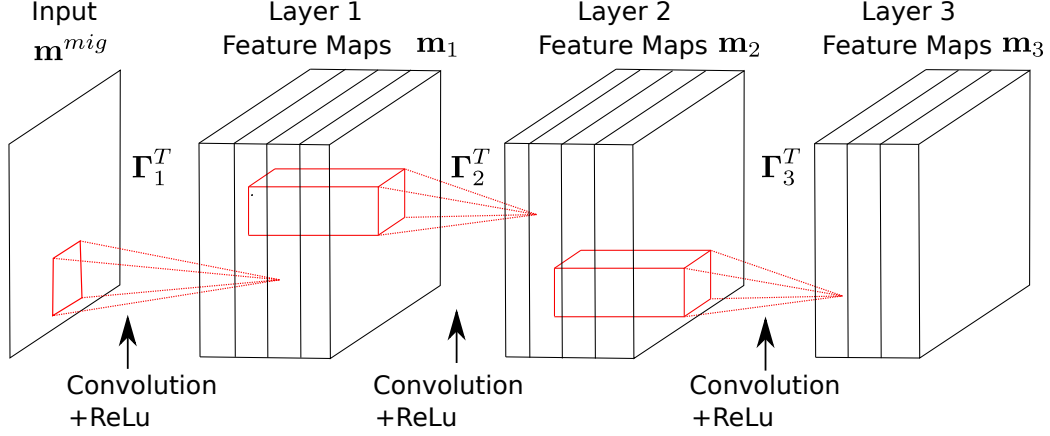


Figure 24.1: The forward modeling procedure for a multilayer CNN is equivalent to the multilayer sparse solution.

by forward and backward solutions to the wave equation. Unlike a neural network, the physics of wave propagation is included with the sparse LSM solution in equation 24.8.

### 24.2.3 Neural Network Least Squares Migration

We now propose the neural network version of SLSM that finds both  $\mathbf{\Gamma}^*$  and  $\mathbf{m}^*$  which minimize equation 24.5, which is equivalent to the convolutional sparse coding (CSC) problem. We denote the optimal solution  $\mathbf{m}$  as the neural network least squares migration (NNLSM) image. Here, we assume that the migration image  $\mathbf{m}^{mig}$  can be decomposed into components that have the form  $\mathbf{\Gamma}_1 \mathbf{m}_1$ , where  $\mathbf{m}_1$  represents a sparse quasi-reflectivity structure for the 1<sup>th</sup> CNN layer in Figure 24.1 and  $\mathbf{\Gamma}_1$  has a convolutional structure. The solution can be found by using the Alternating Direction Method of Multipliers (ADMM) method either in the Fourier domain (Heide et al., 2015) or in the space domain (Papayan et al., 2017b), which alternates between finding  $\mathbf{\Gamma}^*$  (dictionary learning problem) and then finding  $\mathbf{m}^*$  (sparse pursuit problem). The key difference between least squares sparse inversion and NNLSM is that  $\mathbf{\Gamma}$  is not computed by numerical solutions to the wave equation. Instead, the coefficients of  $\mathbf{\Gamma}$  are computed by the usual gradient-descent learning algorithm of a convolutional neural network. For this reason, we denote  $\mathbf{m}^*$  as the quasi-reflectivity distribution and  $\mathbf{\Gamma}^*$  as the quasi-migration Green's function.

Appendix 24.8 shows the general solution for NNLSM for a single-layer neural network, where the optimal  $\mathbf{\Gamma}^*$  is composed of the quasi-migration Green's functions, which are denoted as convolutional filters in the machine learning terminology (Liu and Schuster, 2018; Liu et al., 2018). Each filter is used to compute a feature map that corresponds to a sub-image of quasi-reflection coefficients in the context of LSM.

We now compute the NNLSM image for a 1D model, where we assume  $\mathbf{m}^{mig}$  is a N-dimensional vector which can be expressed as,

$$\mathbf{m}^{mig} = \sum_i^{k_0} \mathbf{m}\gamma_i * \mathbf{m}'_{1i}. \quad (24.11)$$

Here,  $\mathbf{m}\gamma_i$  is the  $i^{th}$  local filter with length of  $n_0$ ,  $\mathbf{m}'_{1i}$  is the  $i^{th}$  feature map, “\*” denotes the convolution operator and  $k_0$  is the number of the filters. Alternatively, following Figure 24.2a, equation 24.11 can be written in matrix form as  $\mathbf{m}^{mig} = \mathbf{\Gamma}_1 \mathbf{m}_1 = \mathbf{\Gamma}'_1 \mathbf{m}'_1$  (Papayan et al., 2017a,

2017b), where  $\mathbf{\Gamma}_1$  is a convolutional matrix where the columns contain the  $k_0$  filters with all of their shifts.  $\mathbf{\Gamma}'_1$  is a concatenation of banded and circulant<sup>2</sup> matrices, which is the same as  $\mathbf{\Gamma}_1$  except that the order of the columns is different.  $\mathbf{m}'_1$  is a concatenation of the feature map vectors  $\mathbf{m}'_{1i}$  for  $i = 1, 2, \dots, k_0$ .

The advantage of NNLSM is that only inexpensive matrix-vector multiplications are used and no expensive solutions to the wave equation are needed for backward and forward wavefield propagation. As will be seen later, convolutional filters that appear to be coherent noise can be excluded for denoising the migration image.

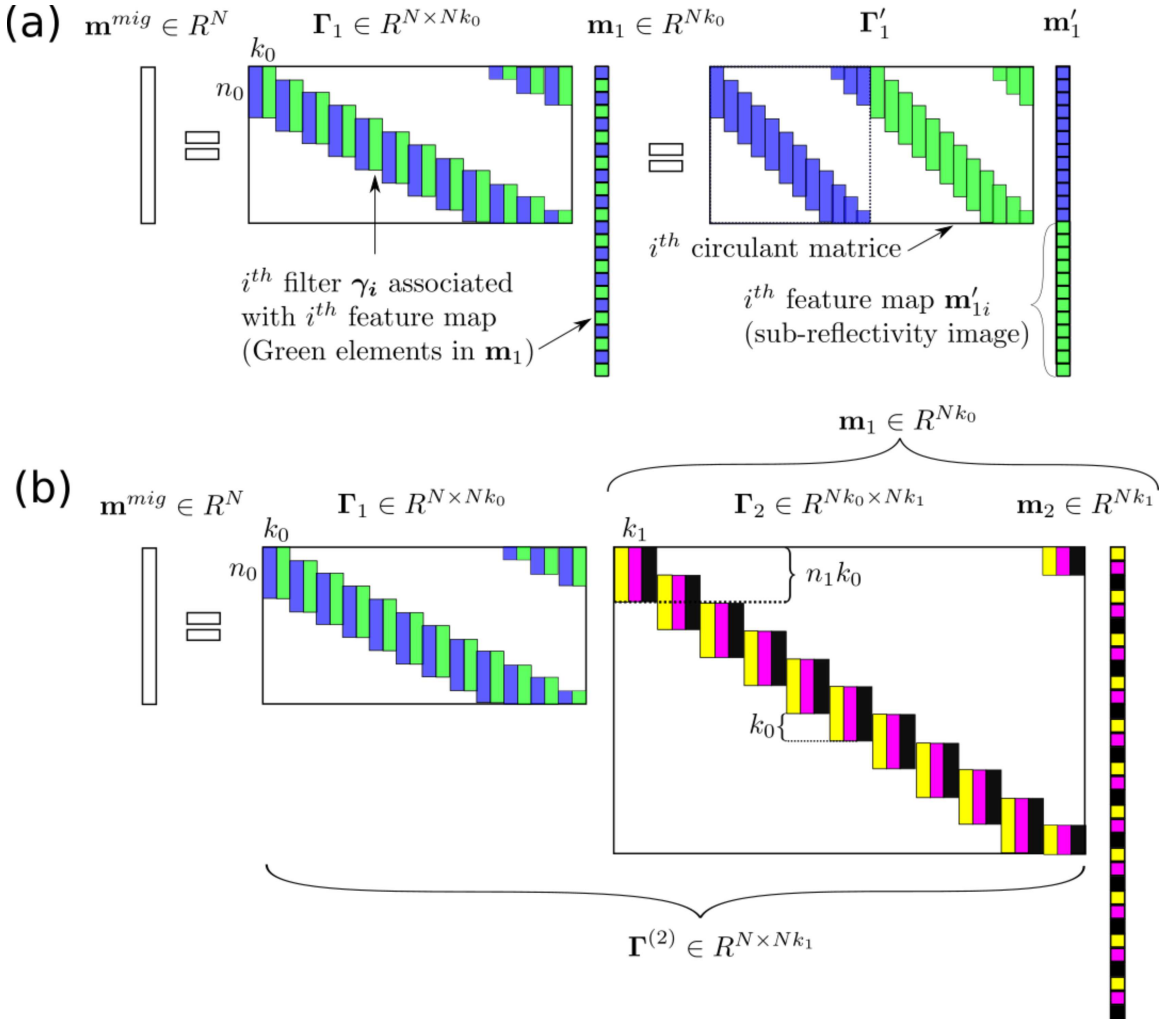


Figure 24.2: a) Single-layer NNLSM and b) multilayer NNLSM for a one-dimensional migration image  $\mathbf{m}^{mig}$ .

<sup>2</sup>We shall assume throughout this paper that boundaries are treated by a periodic continuation, which gives rise to the cyclic structure.



### 24.2.4 Multilayer Neural Network LSM

The multilayer NNLSM is a natural extension of the single-layer NNLSM. For NNLSM, the migration image  $\mathbf{m}^{mig}$  can be expressed as  $\mathbf{m}^{mig} = \mathbf{\Gamma}_1 \mathbf{m}_1$  (Figure 24.2a), where there are  $k_0$  filters in  $\mathbf{\Gamma}_1$  and  $k_0$  sub-quasi-reflectivity images in  $\mathbf{m}_1$ . Following Sulam et al. (2018), we can cascade this model by imposing a similar assumption to the sparse representation  $\mathbf{m}_1$ , i.e.,  $\mathbf{m}_1 = \mathbf{\Gamma}_2 \mathbf{m}_2$ , for a corresponding convolutional matrix  $\mathbf{\Gamma}_2$  with  $k_1$  local filters and a sparse sub-quasi-reflectivity image  $\mathbf{m}_2$ , as depicted in Figure 24.2b. In this case, the filter size is  $n_1 \times k_0$  and there are  $k_1$  sub-quasi-reflectivity images in  $\mathbf{m}_2$ .

Similar to the derivation by (Papayan et al., 2017a) and (Sulam et al., 2018), the multilayer neural network LSM problem is defined as the following.

$$\begin{aligned}
 \text{Find: } & \mathbf{m}_i, \mathbf{\Gamma}_i \quad \text{such that} \\
 \mathbf{m}_1^* &= \arg \min_{\mathbf{m}_1, \mathbf{\Gamma}_1} \left[ \frac{1}{2} \|\mathbf{\Gamma}_1 \mathbf{m}_1 - \mathbf{m}^{mig}\|_2^2 + \lambda S(\mathbf{m}_1) \right], \\
 \mathbf{m}_2^* &= \arg \min_{\mathbf{m}_2, \mathbf{\Gamma}_2} \left[ \frac{1}{2} \|\mathbf{\Gamma}_2 \mathbf{m}_2 - \mathbf{m}_1^*\|_2^2 + \lambda S(\mathbf{m}_2) \right], \\
 & \vdots \\
 \mathbf{m}_N^* &= \arg \min_{\mathbf{m}_N, \mathbf{\Gamma}_N} \left[ \frac{1}{2} \|\mathbf{\Gamma}_N \mathbf{m}_N - \mathbf{m}_{N-1}^*\|_2^2 + \lambda S(\mathbf{m}_N) \right],
 \end{aligned} \tag{24.12}$$

where  $\mathbf{\Gamma}_i$  is the  $i$ th Hessian matrix in the  $i$ th layer. The first iterate solution to the above system of equations can be cast in a form similar to equation 24.10, except we have

$$\mathbf{m}_N^* \approx \sigma_\lambda (\mathbf{\Gamma}_N^T \sigma_\lambda (\mathbf{\Gamma}_{N-1}^T (\dots \sigma_\lambda (\mathbf{\Gamma}_1^T \mathbf{m}^{mig}) \dots)), \tag{24.13}$$

which is a repeated concatenation of the two operations of a multilayered neural network: matrix-vector multiplication followed by a thresholding operation. In all cases, we use a convolutional neural network where different filters are applied to the input from the previous layer to give feature maps associated with the next layer, as shown in Figure 24.1.

For a perfect prediction of the migration image,  $\mathbf{m}^{mig}$  can also be approximated as  $\mathbf{m}^{mig} = \mathbf{\Gamma}_1 \mathbf{\Gamma}_2 \dots \mathbf{\Gamma}_N \mathbf{m}_N$ . We refer to  $\mathbf{\Gamma}^{(i)}$  as the effective filter at the  $i$ th level,

$$\mathbf{\Gamma}^{(i)} = \mathbf{\Gamma}_1 \mathbf{\Gamma}_2 \dots \mathbf{\Gamma}_i, \tag{24.14}$$

so that

$$\mathbf{m}^{mig} = \mathbf{\Gamma}^{(i)} \mathbf{m}_i. \tag{24.15}$$

The next section tests the effectiveness of NNLSM on both synthetic data and field data.

## 24.3 Numerical Results

We now present numerical simulations of NNLSM. Instead of only determining the optimal reflectivity  $\mathbf{m}$  as computed by SLSM, the NNLSM method computes both the quasi-reflectivity  $\mathbf{m}$  and the elements of the Hessian matrix  $\mathbf{\Gamma} = \mathbf{L}^T \mathbf{L}$ . Each block of  $\mathbf{\Gamma}$  is considered to be the *segment response function* (SSF) of the migration operator rather than the *point spread function* (PSF). If the actual Green's functions are used to construct  $\mathbf{\Gamma}$  then each column of the Hessian matrix is the point scatterer response of the migration operator (Schuster and Hu, 2000). In contrast, the NNLSM Hessian is composed of blocks, where each block is the segment scatterer response of the migration operator. An example will be shown later where a reflections from a segment of the reflector are migrated to give the migration segment response of the migration operator. The computational cost

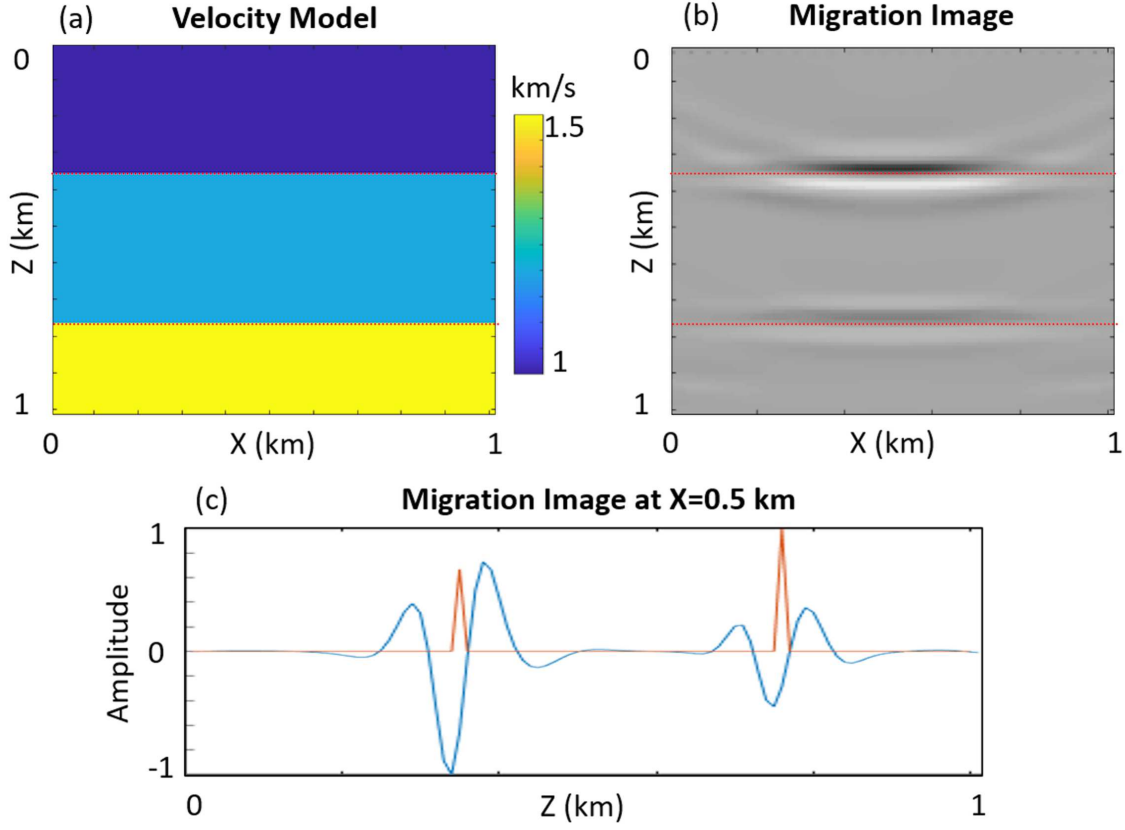


Figure 24.3: a) Three-layer velocity model, b) RTM image, and c) migration image (blue curve) at  $X=0.5$  km, where the red curve is the normalized reflectivity model.

for computing SSF's is several orders of magnitude less than that for PSFs because no solutions to the wave equation are needed. The penalty, however, is that the resulting solution  $\mathbf{m}$  is not the true reflectivity, but a sparse representation of it we denote as the quasi-reflectivity distribution.

Using the terminology of neural networks, we can also denote the sparse sub-quasi-reflectivity images as feature maps. Each block in  $\mathbf{\Gamma}$  will be denoted as a filter. Therefore the vector output of  $\mathbf{\Gamma}\mathbf{m}$  can be interpreted as a weighted sum of filter vectors  $\mathbf{m}\gamma_i$  weighted by the coefficients in  $\mathbf{m}$ , where  $\mathbf{m}\gamma_i$  is the  $i^{th}$  column vector of  $\mathbf{\Gamma}$ .

### 24.3.1 Three-layer Velocity Model

The interpretation of feature maps and filters can be understood by computing them for the Figure 24.3a model. The grid size of the model is  $101 \times 101$ , and the grid interval is 10 m in both the x and z directions. There are 26 shots evenly spaced at a distance of 40 m on the surface, and each shot is recorded by 101 receivers with a sampling interval of 10 m. Figure 24.3b show the reverse time migration (RTM) image.

The first test is for a 1D model where we extract the image located at  $X=0.5$  km, which is displayed as the blue curve in Figure 24.3c. The red curve in Figure 24.3c is the reflectivity model. Assume that there is only one filter in  $\mathbf{\Gamma}$  and it extends over the depth of 400 m (41 grid points). We now compute the NNLSM image by finding the optimal  $\mathbf{m}$  and  $\mathbf{\Gamma}$  by the two-step iterative procedure

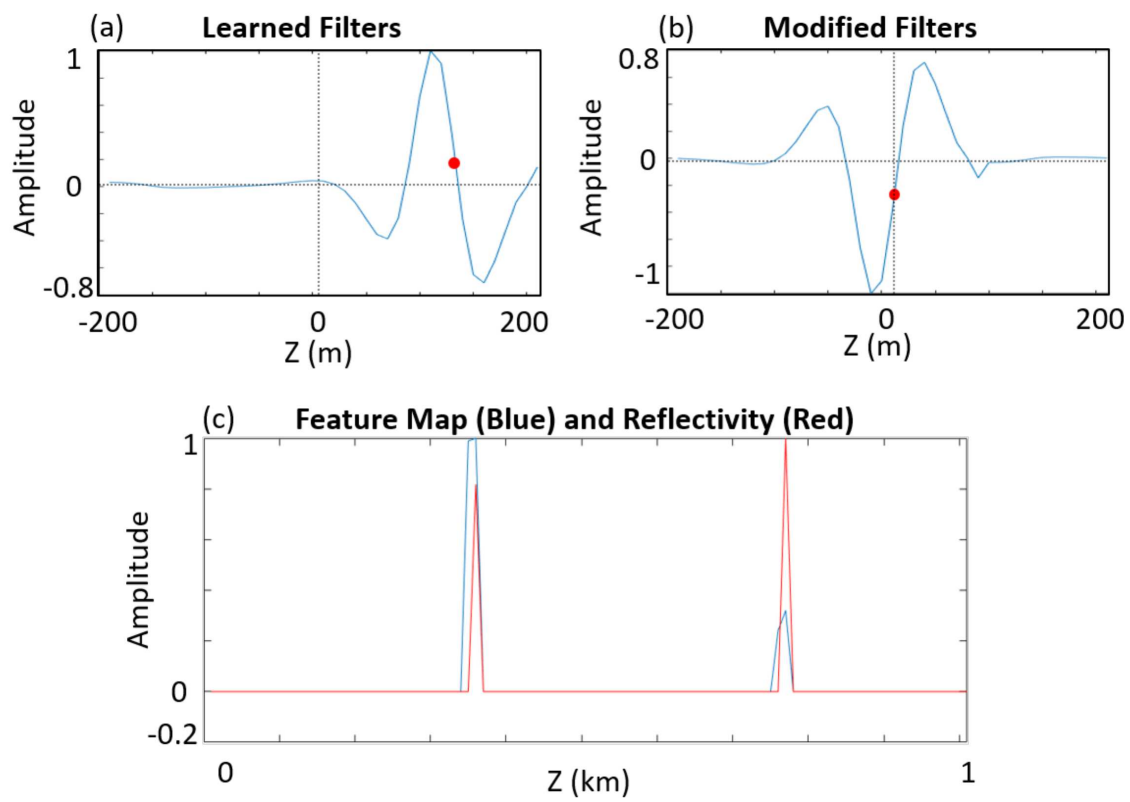


Figure 24.4: a) Learned filter, b) modified filter, and c) feature map (blue) and reflectivity model (red).

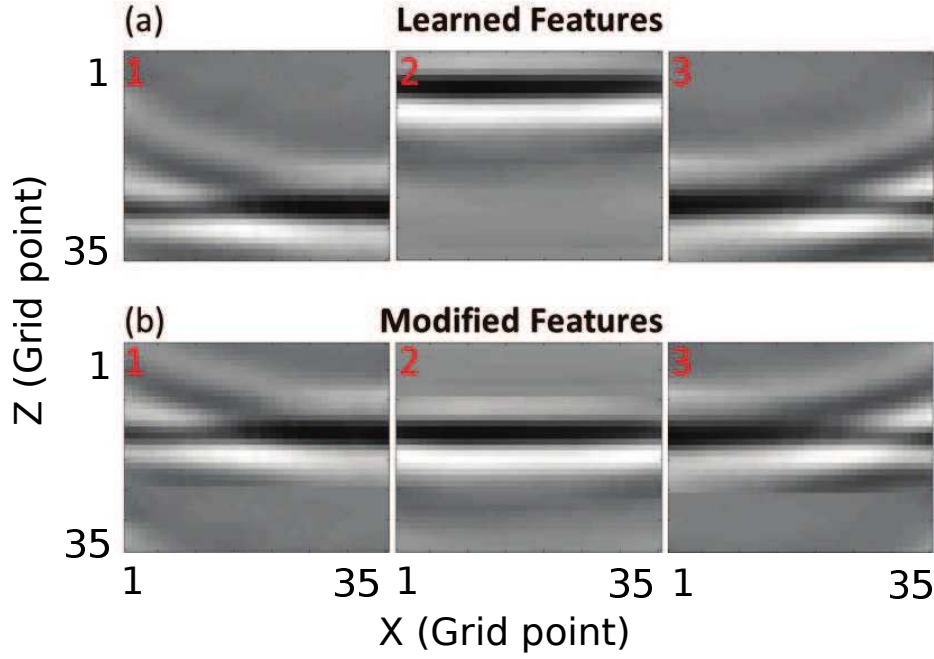


Figure 24.5: a) Learned and b) modified features.

denoted as the alternating descent method (see Liu et al. (2018) and Liu and Schuster (2018)). The computed filter  $\mathbf{m}\gamma_i$  is shown in Figure 24.4a where the phase of the filter  $\gamma_i$  is nonzero. If we use a filter with a non-zero time lag to calculate its feature map  $\mathbf{m}$ , the phases of the feature map and the true reflectivity  $\mathbf{m}$  will be different. So, we need to modify the time lag and polarity of the basis function  $\tilde{\Gamma}_i$ . The modified basis function is shown in Figure 24.4b, and its coefficients are displayed as the blue curve in Figure 24.4c. Compared with the true reflectivity  $\mathbf{m}$  (red curve in Figure 24.4), the feature map can give the correct positions but also give the wrong values of the reflectivity distribution.

Next, we perform a 2D test where the input is the 2D migration image in Figure 24.3b. Three  $35 \times 35$  (grid point) filters are learned (see Figure 24.5a). The modified filters are shown in Figure 24.5b. Appendix 24.9 describes how we align the filters by using the cross-correlation method. The feature maps of these three filters are displayed in Figures 24.6a-24.6c. Figure 24.6d shows the sum of these three feature maps. It is evident that the stacked feature maps can estimate the correct locations of the reflectivity spikes.

### 24.3.2 SEG/EAGE Salt Model

The multilayer NNLSM procedure (see equation 24.12) is now applied to the migration image associated with the 2D SEG/EAGE salt velocity model in Figure 24.7a. The grid size of the model is 101 grid points in both the z- and x-directions. The grid interval is 40 m in the x-direction and 20 m in the z-direction. Figure 24.7b shows the reverse time migration (RTM) image. The multilayer NNLSM consists of three convolutional layers: the first one contains 15 basis functions, i.e., filters, of size  $11 \times 11$  grid points, the second one consists of 15 basis functions with dimensions  $11 \times 11 \times 15$ , and the last one contains contains 15 basis function of dimensions  $11 \times 11 \times 15$ . Equation 24.12 is solved for both  $\mathbf{m}_i$  and  $\tilde{\Gamma}_i$  ( $i \in 1, 2, 3$ ) by the two-step iterative procedure denoted as the alternating descent method. The multilayered structure is shown in Figure 24.8, where the black

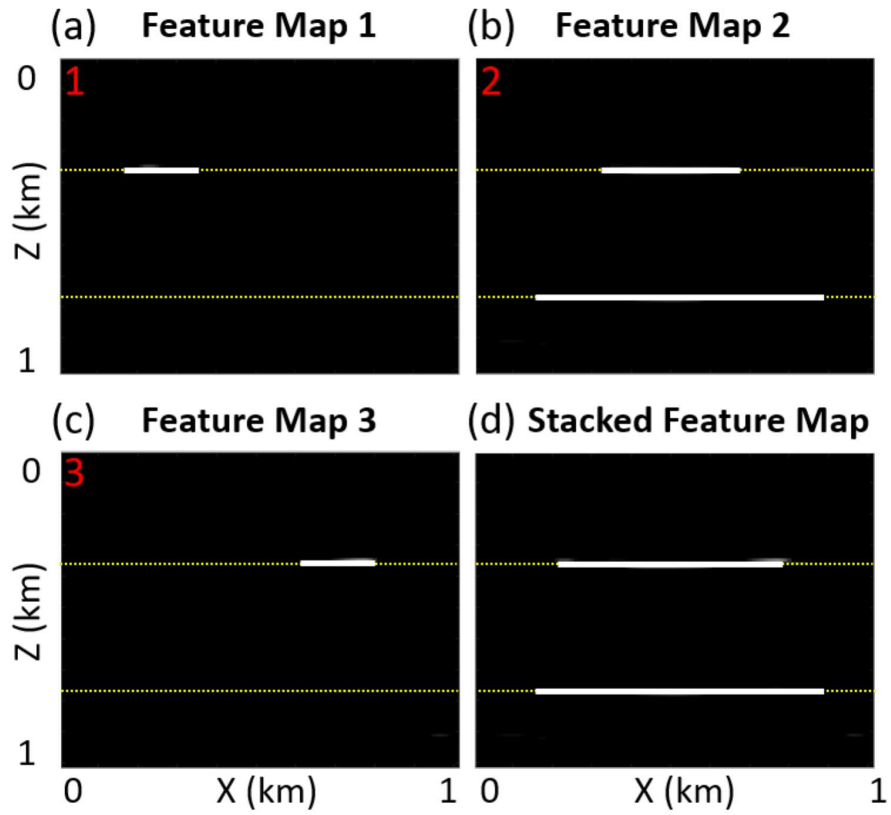


Figure 24.6: Feature maps for the features a) 1, b) 2, and c) 3 shown in Figure 24.5. The stacked feature map is shown in d). Here, the white lines show the locations of non-zero points and the yellow lines indicate the locations of the reflectivity distributions.

dots in  $\mathbf{m}_i$  represent the nonzero values of the quasi-reflectivity distribution. The effective basis functions computed for these layers are shown in Figures 24.7c-24.7e, where the yellow, red and green boxes indicate the sizes of the effective basis functions, which can be considered as quasi-migration Green's functions. It indicates that the basis functions of the first layer  $\Gamma_1$  contain very simple small-dimensional edges, which are called "atoms" by Sulam et al. (2018). The non-zeros of the second group of basis functions  $\Gamma_2$  combine a few atoms from  $\Gamma_1$  to create slightly more complex edges, junctions and corners in the effective basis functions in  $\Gamma^{(2)}$ . Lastly,  $\Gamma_3$  combines atoms from  $\Gamma^{(2)}$  in order to reconstruct the more complex parts of the migration image. The corresponding stacked coefficient images, also known as feature maps, are shown in Figures 24.7f-24.7h, which give the quasi-reflectivity distributions. The reconstructed migration images are shown in Figures 24.7i-24.7k.

For comparison, we computed the standard LSM image using the deblurring method described in Chen et al. (2017) and Chen et al. (2019). Here, the deblurring filter size is 17x17 grid points (black boxes in Figure 24.9) and computed for a 50x50 grid (red boxes in Figure 24.9) of evenly spaced point scatterers with the same migration velocity model as used for the data migration in Figure 24.7a. The standard LSM images for the first and 50<sup>th</sup> iterations are shown in Figures 24.10b and 24.10c, respectively, next to the NNLSM image in Figure 24.10d. It is clear that the NNLSM image is better resolved than the LSM image, although there are discontinuities in some of the NNLSM interfaces not seen in the LSM image. Some of the detailed geology is lost in the LSM image as seen in the wiggly interface in the red-rectangular area of Figure 24.10. The practical application of the NNLSM image is that it might serve a super-resolved attribute image that can be combined with other attributes to delineate geology. For example, combining the depth-slice of the NNLSM image with a spectral decomposition image can help delineate the lithological edges of meandering channels.

NNLSM can filter out random and coherent noises in the migration image after reconstructing the migration image by eliminating the noisy learned basis functions and their coefficients in the NNLSM image. For example, Figure 24.11a shows the RTM image with a sparse acquisition geometry so that the image contains a strong acquisition footprint. The reconstructed migration image in Figure 24.11b shows significant mitigation of this noise in Figure 24.11a. However, the migration swing noise is still prominent near the red arrows in Figure 24.11b. Such noise is reconstructed from the noisy basis function shown in Figure 24.12a and the coefficients in Figure 24.12b. Figure 24.12c is the image reconstructed by correlating the basis function in Figure 24.12a with the coefficients in Figure 24.12b. After filtering out the basis functions from noise, the reconstructed image is shown in Figure 24.11c, which is free from aliasing noise at the locations indicated by the red arrows.

### 24.3.3 North Sea Data

We apply the NNLSM method to field data collected in the North Sea (Schroot and Schuttenhelm, 2003), where the time migration image is shown in Figure 24.13a. The time axis is gridded with 213 evenly-spaced points and there are 301 grid points along the x-axis. We compute 21 13-by-5 (grid point) convolutional basis functions, i.e. filters  $\gamma_i$  for ( $i = 1, 2, \dots, 21$ ), by the NNLSM procedure (see Figure 24.13b). These filters approximate the dip-filtered migration Green's functions, and the basis function is marked as the yellow boxes in Figure 24.13a and 24.13b. The stacked feature maps (quasi-reflectivity distribution) are displayed in Figure 24.13c. It is evident that the stacked feature maps can provide a high-resolution migration image. After reconstruction from the learned filters and feature maps, the migration image is shown in Figure 24.13d with less noise.

Finally, we apply NNLSM to a time slice of the migration image, which is shown in Figure 24.14a, and the image size is 301 by 301 gridpoints. Figure 24.14b shows the 21 13x5 filters estimated by the NNLSM procedure. The stacked feature map is displayed in Figure 24.14c, which may be used as a super-resolution attribute image for high-resolution delineation of geologic bodies. The reconstructed migration image is shown in Figure 24.14d and we can see there is less noise.

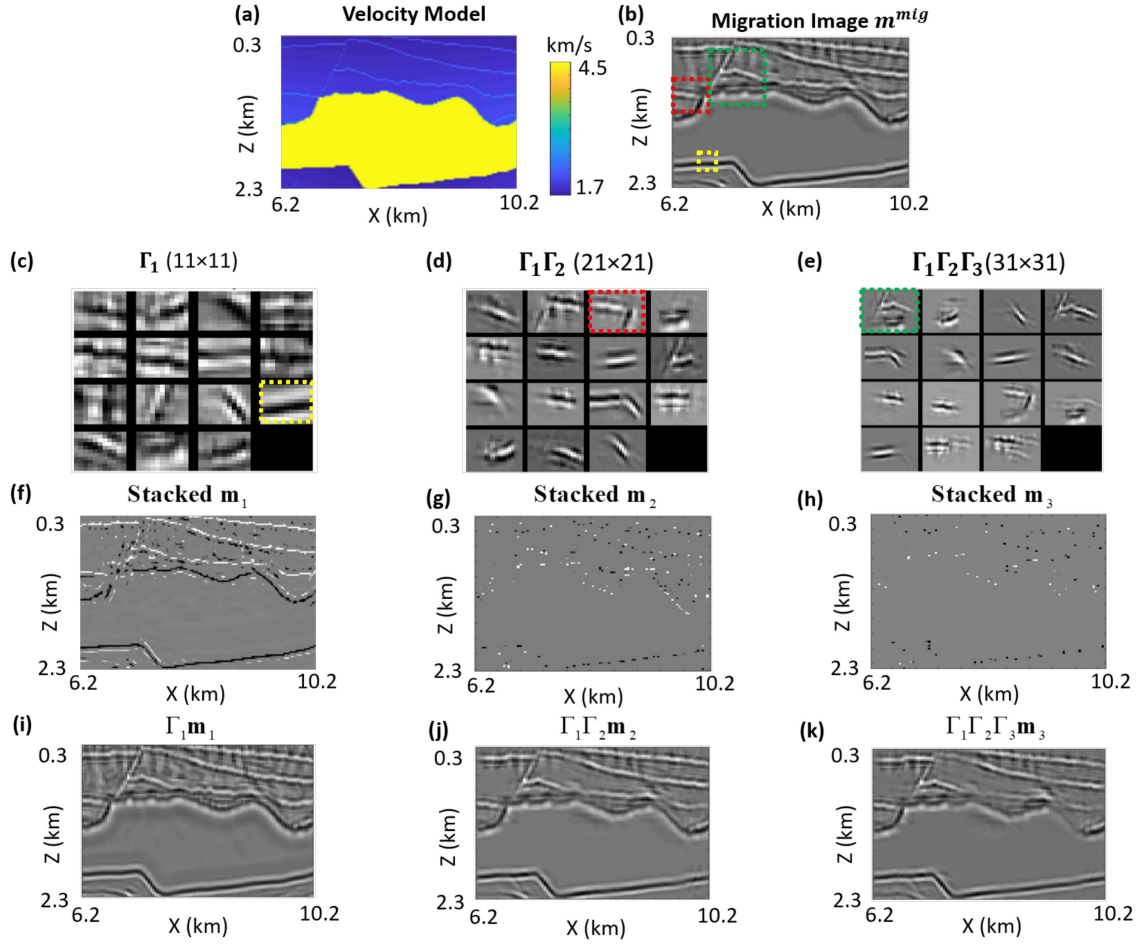


Figure 24.7: a) 2D SEG/EAGE salt model, b) RTM image, c)-(e) learned effective filters  $\Gamma^{(1)}$ ,  $\Gamma^{(2)}$  and  $\Gamma^{(3)}$ , (f)-(h) stacked quasi-reflectivity coefficients for  $\mathbf{m}_1$ ,  $\mathbf{m}_2$  and  $\mathbf{m}_3$ , (i)-(k) reconstructed migration images  $\Gamma^{(1)}\mathbf{m}_1$ ,  $\Gamma^{(2)}\mathbf{m}_2$  and  $\Gamma^{(3)}\mathbf{m}_3$ .

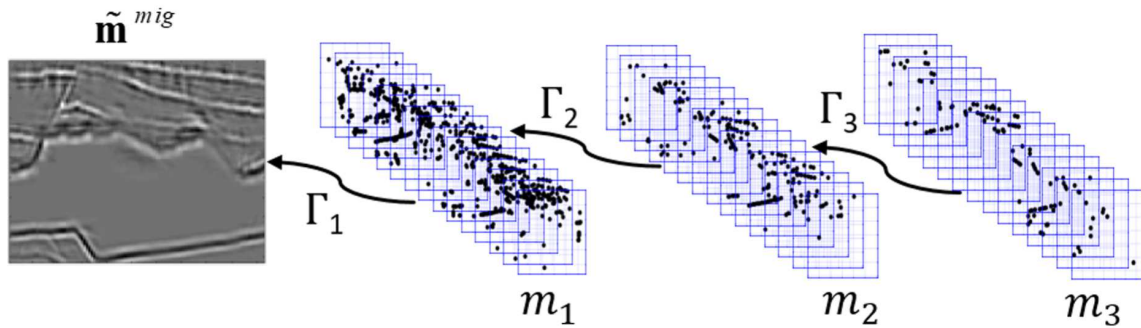


Figure 24.8: Decomposition of the migration image of the 2D SEG/EAGE salt model in terms of the multi-layer filter  $\Gamma_i$  and pre-stacked quasi-reflectivity coefficient  $\mathbf{m}_i$ , where the black dots in  $\mathbf{m}_i$  represent the nonzero reflectivity values.

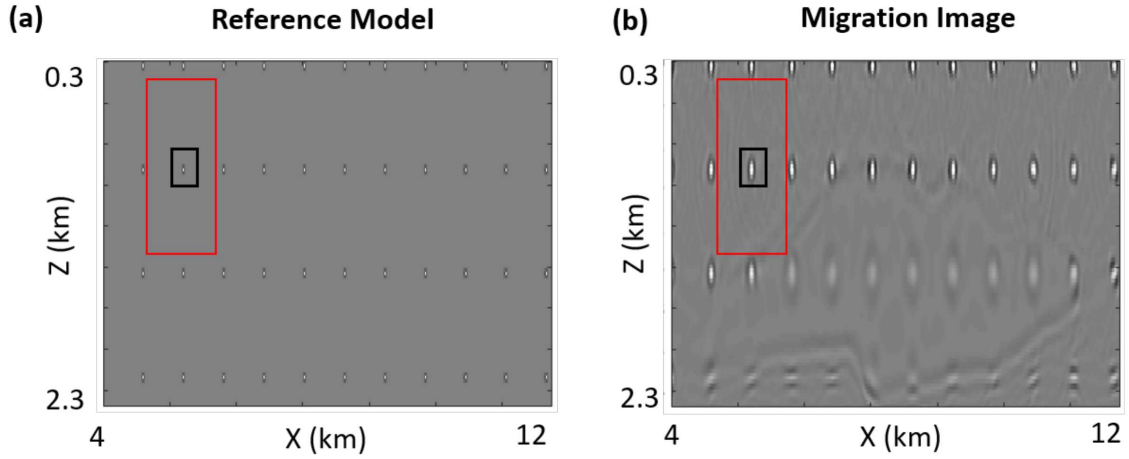


Figure 24.9: a) Reference model and b) its migration image for the standard deblurring LSM method.

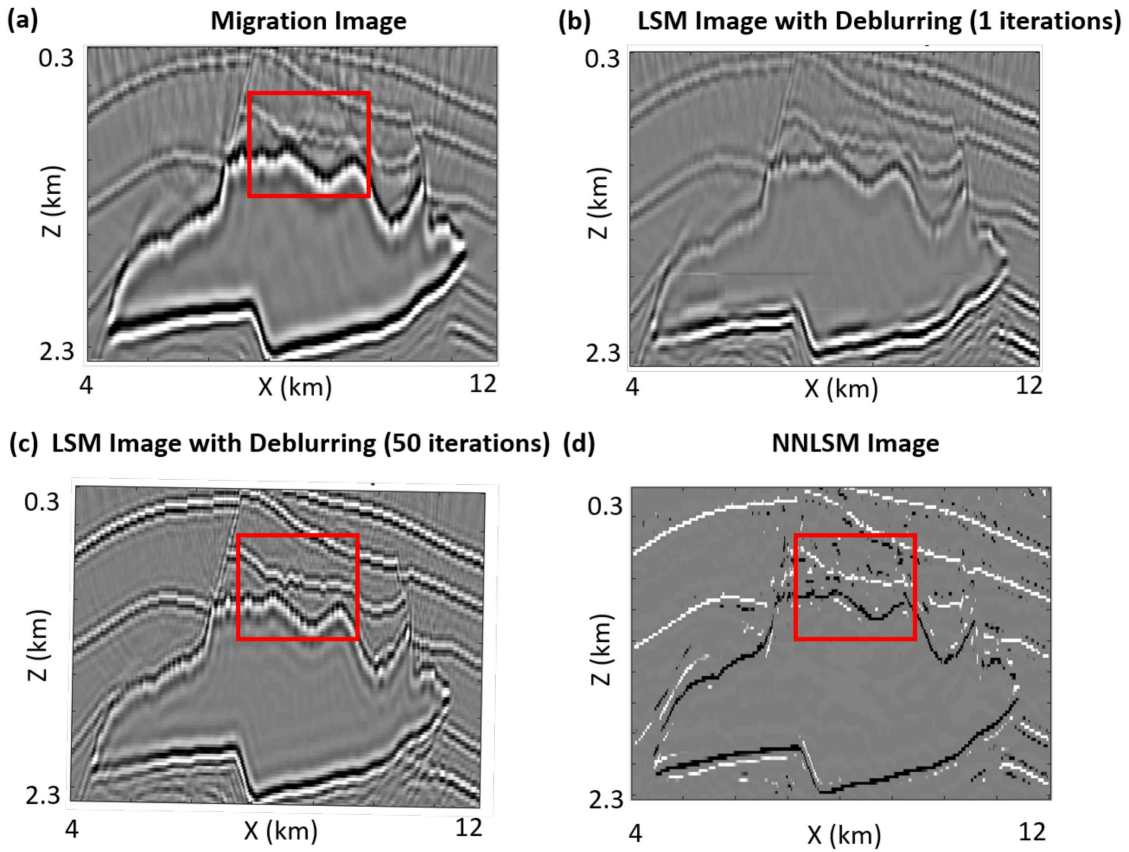


Figure 24.10: a) RTM image, b) the first and c) 50<sup>th</sup> iteration results by LSM with deblurring, and d) NNLSM image.



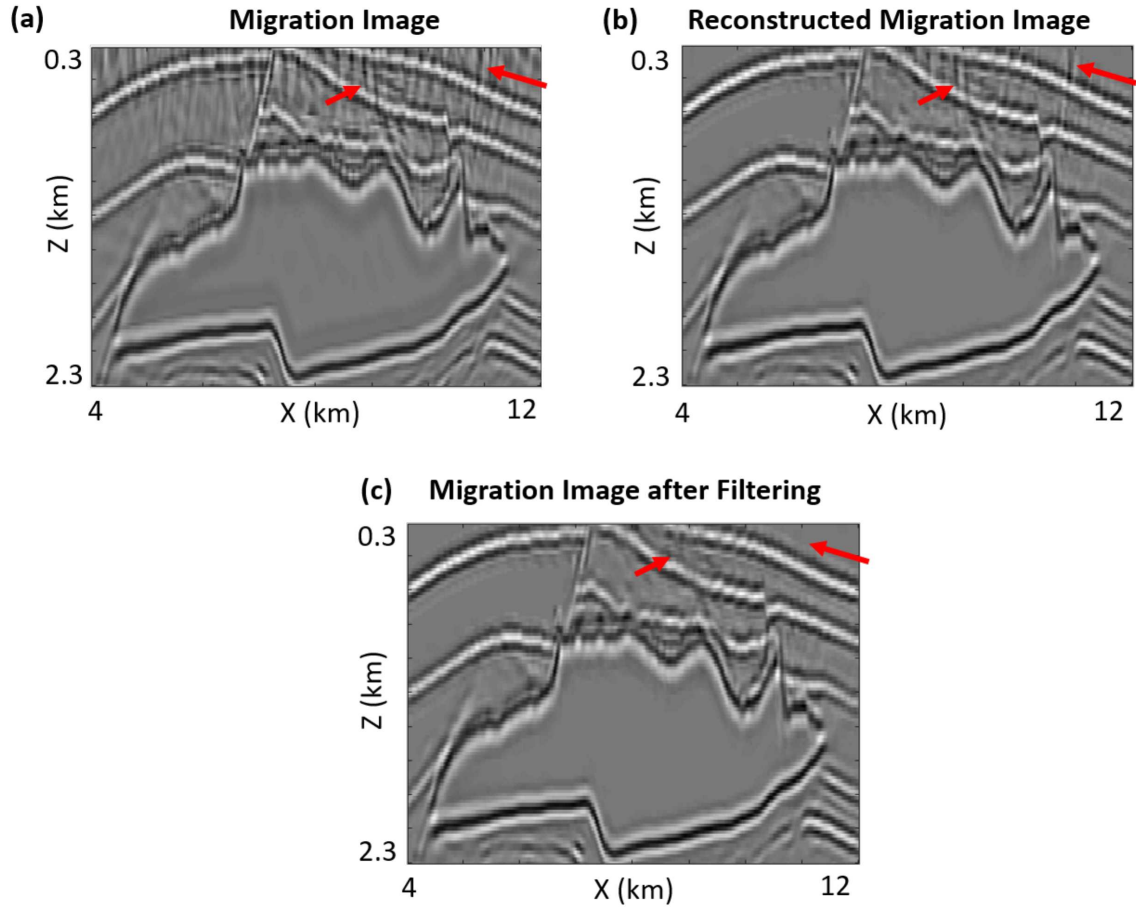


Figure 24.11: a) RTM image, b) reconstructed image with all basis functions, and c) reconstructed image with selected basis functions.

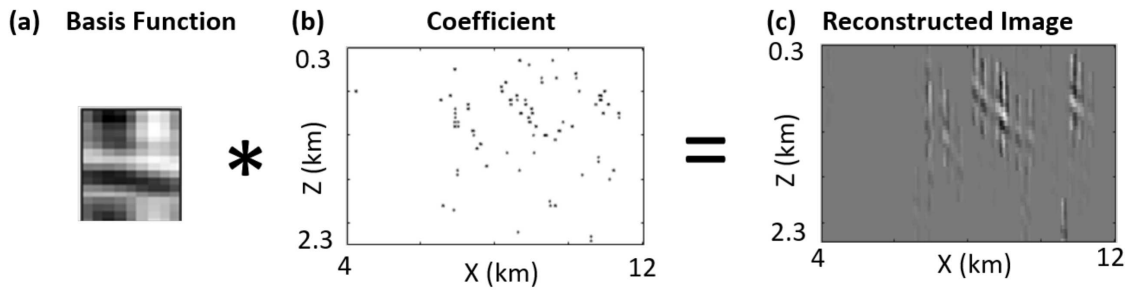


Figure 24.12: a) Basis function from noise, b) the coefficients, and c) reconstructed image.

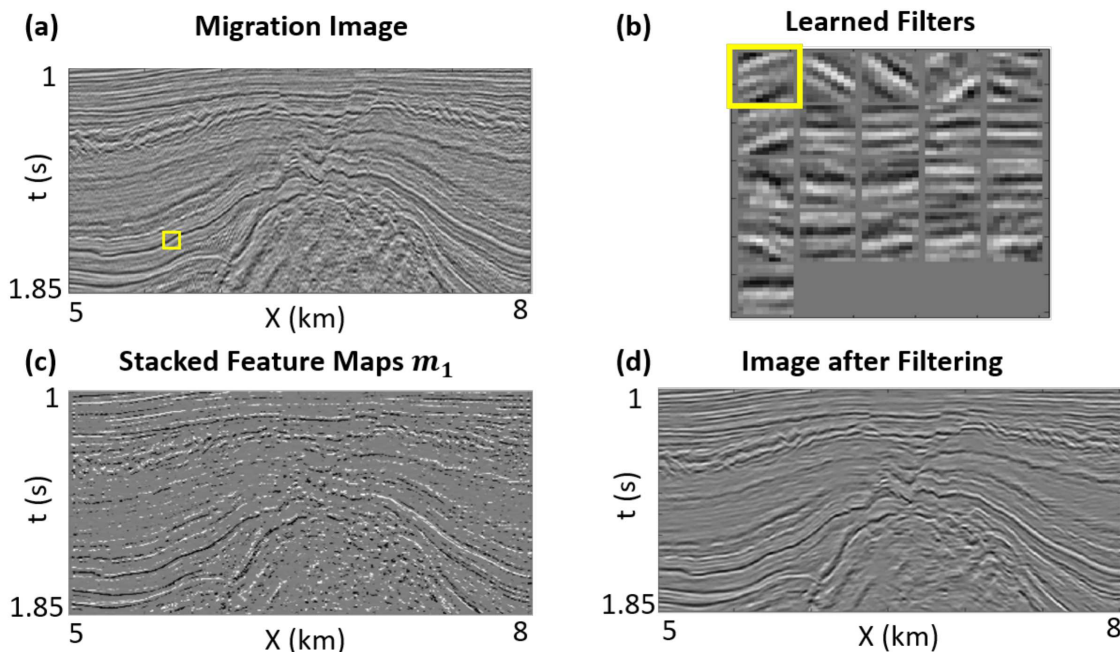


Figure 24.13: a) Migration image computed from the F3 offshore block data, b) learned filters, c) stacked feature maps and d) migration image after filtering.

## 24.4 Discussion

The forward modeling for a multilayered neural network is shown to be equivalent to a single-iterate solution of a multilayered LSM problem. This assumes positivity and shrinkage constraints on the soft thresholding operation, so it reduces to the ReLU operation. This equivalence relates the physics of seismic imaging to architectural features in the neural network.

- The size of the filters in the first layer should be about the same size as 1-2 wavelengths for the depths of interest, which are validated by numerical experiments. In this case, the filter is interpreted as an approximation to the migration Green's function, except it is that for a reflecting segment. Thus, we interpret the approximate migration Green's function as a migration segment spread function (SSF) rather than a migration point spread function. Sulam et al. (2018) classifies each feature in the first layer as an atom which takes on the role of a SSF.
- The output of the first layer provides the small scale, i.e., high-wavenumber, features associated with the input data. For an input migration image, the feature maps of the first layer resemble sub-quasi-reflectivity maps of the subsurface. Adding the sub-quasi-reflectivity maps together gives a close approximation to the actual reflectivity model as shown in Figures 24.6d and 24.7f.
- The output of the second layer is a weighted sum of the first-layer features, which create sparser feature maps. Sulam et al. (2018) classifies the concatenation of the filters from the first and second layers as molecules (see equation 24.14). In the migration problem, the resulting filters are SSFs for even larger segments of the original reflector boundaries. The feature maps of the third layer are a weighted sum of the second layer's features to produce

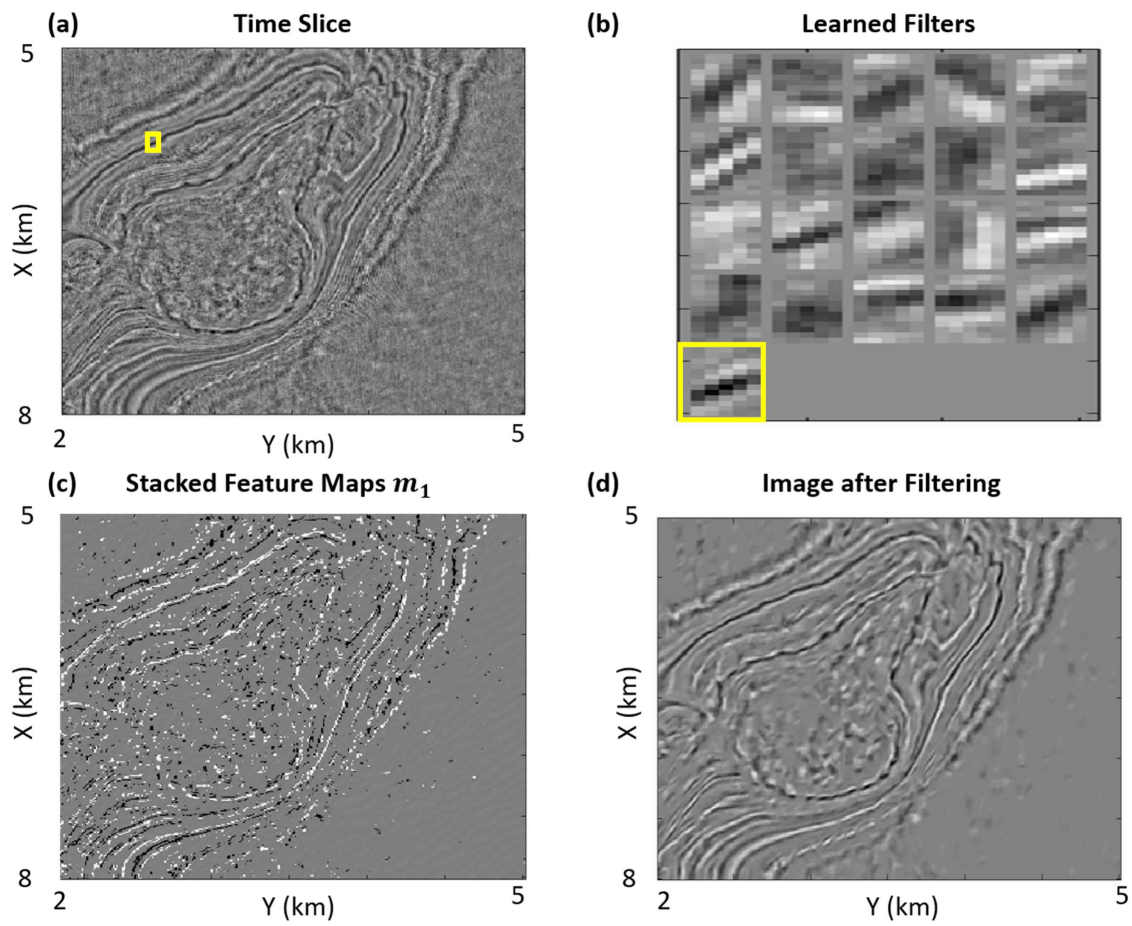
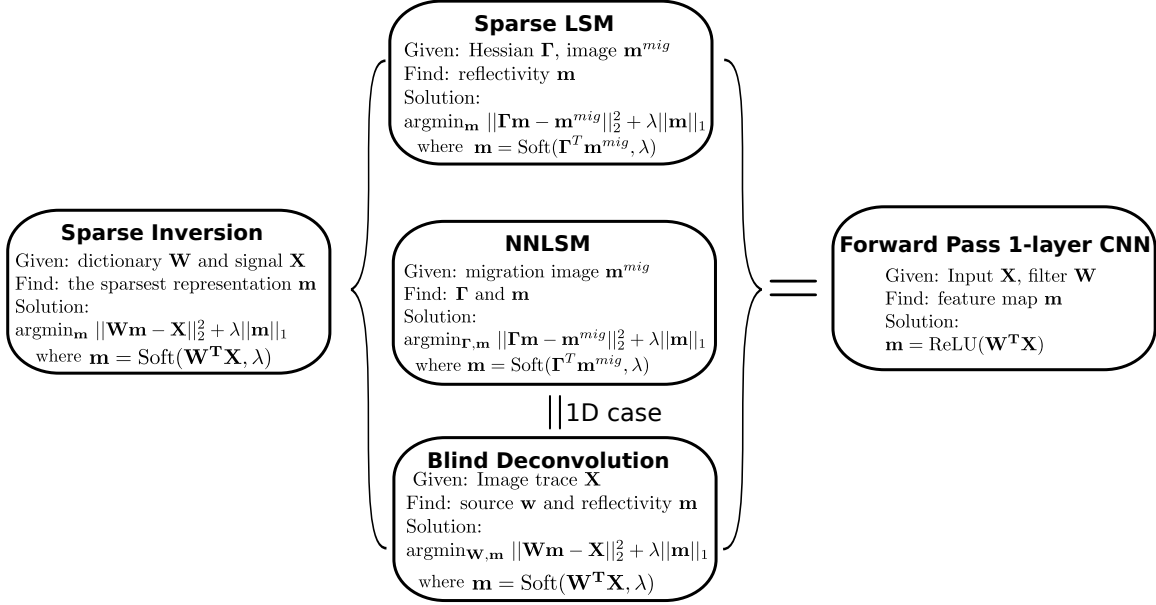


Figure 24.14: a) Time slice of the migration image computed from the F3 offshore block data, b) learned filters, c) stacked feature maps and d) migration image after filtering.



\*  $\mathbf{W}$  is a convolutional matrix of  $\mathbf{w}$  in Blind Deconvolution

Figure 24.15: The relationship between sparse inversion and CNN, where the sparse inversion methods include sparse LSM, NNLSM, and blind deconvolution.

even the sparsest feature maps. For migration, the final feature maps are very sparse while the concatenated filters are associated with large-scale features of the migration image.

- The computational cost of computing NNLSM images is significantly less than that for LSM images because no solutions of the wave equation are needed. For example, we consider the 2D FDTD forward modeling of the acoustic equation with an eighth-order scheme in space and a second-order scheme in time, and its computational complexity is  $O(n_t n^2)$  for one shot, where  $n$  is the number of grid points in one direction and  $n_t$  is the number of the time steps. According to Plessix (2007),  $n_t$  is approximate 30  $n$  to satisfy the stability condition and to make sure there is enough recoding time when  $v_{max}/v_{min} \simeq 3$ , where  $v_{max}$  and  $v_{min}$  are the maximum and minimum velocities, respectively. So, The complexity of 2D FDTD forward modeling of the acoustic equation is  $O(n_s n^3)$  where  $n_s$  is the number of the shots. The complexity of LSRTM is  $O(6n_s N^{iter} n^3)$  (Schuster, 2017), where  $N^{iter}$  is the iteration number. For NNLSM, the complexity is  $O(N^{iter} n^2 \log n)$  (Heide et al., 2015) and can be reduced to  $O(N^{iter} n^2)$  if a local block coordinate-descent algorithm is used (Zisselman et al., 2019).

The 1D NNLSM can be interpreted as a blind deconvolution (BD) problem in seismic data processing (Kaaresen and Taxt, 1998; Bharadwaj et al., 2018). It can be seen in Figure 24.5 that the filter of NNLSM is the source wavelet of BD and the coefficients of NNLSM are the quasi-reflectivity coefficients of BD. However, NNLSM can have more than one filter and the filters can be 2D or 3D filters. We show the relationship between BD, sparse LSM, NNLSM, and CNN in Figure 24.15.

NNLSM is an unsupervised learning method. Compared to a supervised learning method, it does not heavily depend on the huge amount of training data. However, it may need human intervention for inspecting the quality of the quasi-reflectivity images and we need to align the filters to get more consistent quasi-reflectivity.

In Figure 24.14, we apply 2D NNLSM to a time slice of the 3D North Sea Data. The 2D NNLSM method can be extended to a 3D implementation to learn a set of 3D filters. Incorporating the third dimension of information from 3D filters will likely lead to a better continuity of the reflectors in Figure 24.14c. However, the computational cost will increase by more than a multiplicative factor of  $n$ .

## 24.5 Conclusions

Neural network least squares migration finds the optimal quasi-reflectivity distribution and quasi-migration-Green's functions that minimize a sum of migration misfit and sparsity regularization functions. The advantages of NNLSM over standard LSM are that its computational cost is significantly less than that for LSM and it can be used for filtering both coherent and incoherent noise in migration images. A practical application of the NNLSM image is as an attribute map that provides super-resolution imaging of the layer interfaces. This attribute image can be combined with other attributes to delineate both structure and lithology in depth/time slices of migration images. Its disadvantage is that the NNLSM quasi-reflectivity image is only an approximation to the actual reflectivity distribution.

A contribution of our work is that we show that the filters and feature maps of a multilayered CNN are analogous to the migration Green's functions and reflectivity distributions. For the first time we now have a physical interpretation of the filters and feature maps in deep CNN in terms of the operators for seismic imaging. Such an understanding has the potential to lead to better architecture design of the network and extend its application to waveform inversion. In answer to Donoho's plea for more rigor, NNLSM represents a step forward in establishing the mathematical foundation of CNN in the context of least squares migration.

## 24.6 Appendix: Migration Green's Function

Schuster and Hu (2000) show that the poststack migration (Yilmaz, 2001) image  $m(\mathbf{x})^{mig}$  in the frequency domain is computed by weighting each reflectivity value  $m(\mathbf{z})$  by  $\Gamma(\mathbf{x}|\mathbf{z})$  and integrating over the model-space coordinates  $\mathbf{z}$ :

$$\text{for } \mathbf{x} \in D_{model} : \quad m(\mathbf{x}) = \eta \int_{D_{model}} d\mathbf{z} \overbrace{\int_{\mathbf{y} \in D_{data}} d\mathbf{y} \omega^4 G(\mathbf{x}|\mathbf{y})^{2*} G(\mathbf{y}|\mathbf{z})^2 m(\mathbf{z})}^{\Gamma(\mathbf{x}|\mathbf{z})}, \quad (24.16)$$

where  $\eta$  represents terms such as the frequency variable raised to the 4th power. The migration Green's function  $\Gamma(\mathbf{x}|\mathbf{z})$  is given by

$$\text{for } \mathbf{x}, \mathbf{z} \in D_{model} : \quad \Gamma(\mathbf{x}|\mathbf{z}) = \eta \int_{\mathbf{y} \in D_{data}} d\mathbf{y} \omega^4 G(\mathbf{x}|\mathbf{y})^{2*} G(\mathbf{y}|\mathbf{z})^2. \quad (24.17)$$

Here we implicitly assume a normalized source wavelet in the frequency domain, and  $D_{model}$  and  $D_{data}$  represent the sets of coordinates in, respectively, the model and data spaces. The term  $G(\mathbf{x}'|\mathbf{x}) = e^{i\omega\tau_{xx'}}/||\mathbf{x} - \mathbf{x}'||$  is the Green's function for a source at  $\mathbf{x}$  and a receiver at  $\mathbf{x}'$  in a smoothly varying medium<sup>3</sup>. The traveltime  $\tau_{xx'}$  is for a direct arrival to propagate from  $\mathbf{x}$  to  $\mathbf{x}'$ .

---

<sup>3</sup>If the source and receiver are coincident at  $\mathbf{x}$  then the zero-offset trace is represented by the squared Green's function  $G(\mathbf{x}|\mathbf{x})^2$ .

The physical interpretation of the kernel  $\Gamma(\mathbf{x}'|\mathbf{x})$  is that it is the migration operator's<sup>4</sup> response at  $\mathbf{x}'$  to a point scatterer at  $\mathbf{x}$ , otherwise known as the MGF or the migration Green's function (Schuster and Hu, 2000). It is analogous to the point spread function (PSF) of an optical lens for a point light source at  $\mathbf{x}$  in front of the lens and its optical image at  $\mathbf{x}'$  behind the lens on the image plane. In discrete form, the modeling term  $[\mathbf{\Gamma m}]_i$  in equation 24.16 can be expressed as

$$[\mathbf{\Gamma m}]_i = \sum_j \Gamma(\mathbf{x}_i|\mathbf{z}_j)m_j. \quad (24.18)$$

with the physical interpretation that  $[\mathbf{\Gamma m}]_i$  is the migration Green's function response at  $\mathbf{x}_i$ . An alternative interpretation is that  $[\mathbf{\Gamma m}]_i$  is the weighted sum of basis functions  $\Gamma(\mathbf{x}_i|\mathbf{z}_j)$  where the weights are the reflection coefficients  $m_j$  and the summation is over the  $j$  index. We will now consider this last interpretation and redefine the problem as finding both the weights  $m_i$  and the basis functions  $\Gamma(\mathbf{x}_i|\mathbf{z}_j)$ . This will be shown to be equivalent to the problem of a convolutional neural network.

## 24.7 Appendix: Soft Thresholding Function

Define the sparse inversion problem as finding the optimal value  $x^*$  that minimizes the objective function

$$\epsilon = \frac{1}{2} \|\mathbf{z} - \mathbf{x}\|_2^2 + \lambda \|\mathbf{x}\|_1, \quad (24.19)$$

where the  $L^1$  norm demands sparsity in the solution  $\mathbf{x}$ . An example is where  $\mathbf{z}$  is a noisy  $M \times N$  image such that  $\mathbf{z} = \mathbf{x} + \text{noise}$ , and we seek the optimal vector  $\mathbf{x}$  that satisfies equation 24.19. Here, the noisy  $M \times N$  image has been flattened into the tall  $MN \times 1$  vector  $\mathbf{z}$ .

The stationary condition for equation 24.19 is

$$\begin{aligned} \frac{\partial \epsilon}{\partial x_i} &= (x_i - z_i) + \lambda \frac{\partial \|\mathbf{x}\|_1}{\partial x_i}, \\ &= 0, \end{aligned} \quad (24.20)$$

where

$$\frac{\partial \|\mathbf{x}\|_1}{\partial x_i} = 1 \text{ for } x_i \geq 0; \quad \frac{\partial \|\mathbf{x}\|_1}{\partial x_i} = -1 \text{ for } x_i < 0. \quad (24.21)$$

Equations 24.20-24.21 can be combined to give the optimal  $x^*$  expressed as the two-sided ReLU function

$$x_i = \text{soft}(z_i, \lambda) = \begin{cases} z_i - \lambda & \text{if } z_i \geq \lambda \\ 0 & \text{if } |z_i| < \lambda \\ z_i + \lambda & \text{if } z_i < -\lambda \end{cases}. \quad (24.22)$$

More generally, the iterative-soft-threshold-algorithm (ISTA) that finds  $\mathbf{x}^*$

$$\mathbf{x}^* = \arg \min_{\mathbf{x}} \left[ \frac{1}{2} \|\mathbf{z} - \mathbf{Wx}\|_2^2 + \lambda \|\mathbf{x}\|_1 \right], \quad (24.23)$$

---

<sup>4</sup>This assumes that the zero-offset trace is generated with an impulsive point source with a smoothly varying background velocity model, and then migrated by a poststack migration operation. It is always assumed that the direct arrival is muted and there are no multiples.

is

$$x_i^{(k+1)} = \text{soft}\left(\mathbf{x}^{(k)} - \frac{1}{\alpha} \mathbf{W}^T (\mathbf{W} \mathbf{x}^{(k)} - \mathbf{z}), \frac{\lambda}{\alpha}\right)_i. \quad (24.24)$$

There are several more recently developed algorithms that have faster convergence properties than ISTA. For example, FISTA (Fast-ISTA) has quadratic convergence (Beck and Teboulle, 2009).

## 24.8 Appendix: Neural Network Least Squares Migration

The neural network least squares migration (NNLSM) algorithm in the image domain is defined as solving for *both* the basis functions  $\tilde{\Gamma}(\mathbf{x}_i|\mathbf{x}_j)$  and  $\tilde{m}_j$  that minimize the objective function defined in equation 24.5. In contrast, SLSM only finds the least squares migration image in the image domain and uses the pre-computed migration Green's functions that solve the wave equation.

The NNLSM solution is defined as

$$(\tilde{\mathbf{m}}^*, \tilde{\Gamma}^*) = \arg \min_{\tilde{\mathbf{m}}, \tilde{\Gamma}} \left[ \frac{1}{2} \|\tilde{\Gamma} \tilde{\mathbf{m}} - \mathbf{m}^{mig}\|_2^2 + \lambda S(\tilde{\mathbf{m}}) \right], \quad (24.25)$$

where now both  $\tilde{\Gamma}^*$  and  $\tilde{\mathbf{m}}^*$  are to be found. The functions with tildes are mathematical constructs that are not necessarily identical to those based on the physics of wave propagation as in equation 24.5.

The explicit matrix-vector form of the objective function in equation 24.25 is given by

$$\epsilon = \frac{1}{2} \sum_i \left[ \sum_j \tilde{\Gamma}(\mathbf{x}_i|\mathbf{z}_j) \tilde{m}_j - m_i^{mig} \right]^2 + \lambda S(\tilde{\mathbf{m}}). \quad (24.26)$$

and its Fréchet derivative with respect to  $\tilde{\Gamma}(\mathbf{x}_{i'}|\mathbf{z}_{j'})$  is given by

$$\frac{\partial \epsilon}{\partial \tilde{\Gamma}(\mathbf{x}_{i'}|\mathbf{z}_{j'})} = \sum_j (\tilde{\Gamma}(\mathbf{x}_{i'}|\mathbf{z}_j) \tilde{m}_j - \tilde{m}_{i'}^{mig}) \tilde{m}_{j'}. \quad (24.27)$$

The iterative solution of equation 24.25 is given in two steps (Olshausen and Field, 1996).

1. Iteratively estimate  $\tilde{m}_i$  by the gradient descent formula used with SLSM:

$$\tilde{m}_i^{(k+1)} = \tilde{m}_i^{(k)} - \alpha [\tilde{\Gamma}^T (\tilde{\Gamma} \tilde{\mathbf{m}} - \mathbf{m}^{mig})]_i - \lambda S(\tilde{\mathbf{m}})'_i. \quad (24.28)$$

However, one migration image  $\mathbf{m}^{mig}$  is insufficient to find so many unknowns. In this case the original migration image is broken up into many small pieces so that there are many migration images to form examples from a large training set. For prestack migration, there will be many examples of prestack migration images, one for each shot, and the compressive sensing technique denoted as VISTA (Ahmad et al., 2015) is used for the calculations.

2. Update the basis functions  $\tilde{\Gamma}(\mathbf{x}_i|\mathbf{z}_j)$  by inserting equation 24.27 into the gradient descent formula to get

$$\begin{aligned} \tilde{\Gamma}(\mathbf{x}_{i'}|\mathbf{z}_{j'})^{(k+1)} &= \tilde{\Gamma}(\mathbf{x}_{i'}|\mathbf{z}_{j'})^{(k)} - \alpha \frac{\partial \epsilon}{\partial \tilde{\Gamma}(\mathbf{x}_{i'}|\mathbf{z}_{j'})}, \\ &= \tilde{\Gamma}(\mathbf{x}_{i'}|\mathbf{z}_{j'})^{(k)} \\ &\quad - \alpha \left( \left\{ \sum_j \tilde{\Gamma}(\mathbf{x}_{i'}|\mathbf{z}_j) \tilde{m}_j \right\} - m_{i'}^{mig} \right) \tilde{m}_{j'}. \end{aligned} \quad (24.29)$$

It is tempting to think of  $\tilde{\Gamma}(\mathbf{x}|\mathbf{x}')$  as the migration Green's function and  $\tilde{m}_i$  as the component of reflectivity. However, there is yet no justification to submit to this temptation and so we must consider, unlike in the SLSM algorithm, that  $\tilde{\Gamma}(\mathbf{x}|\mathbf{x}')$  is a sparse basis function and  $\tilde{m}_i$  is its coefficient. To get the true reflectivity then we should equate equation 24.18 to  $\sum_j \tilde{\Gamma}(\mathbf{x}_i, \mathbf{x}_j) \tilde{m}_j$  and solve for  $m_j$ .

## 24.9 Appendix: Alignment of the Filters

To align the learned filters, we first choose a “target” filter, which is denoted as a 2D matrix  $\mathbf{A}$  with the size of  $M \times N$ . Then we try to align all the other filters with the target filter through their cross-correlation. For example, if we choose one filter denoted as matrix  $\mathbf{B}$  with the same size as  $\mathbf{A}$ , we can get the cross-correlation matrix  $\mathbf{C}$  with its elements defined as,

$$C_{i+M, j+N} = \sum_{m=1}^M \sum_{n=1}^N a_{m,n} \cdot b_{m+i, n+j}, \quad (24.30)$$

where  $-M < i < M$  and  $-N < j < N$ .  $a_{i,j}$ ,  $b_{i,j}$  and  $C_{i,j}$  indicate the element at position  $(i, j)$  in matrices  $\mathbf{A}$ ,  $\mathbf{B}$  and  $\mathbf{C}$ , respectively. The location of the maximum absolute value of the elements in matrix  $\mathbf{C}$  indicates how much should we shift filter  $\mathbf{B}$  to filter  $\mathbf{A}$  in each direction. Figure 24.16 shows the calculation of the cross-correlation matrix  $\mathbf{C}$  for two filters  $A$  and  $B$ .  $c_{1,0}$  (or  $C_{4,3}$ ) is the maximum absolute value of the elements in matrix  $\mathbf{C}$ , which indicates filter  $\mathbf{B}$  should be shifted 1 position along the first direction. Here, we need to pad zeros along all the dimensions of filter  $\mathbf{B}$  before shifting it, which is displayed in Figure 24.17.

Figure 24.18a shows the learned filters with a size of  $17 \times 9$  from the migration image of the SEG/EAGE salt model. Filter No. 7 (yellow box) is chosen as the target filter. The aligned filters are shown in Figure 24.18b without zero padding and the stacked feature maps from the original and aligned filters are displayed in Figures 24.18c and 24.18d, respectively. It is evident that the reflector interfaces from the aligned filters are more continuous especially in the red box compared with those of the original filters.



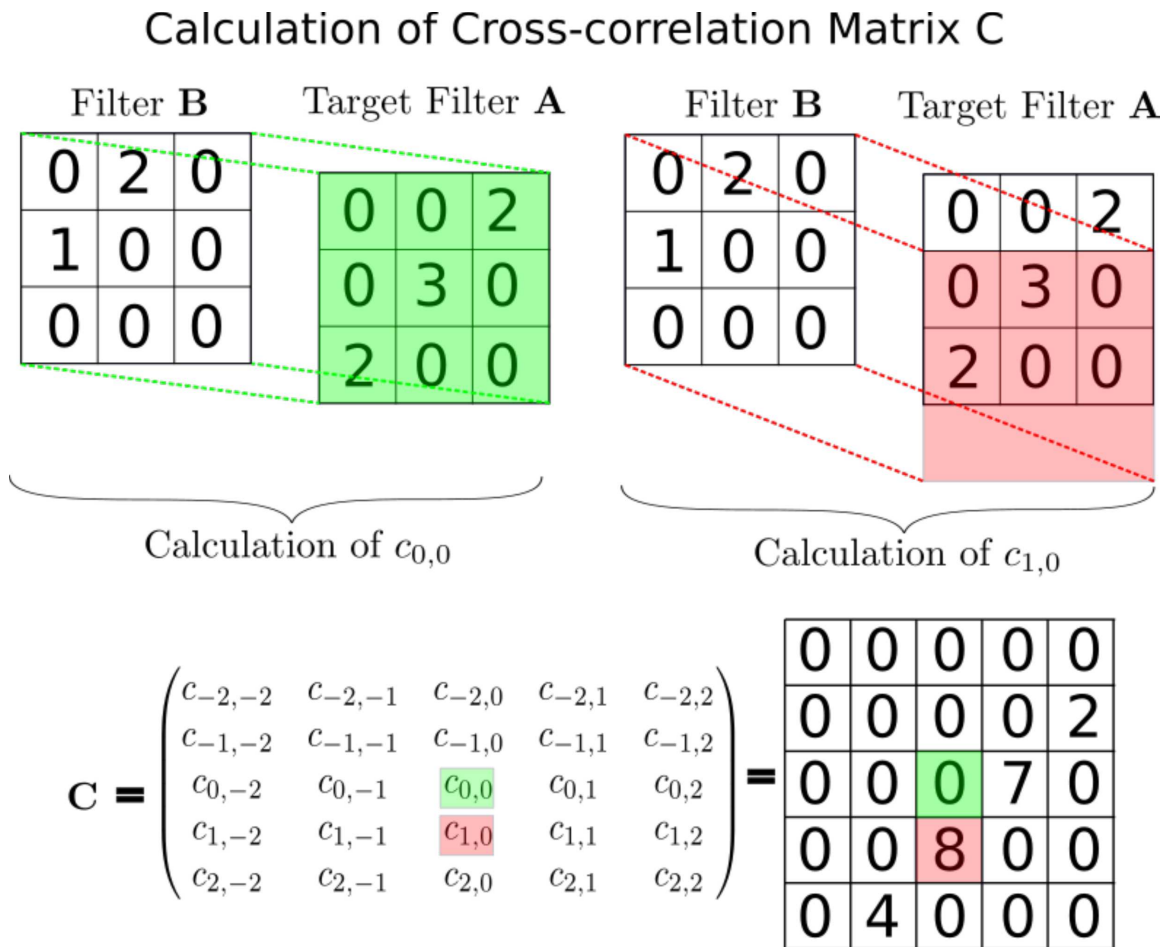
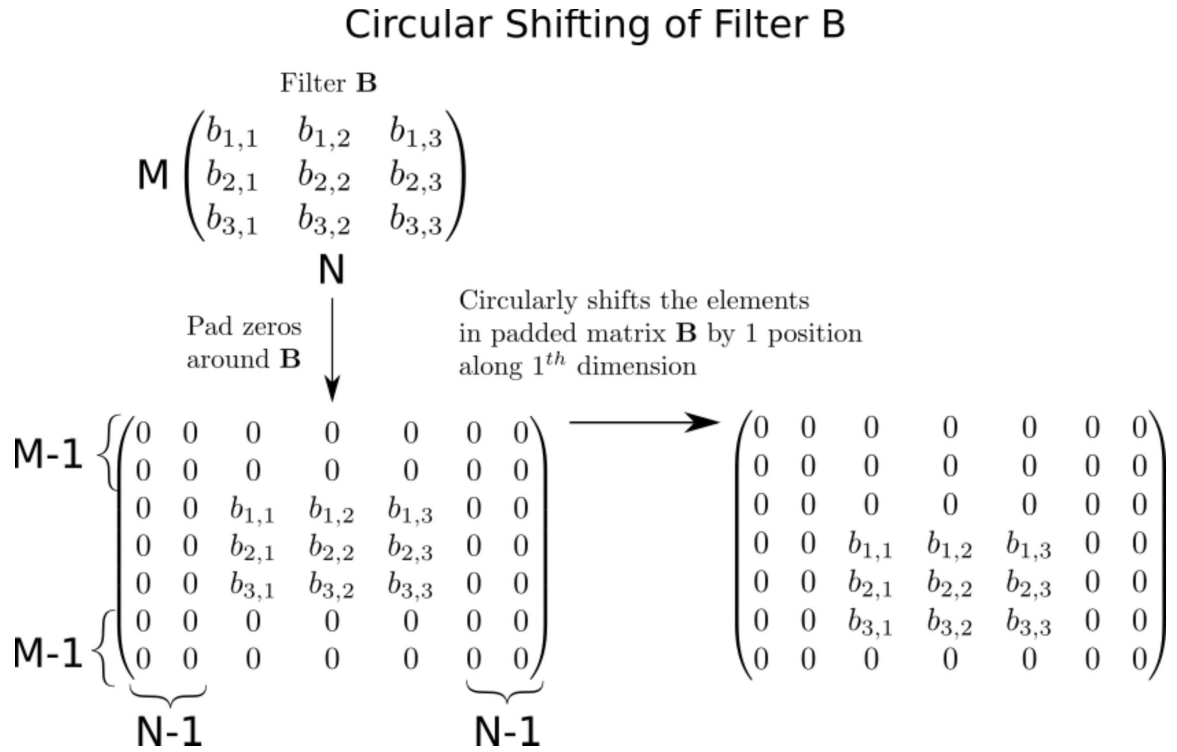


Figure 24.16: Calculation of the cross-correlation matrix C.

Figure 24.17: Diagram that illustrates the circular shifting of padded filter **B**.

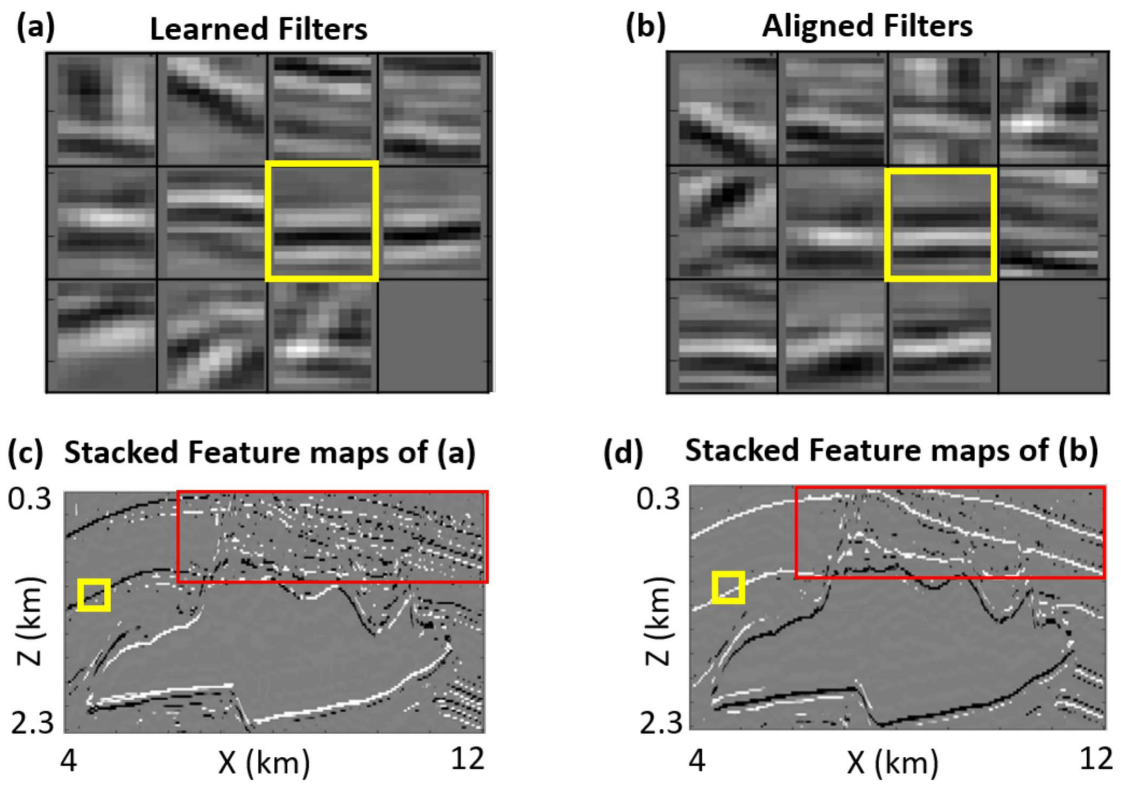


Figure 24.18: a) Learned filters from the migration image of SEG/EAGE salt model; b) the aligned filters; stacked feature maps of the c) original and d) aligned filters, where the yellow boxes show the sizes of the filters for the filters, respectively.



## Chapter 25

# Wave Equation Inversion and Neural Networks

The previous chapter presented an example where the latent variables in an autoencoder were used to invert for the velocity model from seismic data. Now we present the theoretical details for this methodology, denoted as Newtonian machine learning (NML), and apply it to crosswell and refraction seismic data. We compare its performance to that of full waveform inversion (FWI), where the advantage of NML over FWI is it is more robust and has less of a tendency in getting stuck at local minima. This is because NML only needs to explain a sparse representation of a low-dimensional latent-space representation of the input data. This is far less demanding than FWI's difficult task of explaining all of the wiggles in every arrival. The disadvantage of NML is that it sometimes provides a tomogram with theoretically less resolution than the ideal FWI tomogram. In this case, the NML tomogram can be used as an excellent starting model for FWI.

### 25.1 Introduction

Full waveform inversion is very ambitious, it seeks an earth model  $\mathbf{m}$  that explains every complicated wiggle in a large set of very wiggly seismograms denoted by  $\mathbf{d}$  (Tarantola, 1987). A basic assumption is that the forward modeling operator  $\mathbf{L}$  (see Figure 25.1a) is a "good enough" representation of the actual physics of wave propagation in the Earth. Such ambitions and assumptions can sometimes lead to getting stuck in local minima, slow convergence and tomograms with unacceptable errors. To partly mitigate these problems a multiscale strategy (Bunks et al., 1995) can be employed. As an alternative, we present wave-equation inversion of skeletonized data. Here, the large complicated data set is reduced to skeletonized data  $\tilde{\mathbf{d}}$  which are much less complicated yet still contain essential information about the model parameters of interest. This means that the skeletonized data lead to a less complicated objective function that does not contain so many local minima associated with the one for FWI.

To invert the skeletonized data, skeletonized wave-equation inversion (SWI) uses numerical solutions to the wave equation to update the model by back-propagation of the weighted skeletonized data residual (Luo and Schuster, 1991a, 1991b; Lu et al., 2017). The result is a largely robust inversion algorithm free of layered media or high-frequency approximations. The main benefits are that the inversion algorithm is less prone to getting stuck in a local minimum, but at the cost of a reduction in model resolution. However, the inverted tomogram can be used as the starting model for FWI because it is likely to be sufficiently close to the actual earth model.

Recent examples (Lu et al., 2017) of skeletonized wave-equation inversion include the following cases.

1. **Inversion of  $Q(\mathbf{x})$  from diving waves and surface waves.** Here the skeletonized data  $\tilde{\mathbf{d}}$  are the peak frequencies associated with the arrivals of diving waves or surface waves, and the output model is  $Q(\mathbf{x})$  (Li et al., 2017b; Li et al., 2018d; Li et al., 2019c).
2.  **$V(\mathbf{x})$  inversion by migration velocity analysis.** The skeletonized data  $\tilde{\mathbf{d}}$  are the depth residuals or static shifts associated with plane-wave migration images in the common image gather domain (Guo et al., 2017).
3. **Inversion of  $V(\mathbf{x})$  from the dispersion curves of surface waves.** Here, the input skeletonized data  $\tilde{\mathbf{d}}$  are the dispersion curves of surface waves in the  $k - \omega$  domain and the output model is the shear velocity for Rayleigh waves, Love waves, and the P-velocity model for near-surface guided waves (Li et al., 2017a).

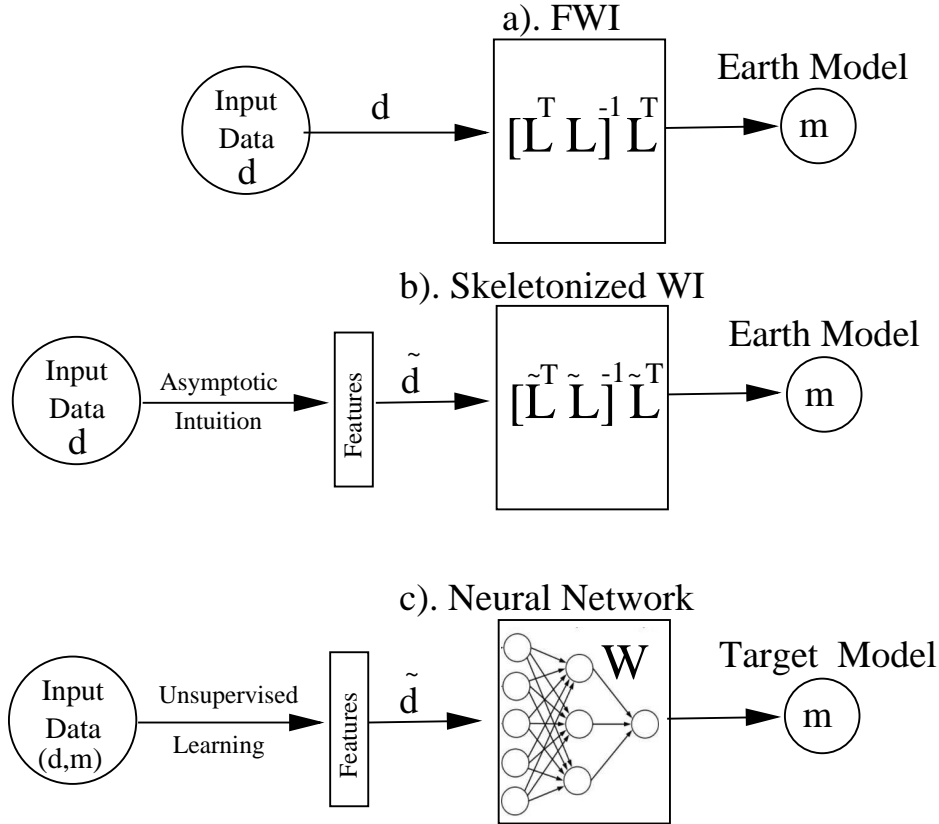


Figure 25.1: System diagrams for a) FWI, b) SWI, and c). a neural network. The neural network and SWI are similar in that the input data are skeletonized features of the raw data strongly influenced by small changes in the model. However, the neural network is first trained to get an estimate of  $\mathbf{W}$ , which is then used to estimate the target model  $\mathbf{m}$  from untrained data.

The skeletonization strategy of SWI is similar to that of neural networks, where a complex data set is often reduced to its *essential features* (Bishop, 2006). These skeletonized "features" are input into the system of neural network layers depicted in Figure 25.1c. Here, the original data can be very complex but complexity can be reduced by an unsupervised learning algorithm such as a principle component analysis, singular value decomposition (Bishop, 2006) or some statistical measure of

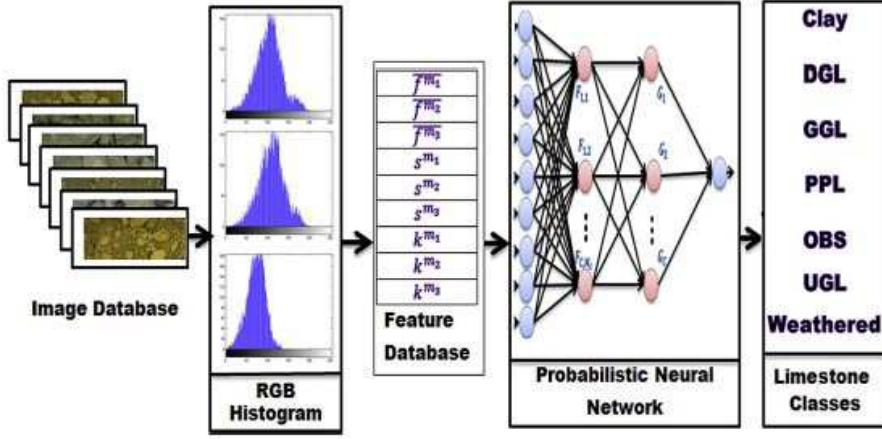


Figure 25.2: System diagram for the probabilistic neural network for classifying thin sections (from Patel and Chatterjee, 2016).

important features (Patel and Chatterjee, 2016). In principle, these less complex features are still rich in information about the model. As an example, Figure 25.2 depicts the workflow for classifying rock types using a supervised learning method (Patel and Chatterjee, 2016). Here the original input data consist of images  $\mathbf{d}$  of thin sections and their classification  $\mathbf{m}$ , each one classified as a type of limestone by a geologist. For example, if there are three types of classified rocks then the  $3 \times 1$  target model vector  $\mathbf{m} = (m_1, m_2, m_3)$  might be assigned  $\mathbf{m} = (1, 0, 0)$  if it is a pure limestone,  $\mathbf{m} = (0, 1, 0)$  if it is a weathered rock, and  $\mathbf{m} = (0, 0, 1)$  if it is a shale-limestone. In practice, the assignment is over a range of classification numbers and  $\mathbf{m}$  is a  $7 \times 1$  vector according to Figure 25.2.

To simplify the input data, the thin-slice images are skeletonized into histograms of red, blue, and green colors. These histograms are further skeletonized into second-order (variance), 3rd-order (skewness), and 4th-order (kurtosis) statistical parameters for each color histogram. The examples from this skeletonized training set are denoted as  $(\tilde{\mathbf{d}}, \mathbf{m})$ , where for each image the input is a  $9 \times 1$  vector of statistical values from the three histograms. For pedagogical simplicity we avoid the superscript notation that denotes the  $i^{th}$  example from the training set. After sufficient training with  $N$  training examples, the coefficients of the neural network are inverted for and used to classify thin sections from out of the training set. In their case history, Patel and Chatterjee achieved more than a 90% accuracy in the classification of thin sections taken from outside the training set.

## 25.2 Theory for Wave Equation Inversion of Skeletonized Data

The starting point for wave-equation inversion of skeletonized data is to define an objective function

$$\epsilon = \frac{1}{2} \sum_{j=1}^N (\tilde{d}_j - \tilde{d}_j^{obs})^2, \quad (25.1)$$

where  $\tilde{d}_j$  is the predicted data for  $j \in [1, 2, \dots, N]$  and the *obs* superscript denotes the observed skeletal data. Different norms can be employed, and different objective functions can be used such as a correlation objective function. For simplicity of exposition, we do not include regularization or

preconditioning terms and assume that  $\tilde{d}_j$  is always real. Note,  $d_j$  without the tilde denotes the raw data but  $\tilde{d}_j$  with the tilde describes the data skeletonized from the raw data.

An iterative gradient descent method can be used to find the model parameters that best explain the data in the least squares sense:

$$m_i^{(k+1)} = m_i^{(k)} - \alpha \sum_{j=1}^N \overbrace{(d_j - \tilde{d}_j^{obs})}^{r_j=residual} \overbrace{\frac{\partial \tilde{d}_j}{\partial m_k}}^{Frechet}, \quad (25.2)$$

where  $\alpha$  is the step length and  $m_i^{(k)}$  is the model parameter of interest in the  $i^{th}$  cell at the  $k^{th}$  iteration. If the skeletal data value  $\tilde{d}_i$  is not explicitly contained in the wave equation, e.g. reflection traveltimes are not explicit variables in the wave equation, then a connective function  $\Phi(\tilde{d}_j, m_k)$  is required to get a formula for the Fréchet derivative  $\frac{\partial \tilde{d}_j}{\partial m_k}$ . For example, Luo and Schuster (1991a) proposed the crosscorrelation between the observed and predicted pressure traces for wave-equation travelttime inversion. Li et al. (2017a) presents examples where the correlation is between the observed and predicted magnitude spectra of dispersion curves from surface waves. There are many other examples, some of which are described in Lu et al. (2017). Once the connective function is properly defined then the implicit function theorem (Luo and Schuster, 1991) can be used to define the formula for the Fréchet derivative:

$$\frac{\partial \tilde{d}_j}{\partial m_k} = A \frac{\partial P_j}{\partial m_k}, \quad (25.3)$$

where  $A$  is a scalar that acts as a normalization term and  $P_j$  is defined as a fundamental field variable that explicitly appears in the wave equation. Fortunately, the formula for the Fréchet derivative  $\frac{\partial P_j}{\partial m_k}$  of a fundamental field variable is straightforward to derive by the adjoint-state method. For acoustic data, the field variable is the pressure field recorded at the surface and the formula for its Fréchet derivative is well known. If more than one type of unknown is inverted for the multidimensional implicit function theorem can be used to compute the Fréchet derivatives for each type of model parameter (described by B. Guo in Schuster, 2017). Note, if  $d_i$  is substituted for  $P_i$  and  $m_i$  denotes the velocity value in the  $i^{th}$  cell then the update formula in equation 25.1 is that for FWI.

### 25.2.1 Feature Extraction

The most important step, or goal, in SWI is the identification of skeletonized features that are simple and also greatly influenced by changes in the model parameter of interest. Towards this goal we look to the past where theoreticians identified skeletal data and derived asymptotic formulas that connected the skeletal data with the model parameter of interest. An obvious example is that of asymptotic ray-based tomography where, under the high-frequency approximation, traveltimes can be quickly generated by tracing rays through a sufficiently smooth model (Aki and Richards, 2002). Another example is frequency-domain inversion which inverts important signal features such as phase and/or amplitudes of arrivals for a sparse set of frequencies. If the physics of the problem is well understood, then the identification of the skeletonized data can be straightforward. If the physics between skeletal data and the model parameters is not well understood, then the sensitivity of the skeletal Fréchet derivative can be estimated by a combination of intuition and numerical sensitivity tests.

Can we go one step further by asking a machine learning algorithm to identify an important skeletal data set. Once identified, these skeletonized data can be inverted by a SWI method. For example, Figure 25.3 suggests a hybrid machine learning and SWI algorithm where an unsupervised machine learning method can be used to extract simplified but important features in the data set.



Then the SWI method and the implicit function theorem can be used to discover the skeletal Fréchet derivative to be used in the update formula in equation 25.2. One possibility is that the skeletonized data can be obtained by principle component analysis (PCA) and the connective function is the spatial crosscorrelation between the observed and predicted PCA images. These PCA images can be localized in space. Can a general machine learning algorithm be used to identify the connection between skeletal features which are most sensitive to the model parameters of interest (Schuster, 2018)? This might be possible using a suitable training set obtained with numerically simulated data in realistic models.

In the next section, we use an autoencoder’s latent-space variables as the skeletonized features of the data. The implicit function theorem is then used to derive the misfit gradient in equation 25.3 that can be used invert for the velocity model. This procedure (Chen and Schuster, 2020; Chen et al., 2020b; Yu et al., 2020) is denoted as Newtonian machine learning (NML).

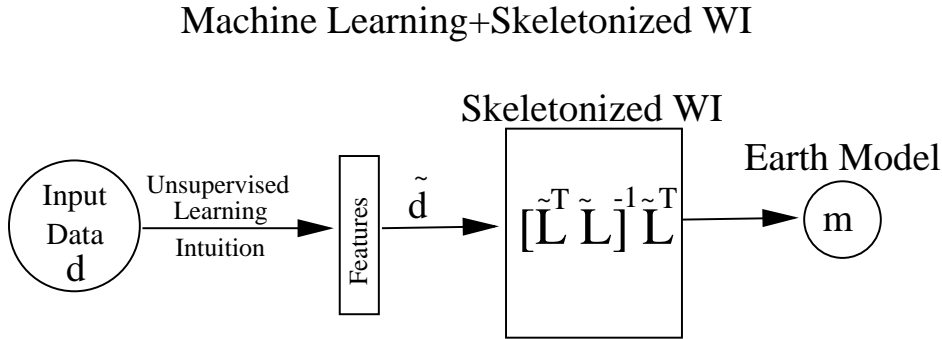


Figure 25.3: System diagram for a hybrid machine learning and SWI algorithm. Here, the skeletonized data  $\tilde{d}$  can be obtained by, for example, a principle component analysis (PCA), where the connective function is the crosscorrelation between the observed and predicted PCA images. Machine learning combined with WI gives rise to a Newtonian machine learning algorithm, where the optimal skeletonized features are learned from the data and then inverted by numerical solutions to the wave equation. Figure from Schuster (2018).

## 25.3 NML Theory

An  $nt \times 1$  seismic trace can be viewed as a data point in a  $nt$ -dimensional space, where  $nt$  is the number of time samples in the trace. In this case, FWI finds the velocity model that minimizes the  $L^2$  distance between the observed and predicted data points in an  $nt$ -dimensional space. Similarly, the NML method finds the velocity model that minimizes the  $L^2$  distance between the  $n \times 1$  observed and predicted latent-space (LS) vectors, where  $n \ll nt$  is much smaller than  $nt$ .

Figure 25.4 illustrates the idea of the NML method. An autoencoder (AE) compresses the high-dimensional observed and predicted seismic trace to a two-dimensional LS. The  $L^2$  distances between the observed and predicted LS vectors can be used to quantify the accuracy of the inverted model.

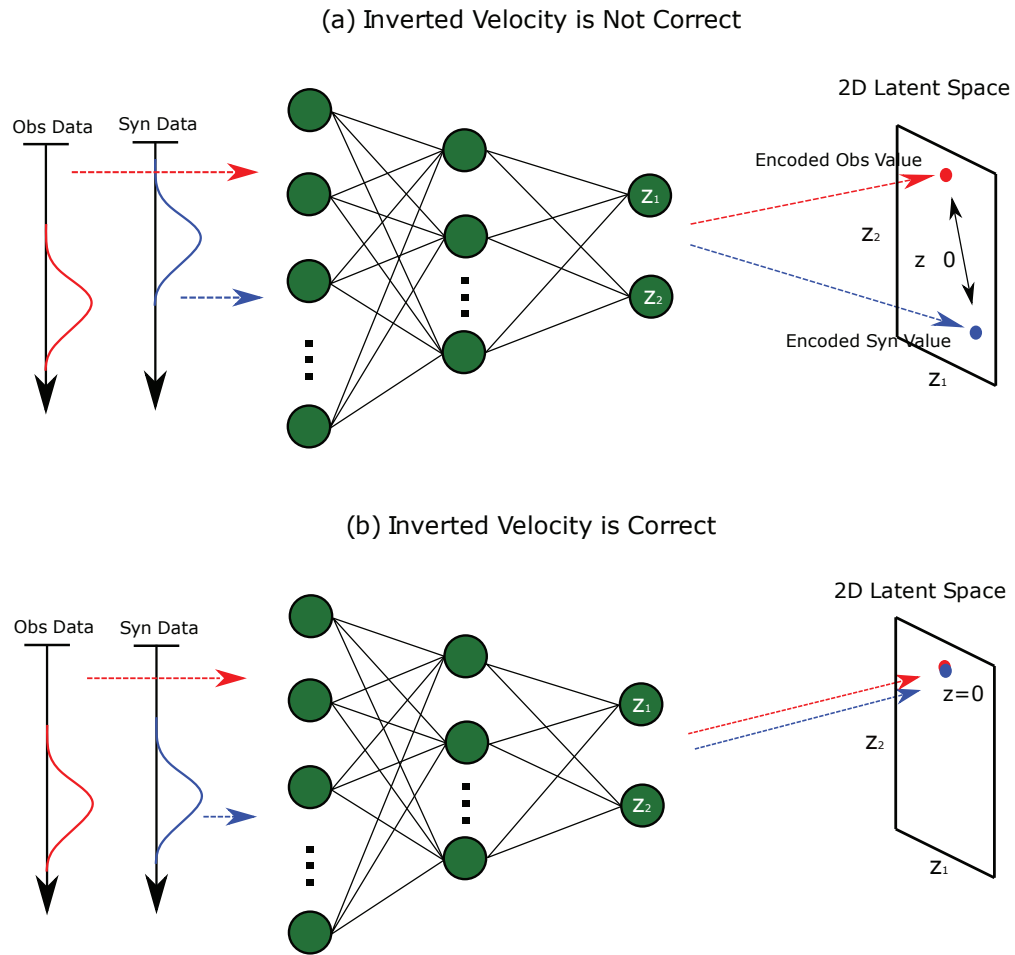


Figure 25.4: The demonstration of NML misfit function when LS dimension equals two. (a) If the inverted velocity is incorrect, it appears a large spatial distance in the two-dimensional latent space. (b) If the inverted velocity model is correct, the distance goes to zero.

### Misfit function

The NML misfit function can be written as

$$\epsilon = \sum_s \sum_r \sum_k \Delta z_k(\mathbf{x}_r, \mathbf{x}_s)^2, \quad (25.4)$$

where  $\Delta z$  represents the difference between the observed and predicted LS vectors. The source and receiver locations are represented by  $\mathbf{x}_s$  and  $\mathbf{x}_r$ , respectively, and  $s$  and  $r$  indicate the source and receiver indices. Here, the subscript  $k$  indicates the dimension index of the LS feature vector.

The gradient  $\gamma(\mathbf{x})$  can be obtained by taking the derivative of the misfit function  $\epsilon$  with respect to the velocity parameter  $v(\mathbf{x})$  as

$$\gamma(\mathbf{x}) = -\frac{\partial \epsilon}{\partial v(\mathbf{x})} = -\sum_s \sum_r \sum_k \left[ \left( \frac{\partial \Delta z_k(\mathbf{x}_r, \mathbf{x}_s)}{\partial v(\mathbf{x})} \right)^T \Delta z_k(\mathbf{x}_r, \mathbf{x}_s) \right]. \quad (25.5)$$

Equation 25.5 shows that the velocity gradient is computed for each of the coordinates in the LS, and the final NML gradient  $\gamma(\mathbf{x})$  is obtained by summing all of these components of the gradient. The details for deriving the gradient formula are given in Appendix 25.8

## 25.4 NML Inversion of Crosswell Data

A portion of the Pluto model is selected as the true model (shown in Figure 25.5a) to test the NML method. A crosswell acquisition system is used to generate the seismic data, where the source well is deployed at  $x = 10$  m and contains 59 shots with a 30 m shot interval. Each shot has 177 receivers evenly distributed along the receiver well, which is located at  $x = 1750$  m. A 20 Hz Ricker wavelet is used as the seismic source, and we assume the initial velocity model is linearly increasing with the minimum and maximum velocities equal to 3100 m/s and 3700 m/s, respectively. An acoustic finite-difference modeling method is used to generate the observed data based on the true model.

In the first step, we design a three-layer AE with a single-dimensional LS (la=1) to extract the kinematic information of the seismic data. The NN matrices for the first, second, and third encoder layers have dimensions of  $2000 \times 200$ ,  $200 \times 15$ , and  $15 \times 1$ , respectively. The decoder architecture is the mirror of the encoder network. The training dataset is composed of the early arrivals in the recorded traces. We train this AE (la=1) network for 50 epochs using the Adam optimizer, and achieved a training accuracy of 91%. Once the training is complete, we fix the parameters of this AE (la=1) network during NML (la=1) inversion. Figures 25.5c and 25.5d show the first iteration gradient and inverted velocity model of the NML (la=1) inversion, which are dominated by low-wavenumber velocity updates. However, the detailed velocity structures are missing in the NML (la=1) inverted model. These are missing because the single-dimensional LS feature is too small to represent the amplitude information in the seismic data.

In the next step, we use the NML (la=1) inverted model as the starting model to further recover the high-wavenumber velocity details. We build a new AE that has the same encoder and decoder network as in the previous AE. But we set its LS dimension to ten, which is the optimal LS dimension found by the "data reconstruction error versus LS dimension" curve in Figure 25.6. The training set is composed of the original observed traces, and the same training strategy is used for this AE (la=10) network training. Figure 25.7a shows the first iteration gradient of NML (la=10) which is dominated by the high-wavenumber updates. The inverted model of NML (la=10) (shown in Figure 25.7b) shows a dramatic resolution increase compared to the NML (la=1) inverted result. This resolution increase is because the ten-dimensional LS feature vector can better preserve the entire data content compared to the one-dimensional LS feature that was used in NML (la=1)

inversion.

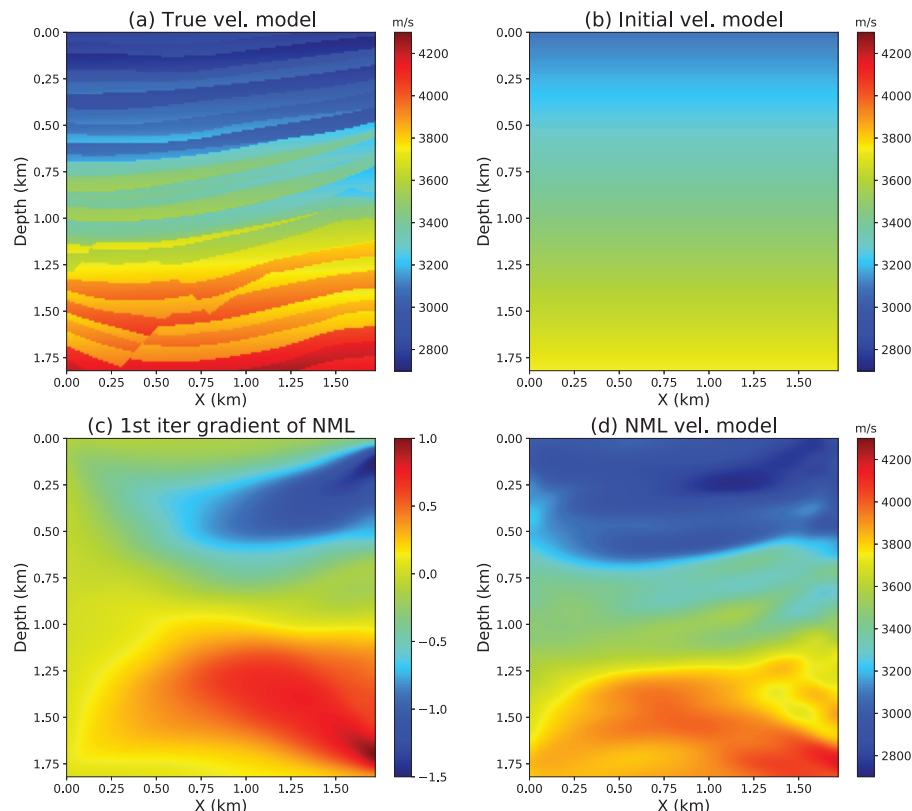


Figure 25.5: (a) The true and (b) initial velocity model. (c) The first iteration gradient and (d) inverted velocity model of NML ( $l_a=1$ ) method.

To evaluate the performance of the NML ( $l_a=10$ ) inversion, we also use FWI to reconstruct a high-resolution model. Here, the starting velocity model for FWI is the NML ( $l_a=1$ ) tomogram. Figures 25.7c and 25.7d show the first iteration gradient and inverted velocity model of FWI, which shows the same level of image resolution compared to the NML ( $l_a=10$ ) tomogram. Therefore we can conclude that NML ( $l_a=10$ ) can well recover the high-wavenumber velocity details by inverting a multi-dimensional LS feature vector, and the inverted model is as good as the FWI result for this model. Moreover, NML ( $l_a=10$ ) requires a much smaller storage space than FWI (about 200 times smaller), because only the low-dimensional LS features of the observed data need to be stored.

### 25.4.1 Assessing the Optimal Dimension of the Latent Space

Two AEs are used, one with a two-dimensional LS and the other with a four-dimensional LS. The seismic traces in Figure 25.8a are used to train these two AEs. Figure 25.8b shows the decoded waveform of the AE ( $l_a=2$ ) network and suggests that the first-arrival energy has been well recovered, but some reflection events are missing. These omissions indicate that the two-dimensional LS is too small to preserve the whole reflection energies. The missing reflection events can be clearly seen in Figure 25.8c which is computed by subtracting the decoded data ( $l_a=2$ ) from the input data. Here  $l_a=2$  represents a two-dimensional LS.

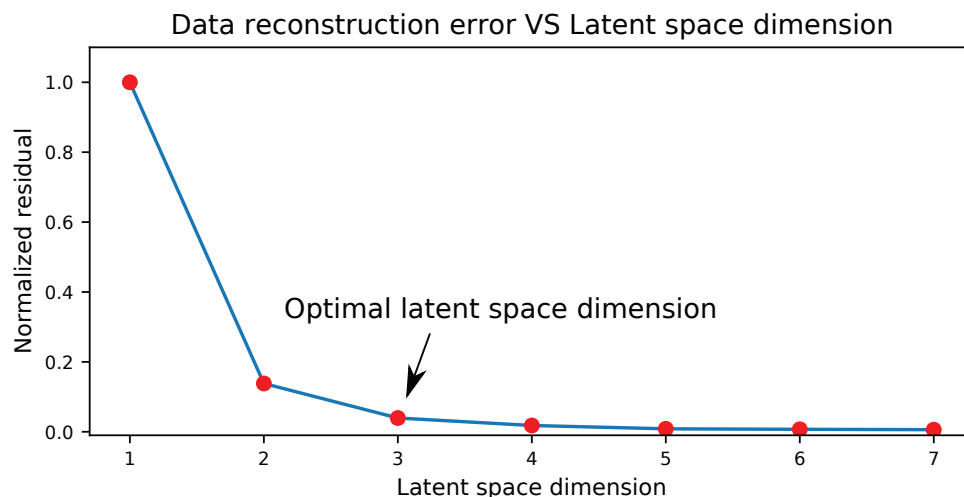


Figure 25.6: The normalized data reconstruction error versus latent-space dimension, where the normalized  $L^2$  norm of the data residual is the reconstruction error. The black arrow points to the optimal latent-space dimension point.

Figure 25.8d shows the decoded waveforms of the AE ( $la=4$ ) where both the first-arrival and reflection energies have been well recovered. The data residual shown in Figure 25.8e is much smaller compared to Figure 25.8c. Therefore we can conclude that an AE with a larger LS has a stronger ability in recovering the input data information than an AE with a smaller LS dimension.

However, there is little improvement of the AE data reconstruction when the LS dimension reaches a certain level. Figure 25.6 shows the residual curve of the data reconstruction error versus the LS dimension, where the reconstruction error drops rapidly when the LS dimension goes from one to three. But it barely changes when the LS dimension is larger than three, which means the three-dimensional LS is the optimal dimension that should be used to represent the whole input data information. The optimal LS dimension of 3 is approximately at the *elbow* of this curve, which can be used to assess the optimal dimension of the LS.

## 25.5 NML Inversion of Refraction Data

Seismic refraction data can also be inverted by NML (Yu et al., 2020). In this case the sources and receivers are deployed on the Earth's free surface, and refraction data are the first arrivals refracting from a layer interface where the velocity of the underlying layer is faster than that of the overlying layer.

As an example, Yu et al. (2020) tested NML on refraction data recorded next to the Gulf of Aqaba on the Saudi Arabian peninsula shown in Figure 25.9a. A 12-lb hammer striking a plate on the ground is used as the seismic source, and each source excitation was recorded by 120 vertical-component receivers placed along a line. The source and receiver spacings are 2.5 m, where each source was excited next to a receiver position. The white receiver line in Figure 25.9a is 300 m long, and an example of a recorded CSG recorded is shown in Figure 25.9b.

The AE architecture in Figure 25.10 is used to generate the latent-space vectors, where  $Dim1 = 2400$  is the same number of samples in an input trace. Empirical tests showed that  $Dim2 = 800$  and  $Dim3 = 150$  were sufficient for useful results. The dimensions of the encoding matrices  $W_1$ ,  $EW_2$  and  $W_3$  are, respectively,  $800 \times 2400$ ,  $150 \times 800$ , and  $1 \times 150$ . Similar to the crosswell examples, the

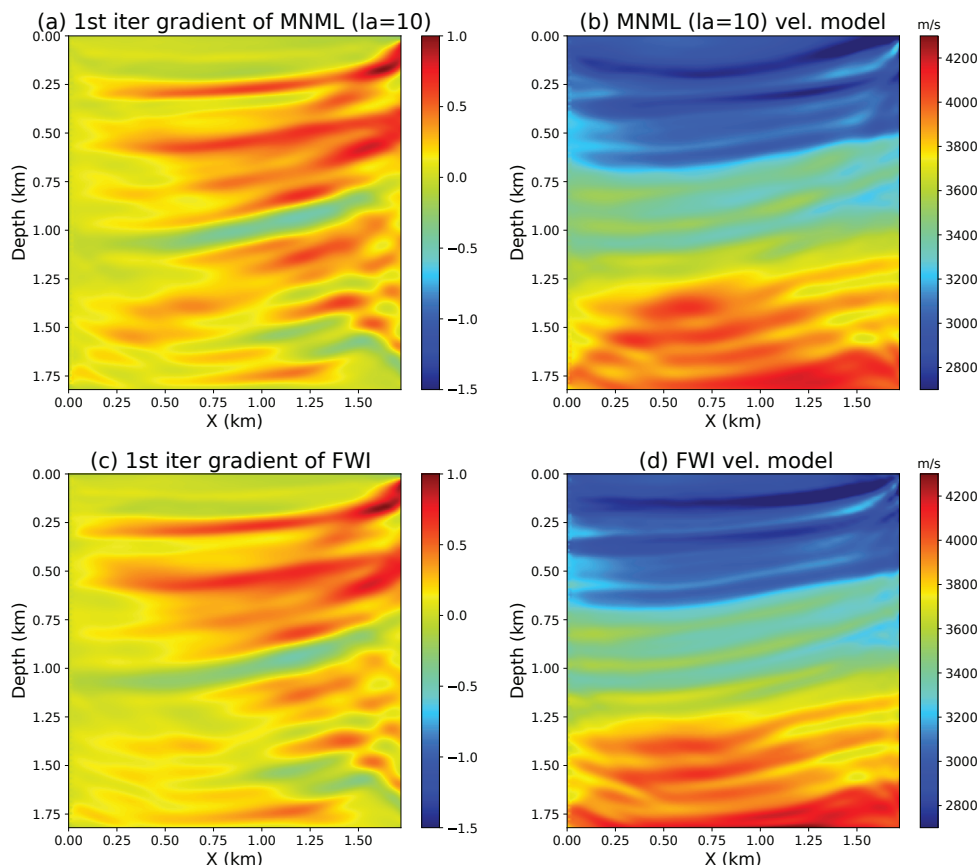


Figure 25.7: (a) The first iteration gradient and (b) inverted velocity model of NML ( $l_a=10$ ) inversion. (c) The first iteration gradient and (d) inverted velocity model of FWI.

velocity tomograms for using a latent-space dimension of 2 are much more accurate than those for a latent-space dimension of 1.

We only are interested in inverting the early arrivals so the data in Figure 25.11a are muted to give the processed data in Figure 25.11b. Noisy wavelets can be tamed by transforming the source wavelets in Figure 25.11b into envelopes to get the Figure 25.12a shot gather. The AE architecture in Figure 25.10 is used to get the AE output shown in Figure 25.12b. Here, a 1-dimensional latent space is used to give the latent-space vectors shown in Figure 25.12c.

A two-dimensional latent-space representation of the windowed seismic data is shown in Figure 25.13. The seismic data and their different features were inverted to give the velocity tomograms in Figure 25.14. A variety of different methods were used to invert the data. The wave-equation traveltime inversion (WT) method is used to compute Figure 25.14a, where the input windowed data were bandpass filtered peaked at 50 Hz. The starting model is a linear gradient velocity model (Yu et al., 2020). The same data were inverted by the NML method to give the tomogram in Figure 25.14b. The envelope inversion is shown in Figure 25.14c and the FWI tomogram is displayed in Figure 25.14e. These results were obtained after 20 iterations. The first four inverted tomograms, especially the first and the fourth tomograms in Figure 25.14a and Figure 25.14d indicate the possible location of the Aqaba fault location at  $X = 125$  m and  $Z = 20$  m, as indicated by the red line that crosses the white receiver line in Figure 25.9a. The result is also consistent with the travel-time

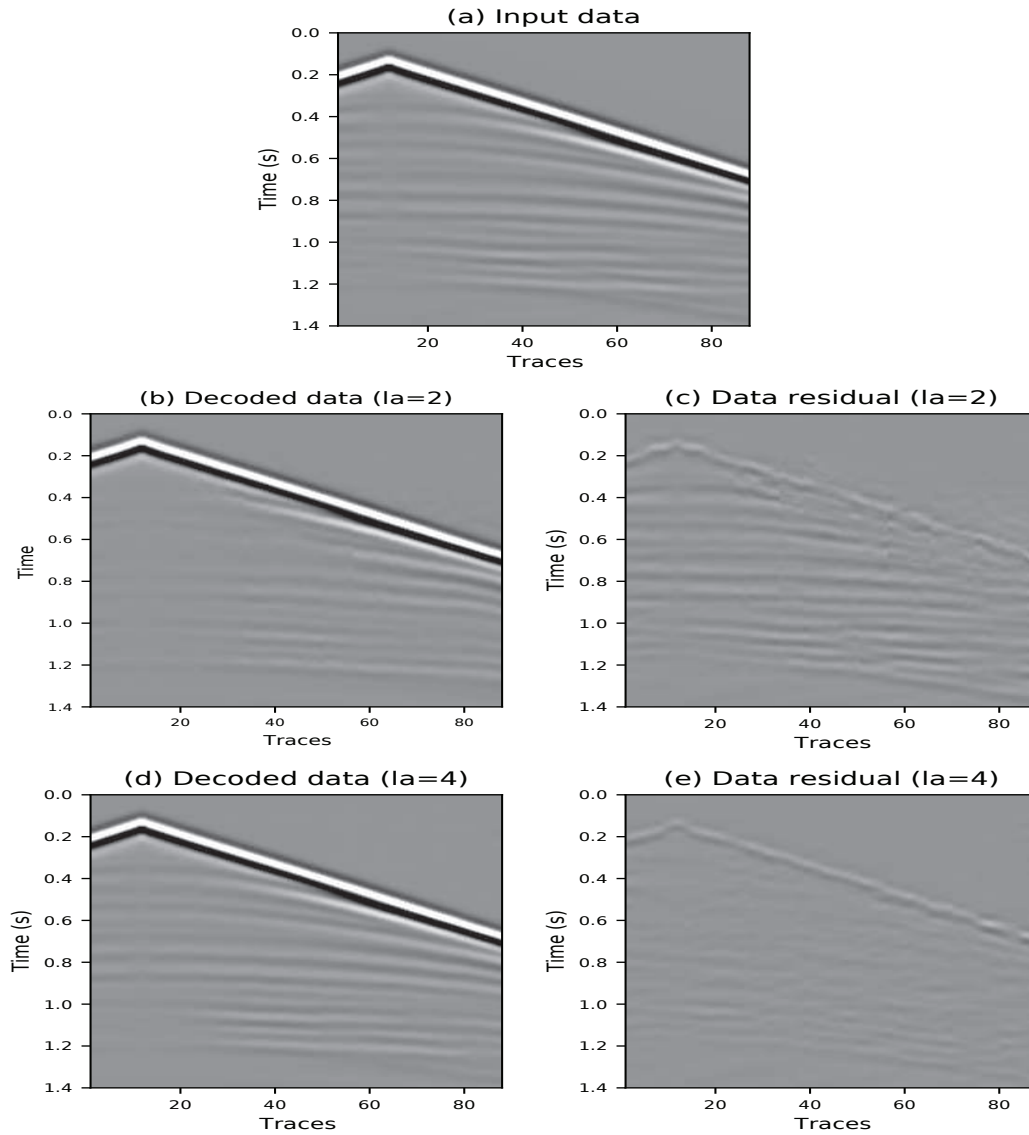


Figure 25.8: a) The decoded waveform and b) data residual when the latent-space dimension is equal to two. c) The decoded waveform and d) data residual when the latent-space dimension is equal to four. e) The decoded waveform and (f) data residual when the latent-space dimension is equal to six.

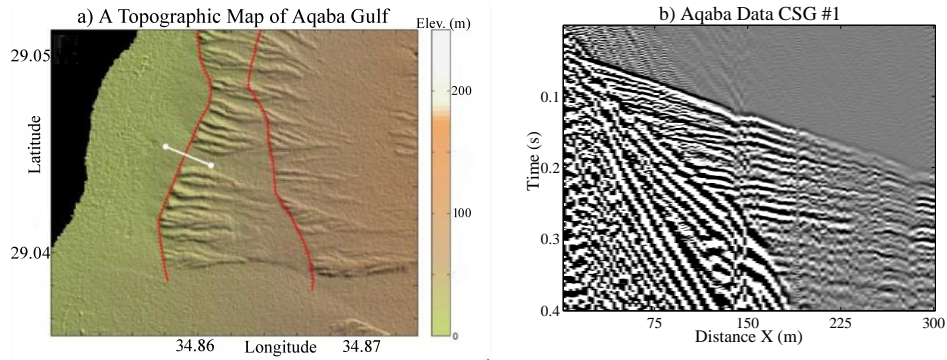


Figure 25.9: a) Aerial view of Aqaba area with the 300-m white line indicating the seismic line. The black region indicates the Gulf of Aqaba, and b) depicts a raw CSG. Figure from Yu et al. (2020).

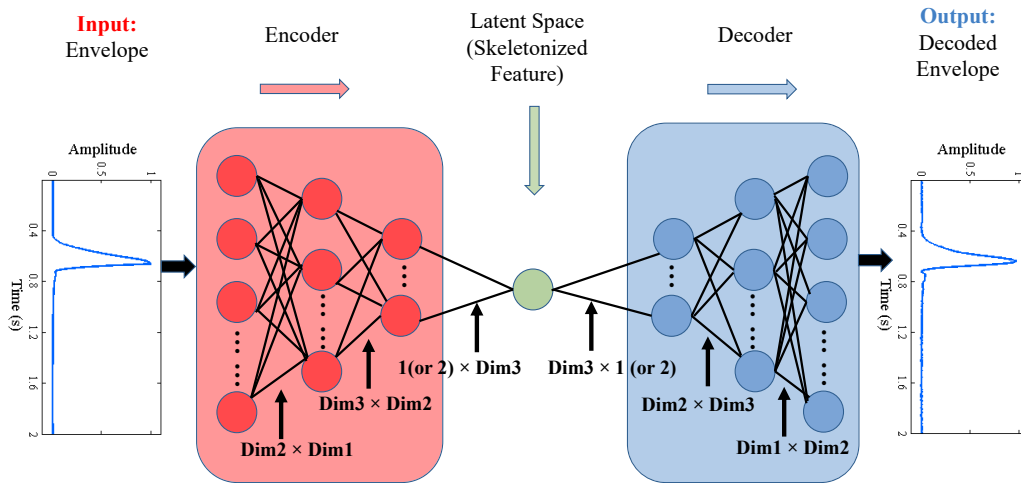


Figure 25.10: Autoencoder architecture for inverting for the velocity tomogram from the latent-space vectors of windowed seismic data. Figure from Yu et al. (2020).

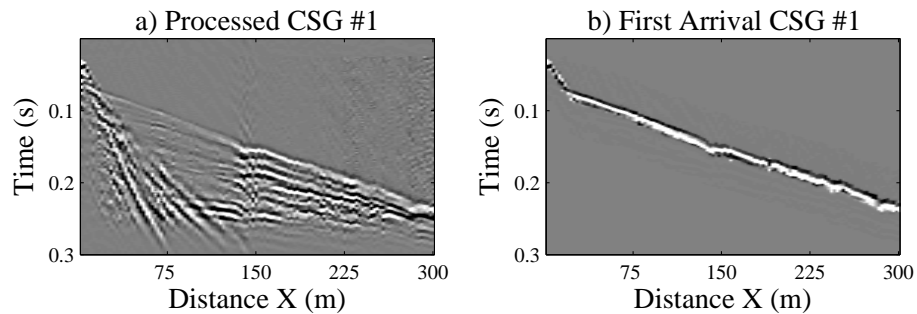


Figure 25.11: a) Processed CSG #1, and b) windowed first arrivals. Figure from Yu et al. (2020).



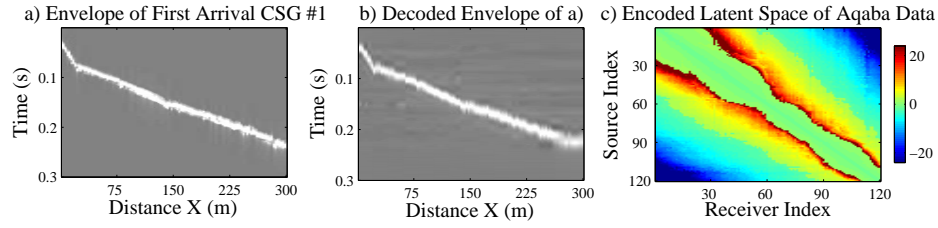


Figure 25.12: a) Envelopes of first arrivals in Figure 25.11b, b) the envelopes decoded by a trained 1D AE, and the c) the encoded latent-space points  $\mathbf{z}$  for all of the traces and a function of the source and receiver coordinates. Figure from Yu et al. (2020).

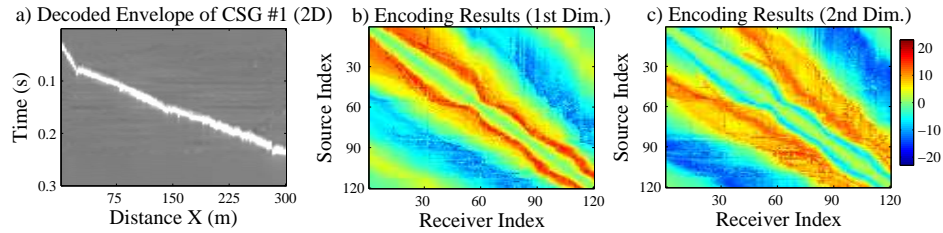


Figure 25.13: Similar to Figures 25.12a and 25.12c except a 2D latent space is used. Here, b) is for  $z_1$  and c) is for  $z_2$  in the latent space vector  $\mathbf{z} = (z_1, z_2)$ . Figure from Yu et al. (2020).

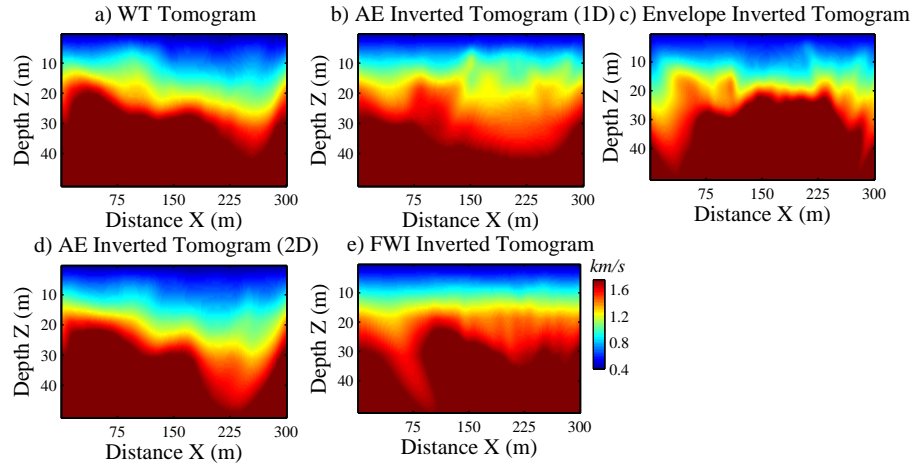


Figure 25.14: Tomograms inverted by a) the 50 Hz wave equation traveltime (WT) inversion method (Luo and Schuster, 1991a), b) AE inversion with a 1D latent space and a 50-Hz bandpass filter applied to the data, c) envelope inversion, d) AE inversion with a 2D latent space, and e) FWI. Figure from Yu et al. (2020).

tomogram by Hanafy et al. (2014).

The reconstructed early arrivals from each of the tomograms in Figure 25.14 are computed by

a finite-difference modeling method and the results are shown in Figure 25.15. Here, the predicted arrivals are subtracted from the actual arrivals to get the residual CSGs in Figure 25.16. It is clear that the 2D NML residual in Figure 25.16d has the smallest residual compared to the other methods.

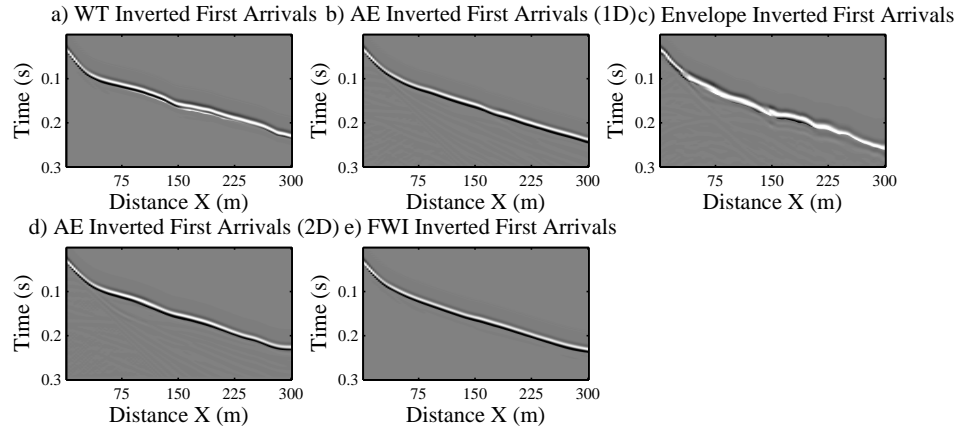


Figure 25.15: Windowed CSGs generated from the a) WT, b) 1D AE, c) envelope, d) 2D AE and e) FWI tomograms in the previous figure. These CSGs should closely match the early arrivals in the raw data in Figure 25.9. Figure from Yu et al. (2020).

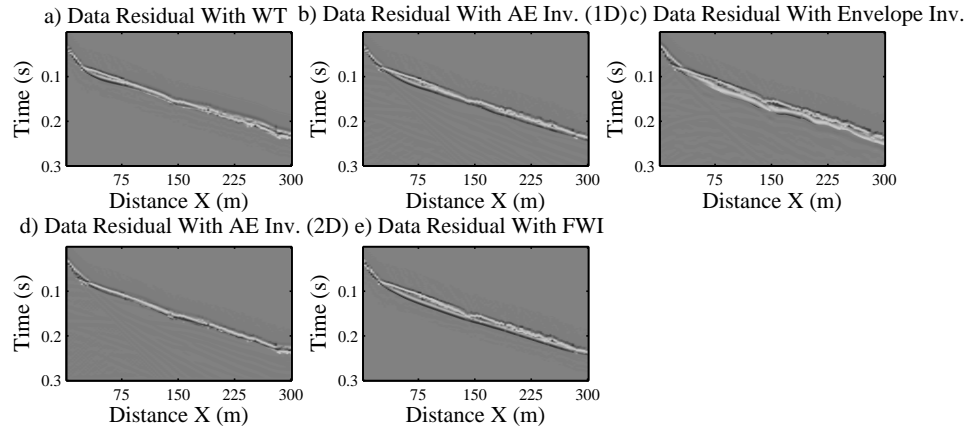


Figure 25.16: Residual CSGs associated with the data generated by the a) WT, b) 1D AE, c) envelope, d) 2D AE and e) FWI methods. Figure from Yu et al. (2020).

The data misfit curves with the five algorithms are presented in Figure 25.17. These curves show that the WT and 2D AE methods converge the quickest to a small data residual.

## 25.6 Summary

The theory of multidimensional NML is presented. NML uses machine learning to extract the skeletonized and important features of seismic data, and then Newton's laws are used to invert

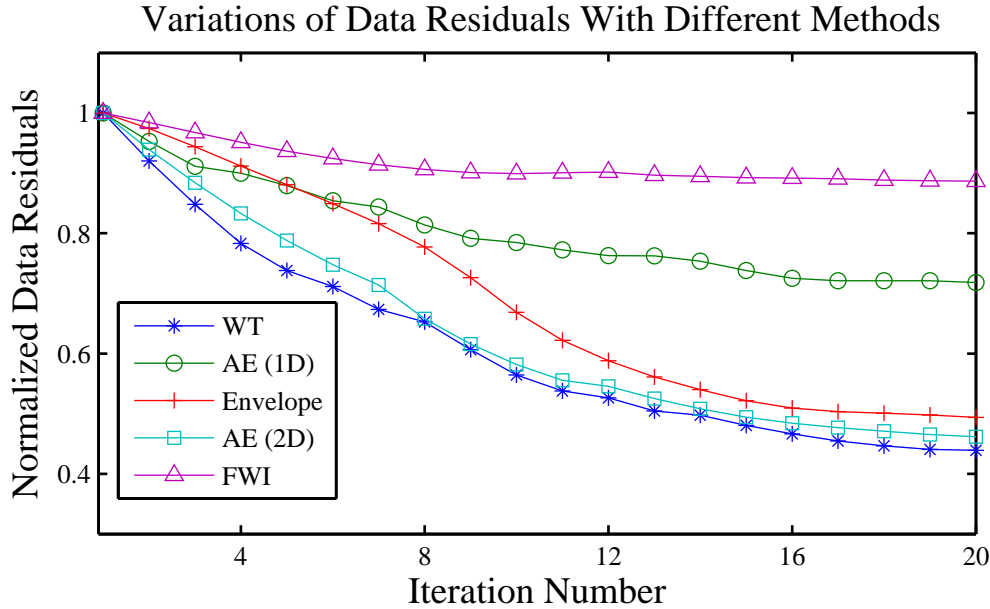


Figure 25.17: Comparisons of data residuals vs iteration number for different methods.

for the model parameters. This is a special case of skeletonized wave-equation inversion (Luo and Schuster, 1991b) where a ML method (Schuster, 2018) is used to generate the skeletonized data. For the examples presented in this chapter, the latent-space vectors in an AE are inverted to estimate the velocity model from refraction and crosswell data. In general any ML method such as PCA might be used to skeletonize the data.

Inversion of higher-dimensional latent-space vectors typically gives more accurate tomograms compared to inversion of lower-dimensional latent-space vectors. A multiscale approach is recommended where the velocity model of lower-dimensional vector is used as the starting model for NML with higher-dimensional latent vectors. The optimal dimension of the latent-space vector is estimated by increasing the dimension of the latent space until there is little change in the final data residual.

Empirical tests suggest that the advantages of NML over FWI are that it requires less memory and appears to be more robust. The main disadvantage is that, theoretically, NML provides less resolution than FWI for low-dimensional latent spaces. In this case, the NML tomogram can be used as the starting model for FWI. The testing of NML is in its early stages, and undoubtedly improvements will be needed in order for NML to be effective with complicated and noisy seismic traces. One such improvement is to include regularization of the latent-space vectors in the AE objective function (Oring et al., 2020).

## 25.7 Exercises

1. Devise the NML inversion algorithm for a linear PCA ML method. Derive the formula for the gradient.
2. Devise the NML inversion algorithm for a non-linear PCA ML method. Derive the formula for the gradient.
3. Derive the AE formulas for the objective function and the gradient where the regularization

term is  $\|\mathbf{z}\|^2$ . Discuss the benefits of using regularization for both the  $\mathbf{z}$  and the velocity-model parameters for NML inversion.

4. Derive the AE formulas for the objective function and the gradient where the regularization term is  $\sum_i (z_i - z_{i-1})^2$ . What is the physical meaning of this regularization term in terms of smoothness? Discuss the advantages of using this type of regularization for inversion in the context of the interpolation paper by Oring et al. (2020).
5. Derive the formula for the gradients of the AE objective function and data misfit functions using a total variation regularization (Estrela et al., 2016) of the latent space vectors and the velocity model vectors.
6. Devise an NML inversion algorithm using the U-Net. Discuss the benefits of using a U-Net architecture instead of an AE architecture for the NML method.

## 25.8 Connective Function

To compute the gradient  $\gamma(\mathbf{x})$ , a connective function is required to connect the perturbation of the LS vector  $\Delta\mathbf{z}$  with respect to the velocity perturbation  $\Delta v(\mathbf{x})$ . We define a connective function that measures the similarity between the observed and predicted seismic traces:

$$F(\tilde{\mathbf{z}}; v) = \int p_{\mathbf{z}-\tilde{\mathbf{z}}}^{obs}(\mathbf{x}_r, t; \mathbf{x}_s) p_{\mathbf{z}}^{pred}(\mathbf{x}_r, t; \mathbf{x}_s) dt, \quad (25.6)$$

where  $p_{\mathbf{z}}^{pred}(\mathbf{x}_r, t; \mathbf{x}_s)$  indicates the predicted trace recorded at the receiver location  $\mathbf{x}_r$  for a source at  $\mathbf{x}_s$ . The subscript vector  $\mathbf{z}$  represents the LS vector of the trace. Similarly,  $p_{\mathbf{z}-\tilde{\mathbf{z}}}^{obs}(\mathbf{x}_r, t; \mathbf{x}_s)$  represents the observed seismic trace for the same source-receiver location where the LS feature vector is  $\mathbf{z} - \tilde{\mathbf{z}}$ . Here,  $\tilde{\mathbf{z}}$  represents the shift between the observed and predicted feature vectors, which becomes zero ( $\tilde{\mathbf{z}} = (0, 0, \dots, 0)$ ) if the predicted velocity model is sufficiently accurate, otherwise  $\tilde{\mathbf{z}} \neq (0, 0, \dots, 0)$ .

The goal is to 1) find the optimal LS feature shift  $\tilde{\mathbf{z}} = (\Delta z_1, \Delta z_2, \dots, \Delta z_n)$  that maximizes equation 25.6; then 2) recover the accurate velocity model by finding the shift vector that minimizes the gradient of equation 25.6. The optimal shift can be found by solving equation  $\nabla F(\tilde{\mathbf{z}})|_{\tilde{\mathbf{z}}=\Delta\mathbf{z}} = 0$  as

$$\begin{aligned} \nabla F(\tilde{\mathbf{z}})|_{\tilde{\mathbf{z}}=\Delta\mathbf{z}} &= \left( \frac{\partial F}{\partial \tilde{z}_1}, \frac{\partial F}{\partial \tilde{z}_2}, \dots, \frac{\partial F}{\partial \tilde{z}_n} \right) \\ &= \int p_{\mathbf{z}}^{pred}(\mathbf{x}_r, t; \mathbf{x}_s) \left( \frac{p_{\mathbf{z}-\Delta\mathbf{z}}^{obs}}{\partial \tilde{z}_1}, \frac{p_{\mathbf{z}-\Delta\mathbf{z}}^{obs}}{\partial \tilde{z}_2}, \dots, \frac{p_{\mathbf{z}-\Delta\mathbf{z}}^{obs}}{\partial \tilde{z}_n} \right) dt = 0. \end{aligned} \quad (25.7)$$

We can re-write equation 25.7 as a system of equations

$$\begin{aligned} \frac{\partial F}{\partial \tilde{z}_1} &= \int p_{\mathbf{z}}^{pred} \frac{p_{\mathbf{z}-\Delta\mathbf{z}}^{obs}}{\partial \tilde{z}_1} dt = 0 \\ \frac{\partial F}{\partial \tilde{z}_2} &= \int p_{\mathbf{z}}^{pred} \frac{p_{\mathbf{z}-\Delta\mathbf{z}}^{obs}}{\partial \tilde{z}_2} dt = 0. \\ &\vdots \\ \frac{\partial F}{\partial \tilde{z}_n} &= \int p_{\mathbf{z}}^{pred} \frac{p_{\mathbf{z}-\Delta\mathbf{z}}^{obs}}{\partial \tilde{z}_n} dt = 0. \end{aligned} \quad (25.8)$$

Combining equation 25.8 with the multi-variable implicit function theorem (Guo, 2016), we can get

the gradient formula for each LS dimension as

$$\begin{bmatrix} \frac{\partial \Delta z_1}{\partial v} \\ \frac{\partial \Delta z_2}{\partial v} \\ \vdots \\ \frac{\partial \Delta z_n}{\partial v} \end{bmatrix} = \begin{bmatrix} \frac{\partial^2 F}{\partial \tilde{z}_1^2} & \frac{\partial^2 F}{\partial \tilde{z}_1 \partial \tilde{z}_2} & \cdots & \frac{\partial^2 F}{\partial \tilde{z}_1 \partial \tilde{z}_n} \\ \frac{\partial^2 F}{\partial \tilde{z}_2 \partial \tilde{z}_1} & \frac{\partial^2 F}{\partial \tilde{z}_2^2} & \cdots & \frac{\partial^2 F}{\partial \tilde{z}_2 \partial \tilde{z}_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 F}{\partial \tilde{z}_n \partial \tilde{z}_1} & \frac{\partial^2 F}{\partial \tilde{z}_n \partial \tilde{z}_2} & \cdots & \frac{\partial^2 F}{\partial \tilde{z}_n^2} \end{bmatrix}^{-1} \begin{bmatrix} \frac{\partial^2 F}{\partial \tilde{z}_1 \partial v} \\ \frac{\partial^2 F}{\partial \tilde{z}_2 \partial v} \\ \vdots \\ \frac{\partial^2 F}{\partial \tilde{z}_n \partial v} \end{bmatrix}, \quad (25.9)$$

where

$$\frac{\partial^2 F}{\partial \tilde{z}_k \partial v} = \int \frac{\partial p_{\mathbf{z}-\Delta \mathbf{z}}^{obs}(\mathbf{x}_r, t; \mathbf{x}_s)}{\partial \tilde{z}_k} \frac{\partial p_{\mathbf{z}}^{pred}(\mathbf{x}_r, t; \mathbf{x}_s)}{\partial v} dt, \quad (25.10)$$

and

$$\begin{aligned} \frac{\partial^2 F}{\partial \tilde{z}_k^2} &= \int \frac{\partial^2 p_{\mathbf{z}-\Delta \mathbf{z}}^{obs}(\mathbf{x}_r, t; \mathbf{x}_s)}{\partial \tilde{z}_k^2} p_{\mathbf{z}}^{pred}(\mathbf{x}_r, t; \mathbf{x}_s) dt, \\ \frac{\partial^2 F}{\partial \tilde{z}_k \partial \tilde{z}_i} &= \int \frac{\partial^2 p_{\mathbf{z}-\Delta \mathbf{z}}^{obs}(\mathbf{x}_r, t; \mathbf{x}_s)}{\partial \tilde{z}_k \partial \tilde{z}_i} p_{\mathbf{z}}^{pred}(\mathbf{x}_r, t; \mathbf{x}_s) dt. \end{aligned} \quad (25.11)$$

### 25.8.1 Gradient

Substituting equation 25.9 into equation 25.5, we can get the gradient formula of NML for an  $n$ -dimensional LS vector as

$$\gamma(\mathbf{x}) = - \sum_s \sum_r \left\langle \begin{bmatrix} \frac{\partial^2 F}{\partial z_1^2} & \frac{\partial^2 F}{\partial z_1 \partial z_2} & \cdots & \frac{\partial^2 F}{\partial z_1 \partial z_n} \\ \frac{\partial^2 F}{\partial z_2 \partial z_1} & \frac{\partial^2 F}{\partial z_2^2} & \cdots & \frac{\partial^2 F}{\partial z_2 \partial z_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 F}{\partial z_n \partial z_1} & \frac{\partial^2 F}{\partial z_n \partial z_2} & \cdots & \frac{\partial^2 F}{\partial z_n^2} \end{bmatrix}^{-1} \begin{bmatrix} \frac{\partial^2 F}{\partial z_1 \partial v(x)} \\ \frac{\partial^2 F}{\partial z_2 \partial v(x)} \\ \vdots \\ \frac{\partial^2 F}{\partial z_n \partial v(x)} \end{bmatrix}, \begin{bmatrix} \Delta z_1 \\ \Delta z_2 \\ \vdots \\ \Delta z_n \end{bmatrix} \right\rangle. \quad (25.12)$$

If we assume  $z_k$  and  $z_i$  are weakly correlated, then  $\frac{\partial^2 F}{\partial z_k \partial z_i} \approx 0$  for  $k \neq i$ . Then equation 25.12 can be re-written as

$$\begin{aligned} \gamma(\mathbf{x}) &= - \sum_s \sum_r \left\langle \begin{bmatrix} E_1 & 0 & \cdots & 0 \\ 0 & E_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & E_n \end{bmatrix}^{-1} \begin{bmatrix} \frac{\partial^2 F}{\partial z_1 \partial v(x)} \\ \frac{\partial^2 F}{\partial z_2 \partial v(x)} \\ \vdots \\ \frac{\partial^2 F}{\partial z_n \partial v(x)} \end{bmatrix}, \begin{bmatrix} \Delta z_1 \\ \Delta z_2 \\ \vdots \\ \Delta z_n \end{bmatrix} \right\rangle, \\ &= - \sum_s \sum_r \sum_k E_k \frac{\partial^2 F}{\partial \tilde{z}_k \partial v} \Delta z_k \\ &= - \sum_s \sum_r \int \frac{\partial p_{(z_1, z_2)}^{pred}(\mathbf{x}_r; \mathbf{x}_s)}{\partial v} \left( \sum_k \frac{\partial p_{(z_1 - \Delta z_1, z_2 - \Delta z_2)}^{obs}(\mathbf{x}_r; \mathbf{x}_s)}{\partial \tilde{z}_k} E_k \Delta z_k \right) dt. \end{aligned} \quad (25.13)$$

The Fréchet derivative  $\frac{\partial p}{\partial v}$  of the first-order acoustic equation can be written as (Chen and Schuster, 2020)

$$\frac{\partial p}{\partial v} = -2\rho v g_p(\mathbf{x}_r, t; \mathbf{x}, 0) * \nabla \cdot \mathbf{v}(\mathbf{x}, t; \mathbf{x}_s), \quad (25.14)$$

where  $\rho$  and  $v$  represent the density and propagation velocity, respectively. The particle velocity is indicated by  $\mathbf{v}$ , and  $g_p$  is the Green's function. Here,  $*$  means temporal convolution. Substituting equation 25.14 into equation 25.13 allows the multi-dimensional NML gradient to be expressed as

$$\gamma(\mathbf{x}) = \sum_s \sum_r \int dt (2\rho v g_p(\mathbf{x}_r, t; \mathbf{x}, 0) * \nabla \cdot \mathbf{v}(\mathbf{x}, t; \mathbf{x}_s)) \Delta p_{\mathbf{z}}(\mathbf{x}, t; \mathbf{x}_s), \quad (25.15)$$

where  $\Delta p_{\mathbf{z}}(\mathbf{x}, t; \mathbf{x}_s) = \sum_k \frac{\partial p_{(z_1 - \Delta z_1, z_2 - \Delta z_2)}^{obs}(\mathbf{x}_r; \mathbf{x}_s)}{\partial \tilde{z}_k} E_k \Delta z_k$  denotes the adjoint-source at the receiver location. For each LS feature dimension, a local adjoint-source is computed by weighting the LS feature difference  $\Delta z_k$  with the partial derivative of the shifted observed seismic trace  $\frac{\partial p_{\mathbf{z} - \Delta \mathbf{z}}^{obs}(\mathbf{x}_r; \mathbf{x}_s)}{\partial \tilde{z}_k}$ , and then scale it by using the weighting parameter  $E_k$ . The final adjoint-source  $\Delta p_{\mathbf{z}}(\mathbf{x}, t; \mathbf{x}_s)$  can be obtained by summing all the local adjoint-sources together. Once we have the gradient, the velocity model can be updated by using the steepest descent formula as

$$v(\mathbf{x})_{k+1} = v(\mathbf{x})_k + \alpha_k \gamma(\mathbf{x})_k, \quad (25.16)$$

where  $k$  represents the iteration index and  $\alpha_k$  indicates the step length.

## Chapter 26

# Automatic Differentiation

Automatic differentiation (AD), aka computational differentiation (Linnainmaa, 1976; Sambridge et al., 2007; Margossian, 2018; Sun et al., 2020), computes the exact value of the derivative of a complicated function. It uses the chain rule to break up the derivative of a complicated composite function  $f(g(h(x)))$  into a chain of simple derivatives: for example,  $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial h} \frac{\partial h}{\partial x}$ . The instructions for carrying out these simple derivatives are well known and can be transformed into coded instructions. If  $h(x) = x^2$  then the computer instruction would be to *compute  $dx^2/dx$  by multiplying the value of  $x$  by 2.* to get the exact value of  $\frac{\partial f(\mathbf{x})}{\partial x_i}$  up to floating-point roundoff. This is equivalent to the results presented in Chapter 6.3. However, AD is more general in that it is used to produce exact values of partial derivatives for almost any complicated function. AD is now commonly used to compute the gradients associated with both FANS and CNNs (Baydin et al., 2018).

### 26.0.1 Automatic Differentiation Algorithm

The automatic differentiation algorithm will be presented by computing the gradient  $\nabla g$  of

$$g(x, y, z) = (x + y) \cdot (y + z), \quad (26.1)$$

where  $(x, y, z)$  are variables. The standard procedure for getting the components of the gradient is to differentiate  $g(x, y, z)$  w/r to the independent  $x$ ,  $y$ , and  $z$  variables:

$$\frac{\partial g(x, y, z)}{\partial x} = y + z; \quad \frac{\partial g(x, y, z)}{\partial y} = x + 2y + z; \quad \frac{\partial g(x, y, z)}{\partial z} = x + y. \quad (26.2)$$

In a neural network, however, the expressions for the gradients of the objective function  $g$  are very complicated and their computation can be inefficient. Instead, we use the approach in the previous section where the chain rule was recursively used to find the partial derivatives of composite functions. This recursion is the key to computational efficiency because the derivative calculations from the shorter links of the chain are reused for computations in the longer parts.

The following four steps describe the automatic differentiation algorithm for obtaining the gradients of equation 26.1.

1. **Decomposition of  $g(\mathbf{x})$  into Primitive Functions.** The first step is to create a formula for  $g(x, y, z)$  to be a composition of primitive functions, where the analytical derivative of each primitive, i.e. simpler, function is already known. For example, we can redefine the terms of equation 26.1 as

$$p(x, y) = x + y; \quad q(y, z) = y + z, \quad (26.3)$$

so  $g(x, y, z)$  becomes the composite function

$$g = g(p(x, y), q(y, z)) = p \cdot q, \quad (26.4)$$

where  $g$  is now a composite function<sup>1</sup> of the functions  $p(x, y)$  and  $q(y, z)$ . A NN can have many layers so there is a multiple nesting of composition functions such as  $g = g(h(i(j(\mathbf{x})))) = g \circ h \circ i \circ j(\mathbf{x})$ , for the functions  $g(\mathbf{x})$ ,  $h(\mathbf{x})$ ,  $i(\mathbf{x})$ ,  $j(\mathbf{x})$ .

2. **Computational Graph.** Figure 26.1b is the *computational graph* that depicts the sequential flow of feedforward operations for computing the primitive functions in equation 26.4. The symbol in each node represents a mathematical operation that is applied to the variables that enter the node from the left. The output of this node operation gives the node variable, aka an intermediate variable. For example,  $p = x + y$  is the node variable for the top node in the middle hidden layer and  $q = y + z$  is for the one below it. Starting from the input layer on the left and executing the algebraic operations from left to right is denoted as feedforward modeling in Chapter 6.

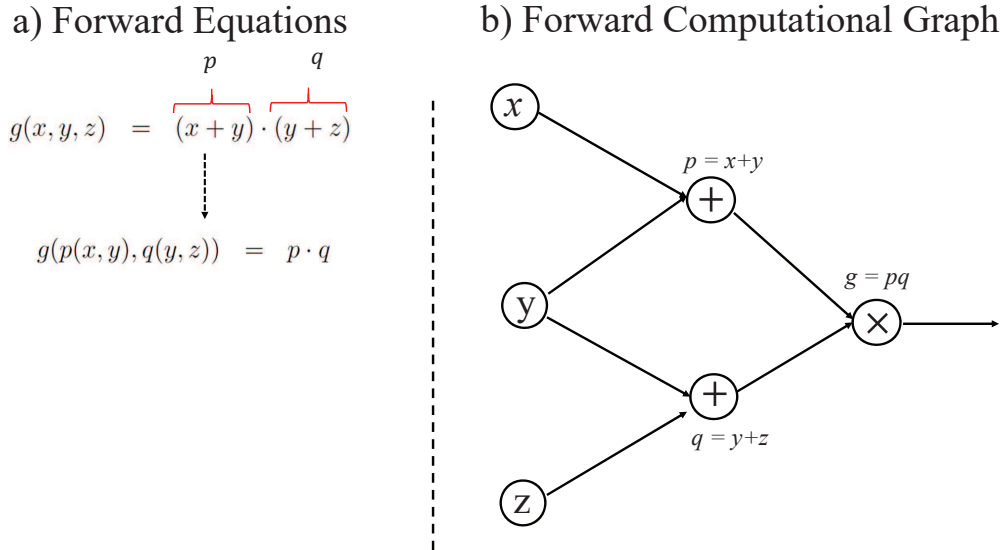


Figure 26.1: a) Feedforward composite equations and b) computational graph for computing equation 26.4 by AD.

3. **Fréchet Derivatives and the Chain Rule.** The goal of AD is to automatically compute the exact partial derivatives of the output  $g(x, y, z)$  w/r to the input variables  $\mathbf{x} = (x, y, z)$ . This is efficiently accomplished by breaking up  $g(x, y, z) = g(p, q)$  into a chain of primitive functions, where the *local* derivative of each one is known and its computation can be easily coded.

As an example, the chain rule applied to equation 26.4 is

$$\underbrace{\frac{\partial g}{\partial x}}_{\text{global}} = \underbrace{\frac{\partial g}{\partial p}}_{\text{regional}} \underbrace{\frac{\partial p}{\partial x}}_{\text{local}}, \quad (26.5)$$

<sup>1</sup>The notation for a composite function is defined as  $f = f(h(\mathbf{x})) = f \circ h(\mathbf{x})$ . Here,  $h$  is nested in the argument of  $f(\cdot)$ .



where the global derivative  $\partial g/\partial x$  characterizes the sensitivity of the output variable  $g$  to changes in the input variable  $x$ . This global derivative can be decomposed into a product of local and regional derivatives. The regional derivative  $\frac{\partial g}{\partial p}$  measures the sensitivity of  $g$  from the  $g$  node to the  $p$  node, where  $\overline{gp}$  spans a shorter path than  $\overline{gx}$  in Figure 26.1. An even shorter sensitivity path is that for the local derivative  $\frac{\partial p}{\partial x}$  which determines the sensitivity of  $p$  to changes to its neighboring node at  $x$ . Decomposing the global derivatives into a product of regional and local derivatives is the trick that makes AD work, as discussed in the section on back substitution!

The ensemble of colored backpropagation arrows and partial derivatives for all six nodes are depicted as in Figure 26.2b. For example, the blue reversed arrows denote the sensitivity paths for partial derivatives in equation 26.5. The blue dashed line intersects three nodes so three partial derivatives are required, one of which is equation 26.5 and the others are below.

$$\begin{aligned} \overbrace{\frac{\partial g}{\partial p}}^{\text{global}} &= \overbrace{\frac{\partial g}{\partial g}}^{\text{regional}} \overbrace{\frac{\partial g}{\partial p}}^{\text{local}} = q, \\ \overbrace{\frac{\partial g}{\partial g}}^{\text{global}} &= 1, \end{aligned} \quad (26.6)$$

The decomposition of the partial derivatives along the blue dashed line is a simple concatenation of local and regional derivatives. This is because only  $p(x, y)$  in  $g(p(x, y), q(y, z))$  is a function of  $x$ , so  $g(p)$  can be considered as a univariate function of  $x$  when  $y$  is fixed. Hence, the univariate chain rule is used for equation 26.6.

In contrast,  $g(p(x, y), q(y, z))$  is a multivariate function of  $y$  because both  $p(x, y)$  and  $q(y, z)$  depend on  $y$ . Therefore, the multivariate Taylor's theorem says that the partial derivative of  $g(p, q)$  w/r to  $y$  should be

$$\partial g/\partial y = \overbrace{\frac{\partial g}{\partial p}}^{\text{regional}} \overbrace{\frac{\partial p}{\partial y}}^{\text{local}} + \overbrace{\frac{\partial g}{\partial q}}^{\text{regional}} \overbrace{\frac{\partial q}{\partial y}}^{\text{local}}, \quad (26.7)$$

where the derivatives are denoted near the green and red arrows in Figure 26.2b. Equation 26.7 contains an extra term on the RHS compared to equation 26.5. This makes sense from the Figure 26.2b diagram because a small variation  $\delta y$  at the  $y$  node will affect the values of both the  $p$  and  $q$  variables, which in turn, will affect  $g$ . Therefore,  $\partial g/\partial y$  in equation 26.7 will be a weighted sum of the changes in the  $q$  and  $p$  nodes. Anytime we take a derivative of a bivariate function  $g(p(y), q(y))$  with respect to  $y$  then there will be two arrows (green and red) bifurcating from the  $y$  node, which then come together at the  $g$  node in Figure 26.2b. See exercise 4.

In summary, Figure 26.2b can be used to guide the chain rule in forming the partial derivatives

## a) Backpropagation Equations

$$\begin{aligned}
 g(x, y, z) &= \overbrace{(x+y)}^p \cdot \overbrace{(y+z)}^q \\
 g(p(x, y), q(y, z)) &= p \cdot q \\
 \frac{\partial g(p, q)}{\partial g} &= 1; \\
 \frac{\partial g(p, q)}{\partial p} &= q; \\
 \frac{\partial g(p, q)}{\partial q} &= p; \\
 \frac{\partial g(p, q)}{\partial x} &= \frac{\partial g(p, q)}{\partial p} \frac{\partial p}{\partial x} = \hat{g}_p; \\
 \frac{\partial g(p, q)}{\partial y} &= \frac{\partial g(p, q)}{\partial p} \frac{\partial p}{\partial y} + \frac{\partial g(p, q)}{\partial q} \frac{\partial q}{\partial y} = \hat{g}_p + \hat{g}_q; \\
 \frac{\partial g(p, q)}{\partial z} &= \frac{\partial g(p, q)}{\partial q} \frac{\partial q}{\partial z} = \hat{g}_q,
 \end{aligned}$$

## b) Backpropagation Computational Graph

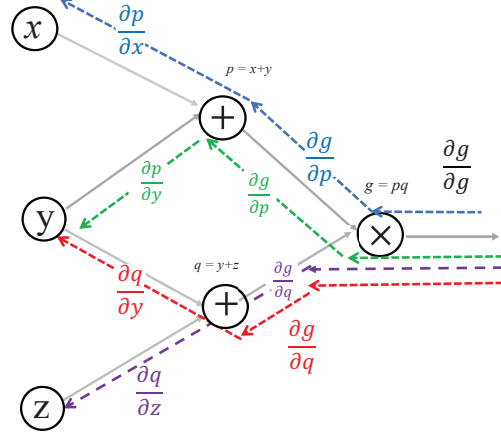


Figure 26.2: a) Backpropagation equations and b) computational graph associated with equations 26.8-26.13. The local partial derivatives on the RHS can easily be computed by differentiating equations 26.3-26.4.

of  $g(p, q)$  w/r to each of the node variables:

$$\hat{g}_g = \frac{\partial g(p, q)}{\partial g} = 1; \quad (26.8)$$

$$\hat{g}_p = \frac{\partial g(p, q)}{\partial p} = q; \quad (26.9)$$

$$\hat{g}_q = \frac{\partial g(p, q)}{\partial q} = p; \quad (26.10)$$

$$\hat{g}_x = \frac{\partial g(p, q)}{\partial x} = \frac{\partial g(p, q)}{\partial p} \frac{\partial p}{\partial x} = \hat{g}_p; \quad (26.11)$$

$$\hat{g}_y = \frac{\partial g(p, q)}{\partial y} = \frac{\partial g(p, q)}{\partial p} \frac{\partial p}{\partial y} + \frac{\partial g(p, q)}{\partial q} \frac{\partial q}{\partial y} = \hat{g}_p + \hat{g}_q, \quad (26.12)$$

$$\hat{g}_z = \frac{\partial g(p, q)}{\partial z} = \frac{\partial g(p, q)}{\partial q} \frac{\partial q}{\partial z} = \hat{g}_q, \quad (26.13)$$

where the  $6 \times 1$  vector of global Fréchet derivatives  $\alpha = (\hat{g}_g, \hat{g}_p, \hat{g}_q, \hat{g}_x, \hat{g}_y, \hat{g}_z)$  are also known as adjoint variables. Note, there are six nodes in Figure 26.2b so there are six node variables and six partial derivatives of  $g(p, q)$  with respect to the node variables<sup>2</sup>. Each of these adjoint variables can be considered to be an unknown, and they can be found by solving the six equations of constraint 26.8-26.13. The derivatives on the far left are the *global* derivatives

<sup>2</sup>The general procedure for constructing the partial derivative  $\partial g / \partial w^{[n]}$  for any node variable  $w^{[n]}$  in the  $n^{th}$  layer of a NN is to write it as  $\partial g / \partial w^{[n]} = (\partial g / \partial w^{[n+1]})(\partial w^{[n+1]} / \partial w^{[n]})$  because  $\partial w^{[n+1]} / \partial w^{[n]}$  can be explicitly computed and  $\partial g / \partial w^{[n+1]}$  is obtained from the calculations in the neighboring deeper layer.

and the products to the right are formed by multiplying a regional derivative with a local derivative according to the chain rule. This is the key trick, the local derivatives can be computed from the analytical derivatives<sup>3</sup> of equation 26.4 and the global derivatives can be computed by the back substitution procedure discussed below.

Equations 26.8-26.13 can be rearranged as the matrix-vector product  $\mathbf{A}\boldsymbol{\alpha} = \boldsymbol{\beta}$ :

$$\overbrace{\begin{bmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 & -1 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}}^{\mathbf{A}} \overbrace{\begin{pmatrix} \hat{g}_z \\ \hat{g}_y \\ \hat{g}_x \\ \hat{g}_q \\ \hat{g}_p \\ \hat{g}_g \end{pmatrix}}^{\boldsymbol{\alpha}} = \overbrace{\begin{pmatrix} 0 \\ 0 \\ 0 \\ p \\ q \\ 1 \end{pmatrix}}^{\boldsymbol{\beta}}. \quad (26.14)$$

This is an upper-triangular matrix, so back substitution starting from the bottom row will provide the values of the Fréchet derivatives at each node for any input vector  $\boldsymbol{\beta}$  computed from  $p = x + y$  and  $q = y + z$ . These will agree with the components of the gradient computed by equation 26.2. See exercise 8.

4. **Back substitution=Recursive Backpropagation.** Backward substitution of the upper-triangular matrix  $\mathbf{A}$  starts at the bottom row, solves for the unknown global variable  $\alpha_N$  in the last row, and then reuses this solution in solving for  $\alpha_{N-1}$  in the second-to-last row. This process is recursively repeated until all of the components of the partial-derivative vector  $\boldsymbol{\alpha}$  are solved for. This is equivalent to the recursive procedure in Chapter 6.3 for computing the Fréchet derivatives of a neural network. Start at the last layer, find the Fréchet derivatives, and reuse this information for computing the Fréchet derivatives at the neighboring shallower layer. This procedure is repeated backward towards shallower layers until it stops at the input layer.

A Python code that implements the AD for obtaining the gradient of  $g = (x + y)(y + z)$  is given below.

---

<sup>3</sup>For example,  $\partial p / \partial x = 1$ ,  $\partial p / \partial y = 1$ ,  $\partial q / \partial y = 1$  and  $\partial q / \partial z = 1$ .

**Code 26.0.1.** *Python Automatic Differentiation of  $g(x, y, z) = (x + y) \cdot (y + z)$* 

```

# Python code computes derivatives of G=(X + Y)*(Y + Z) by AD
# Written by Tushar Gautam
import numpy as np

X = 3.0
Y = 2.0
Z = 3.0

AD = auto_diff(X, Y, Z) # Defining object AD for the class auto_diff
dGwrtX, dGwrtY, dGwrtZ = AD.grad() # Retrieving derivatives from the method grad
# Printing the computed derivatives
print("dG/dX is ", dGwrtX)
print("dG/dY is ", dGwrtY)
print("dG/dZ is ", dGwrtZ)

# Output from the above print statements
# dG/dX is  5.0
# dG/dY is  10.0
# dG/dZ is  5.0

class auto_diff:
    '''
    Defining a class which computes gradients using automatic differentiation
    '''
    def __init__(self, X, Y, Z):
        '''
        Initializing variables
        '''
        self.X = X # Input global variable X
        self.Y = Y # Input global variable Y
        self.Z = Z # Input global variable Z
        self.G = (X + Y) * (Y + Z) # Variable used to define equation from global variables
        self.P = X + Y # Defining local variable P
        self.Q = Y + Z # Defining local variable Q

    def grad(self):
        '''
        Method used to calculate gradients
        '''
        dGwrtP = self.Q # Local Derivative (dG/dP)
        dGwrtQ = self.P # Local Derivative (dG/dQ)

        dPwrtX = 1 # dP/dX
        dPwrtY = 1 # dP/dY
        dPwrtZ = 0 # dP/dZ

        dQwrtX = 0 # dQ/dX
        dQwrtY = 1 # dQ/dY
        dQwrtZ = 1 # dQ/dZ

        # Using multivariate Taylor's Theorem
        dGwrtX = dGwrtP * dPwrtX + dGwrtQ * dQwrtX
        dGwrtY = dGwrtP * dPwrtY + dGwrtQ * dQwrtY
        dGwrtZ = dGwrtP * dPwrtZ + dGwrtQ * dQwrtZ

        return dGwrtX, dGwrtY, dGwrtZ

```

## 26.1 Modeling, Automatic Differentiation and Inversion

Automatic differentiation can also be used to find the Fréchet derivatives associated with seismic data computed by finite-difference solutions to the wave equation. These Fréchet derivatives are then used in the steepest descent formula for inverting the velocity model from seismic data. The next section shows how the Fréchet derivatives can be computed by automatic differentiation of numerical solutions to the wave equation.

### 26.1.1 Automatic Differentiation for the One-way Wave Equation

We now show how AD can be used to compute Fréchet derivatives from the one-way wave equation

$$\frac{\partial d(z, t)}{\partial t} = c(z) \frac{\partial d(z, t)}{\partial z}, \quad (26.15)$$

where the acoustic velocity  $c(z)$  only varies in the  $z$ -coordinate and  $u(z, t)$  is the pressure field. An explicit finite-difference solution to this equation (Yilmaz, 2001) is given by

$$d_i^{t+1} = d_i^t - \alpha_i (d_{i+1}^t - d_i^t), \quad i \in \{1, 2, \dots, N\}, \quad (26.16)$$

where a 1st-order forward-difference approximation is used for both the temporal and spatial derivatives. Here,  $c_i$  is the propagation velocity at the  $i^{th}$  gridpoint,  $\alpha_i = c_i \Delta t / \Delta z$ , and  $d_i^t$  is the pressure field at the  $i^{th}$  node at time step  $t$ . The goal is to compute the Fréchet derivative  $\frac{\partial d(z_i, t)^{t+1}}{\partial c_i}$  at each grid point  $z_i$ . More generally, this same procedure can be applied to finite-difference solutions to the acoustic 2D wave equation described in Appendix 26.4.

Two stencils associated with the forward FD equation 26.16 are plotted in Figure 26.3a, and their computational graphs are in Figure 26.3b. In this case we can denote the intermediate variables for the red stencil pivoted at  $z_i$  as

$$\begin{aligned} p_i^t &= d_{i+1}^t - d_i^t, \\ q_i^t &= p_i^t \cdot \alpha_i, \\ r_i^t &= q_i^t + d_{i+1}^t, \\ d_i^{t+1} &= r_i^t. \end{aligned} \quad (26.17)$$

For convenience, we set  $g = d_i^{t+1}$ ,  $p = p_i^t$ ,  $q = q_i^t$ ,  $r = r_i^t$  and assume  $\Delta t / \Delta z = 1$  so  $\alpha_i = c_i$ . In this case, the sensitivity path for the Fréchet derivative  $\partial d_i^{t+1} / \partial c_i$  is along the red dashed arrow in Figure 26.3b. Here, we can decompose the global derivatives into products of regional and local derivatives:

$$\hat{g}_r = \frac{\partial g(r)}{\partial r} = 1; \quad (26.18)$$

$$\hat{g}_q = \frac{\overbrace{\frac{\partial g(r(d, q))}{\partial r}}^{\text{regional}} \overbrace{\frac{\partial r(d, q)}{\partial q}}^{\text{local}}}{\partial q} = 1; \quad (26.19)$$

$$\hat{g}_v = \frac{\overbrace{\frac{\partial g(r(d, q))}{\partial q}}^{\text{regional}} \overbrace{\frac{\partial q(v)}{\partial v}}^{\text{local}}}{\partial v} = p \hat{g}_q, \quad (26.20)$$

where the local derivatives  $\frac{\partial r(d, q)}{\partial q} = 1$  and  $\frac{\partial q(v)}{\partial v} = p$  are derived by differentiating the expressions in equations 26.17.

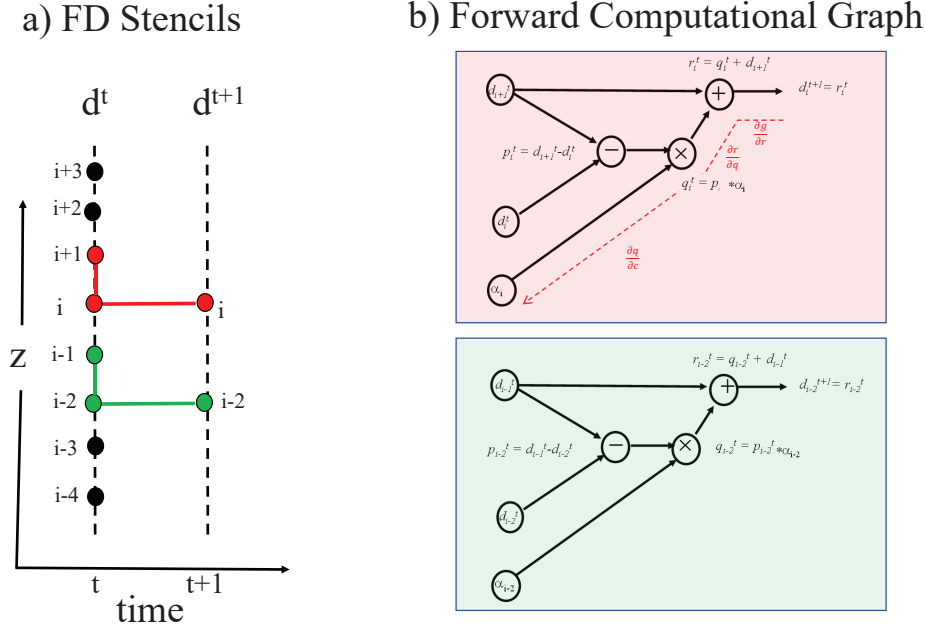


Figure 26.3: a) Finite-difference stencils pivoted at time step  $t$  and depths  $z_i$  and  $z_{i-2}$  and b) computational graphs for the red and green stencils. Here  $\alpha_i = v_i$  because we assume  $\Delta x / \Delta t = 1$ .

Equations 26.18-26.20 can be expressed as the system of equations  $\mathbf{A}\alpha = \beta$  with the upper-triangular matrix:

$$\overbrace{\begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}^{\mathbf{A}} \overbrace{\begin{pmatrix} \hat{g}_v \\ \hat{g}_q \\ \hat{g}_r \end{pmatrix}}^{\alpha} = \overbrace{\begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}}^{\beta}. \quad (26.21)$$

which can be solved for  $\hat{g}_v$  for all points  $z_i$  for  $i \in \{1, 2, \dots, N\}$ . Note, equation 26.20 ignores the implicit dependency of the field variable  $d_i^t$  on the velocity model  $c_i$ . Otherwise taking the derivative of  $g$  w/r to  $v$  would also involve taking the derivative of both  $p$  and  $\alpha$  in the  $q_i^t$  node. Ignoring this bifurcated dashed line is akin to the Born approximation<sup>4</sup> (see Exercise 13).

Figure 26.4 depicts the computational diagrams for FD solutions to the wave equation, which resemble one of the recurrent neural networks (Hughes et al., 2019) discussed in Chapter 12. However, the FD solution does not contain the thresholding functions inherent in a recurrent neural network.

### 26.1.2 Automatic Differentiation and Waveform Inversion

The Fréchet derivative  $\hat{g}_v = \partial g_i^t / \partial v_j$  computed at every spatial gridpoint and time sample is to get the optimal velocity model  $v_i$  that accurately predicts the data  $\bar{d}_i^t$  recorded on the surface. This

<sup>4</sup>Another plausibility argument is to apply a velocity perturbation to the acoustic wave equation. The weak-scattering assumption ignores the higher-order terms in the perturbed wave equation, so it is identical to the one-way wave equation in equation 26.15, except for a source term on the RHS. Hence, computing the Fréchet derivatives associated with equation 26.15 is equivalent to assuming the Born approximation.

## FD Computational Graph = Recurrent Neural Network Diagram

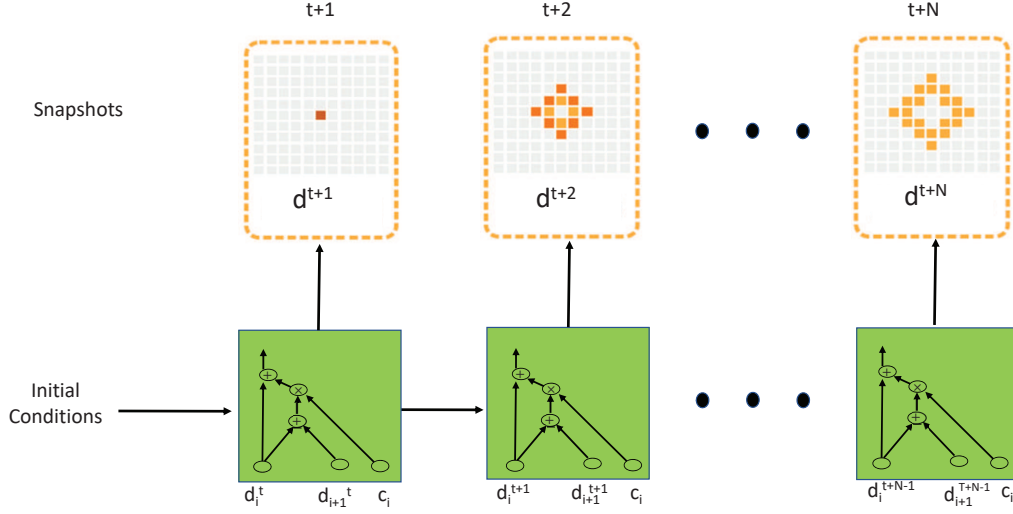


Figure 26.4: Computational graphs from Figure 26.3 arranged in order of increasing time steps to the right so that it resembles the architecture of a recurrent neural network. The top row of images correspond to expanding wavefronts that get larger with increasing time.

procedure (Virieux and Operto, 2009; Sun et al., 2020) is known as full waveform inversion (FWI). For convenience we will assume a 1D velocity model along a line in  $z$  and the recorded data  $\bar{d}_i^t$  are along that same line, just like the model used in the previous section. These data take the form of seismograms such as the one depicted in Figure 26.8b and are recorded along, for example, a well with the source at the surface. A shot gather of traces is known as a vertical seismic profile (Yilmaz, 2001). We shall denote the recorded seismograms as  $\bar{d}_i^t$ . The FWI  $L_2$  misfit function is

$$\epsilon = \frac{1}{2} \sum_t \sum_l \delta_l (d_l^t - \bar{d}_l^t)^2, \quad (26.22)$$

where the summations are over the discrete time  $t$  and offset  $l$  indices of the traces, and  $\delta_l = 1$  if  $l$  is an index for a recorded trace, otherwise  $\delta_l = 0$ . The velocity model is updated by an iterative gradient method, such as the steepest descent formula

$$\begin{aligned} v_i^{(k+1)} &= v_i^{(k)} - \alpha \frac{\partial \epsilon}{\partial v_i}, \\ &= v_i^{(k)} - \alpha \sum_t \sum_l (d_i^t - \bar{d}_l^t) \frac{\partial d_l^t}{\partial v_i}, \end{aligned} \quad (26.23)$$

where  $g_v$  in equation 26.21 is exactly the Fréchet derivative  $\frac{\partial d_l^t}{\partial v_i}$  in the steepest descent formula.

The benefit of this approach compared to traditional FWI is that a general AD program can be used to compute the components of the gradient vector for any type of wave equation (Sun et al., 2020). AD also saves a great deal of time in manual coding compared to coding the explicit physics of FWI. The drawback is that it is reported to sometimes require much more CPU time than the

conventional backpropagation procedure for FWI. In addition, AD is blind to the physics of the problem, so it cannot easily be adapted to expedite the convergence rate by utilizing innovations from the physics.

## 26.2 Summary

The automatic differentiation procedure is presented which efficiently computes the Fréchet derivatives of a complicated objective function. The key idea is to decompose the objective function into a composite chain of primitive functions, where there is an analytical formula for the local derivative of each composite function. Each primitive function is characterized by an intermediate variable at a node of the composite chain. The partial derivatives of the objective function with respect to the node variables are then expressed as a nested chain of local and regional derivatives, where the value of the global derivative at each node is saved for reuse at longer portions of the chain. Reusing these computed derivatives is the key to AD's computational efficiency. For example, AD only costs  $O(NM^2)$  computations to get all of the Fréchet derivatives of an  $N$  link chain, where it costs  $O(M^2)$  calculations to compute a local derivative. If the global derivatives at each link are not recycled then the average computational cost is  $O(NM^2)/2$  to get a global derivative at any node; here, it is assumed that the 2D model is an  $M \times M$  grid of points. Thus, the no-recycle computation of the global derivatives at every node will be  $O(N^2M^2)/2$ . It is reported that AD is used to efficiently compute the gradient values with Pytorch and other machine learning software.

The physics of wave propagation or backpropagation does not need to be understood in order to write or implement a black-box AD code for seismic inversion. However, neglecting the fundamentals of optimization theory and the physics of wave propagation will likely prevent the user from identifying and remedying problems with convergence and solution accuracy. As difficult as it might be, similar insights should be sought when attempting to implement AD for any inversion problem in geoscience.

## 26.3 Exercises

1. If

$$f(z) = \frac{1}{1 + e^{wz}}; \quad g(z) = wz. \quad (26.24)$$

What is the formula for  $f(g(z))$ ?

2. Draw the feedforward computational diagram for  $f(g(z))$  in equation 26.24.
3. Take the derivative of  $g(z) = \frac{1}{1+e^{-wz}}$  w/r to  $z$ . Compare it to the derivative of  $g(h(z))$  using the chain rule where  $g(z) = 1/(1 + e^{-z})$  and  $h(z) = wz$ .
4. Given a multivariate function  $f(x, y)$  and two variables  $x = x(t)$  and  $y = y(t)$ , use Taylor's theorem to show that

$$\frac{df(x, y)}{dt} = \frac{df}{dx} \frac{dx}{dt} + \frac{df}{dy} \frac{dy}{dt}. \quad (26.25)$$

Show how this can be used to derive equation 26.12.

5. For  $f(x, y) = \sqrt{x^2 + y^2}$  and  $g(x, y) = \ln x$ , draw the feedforward computational diagram for  $f(g(x, y))$ . Do the same for  $f(f(x, y))$ .
6. Rewrite equation 26.1 as

$$g(x, y, z) = xy + xz + y^2 + yz, \quad (26.26)$$



where  $h(x, y) = xy$ ,  $i(x, z) = xz$ ,  $j(y) = y^2$ , and  $k(y, z) = yz$ . Is  $g = g(h(x, y), i(x, z), j(y), k(y, z))$  a composite function? Draw the computational graph and compare to the one drawn in Figure 26.1. Which one is more efficient in terms of computations for doing feedforward modeling?

7. Use the chain rule to derive the gradient of  $g = g(h(x, y), i(x, z), j(y), k(y, z))$  in equation 26.26. Compare the result to the formulas after taking derivatives of  $g(x, y)$  w/r to  $x, y, z$  in equation 26.26.
8. Show that the values of the gradient in equation 26.2 agree with those computed by back substitution in equation 26.14.
9. Read Appendix 27 to acquaint yourself with seismograms and the characteristics of wave propagation. Run the MATLAB code 26.4 in Appendix 26.4 to compute snapshots of the evolving wavefield. Identify the direct waves and reflected waves in the snapshots. Use the plotting program *plot* in MATLAB to plot seismograms. Identify the reflections and direct waves.
10. The comments in the MATLAB code 26.4 describe the stability and dispersion conditions. Check to see if these conditions have been violated. Have they been violated? Now double the value of  $dx$ . What happens to the wavelet in the seismograms?
11. Construct the composite function for a single time step in a 1D version of the finite-difference formula in equation 26.30 and Figure 11. Also draw the computational graph for this composite formula.
12. From exercise 11, draw the reverse arrows depicting the Fréchet derivatives  $\partial P / \partial v_i$ , where  $v_i$  is the velocity at the  $i^{th}$  grid point in the 1D velocity model. Write down the equations of constraint similar to those in equations 26.8-26.13.
13. Make the replacements  $d \rightarrow d + \delta d$  and  $c \rightarrow c + \delta c$  in equation 26.15. Assume that  $\delta c$  is small enough so that the scattered field  $\delta d$  from this velocity perturbation is small as well. Now assume that the quadratic perturbation terms in the perturbed wave equation can be neglected, which is called the weak-scattering approximation. Discuss how this weak-scattering wave equation leads to the conclusion that AD applied to the wave equation in equation 26.15 implicitly assumes weak scattering.

## 26.4 Appendix: Finite Difference Solution to the Wave Equation

The 2D acoustic wave equation for a medium with no density variations is given by:

$$\frac{\partial^2 p(\mathbf{x}, t')}{\partial x^2} + \frac{\partial^2 p(\mathbf{x}, t')}{\partial z^2} - \frac{1}{c(x, z)^2} \frac{\partial^2 p(\mathbf{x}, t')}{\partial t'^2} = f(\mathbf{x}, t'), \quad (26.27)$$

where  $c(x, z)$  is the velocity model,  $p(\mathbf{x}, t')$  is the pressure field, and  $f(\mathbf{x}, t')$  is the inhomogeneous source term. The continuous wave equation and its solution can be discretized onto an evenly sampled grid in the space and time domains such that

$$\begin{aligned} (x, z, t') &\longleftrightarrow (i\Delta x, j\Delta z, t\Delta t) \text{ where } i, j, t \text{ are integers} \\ p(x, z, t') &\longleftrightarrow p_{ij}^t, \\ f(x, z, t') &\longleftrightarrow f_{ij}^t, \\ c(x, z) &\longleftrightarrow c_{ij}, \end{aligned} \quad (26.28)$$

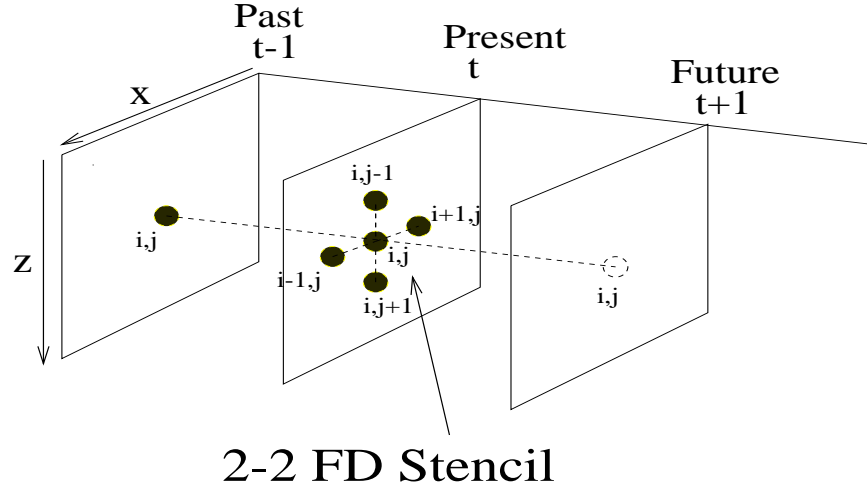


Figure 26.5: Depiction of a 2-2 finite-difference (FD) stencil for the 2D acoustic wave equation. The future value of the pressure at the  $(i, j)$  node (open dashed circle) is computed from the present and past values of the pressure that neighbor the  $(i, j)$  node at the present time  $t$ . The stencil can be shifted within the  $t$  plane to compute all of the pressure values within the  $t + 1$  plane. The pressure values at the boundaries of these planes must be specified.

and for convenience we will assume that the vertical gridpoint spacing  $\Delta z$  is the same as the horizontal spacing  $\Delta x$ . The discrete solution can be visualized with a 3D-data cube (see Figure 26.5) with the  $x$ ,  $z$ , and  $t$  axes. The depth of the data cube is given by  $M\Delta z = D$ , the width is  $N\Delta x = X$ , and the temporal extent is  $L\Delta t = T$ , where  $M$ ,  $N$  and  $L$  are integers.

Pivoting (i.e., evaluating) the pressure field at  $(i, j, t)$  and approximating the second-order derivatives in equation 26.27 by second-order correct central-difference approximations gives

$$\begin{aligned} \frac{\partial^2 p}{\partial t'^2} &\longleftrightarrow [p_{ij}^{t+1} - 2p_{ij}^t + p_{ij}^{t-1}]/(\Delta t)^2 \\ \frac{\partial^2 p}{\partial x^2} &\longleftrightarrow [p_{i+1j}^t - 2p_{ij}^t + p_{i-1j}^t]/(\Delta x)^2 \\ \frac{\partial^2 p}{\partial z^2} &\longleftrightarrow [p_{ij+1}^t - 2p_{ij}^t + p_{ij-1}^t]/(\Delta x)^2, \end{aligned} \quad (26.29)$$

yields the discretized wave equation and its associated stencil is shown in Figure 26.5. The time-stepping scheme for these discrete equations is given by

$$\begin{aligned} &\text{for } t = 2 : nt \\ &\quad p_{ij}^{t+1} = 2p_{ij}^t - p_{ij}^{t-1} + a[p_{i+1j}^t - 2p_{ij}^t + p_{i-1j}^t] + a[p_{ij+1}^t - 2p_{ij}^t + p_{ij-1}^t] - (\Delta x)^2 a f_{ij}^t, \\ &\text{end} \end{aligned} \quad (26.30)$$

where  $a = (c_{ij}\Delta t/\Delta x)^2$ . Here, we assume that the initial conditions are  $p(\mathbf{x}, t' = 0)$  and  $\partial p(\mathbf{x}, t' = 0)/\partial t'$  for all  $(x, z)$ , so that the source wavefield is generated by the body-force term  $f_{ij}^t$ .

Figure 26.7 depicts the high-level computational graph for computing the pressure field  $P^{t+1}$  from two earlier values  $P^{t+1}$  and  $P^{t+1}$  and the velocity model  $v$ . The Laplacian  $\nabla^2$  represents the algebraic formula in equation 26.30, which can be decomposed into a detailed computational

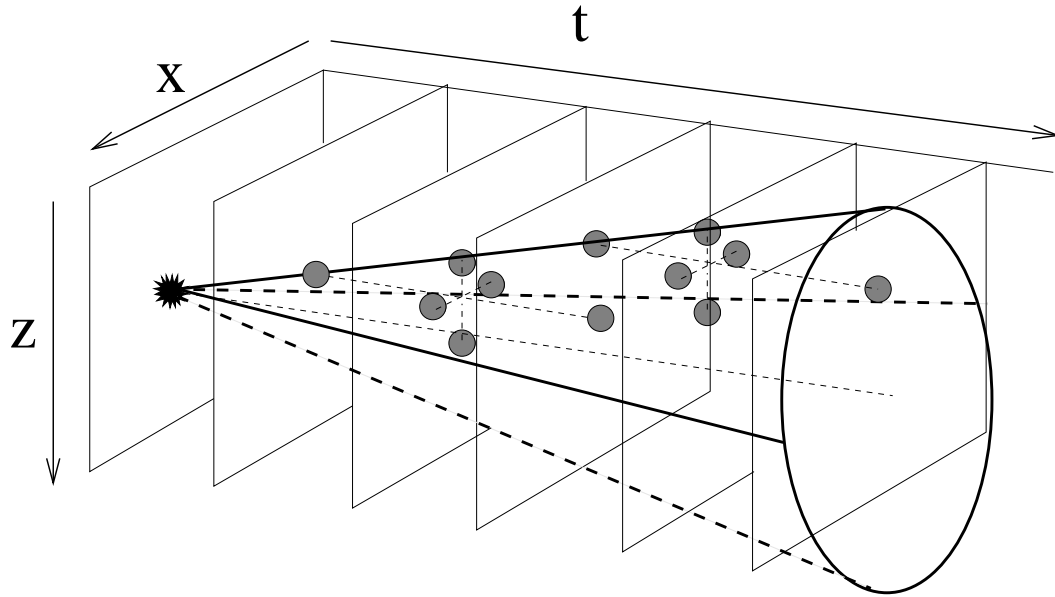


Figure 26.6: Depiction of forward light cone generated by a FD solution (several stencils are depicted) to the wave equation for a point source at depth.

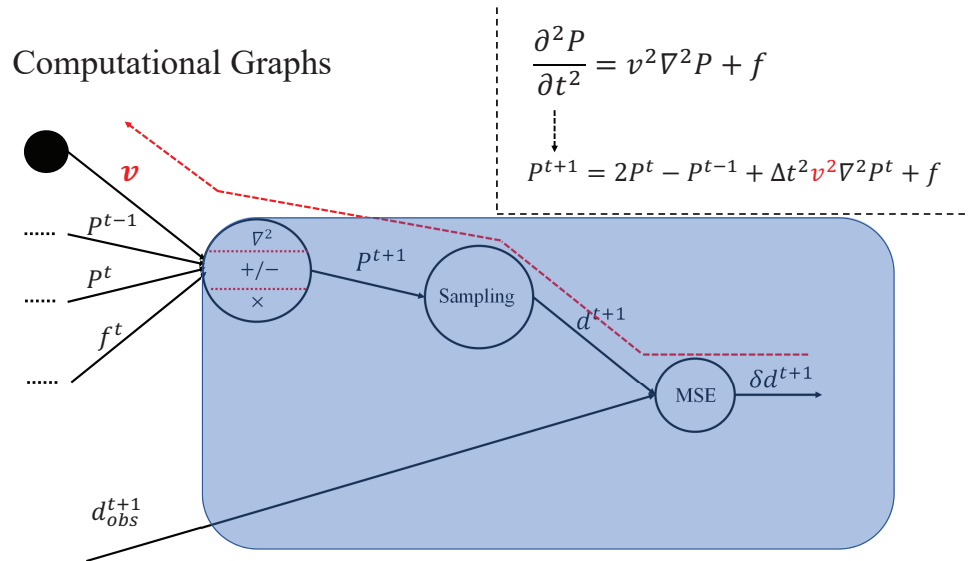


Figure 26.7: High-level computational graph for computing the pressure fields  $P^t$  after specifying the velocity model  $v(\mathbf{x})$ , denoted by the red letter  $v$ , and source wavelet  $f^t$ . Here,  $d^t = P(x, 0, t)$  is the predicted pressure field at the geophone line for the offset coordinate  $x$  and  $z = 0$ .

graph for a composite function. See exercise 11-12. The data recorded at the surface are denoted as  $d_{obs}^t$ , and are subtracted from the predicted data at the surface  $P(x, 0, t) \rightarrow d^t$  to get the residual  $r^t = d^t - d_{obs}^t$ . The squared residual is summed over all the trace offset locations and time samples to get the misfit function  $\epsilon$ , which will be used in the next section for an inversion algorithm.

The following MATLAB code computes the 2-2 FD solutions to the 2D acoustic wave equation with constant density.

**Code 26.4.1. MATLAB FD Solution to the 2D Acoustic Wave Equation**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% (NX,NZ,NT) - input- (Horizontal,Vertical) gridpt dims. of vel
% model & # Time Steps
% FR      - input- Peak frequency of Ricker wavelet
% BVEL    - input- NXxNZ matrix of background velocity model
% (dx,dt) - input- (space, time) sample intervals
% (xs,zs) - input- (x,z) coordinates of line source
% RICKER(NT) - input- NT vector of source time histories
% (p2,p1,p0) - calc.- (future,present,past) NXxNZ matrices of
% modeled pressure field
% (p0,p1) -output- Old and present pressure panels at time NT.
% REALDATA(NX,NT) -output- CSG seismograms at z=2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c=4.0;FRE=20;figure(1)
NX=300;NZ=NX; dx=c/FRE/20;dt=.5*dx/c;
xs=round(NX/2.3); zs=round(NX/2);NT=600;
t=[0:1:NT-1]*dt-0.95/FRE;RICKER=zeros(length(t));
RICKER= (1-t.*t * FRE^2 *pi^2 ) .*exp(- t.^2 * pi^2 * FRE^2 );
plot([0:NT-1]*dt,RICKER);
title('Ricker Wavelet');xlabel('Time (s)')
BVEL=ones(NX,NZ)*c;
BVEL(NX-round(NX/2):NX,:)= BVEL(NX-round(NX/2):NX,:)*1.2;
REALDATA=zeros(NX,NT);
p0=zeros(NX,NZ);p1=p0;p2=p0;
cns=(dt/dx*BVEL).^2;

for it=1:1:NT      %Forward Time Looping
    p2 = 2*p1 - p0 + cns.*del2(p1);
    p2(xs,zs) = p2(xs,zs) + RICKER(it);
    REALDATA(:,it) = p2(xs,:);
    p0=p1;p1=p2;
    if round(it/20)*20==it;p00=p0/max(abs(p0(:))+.001);
        imagesc([1:NX]*dx,[1:NZ]*dx,(p00+BVEL)); colorbar;
        text((zs-3)*dx,(xs+3)*dx,'*', 'FontSize',16,'color','red');
        hold on;plot([0 dx*NX],[xs-1]*dx (xs-1)*dx),'--y');hold off
        pause(.1);
    end
end;

p1=p0;p0=p2;
title('Snapshot of Acoustic Waves');text(xs,zs,'*', 'FontSize',16);
xlabel('X (km)'); ylabel('Z (km)')
figure(2);imagesc([1:NX]*dx,[1:NT]*dt,REALDATA);xlabel('X (km)');ylabel('Time (s)')
title(['Seismograms along Horizontal Line at Depth ',num2str(xs*dx),' km'])
figure(3);plot([1:NT-1]*dt,diff(REALDATA(100,:),1));xlabel('Time (s)')
title('Time Derivative of Pressure Seismogram')
```

The output produces snapshots and seismograms such as those shown in Figure 26.8. No absorbing boundary conditions have been included in the above code so strong reflections will emanate from the sides of the model, as if there is a free surface on each side with zero pressure conditions. These unwanted reflections can be partly attenuated by applying a sponge zone and/or absorbing boundary conditions Schuster (2017) to the bottom and side boundaries.

The fragment from the Python code for the finite-difference solution to the 2D acoustic wave equation is below. An excellent tutorial on finite-difference solutions to the wave equation and its associated Python codes is at [http://hplgit.github.io/INF5620/doc/notes/wave-sphinx/main\\_wave.html](http://hplgit.github.io/INF5620/doc/notes/wave-sphinx/main_wave.html).

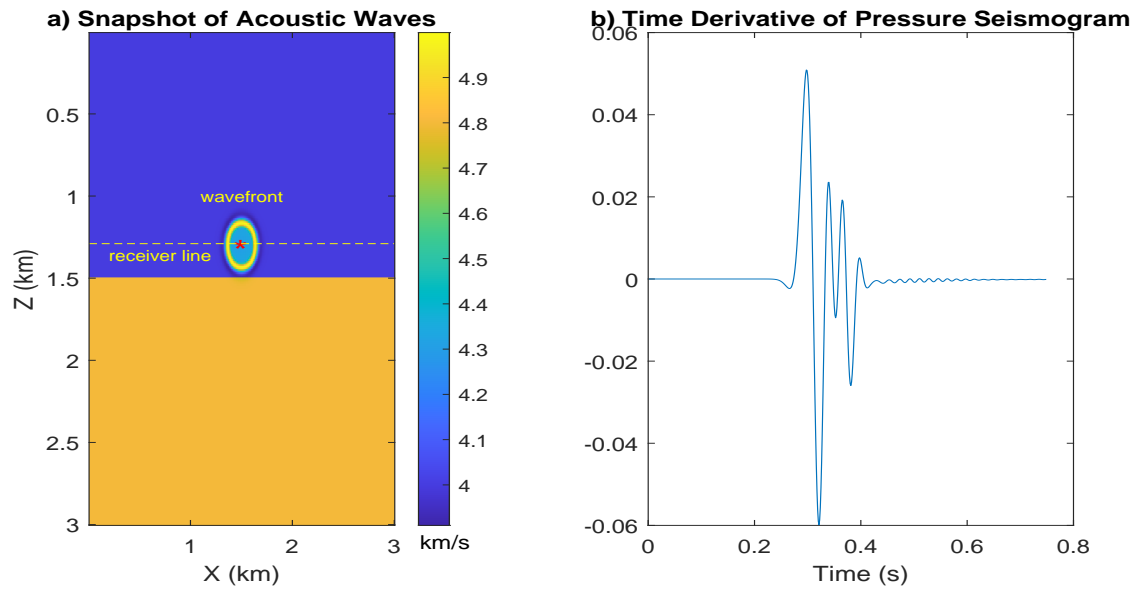


Figure 26.8: a) Snapshot and b) time derivative of the pressure seismogram recorded at  $x = 1 \text{ km}$  and  $z = 1.5 \text{ km}$ . The point source location in a) is denoted by the red asterisk and the receiver line is along the yellow dashed line.

**Code 26.4.2.** *Python FD Solution to the 2D Acoustic Wave Equation (Hans Petter Langtangen)*

```
import numpy as np
cimport numpy as np
cimport cython
ctypedef np.float64_t DT      # data type

@cython.boundscheck(False) # turn off array bounds check
@cython.wraparound(False)  # turn off negative indices (u[-1,-1])
cpdef advance(
    np.ndarray[DT, ndim=2, mode='c'] u,
    np.ndarray[DT, ndim=2, mode='c'] u_1,
    np.ndarray[DT, ndim=2, mode='c'] u_2,
    np.ndarray[DT, ndim=2, mode='c'] f,
    double Cx2, double Cy2, double dt2):

    cdef int Nx, Ny, i, j
    Nx = u.shape[0]-1
    Ny = u.shape[1]-1
    for i in xrange(1, Nx):
        for j in xrange(1, Ny):
            u[i,j] = 2*u_1[i,j] - u_2[i,j] + \
                Cx2*(u_1[i-1,j] - 2*u_1[i,j] + u_1[i+1,j]) + \
                Cy2*(u_1[i,j-1] - 2*u_1[i,j] + u_1[i,j+1]) + \
                dt2*f[i,j]

    # Boundary condition u=0
    j = 0
    for i in range(0, Nx+1): u[i,j] = 0
    j = Ny
    for i in range(0, Nx+1): u[i,j] = 0
    i = 0
    for j in range(0, Ny+1): u[i,j] = 0
    i = Nx
    for j in range(0, Ny+1): u[i,j] = 0
    return u
```



Part VII

Background Appendices





## Chapter 27

# Appendix: Geophysics Background

This appendix provides a high-level overview of the acquisition and processing procedures in exploration seismology. It is presented at a high level in order to help the non-specialist understand how they might benefit from machine learning applications. We also describe the seismic processing procedure of diffraction migration that provides maps of the earth's reflectivity distribution from seismic data. It is inarguably the most important processing tool in exploration seismology.

### 27.1 Seismic Data Recording and Processing

The principal goal of exploration seismology is to map out oil and gas reservoirs and aquifers by imaging the earth's seismic reflectivity and velocity distributions. In this regard, exploration geophysicists perform seismic experiments ideally equivalent to that shown in Figure 27.1, where the source (red stars) excites seismic waves, and the resulting primary reflections are recorded by a geophone (inset next to a coin) located at the source position. For this ideal zero-offset (ZO) experiment we assume only primary reflections<sup>1</sup> in the records and that waves only travel in the vertical direction.

After recording at one location, the source and receiver are laterally moved by about  $1/2$  source wavelength and the experiment is repeated at different ground positions. All recorded traces are lined up next to one another and the resulting section is defined as the zero-offset (ZO) or poststack seismic section, as shown on the rightside of Figure 27.1. This section resembles the actual geology, where one side of the signal is colored red and the other is blue to help enhance visual identification of the interface<sup>2</sup>. Note that the depth  $d$  of the first reflector can be calculated by multiplying the two-way reflection time  $t_{refl}$  by half the P-wave velocity  $v$  of the first layer, i.e.,  $d = t_{refl}v/2$ .

Seismic images of the subsurface are used to understand the geology of the earth. For example, Figure 27.2 shows both optical and seismic pictures of faults. These images reveal the characteristics of buried faults that can aid geologists in deciphering the tectonic forces that shaped the earth. Faults can also serve as impermeable traps for oil and gas deposits, waiting to be found by the explorationist with the most capable seismic camera and digital processing algorithms.

---

<sup>1</sup>A primary reflection is a wave that propagates downward, reflects off a reflector, and propagates back up to the surface, without any reverberations between layers.

<sup>2</sup>There are 4 hand-drawn seismograms on the right, but the actual seismograms are displayed in the background. There are hundreds of seismograms displayed next to one another, but the separation between each seismogram is not visible with this plotting format.

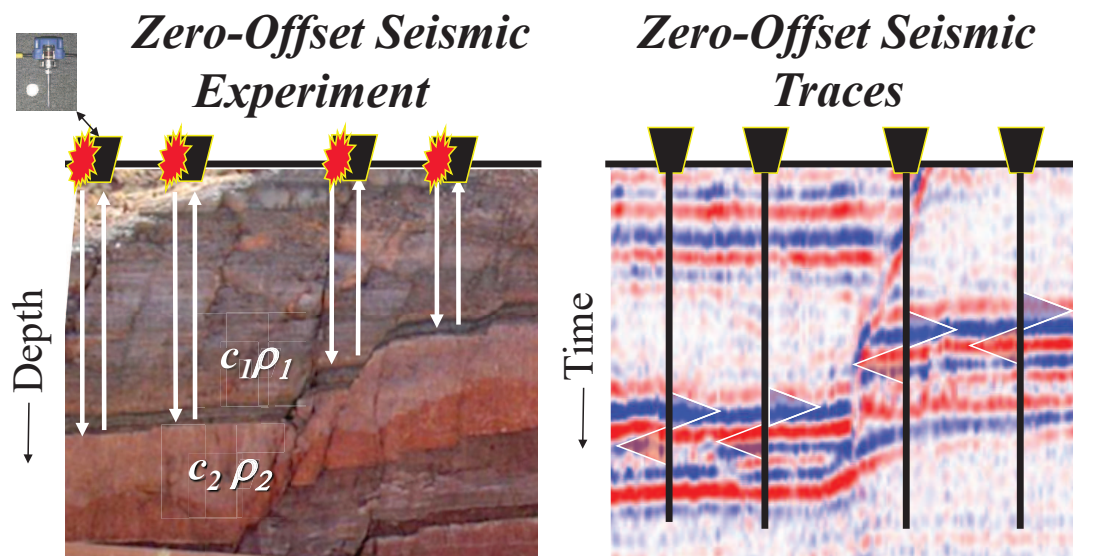


Figure 27.1: Ideal zero-offset (ZO) experiment on the left and the ideal traces (4 line-drawn traces) and actual ZO seismic sections to the right. Each trace is recorded by a geophone coincident with the source position; and the red- (blue-) colored amplitudes correspond to the particle velocity of the ground in the upward (downward) direction. An implicit assumption here is that the specular reflection point is directly beneath the geophone, which is incorrect for earth models with dipping structures or lateral velocity variations. Thus, the need for migrating the data.

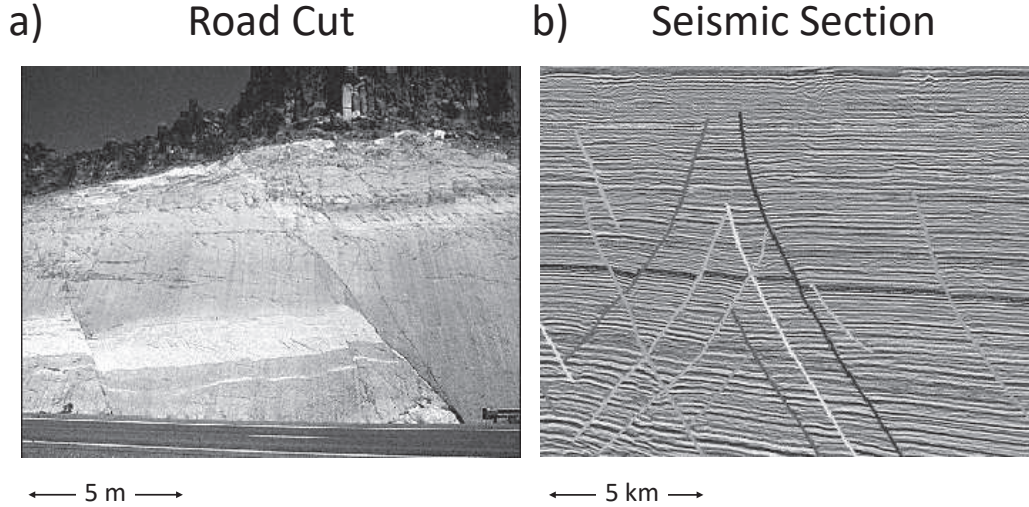


Figure 27.2: Geologic faults revealed by a) road cut and b) marine seismic section. The length scales above are roughly estimated.

### Seismic Sources

A land seismic source consists of a mechanical device or explosive located at  $\mathbf{s}$  that thumps the Earth (see Figure 27.3a) at time  $t = 0$ . A geophone (see Figure 27.3b) at  $\mathbf{g}$  records the time history of the vertical-particle velocity, denoted as the seismic trace  $d(\mathbf{g}, t | \mathbf{s}, 0)$ . A marine source is usually an array of air guns deployed as in Figure 27.4b. The lobe amplitudes in the Figure 27.1 traces are roughly proportional to the reflectivity strength  $m(\mathbf{x})$  of the corresponding reflector at  $\mathbf{x} = (x, y, z)$ . Assuming a constant density and a layered medium, the reflectivity model  $m(\mathbf{x})$  is sometimes approximated as

$$m(\mathbf{x}) \approx \frac{v(z + dz) - v(z)}{v(z + dz) + v(z)}, \quad (27.1)$$

where  $v(z)$  is the P-wave propagation velocity at depth  $z$  and  $dz$  is a small increment in depth. This formula is also the reflection coefficient for a pressure wave normally incident on a flat reflector with no density contrast.

### Non-zero Offset Seismic Experiment

In practice, traces from a ZO experiment cannot generate the ideal seismic section because the source also generates strong coherent noise and near-source scattering energy. In addition, the waves are propagating in all directions and contain distracting noise such as multiples, surface waves, scattered arrivals, out-of-the plane reflections, and converted waves. To account for these complexities, geophysicists perform non-zero offset experiments where the vibrations are recorded by many receivers as shown in Figure 27.4a. As before, each experiment consists of a shot at a different location except hundreds of active receivers are spread out over a long line for a 2D survey and a large area for a 3D survey. Figure 27.5 depicts a pictorial summary of a 3D marine survey, along with the key equation  $\mathbf{m} = \mathbf{L}^{-1}\mathbf{d}$  that should be solved for the reflectivity model  $m(x, y, z)$ .

a) Vibroseis Truck

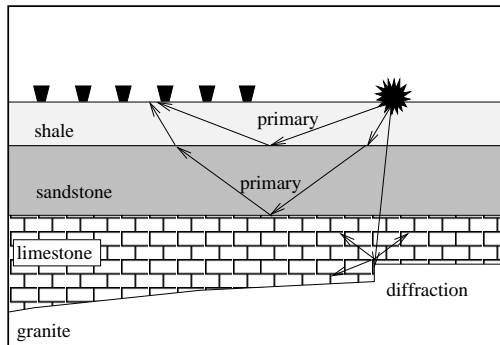


b) Geophones and Cables



Figure 27.3: a) Vibroseis truck and b) geophones attached to cables at a desert base camp. Inset is a particle-velocity geophone that is about 9 cm long.

a) 2D SSP Land Survey



b) 3D SSP Marine Survey (plan view)

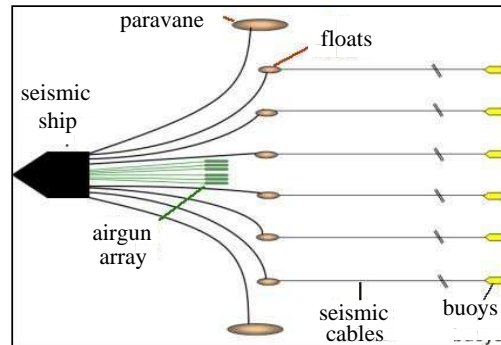


Figure 27.4: a) 2D land and b) 3D marine (courtesy of [www.blendspace.com](http://www.blendspace.com)) survey geometries to record surface seismic profiles (SSPs). The hydrophone streamer for a marine survey can be as long as 12 km with a 15-30 m hydrophone spacing. The typical 3D survey might consist of several source boats and often more than a dozen parallel hydrophone streamers, with a separation distance of up to 100 m. A survey where the sources and receivers are along the free surface is known as a surface seismic profile (SSP) survey.

## Marine Seismic Survey

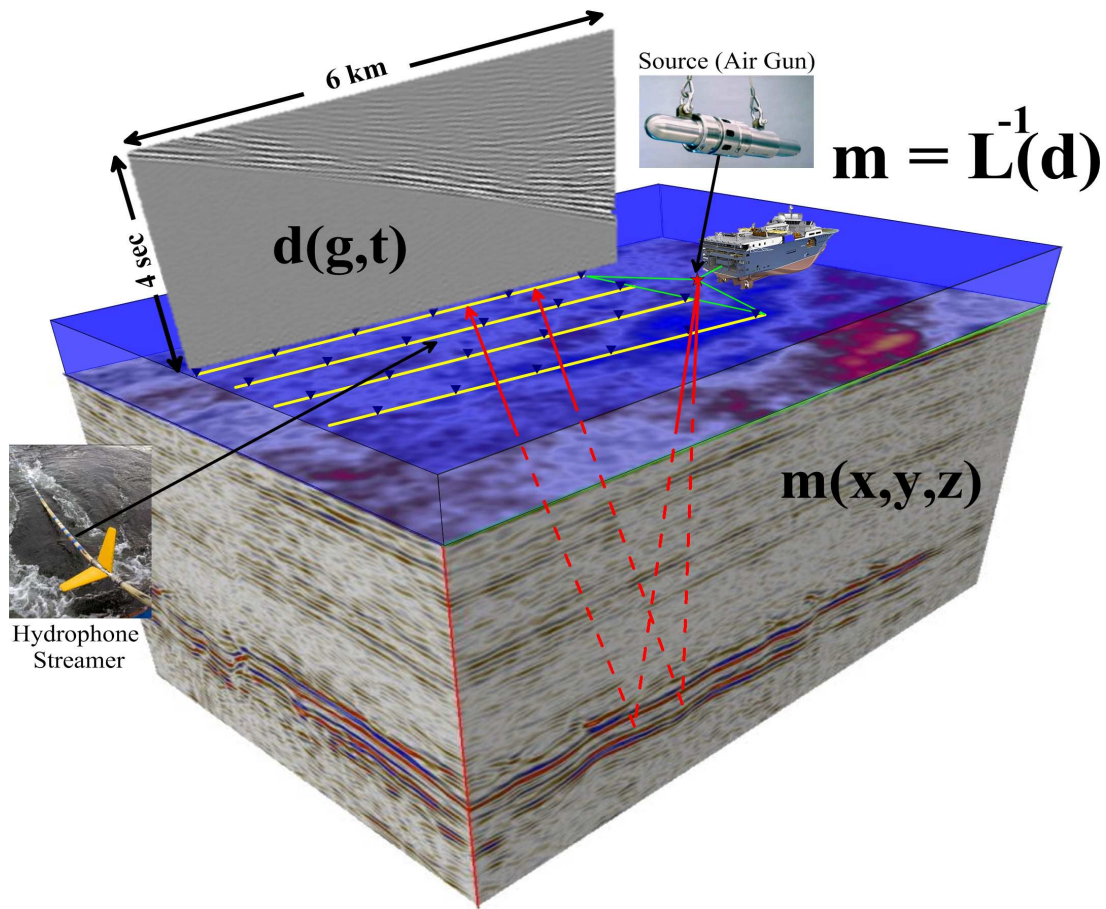


Figure 27.5: Picture of a marine survey boat towing hydrophone streamers (insets along right and lower left). A common shot gather recorded along one streamer is depicted in the upper left inset with time increasing upward, where the data  $\mathbf{d} = \mathbf{L}\mathbf{m}$  are related to the reflectivity model  $\mathbf{m}$  by the wave-equation modeling operator  $\mathbf{L}$ . A 3D reflectivity image computed from Gulf of Mexico data is depicted along the faces of the cube. The goal of seismic imaging is to estimate the earth model  $\mathbf{m}$  from the data  $\mathbf{d}$  by  $\mathbf{m} = \mathbf{L}^{-1}\mathbf{d}$ . Courtesy of Sherif Hanafy.



## Reflection Amplitudes.

The arrival times of reflections are influenced by velocity, but the reflection amplitudes are strongly controlled by density  $\rho$  and velocity  $v$  changes across an interface. The product  $I = \rho v$  is known as impedance, and the normalized difference  $(I_2 - I_1)/(I_1 + I_2)$  across an interface is the reflection coefficient of a pressure wave at zero-incidence angle, where  $I_1$  and  $I_2$  are the impedances above and below the interface. Figure 27.6a depicts a shot gather of traces recorded by a marine experiment where refractions, diffractions, and reflections are extant.

## Seismic Processing

For surveys over a mostly layered medium, data processing consists of the following steps:

1. Filtering of noise and near-surface statics corrections.
2. Reassembly of common shot gather (CSG) traces in Figure 27.6a into the common midpoint gather (CMG) shown in Figure 27.6b.
3. The traces in the CMG are time shifted to align the common midpoint reflections with the ZO reflection event in Figure 27.6c. The time shift that aligns the non-zero offset reflections with the ZO reflection is known as the normal moveout operation (NMO).
4. Stack<sup>3</sup> the traces in the time-shifted CMG to form a single trace at the common midpoint position in Figure 27.6d. This stacked trace approximates a ZO trace at that position.
5. Repeat steps 3-4 for all midpoint gathers to give the seismic section shown in Figure 27.2b. If the subsurface reflectivity is extremely complex then steps 2-5 are skipped and instead the algorithm known as prestack migration (see section 27.2) is used. However, an accurate migration image requires an accurate migration velocity model so we should first use travel-time tomography (see Chapter 2), full waveform inversion, and/or migration velocity analysis (Yilmaz, 2000) to estimate the migration velocity model.

In many exploration data sets, the above stacking process is skipped and instead the data are migrated to form the reflectivity section.

## Reflection Imaging

The problem with the ZO section is that it assumes that the reflections along the same hyperbola<sup>4</sup> emanate from the same reflection point directly beneath the midpoint geophone. This is not true for irregular layering or large lateral contrasts in velocity. Therefore, the reflections must be *migrated* to their original reflection points before they are stacked. Relocating reflections recorded in  $x - t$  to their points of origin in  $x - z$  is known as seismic migration. This yields an approximation to the reflectivity distribution, otherwise known as the migration image  $m(\mathbf{x})$ .

## Key Problem with Migration Images

A key difficulty in obtaining an accurate migration image is the estimation of a sufficiently accurate velocity model for migration. The migration velocity model is used to map reflection

---

<sup>3</sup>Summing, or stacking,  $N$  traces together gives a stacked trace, where the signal-to-noise ratio can be enhanced by  $\sqrt{N}$ . This assumes that the coherent signals are aligned with one another from trace-to-trace and the zero-mean additive noise is white.

<sup>4</sup>The primary reflections in Figure 27.6a have a hyperbolic moveout pattern in the shot gather because the traveltimes from a flat reflector at depth  $d$  honors the hyperbolic formula  $t(x) = \sqrt{\frac{x^2}{v^2} + \frac{4d^2}{v^2}}$ . Here,  $v$  is the effective velocity associated with the two-layer medium.

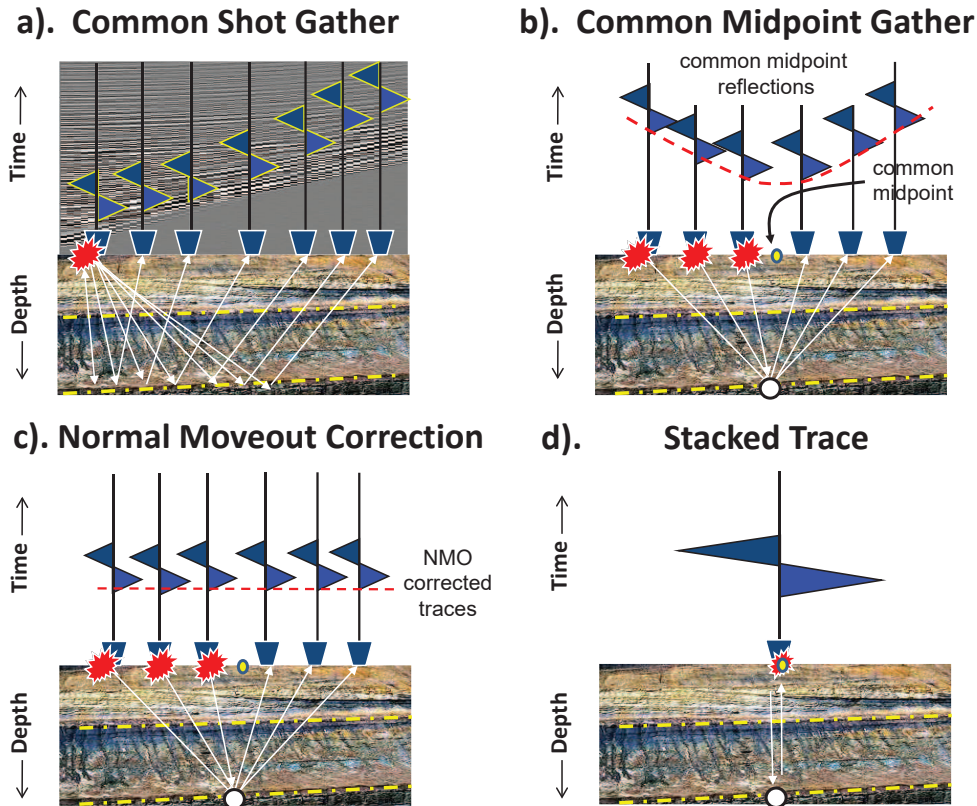


Figure 27.6: Simple processing steps for seismic data recorded over a layered medium, where the layers are mostly horizontal. a) common shot gather where a single source is recorded by all of the geophones to form a CSG of traces. b) The traces from many CSGs can be reassembled to form a common midpoint gather (CMG) where a trace associated with each source-receiver pair has the same source-receiver midpoint position on the surface. The common subsurface reflection point is known as the common reflection point; if the interface is dipping then rays associated with traces in the same CMG do not share a common reflection point. c) Flattened traces after time shifting the CMP traces to align with the zero-offset trace, aka NMO corrected traces. d) The stacked trace is formed by adding the NMO corrected traces. In this case the fold is 7 and the signal/noise is enhanced by a factor of  $\sqrt{7}$  if there is only additive white noise in the traces. This stacked trace approximates a zero-offset (ZO) trace where the source and geophone are coincident with one another.

events to their  $x - z$  reflection points. Large errors in this velocity model will lead to inaccurate, defocused and sometimes unusable migration images. As an example, Figure 27.7 shows different migration images obtained with variable velocity errors, where velocity errors should typically be less than 5% in order to obtain an accurate rendering of the reflectivity model.

## 27.2 Diffraction Stack Migration

Integral equation migration is a popular seismic imaging method (Yilmaz, 2001) used in the oil industry. It can be described as summing the weighted reflections in CSGs along pseudo-hyperbolas associated with each trial image point at  $\mathbf{x}$  to give the migration image  $m(\mathbf{x})$ . This summation is described by the diffraction stack migration formula

$$\begin{aligned} m(\mathbf{x}) &= \sum_A \sum_B \sum_{\omega} [i\omega D(\mathbf{B}|\mathbf{A}) e^{-i\omega(\tau_{Ax} + \tau_{xB})}], \\ &= \sum_A \sum_B \dot{d}(\mathbf{B}, \tau_{Ax} + \tau_{xB} | \mathbf{A}), \end{aligned} \quad (27.2)$$

where  $D(\mathbf{B}|\mathbf{A})$  is the frequency-domain representation of the trace in Figure 27.8, its inverse Fourier transform is given by  $d(\mathbf{B}, t | \mathbf{A})$ , and the summations run over the indices for the source  $\mathbf{A}$  and receiver  $\mathbf{B}$  locations. For a homogeneous medium having velocity  $c$  the time for waves to propagate from the source at  $\mathbf{A}$  down to the scatterer at  $\mathbf{x}_0$  and back up to the receiver at  $\mathbf{B}$  is given by

$$\tau_{Ax} + \tau_{xB} = \frac{\sqrt{(x_A - x_0)^2 + (z_A - z_0)^2} + \sqrt{(x_B - x_0)^2 + (z_B - z_0)^2}}{c}, \quad (27.3)$$

where  $z_A = z_B = 0$  for sources and receivers on a horizontal datum. This two-way traveltime equation describes a hyperbola in  $x_B - \text{time}$  coordinates for any image point at  $\mathbf{x}$  and a source fixed at  $\mathbf{A}$ ; the apex of the hyperbola is centered over the point scatterer at  $\mathbf{x}_0$ . If the reflection time on the leftside of equation 27.3 is fixed at a specific time then the values of  $\mathbf{x}$  that satisfy this traveltime equation form an ellipse with the foci at  $\mathbf{A}$  and  $\mathbf{B}$ , as depicted in Figure 27.8.

The observed data for a single scatterer can be approximated by

$$D(\mathbf{B}|\mathbf{A}) = e^{-i\omega(\tau_{Ax_0} + \tau_{x_0B})}, \quad (27.4)$$

where geometric spreading and the reflection coefficient are conveniently ignored. Substituting equation 27.4 into equation 27.2 gives

$$m(\mathbf{x}) = \sum_A \sum_B \sum_{\omega} [i\omega e^{-i\omega(\tau_{Ax} + \tau_{xB} - \tau_{Ax_0} - \tau_{x_0B})}]. \quad (27.5)$$

If the trial image point at  $\mathbf{x}$  coincides with the actual scatterer at  $\mathbf{x}_0$  then the summations of phases constructively interfere with one another to give a large migration amplitude  $m(\mathbf{x})$ . On the other hand, if  $\mathbf{x}$  is far from the actual scatterer then the predicted hyperbola will not match the actual hyperbola of recorded events. The consequence is that the summations in equation 27.5 will be largely incoherent and sum to a small number to be placed at the trial image point at  $\mathbf{x} \neq \mathbf{x}_0$ .

This type of imaging is also known as diffraction stack migration (Yilmaz, 2001). For a single source and single receiver the above equation becomes

$$m(\mathbf{x}) = \dot{d}(B, \tau_{Ax} + \tau_{xB} | A). \quad (27.6)$$

In a 2D homogeneous medium, the reflection traveltime  $\tau^{ref} = \tau_{Ax} + \tau_{xB}$  associated with the trial



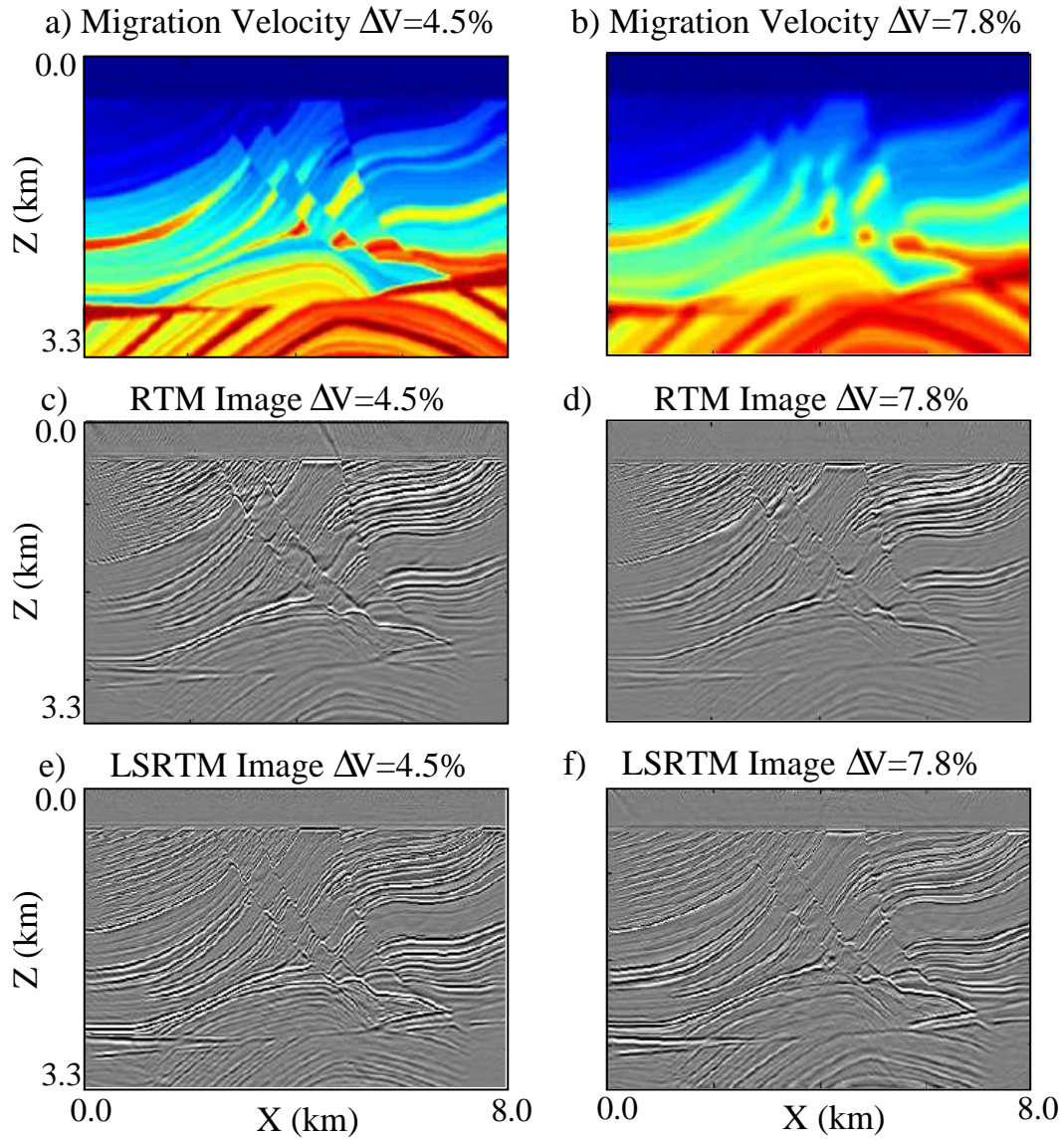


Figure 27.7: Reverse time migration (RTM) and least squares reverse time migration (LSRTM) images for the Marmousi migration velocity model with an RMS velocity error of a)  $\Delta V = 4.45\%$  and b)  $\Delta V = 7.81\%$ . Images courtesy of Gaurav Dutta.

### Migration: Smear and Sum Refl. Amp. Along Ellipses

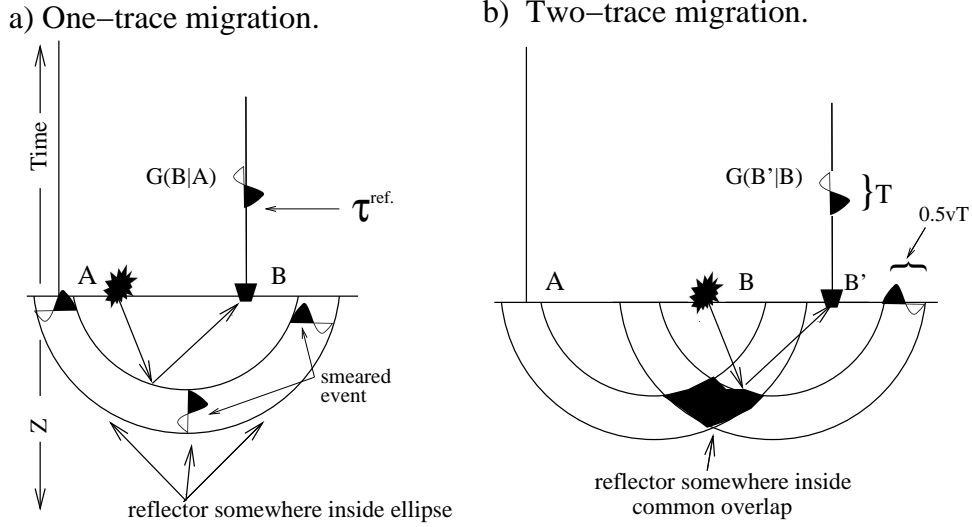


Figure 27.8: prestack migration is the smearing of trace amplitudes along the appropriate fat ellipses in  $(x, z)$  for each source-receiver pair  $\mathbf{A} - \mathbf{B}$  (Yilmaz, 2001). Migration of two traces in b) has better spatial resolution than migrating just one trace in a), and the thickness of each fat ellipse at  $z = 0$  is  $0.5vT$ , where  $T$  is the dominant period of the source wavelet.

scattering point at  $\mathbf{x}$  can be expressed as

$$\tau^{ref.} = [\sqrt{(x_A - x)^2 + (z_A - z)^2} + \sqrt{(x_B - x)^2 + (z_B - z)^2}]/v, \quad (27.7)$$

which, for  $\tau^{ref.} = cst$ , describes an ellipse in model-space coordinates  $(x, z)$  with the foci at  $\mathbf{A} = (x_A, z_A)$  and  $\mathbf{B} = (x_B, z_B)$ . It follows from equation 27.6 that the reflectivity image  $m(\mathbf{x})$  is approximated by smearing the trace amplitude at time  $\tau^{ref.}$  over the ellipse described by equation 27.7 in model space.

Smearing the seismic amplitude over an ellipse is shown in Figure 27.8a; here, the temporal interval  $T$  of the trace's source wavelet determines the thickness  $0.5vT$  of the fat ellipse at  $z = 0$ . Somewhere along this ellipse is a scatterer that gave rise to the event at time  $\tau^{ref.}$  for the receiver at  $\mathbf{d}$ . The scatterer's location can be better estimated by stacking (see equation 27.2) the "smears" from other traces into the model, as illustrated in Figure 27.8b. If the receivers are too far apart then the steeply dipping parts of the migration ellipse do not cancel, and so produce coherent artifacts defined as aliasing noise. Figure 27.9 shows a poststack migration image (where the source is at the receiver) for a homogeneous velocity model with 6 scattering points. The lateral spatial resolution of the image becomes worse with increasing depth.

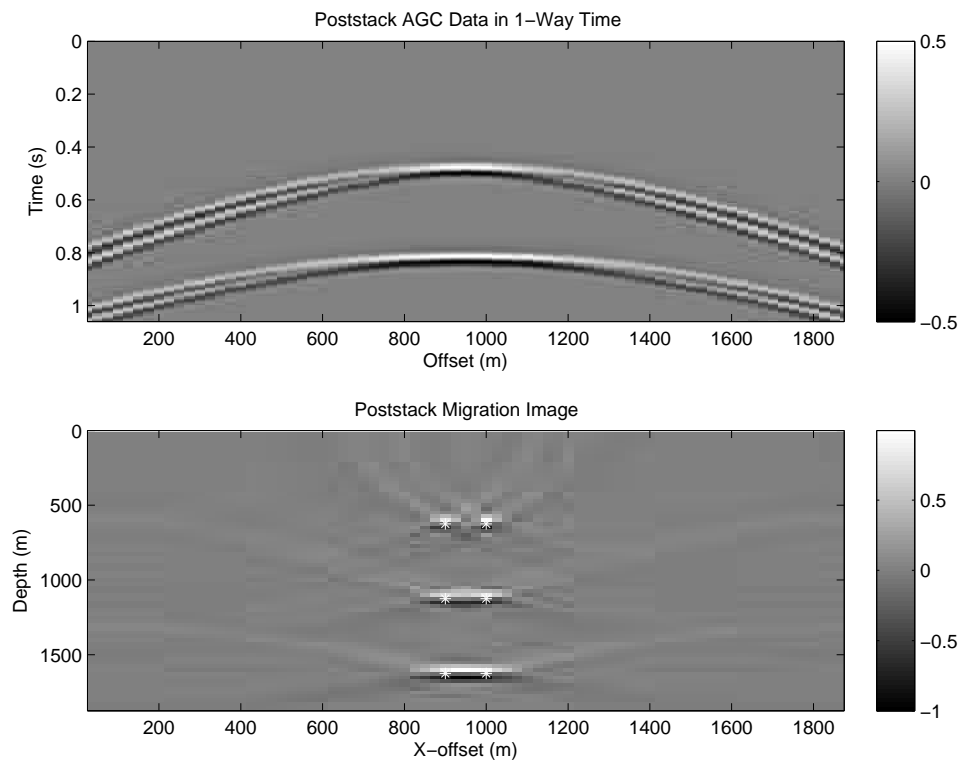


Figure 27.9: (Top) poststack data and (bottom) migration image where there are 6 point scatterers indicated by the white stars. Note, the spatial resolution of the image increases with decreasing depth of the scatterers.

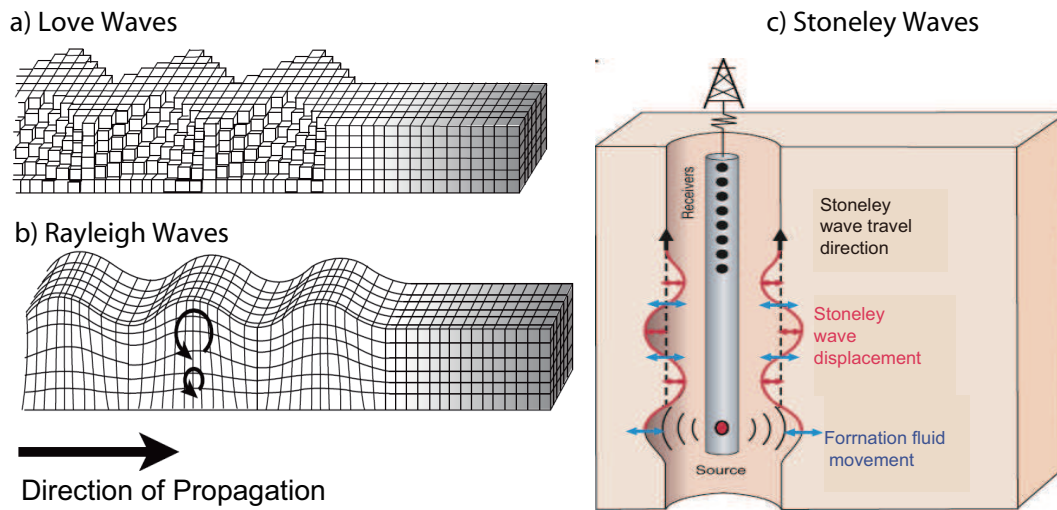


Figure 27.10: Illustrations of propagating a) Love waves, b) Rayleigh waves, and c) Stoneley waves.

### 27.3 Surface Waves

A surface wave is an evanescent wave that mostly propagates along a surface (such as the Earth's free surface or the sea floor), and generally attenuates with increasing depth. If the velocity structure of the medium is layered then different frequency components of the surface waves will propagate laterally with different phase velocities along the horizontal direction. The horizontal phase velocity at a specific frequency is a weighted average of the S-wave velocities down to, for Rayleigh waves, about a third of the shear wavelength ( $z \approx \frac{1}{3}\lambda$ ) below the free surface (Richert et al., 1970). Thus, the average S-wave velocity to the depth of  $1/3$  the wavelength can be approximated by measuring the associated phase velocities over different frequencies.

There are three types of surface waves that are of keen interest to the explorationists: Love waves (see Figure 27.10a) recorded by horizontal geophones with horizontal-particle motion perpendicular to the propagation direction, Rayleigh waves (see Figure 27.10b) recorded by vertical- and horizontal-component geophones, and guided waves (Haney and Stone, 2012; Aki and Richards, 2002; Zhao et al., 1994; Beresford and Janex, 2005; Gaiser and Vasconcelos, 2008; Boiero et al., 2013) that propagate along the water-sediment interface in a borehole (denoted as Stoneley waves in Figure 27.10c) or the sea floor (Scholte waves). Amplitudes of Rayleigh and Love waves exponentially decrease with depth, thus the motion induced by their passage is limited to the shallow subsurface for frequencies greater than 10 Hz. Love waves are simpler than Rayleigh waves in a layered medium because their particle motion is horizontal and perpendicular to the direction of propagation. Stoneley waves are another type of surface wave created along a solid-solid interface and are often exploited in borehole seismic to infer the shear-wave velocities (Stevens and Day, 1986). Previous research about surface waves focused on inverting the dispersion curve for the S-wave velocity profile (Park et al., 1998; Xia et al., 1999, 2007), imaging lateral heterogeneities using backscattered waves (Almuhaidib and Toksöz, 2015), and ground-roll reflection analysis (Sloan et al., 2015).

The particle motion for a Rayleigh wave in a layered medium is retrograde elliptical as denoted by the black ellipse in Figure 27.10b. For a layered medium, different frequency components of surface waves propagate with different velocities, which are known as dispersive waves. The result is that an impulse-like wavelet disperses as it propagates, where different frequency components arrive at different times. This is illustrated in Figure 27.11a where the high-frequency surface waves propagate

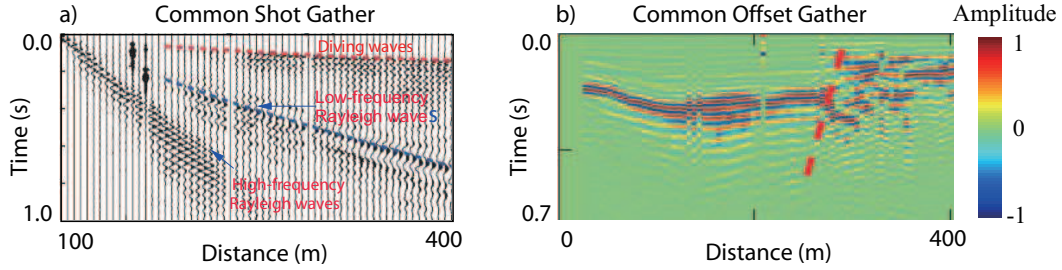


Figure 27.11: a) Common shot gather (CSG) and b) common offset gather (COG, offset=50 m) recorded by a land survey. The red dashed line in b) is an interpreted normal fault.

at the shallow-layer velocities with a steeper slope in the  $x-t$  domain than the low-frequency waves propagating at the faster velocities at deep depths.

The lateral variations of the deeper velocities are indicated in the common offset gathers (COGs) in Figure 27.11b where the dashed red line denotes a normal fault with slower downthrown-side velocities. If the traces are decomposed into separate frequencies by a 2D FFT, then the fundamental-mode dispersion curve can be picked in the wavenumber frequency ( $k-\omega$ ) domain (Figure 5.9b) or the  $C(\omega)-\omega$  domain (Figure 5.9c). Here,  $C(\omega)$  is the phase-velocity at frequency  $\omega$ ,  $D(k, \omega)$  is the spectrum of the traces, and  $n=0$  represents the fundamental mode of the Rayleigh waves. Typically, the fundamental mode is denoted by the curve that coincides with the largest amount of spectral energy near the  $C(\omega)$  axis in Figure 5.9c.

### 27.3.1 Estimating Depth vs Frequency

The S-wave velocity model can be used to assess velocity anomalies along the near surface, which can be used, for example, to identify drilling hazards and assign statics corrections (Xia et al., 1999; Socco et al., 2010). Unfortunately, surface-wave characterization of the subsurface with typical exploration surveys fails to reliably generate and record frequencies lower than 2 Hz. For example, if the lowest frequencies recorded are no less than about 2 Hz and the average S-wave velocity is slightly above 1000 m/s then surface waves are not sensitive to geology below the depth  $z$  of  $1/3$  the dominant wavelength:

$$z \approx \frac{\lambda_S}{3} = \frac{V_r}{3f} = \frac{1}{3} \frac{1000 \text{ m/s}}{2 \text{ Hz}} = 166 \text{ m}, \quad (27.8)$$

where  $f$  is the frequency and  $V_r$  is the Rayleigh-wave phase velocity at that frequency. This problem can be mitigated by recording ambient noise such as in the Long Beach array (Lin et al., 2013) and microtremor surveys where useful frequencies as low as 0.5 Hz can be used to probe more than a kilometer below the free surface. But such ambient noise data usually requires weeks of recording time.

## 27.4 Seismic Reflection Data as an LTI System: $s(t) = r(t) \star w(t)$

The velocity structure of a 1-D layered earth with constant density can be described by  $v(z)$ , where  $v(z)$  is the propagation velocity as a function of depth. Assuming a uniform sampling in depth

with a sampling interval of  $dz$  we have the propagation velocity vector  $\mathbf{v} = [v(0), v(1), \dots, v(N)]$ , and the associated reflection coefficient vector as function of depth  $\mathbf{r} = [r(0), r(1), \dots, r(N)]$  where:

$$r(i) = \{v(i) - v(i-1)\} / \{v(i) + v(i-1)\}. \quad (27.9)$$

The time it takes a plane wave to go from the  $i$   $dz$  depth to the  $(i+1)dz$  depth is  $dz/v(i)$ . Thus the 2-way time  $t(z)$  that seismic energy takes to go from the surface to the  $K$ th (i.e.,  $z = dz \cdot K$ ) depth level and back up to the surface is:

	K		BOOM		Geophone
	+---				
	+		^		v(1)
t(K*dz)	= + 2*dz/v(j)	V			
	+				
	+---		^		v(2)
	j = 1	V			
		.			
		.			
		.			
			^		v(K)
		V			

Using  $t(z)$  we can get  $z(t)$ ; thus we can convert the reflection coefficient  $r(z)$  as a function of depth to the reflection coefficient  $r(t) = r(z(t))$  as a function of time. The physical meaning of  $r(t)$  is that it is the impulse response of the 1-D layered earth for an impulsive source and a receiver at the surface. It assumes no multiples, no attenuation and no transmission losses in the earth.

For example, if the earth model is of uniform velocity  $v = 1$  km/s except for a layer with reflectivity  $-.5$  at  $z = 1$  km then an impulsive source wavelet with amplitude  $A \delta_{n0}$  launched from the surface will generate the following reflection response:

$$r(t) = [r(0) \ r(1) \ r(2) \ \dots] = [A \ 0 \ -.5 \cdot A \ 0 \ 0 \ 0 \ \dots], \quad (27.10)$$

where  $\Delta t = 1$  s. The first term  $r(0) = A$  is the direct wave and  $r(2) = -.5 \cdot A$  is the primary reflection from the first layer interface.

If we consider the earth's impulse response as LTI, the time history of the seismic source wavelet as  $w(t)$ , and the synthetic seismogram  $s(t)$  as the output, then  $s(t) = r(t) \star w(t)$ , where  $r(t)$  is the earth's impulse response. This is known as the 1-D convolution model of a seismogram. Is the earth really an LTI system? Does the earth system really satisfy linearity and scaling properties? What experiments can you devise to test this hypothesis?

An example of computing the seismogram from a synthetic sonic velocity log is given in Figure 27.12 and a field data example is given in Figure 27.13.

The mathematical description for describing two-way traveltime as a function of depth in continuous variables is given by

$$t(z) = 2 \int_0^z dz' / v(z'), \quad (27.11)$$

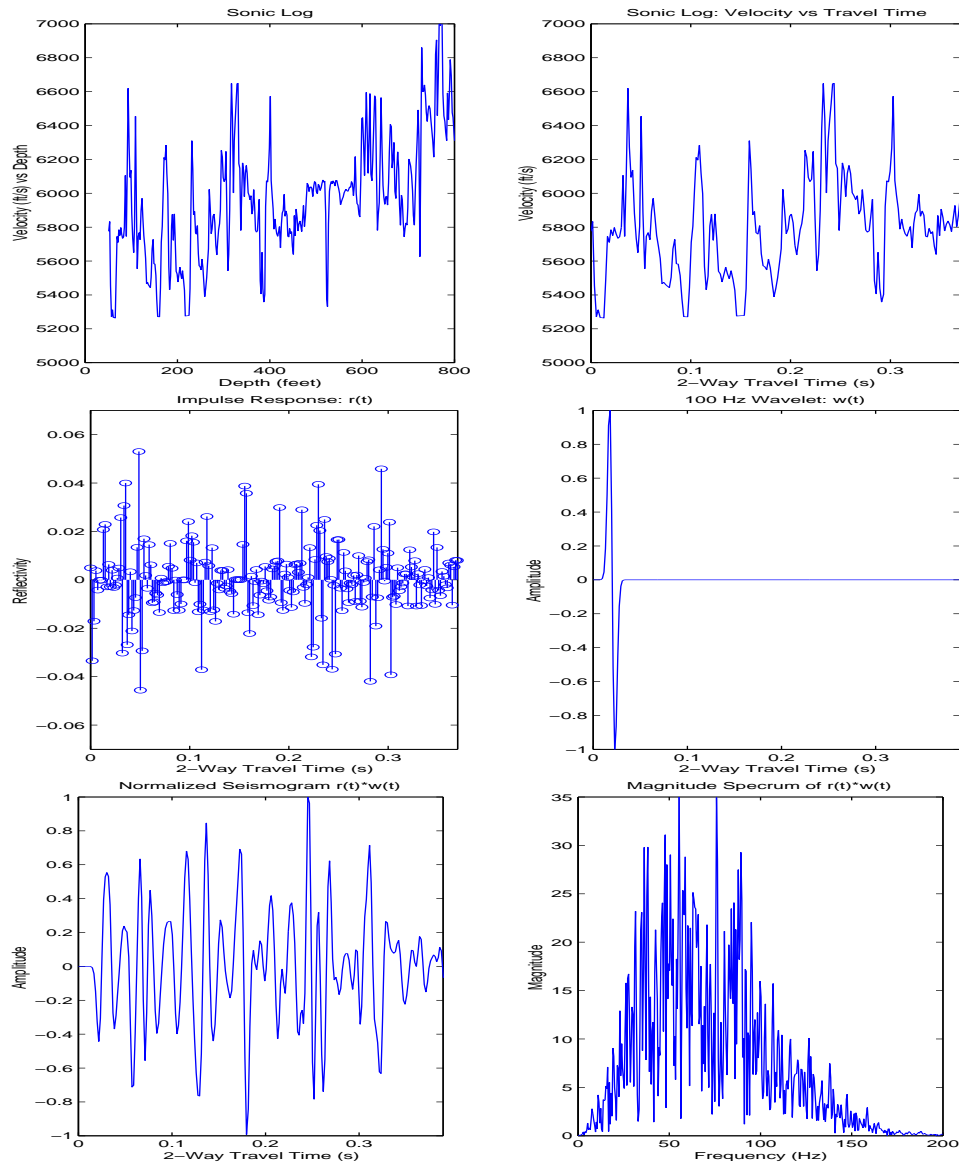


Figure 27.12: Synthetic (a). sonic log in depth, (b). sonic log in 2-way travel time , c). impulse response  $r(t)$ , d). 100 Hz wavelet  $w(t)$ , e). seismogram  $s(t) = r(r) \star w(t)$  and f). associated magnitude spectrum.

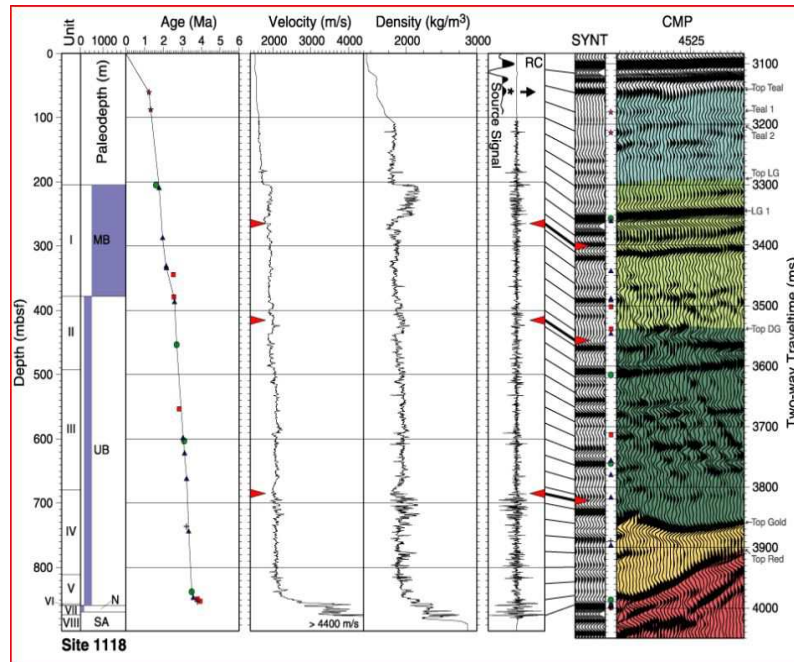


Figure 27.13: Field stacked section on far right and associated logs on left. Synthetic seismograms (derived from well logs) are shown just to the left of stacked section and correlate well with the recorded seismograms.



where  $t(z)$  is the 2-way propagation time for energy to go vertically downward from the surface to the horizontal reflector at depth  $z$  and back up to the surface in the 1D layered model. A MATLAB script for this mapping from depth to time is given as

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Finds t(z) from v(z). Assumes
% v(z) starts at free surface.
% v(z) - input- sonic log as function of z
% dz     - input- depth sampling interval of sonic log
% t(z)   - input- 2-way time as function of z
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [time]=depth2time(v,dz)
nz=length(v);time=zeros(nz,1); time(1)=dz/v(1);
for i=2:nz; time(i)=time(i-1)+dz/v(i); end
time=time*2; plot(dz*[1:nz],time);
xlabel('Depth (ft)'); ylabel('Time (s)')
title('Depth vs 2-way Time');figure
plot(time,v);xlabel('Time (s)');ylabel('Velocity (ft/s)')
```

The velocity model as a function of time  $v(t)$  (see Figure 27.12b) is usually unevenly sampled. To perform convolutional forward modeling, we must convert to an evenly sampled function in time  $v(t) = v(t(z))'$ ; the MATLAB code for getting an even sampled function at the sampling interval  $\Delta t$  from an unevenly sampled function is given in Appendix D.

Assuming that the velocity function  $v(t)$  is now an evenly sampled function, the evenly sampled zero-offset (ZO) reflection coefficients as a function of time can be estimated by  $r(t) = (\rho(t)v(t) - \rho(t - \Delta t)v(t - \Delta t)) / ((\rho(t)v(t) + \rho(t - \Delta t)v(t - \Delta t)))$ , which in MATLAB script becomes for constant density:

```
y=diff(vpp);nl=length(y);dt=diff(time);add=vpp(1:nl)+vpp(2:nl+1);
rc=y(1:nl)./add(1:nl);stem(time(1:nl),rc(1:nl));
```

If the density profile is known then the density can be put into the above reflection coefficient formula.

The bandlimited response of the medium for a plane wave input (with source wavelet  $w(t)$  as shown in Figure 27.12d) into the surface is a combination of arrivals, including primary and multiple reflections. If attenuation, transmission losses and multiples are excluded then the 1D convolutional model of the seismogram  $s(t)$  is given by

$$s(t) = \int_{-\infty}^{\infty} r(\tau)w(t - \tau)d\tau, \quad (27.12)$$

which is the definition of convolution of  $r(t)$  with  $w(t)$ , often abbreviated as  $s(t) = r(t) \star w(t)$ .

The above formula can be derived by taking the special case of the impulse response where the source wavelet is a Dirac delta function that is excited at time equal to zero:  $w(t) = \delta(t)$ , where the delta function is defined as  $\delta(t) = 0$  if  $t \neq 0$ , otherwise  $\delta(t) = 1$  in the sense  $\int \delta(t)\Delta t = 1$ . Plugging this impulse wavelet into the above equation yields:

$$\begin{aligned} s(t) &= \int_{-\infty}^{\infty} r(\tau)\delta(t - \tau)d\tau, \\ &= r(t), \end{aligned} \quad (27.13)$$

which describes the reflection coefficient series shown in Figure 27.12c. Thus, the 1D impulse response of the earth under the above assumptions perfectly describes the reflection coefficient series

as a function of 2-way traveltime. If the source wavelet were weighted by the scalar weight  $w(\tau_i)$  and delayed by time  $\tau_i$  then  $w(t) = w(\tau_i)\delta(t - \tau_i)$  then the delayed impulse response of the earth would be

$$\begin{aligned} s(t)' &= \int_{-\infty}^{\infty} r(\tau)w(\tau_i)\delta(t - \tau_i - \tau)d\tau, \\ &= w(\tau_i)r(t - \tau_i), \end{aligned} \quad (27.14)$$

which is a weighted delayed version of the original impulse response in equation 27.12. If we were to sum these two seismograms we would get, by linearity of integration,

$$s(t)' + s(t) = w(\tau_0)r(t - \tau_0) + w(\tau_i)r(t - \tau_i), \quad (27.15)$$

where  $\tau_0 = 0$  and  $w(\tau_0) = 1$ . By the superposition property of waves (i.e., interfering wave motions add together), we could have performed these two seismic experiments at the same time and the resulting seismograms would be identical mathematically to equation 27.15. More generally, the earth's response to an arbitrary wavelet  $w(\tau)$  is given by

$$\begin{aligned} s(t) &= \sum_i w(\tau_i)r(t - \tau_i), \\ &\approx \int_{-\infty}^{\infty} w(\tau)r(t - \tau)d\tau, \end{aligned} \quad (27.16)$$

and in the limit of vanishing sampling interval  $\Delta t = \tau_{i+1} - \tau_i$  the approximation becomes an equality (see Figure 27.12e). Under the transformation of variables  $\tau' = t - \tau$  equation 27.16 becomes

$$s(t) = \int_{-\infty}^{\infty} w(t - \tau')r(\tau')d\tau', \quad (27.17)$$

which is precisely the convolution equation shown in equation 27.12. The equality of equations 27.16 and 27.17 also shows that convolution commutes, i.e.,  $s(t) = r(t) \star w(t) = w(t) \star r(t)$ . The convolutional modeling equation was practically used by many oil companies starting in the 1950's, and is still in use today for correlation of well logs to surface seismic data.

## Chapter 28

# Probability Theory for Machine Learning

The mathematical foundations of machine learning methods are grounded in the theories of probability, statistics and optimization. The statistical approach for supervised learning defines the functional relationship  $\mathbf{y} = f(\mathbf{x}, \mathbf{w})$  between the input data  $\mathbf{x}$  and output  $\mathbf{y}$ , while optimization theory tells us how to find the optimal values of the model parameters  $\mathbf{w}$ . Therefore, the geoscientist who desires to deeply understand machine learning methods must also comprehend the salient aspects of statistics and probability. This chapter presents a brief overview of many of the important concepts of probability and statistics used in machine learning.

### 28.1 Random Variables, Sample Space, Events and Probability

A random event can be defined as the unpredictable outcome of an experiment, e.g., the flip of a coin gives either tails (T) or heads (H). The random variable (RV) is a mapping, i.e. function  $X(s)$ , that assigns a real number  $r$  to each possible outcome, such as  $r = 0$  for tails and  $r = 1$  for heads. The index  $s$  is the sample space index associated with a specific event. An uppercase letter  $X(s) = X$  is used for the random variable in the case of a finite discrete set of distinct  $N$  outcomes. The specific real values assigned to  $X$  are denoted as lower-case letters such as  $\{r_1, r_2, \dots, r_N\}$ . For example, the RV for the outcome of flipping a coin is given by

$$X(s) = \begin{cases} r_1 = 1 & \text{if heads where } s=1 \\ r_2 = 0 & \text{if tails where } s=100, \end{cases} \quad (28.1)$$

where we assign an arbitrary scalar value  $r_1 = 1$  for heads and  $r_2 = 0$  for tails. Note, the RV  $X(s)$  is a function of the index  $s$  where  $s = 1$  is the index for a head and  $s = 100$  is that for a tail. In this example, we could have assigned the index  $s = 0$  for a tail to get a more interpretable index, i.e.  $s = r$ .

In the case of a finite discrete set of outcomes, the specific scalar values assigned to  $N$  distinct outcomes of  $X$  are denoted as lower-case letters such as  $\{r_1, r_2, \dots, r_N\}$ . In general, a RV is a function  $X(s)$  that assigns a real-valued number  $r$  to each distinct outcome  $s \in S$ . Assigning a real number to each distinct event allows for the quantification of outcomes with numbers rather than with words. This will then allow for the mathematical definition of the probability of outcomes in terms of the RVs. value 1.

**Sample Space and Events.** The sample space  $S$  is the set of all distinct outcomes of an experiment. Each outcome can be thought of as a point in this sample space (DeGroot, 1975).

**Example 1: Three Tosses of a Coin.** Toss a coin three times and define the sample space as consisting of the  $2^3$  possible outcomes. A point in  $S$  represents the unique outcome of the 3-toss experiment and can be defined as a sequence of single-flip outcomes such as

$$\begin{aligned} s_1 : HHH; \quad s_2 : THH; \quad s_3 : HTH; \quad s_4 : HHT; \\ s_5 : HTT; \quad s_6 : THT; \quad s_7 : TTH; \quad s_8 : TTT. \end{aligned} \quad (28.2)$$

Each possible 3-term sequence of heads and tails is assigned a number<sup>1</sup>  $s_i$ , and  $X(s_i)$  can be defined as the number of heads in that sequence indicated by the value of the sample-space index  $s_i$ . If the number  $s_7 = 7$  is assigned to the event  $s_7 : TTH$ , then the number of heads in  $s_7$  is  $X(s_7 = 7) = r_7 = 1$ . This is an example where the sample space index  $s_7$  is not equal to the RV value  $r_7$ .

**Example 2: Points in a Plane.** If points  $(x_i, y_i)$  in a gridded plane are randomly selected according to some probability distribution then each coordinate can be considered as a RV. An outcome in the sample space  $S$  will be a two-dimensional point assigned to the real number  $s_i : (x_i, y_i)$ , and the number of distinct outcomes is equal to the number of points in the grid. The term  $(X, Y)$  is an example of a two-dimensional random vector. If three Cartesian coordinates  $(x_i, y_i, z_i)$  are randomly selected then the RV might be defined as the  $3 \times 1$  vector  $(X, Y, Z)$ , where  $X(s_i) = x_i$ ,  $Y(s_i) = y_i$  and  $Z(s_i) = \sqrt{x_i^2 + y_i^2}$ . Here, we arbitrarily assigned  $Z$  to be the RV which is the distance of  $(x_i, y_i, 0)$  from the origin for a point in a 3D coordinate system.

Subsets of  $S$  are called **events**. For example, if we flip two coins simultaneously then the sample space consists of the mutually exclusive outcomes  $S = \{HH, TT, HT, TH\}$ . Each of these distinct two-coin outcomes can be assigned a scalar value in sample space denoted by the lower-case symbols  $\{s_1, s_2, s_3, s_4\}$ . As an example, we can define an event as the result of a two-coin flip where *the first coin has a head* and is therefore the sample space is described by  $\tilde{S} = \{HH, HT\}$ .

The sample space for flipping one coin in equation 28.1 can also be described by the set notation  $S = \{H, T\}$  where the corresponding set of RVs is given by  $\{X_1, X_2\}$ . In this example,  $X_1, X_2$  are mutually exclusive binary variables that can be either 1 or 0. For example, if the flip is a head then the outcome can be denoted as the  $2 \times 1$  vector  $(1, 0)$ . Similarly, a tail outcome is described by  $(0, 1)$ , and there are no more possibilities in the sample space. The event  $X_1$  cannot simultaneously occur with the event  $X_2$ , so these events are mutually exclusive, aka disjoint, such that  $X_1 \wedge X_2 = \{\}$ .

### 28.1.1 Types of Events

There are three types of events.

- The RVs associated with events are classified as **independent** if they do not influence one another. For example, the event of a single coin flip is either a head (H) or a tail (T) and does not influence the outcome of a second coin flip. Another example is the event defined as the simultaneous flip of two coins, which results in either  $HH, TT, HT$  or  $TH$ . Flipping the two coins again will have an outcome independent of the first pair of coin flips. In the next section, we describe this dependency for finite discrete events by the conditional probability mass function.
- A **dependent (or conditional)** event is one which influences the other. For example, if  $X$

---

<sup>1</sup> $s_i$  can also be assigned a name instead of a number.

is the event of being cloudy and  $Y$  is the event of rain, then  $X$  influences whether it will rain  $Y$ . Thus  $Y$  is dependent on the existence of  $X$ .

- Two events are **mutually exclusive**, aka **disjoint**, if they can't happen at the same time. For example, we cannot flip a coin and get both a head and a tail.

### 28.1.2 Probability

A random variable has a value that represents the numerical outcome of a random phenomenon. If the number of these numerical values is uncountably infinite, i.e. take on any value along a segment of the real line, then this RV is denoted as a continuous RV. For example, the outdoor temperature value for any given day. If, on the other hand, the RV can only take on a countable number of values then this is denoted as a discrete RV.

For a discrete RV  $X$ , the probability  $P(X = x_1)$  is the percentage that the outcome  $X = x_1$  occurs<sup>2</sup> compared to all possible outcomes in the sample space  $S$ . For a discrete RV,  $P(X)$  is also called a probability mass function (PMF).

For example, we can perform  $N$  coin flips, i.e. realizations, to get the number  $N_T$  of tails, and give an estimate of the frequency of tails  $F_T$  and heads  $F_H$  as

$$F_T = \frac{N_T}{N}; \quad F_H = \frac{N_H}{N} = \frac{N - N_T}{N}, \quad (28.3)$$

where  $N_H = N - N_T$  is the number of heads in  $N$  experiments. Assigning  $x_1 = 1 \rightarrow Heads$  and  $x_2 = 0 \rightarrow Tails$ ,  $F_T$  and  $F_H$  are only estimates of  $P(x_i) \forall i \in \{0, 1\}$  because a different set of  $N$  realizations will likely give a slightly different estimate for  $P(x_i)$ .

As the number of experiments approaches infinity the frequency estimates become the probabilities

$$\lim_{N \rightarrow \infty} F_H = P(X = 1); \quad \lim_{N \rightarrow \infty} F_T = P(X = 0) = 1 - P(X = 1). \quad (28.4)$$

The frequency estimates have the properties of  $0 \leq F_H, F_T \leq 1$  and  $F_H + F_T = 1$ . Consequently,  $0 \leq P(x = 1), P(x = 0) \leq 1$  and  $\sum_{i=1}^2 P(x_i) = 1$  for the PMF  $P(X)$ .

A probability mass distribution  $P(X)$  lists the probability of all possible outcomes in the sample space. For example, the RV might be the outcome of the roll of a 6-faced dice describe by the sample space

$$S = \{x_1, x_2, x_3, x_4, x_5, x_6\}, \quad (28.5)$$

listed in Table 28.1 for an unfair dice. Here, the 6 possible outcomes of the roll of an unfair 6-faced dice are mapped to the numbers in the second column of Table 28.1, and their probability mass function  $P(X)$  is given in the third column.

**Continuous RV.** If the random variable is a continuous variable then the probabilities can be nonzero only if they refer to intervals, and these probability functions are known as *probability density functions* (PDFs). For example, the probability of an adult having a height between 1 meters and 3 meters is no less than 98%, and is given by  $\int_1^3 P(x)dx$ . For most of our discussions we will assume finite discrete variables and a PMF, and denote  $P(X)$  as a probability. If this probability is for a continuous variable  $X$  then the probability density function will be denoted by lowercase  $p(X)$ . For a PDF,  $0 \leq p(X)$  and  $\int_{-\infty}^{\infty} p(x)dx = 1$ . Unlike a PMF, the PDF  $p(x) > 1$  can be larger than the value 1 for certain values of  $x$  as long as  $\int_{-\infty}^{\infty} p(x)dx = 1$ .

<sup>2</sup>The equation  $X = x_1 = 1$  says that the RV  $X$  takes the value  $x_1 = 1$ .

Table 28.1: Probability Mass Function  $P(X)$  for an Unfair Dice

# of Dots on Face	Assigned Scalar for x	P(x)
1	1	$P(1)=1/6$
2	2	$P(2)=1/6$
3	3	$P(3)=1/6$
4	4	$P(4)=1/6$
5	5	$P(5)=2/6$
6	6	$P(6) = 0$

### Axioms of PMF

A probability mass function must satisfy three axioms:

- Axiom 1:

$$P(A) \geq 0 \text{ for every event } A. \quad (28.6)$$

- Axiom 2:

$$P(S) = 1, \quad (28.7)$$

where  $S$  is the entire sample space of events.

- Axiom 3: If  $A_1, A_2, \dots$  are disjoint events then

$$P(\cup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} P(A_i). \quad (28.8)$$

For example,  $x_i$  represents one of the six possible outcome values in the second column of Table 28.1. In this case, each outcome is disjoint and so the sum of the probabilities in the third column sum to the value 1, which is consistent with Axioms 2 and 3. That is, the probability is 100% that one of the 6 faces is seen after one roll of the dice. Another example is that summing the first three rows of the third column in Table 28.1 gives the value  $1/2$ . According to Axiom 3, this says that there is a 50% probability of the dice roll showing 1, 2 or 3 dots.

**Mean and Variance.** The average value  $\bar{X}$  of the outcome of a dice roll and its variance  $\sigma^2$  are given by

$$\bar{X} = \langle X \rangle = \sum_{i=1}^I P(x_i) x_i; \quad \sigma^2 = \langle (X - \bar{X})^2 \rangle = \sum_{i=1}^I P(x_i) (x_i - \bar{x})^2, \quad (28.9)$$

where  $\langle \rangle$  indicates the expectation operator and  $I$  is the number of discrete values of the RV. We often approximate the mean and variances by an averaging procedure over a finite sum of weighted

samples, so we call these approximations the sample mean or sample variance. For example, if we have 15 samples of the value  $x^{[k]}$  then the sample mean  $\frac{1}{15} \sum x^{[k]}$  is only an approximation to  $\bar{X}$ .

### Multivariate Probability Distributions

A probability distribution whose sample space is one-dimensional (for example real numbers, list of labels, ordered labels or binary) is called univariate, while a distribution whose sample space is a vector space of dimension 2 or more is called a multivariate distribution. For a two-dimensional random vector  $\mathbf{X} = (X_1, X_2)$ , such as the outcome of two coins being flipped simultaneously, the two independent random variables are denoted as  $X_1$  and  $X_2$ .

Table 28.2 describes the joint probability distribution for the two-dimensional random vector  $(X, Y)$  associated with flipping two fair coins. It assigns the number 1 if the coin is heads and the number 0 if it is a tail. The RV for the first coin is  $X$  and  $Y$  is the RV for the second coin. We often use the binomial distribution for classification of binary variables and multinomial variables.

Table 28.2: Joint Probability Mass Function  $P(X, Y)$  for Simultaneous Coin Flips for Two Fair Coins.

Outcome	$(X, Y)$	$P(X, Y)$
HH	(1,1)	$P(1,1)=1/4$
HT	(1,0)	$P(1,0)=1/4$
TH	(0,1)	$P(0,1)=1/4$
TT	(0,0)	$P(0,0)=1/4$

### Joint Probability Distributions.

The joint probability of two independent events  $X$  and  $Y$  occurring together at the same time is given by

$$P(X, Y) = P(X)P(Y). \quad (28.10)$$

It describes the likelihood, or frequency, of two events  $X$  and  $Y$  occurring at the same time. For example, the joint probability of the outcome  $HT$  with a fair two-dice roll is given by

$$P(X, Y)|_{X=1, Y=0} = P(X)|_{X=1}P(Y)|_{Y=0} = \frac{1}{2} \times \frac{1}{2} = \frac{1}{4}, \quad (28.11)$$

which agrees with  $P(1, 0) = 1/4$  in the second row of Table 28.2.

The first condition for defining the joint probability  $P(X, Y)$  is that both  $X$  and  $Y$  occur at the same time<sup>3</sup>. The other is that  $X$  and  $Y$  must be independent of each other, so the outcome of one does not influence the other. For example, rolling two dice gives outcomes in either dice that are independent of one another.

### Conditional Probability

A conditional distribution gives the probability of  $X$  contingent upon the values of the other random variables. Assume two dependent events denoted by the RVs  $X$  and  $Y$ . Their frequency of occurrence can be plotted as the disks in Figure 28.1, where  $P(X) = A_X/A$  and  $P(Y) = A_Y/A$ . Here,  $A$ ,  $A_X$ , and  $A_Y$  denote the areas of the black box, blue disk and red disk, respectively. The

<sup>3</sup><https://medium.com/mlengineer/joint-probability-vs-conditional-probability-fa2d47d95c4a>

## Black = Boundary of Total Universe

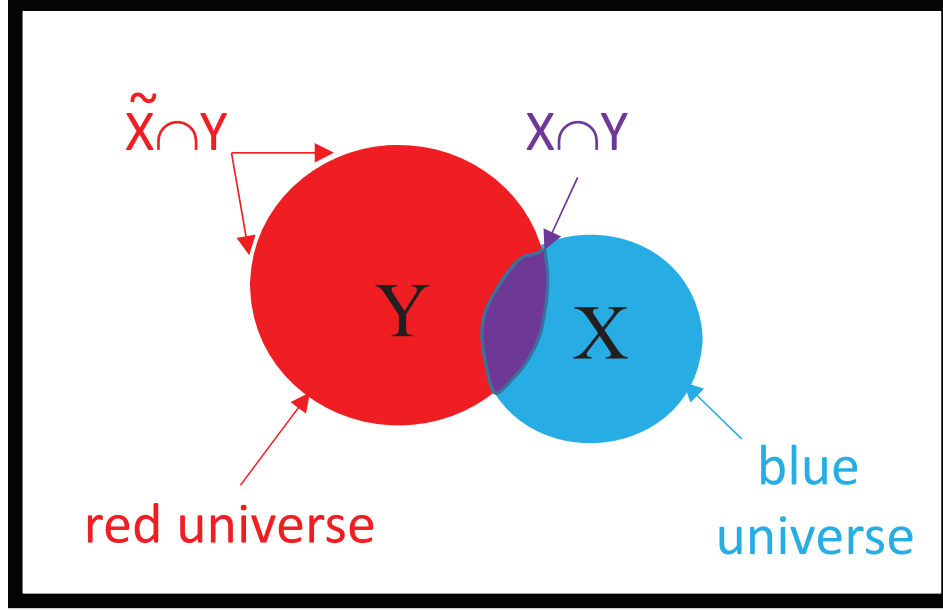


Figure 28.1: Regions outlining areas of the RVs  $X$  (blue disk) and  $Y$  (red disk). The area of intersection  $X \cap Y$  is denoted by the purple region and the total universe of events is represented by the area bounded by the black rectangle. The union of  $X \cap Y$  and  $\tilde{X} \cap Y$  is the red circle region, where  $\tilde{X}$  denotes the sample space outside of  $X$ .

total area of the *black reference universe* is denoted as  $A$  and contains all of the possible events in the universe of experiments.

If  $X$  is known to have occurred then an event associated with the blue disk has occurred with 100% probability. Therefore, the reference area is no longer that of the black box but it is that of the blue disk. In this case the conditional probability  $P(Y|X)$  of  $Y$  occurring if  $X$  has occurred is

$$\begin{aligned}
 P(Y|X) &= \frac{A_Y \cap A_X}{A_X} = \frac{\overbrace{A_Y \cap A_X}^{P(X,Y)}}{\overbrace{A}^{P(X)}} \left[ \frac{A_X}{A} \right]^{-1}, \\
 &= \frac{P(X,Y)}{P(X)}.
 \end{aligned} \tag{28.12}$$

Rearranging equation 28.12 we have

$$P(X,Y) = P(Y|X)P(X), \tag{28.13}$$

which is known as a special case of the product rule in section 28.1.2. Interchanging  $X$  and  $Y$  in equation 28.12 gives

$$P(X,Y) = P(X|Y)P(Y), \tag{28.14}$$



where  $P(X, Y) = P(Y, X)$  and  $P(X|Y)$  is not defined if  $P(Y) = 0$ . If  $X$  and  $Y$  are independent events the  $X$  should not depend on the occurrence of  $Y$  so  $P(X|Y) = P(X)$ . Therefore, equation 28.14 becomes  $P(X, Y) = P(X)P(Y)$ .

**Example: Seeing both Stars and the Moon.** The conditional probability  $P(Y|X)$  gives the frequency of event  $Y$  occurring if event  $X$  has occurred. This compares to the joint probability of events that tells us how often both  $X$  and  $Y$  occur simultaneously. For example, in an area where cloud cover occurs 90% of the time then we can see both the Moon and Stars at night less than 10% of the time; i.e.,  $P(M, S) < 0.1$ . However, the conditional probability  $P(S|M)$  is much more optimistic because the reference universe is that we have seen the moon. For example, in the same cloudy area, if we can see the moon then 98% of the time we also can see the stars; i.e.  $P(S|M) = 0.98$ . The few times we can't see the stars during a moon sighting is because wisps of clouds can obscure the stars while the much brighter moon burns through the wisps. Note,  $P(M|S) < P(S|M)$  because the moon is not always visible when fully obscured by the earth's shadow, but the stars are always unshadowed.

**Example: Internships and Employment.** As another example, assume that 70% ( $P(I)|_{I=1} = 0.7$ ) of students get an **Internship**, and 49% ( $P(J, I)|_{I=1, J=1} = 0.49$ ) of students get both an **Internship** and a fulltime **Job** offer after they graduate. Here, we assume  $I, J = 1$  indicates the internship and job offer and  $I, J = 0$  indicates neither.

What if the percent of students who received a full-time job offer  $P(J|I)|_{I=1, J=1}$  given that they worked as an intern? From equation 28.12 we have

$$P(1|1) = P(J|I)|_{I=1, J=1} = \frac{P(J, I)}{P(I)}|_{I=1, J=1} = \frac{0.49}{0.7} = 0.7, \quad (28.15)$$

which says 70% got a job offer after an internship compared to the  $[P(0|1) = 1 - P(1|1)] \times 100 = 30\%$  of the internees who did not get the full-time job.

### General Product Rule for Conditional Probabilities: Chain Rule

The chain rule, aka the general product rule, can transform any joint distribution of many random variables into products of conditional probabilities. For example, if  $X_1, X_2, X_3$  and  $X_4$  are RVs that satisfy the axioms of probability then the joint probability  $P(X_1, X_2, X_3, X_4)$  can be expressed as a product of conditional probabilities:

$$P(X_4, X_3, X_2, X_1) = P(X_4|X_3, X_2, X_1)P(X_3, X_2, X_1), \quad (28.16)$$

$$= P(X_4|X_3, X_2, X_1)P(X_3|X_2, X_1)P(X_2, X_1), \quad (28.17)$$

$$= P(X_4|X_3, X_2, X_1)P(X_3|X_2, X_1)P(X_2|X_1)P(X_1), \quad (28.18)$$

where equation 28.16 follows from the definition of joint probability in equation 28.14 by treating  $\{X_3, X_2, X_1\}$  as a single event. Equation 28.17 follows by treating  $\{X_2, X_1\}$  as a single event.

More generally, it is straightforward to decompose  $P(X_K, X_{K-1}, \dots, X_1)$  into the product of  $K$  conditional probabilities and a marginal probability to give the probability chain rule<sup>4</sup>:

$$\begin{aligned} P(X_K, X_{K-1}, \dots, X_1) &= P(X_K|X_{K-1}, \dots, X_1)P(X_{K-1}, \dots, X_1), \\ &= P(X_K|X_{K-1}, \dots, X_1)P(X_{K-1}|X_{K-2}, \dots, X_1)P(X_{K-2}, \dots, X_1), \\ &= \prod_{k=1}^K P(X_k|X_1, X_2, \dots, X_{k-1}), \end{aligned} \quad (28.19)$$

<sup>4</sup>The probability chain rule is also called the general product rule, where the joint distribution  $P(X_K, X_{K-1}, \dots, X_1)$  can be expressed as products of conditional probabilities (see [http://en.wikipedia.org/wiki/Chain\\_rule\\_\(probability\)](http://en.wikipedia.org/wiki/Chain_rule_(probability))).

where  $P(X_k|X_1, \dots, X_{k-1}) = P(X_1)$  for  $k = 1$  and  $P(X_k|X_1, \dots, X_{k-1}) = P(X_2|X_1)$  for  $k = 2$ . In other words, the probability chain rule *allows us to transform a joint distribution of random variables into products of conditional probabilities and a marginal distribution*.

A special case of the product rule is

$$P(X, Y|Z) = P(Y|X, Z)P(X|Z), \quad (28.20)$$

where we can consider  $Z$  as a silent variable for  $\tilde{P}(X, Y) = P(X, Y|Z)$ . In this case the product rule says  $\tilde{P}(X, Y) = \tilde{P}(Y|X)\tilde{P}(X)$ , so replacing  $\tilde{P}(*)$  by the conditioned  $P(*|Z)$  and replacing  $\tilde{P}(Y|X)$  by  $P(Y|X, Z)$  gives equation 28.20.

### Computational Infeasibility for $P(x_1, x_2, \dots, x_K|y)$

It is computationally infeasible to compute the conditional probability  $P(x_1, x_2, \dots, x_K|y)$  by sampling for large values of  $K$ . For example, assume  $K = 1000$  and  $x_i \in \{0, 1\}$  is a binary RV for  $i \in \{1, 2, \dots, 1000\}$ . In this case there are  $2^{1000}$  unique combinations of  $(x_1, x_2, \dots, x_{1000})$ . To estimate the covariance matrix by sampling, we might require at least 10 sample averages for each unique combination of  $\{x_i, x_j\}$ . Therefore  $10 \times 2^{1000}$  training examples are needed for estimating the covariance matrix by a counting method. In addition, we need to repeat this for every new class of data such as  $Y \in \{0, 1, 2, \dots, C\}$ , where  $C > 2$  is an integer. Thus, it is practically infeasible to devise a training set with a sufficient number of examples to give an accurate counting estimate of the conditional probability  $P(x_1, x_2, \dots, x_K|y)$  or the elements of the covariance matrix. To overcome this challenge we often assume conditional independence of the RVs, as described below.

### Mutual Independence of $\{X_1, \dots, X_K\}$ Conditioned on $Y$

If we assume that all the features in  $\{X_1, X_2, \dots, X_K\}$  are mutually independent, conditional on the category  $Y$ , then

$$P(X_i|X_{i+1}, \dots, X_K, Y) = P(X_i|Y). \quad (28.21)$$

This assumes  $X_i$  is only dependent on  $Y$ . The mutual independence of the  $X_i$ 's conditioned on  $Y$  is known as *naive conditional independence*, which *Naive Bayes' Theorem* uses to eliminate the infeasible computation of multidimensional conditional probability tables. See Figure 28.2 for an example and section 20.3 for the description of Naive Bayes' Theorem.

Under the assumption of naive conditional independence expressed in equation 28.21, equation 28.19 becomes

$$\begin{aligned} P(Y, X_1, X_2, \dots, X_K) &= P(Y)P(X_1|Y)P(X_2|Y) \dots P(X_K|Y), \\ &= P(Y)\prod_{i=1}^K P(X_i|Y). \end{aligned} \quad (28.22)$$

The computation of the conditional probability table (CPT) for each term in equation 28.22 can be orders of magnitude less expensive than computing the conditional probability table (see Figure 28.2) for  $P(Y, X_1, X_2, \dots, X_K)$  in equation 28.19. This cost reduction is crucial for the practical application of the *Naive Bayes' Theorem* in section 20.3.

### 28.1.3 Law of Total Probability

Assume  $\{X_n : n = 1, 2, 3, \dots\}$  is a finite or countably infinite partition of a sample space  $X$ . That is,  $X_n$  are pairwise disjoint events whose union is the entire sample space  $X$ . Also assume that  $Y$  contains events in  $X$ . Therefore, the marginal probability  $P_X(x)$  of  $X$  can be obtained from the joint probability function  $P(X, Y)$  by summing  $P(X, Y)$  over the values that  $Y$  can take on.

## Conditional Probability Table $P(y|x_1, x_2)$

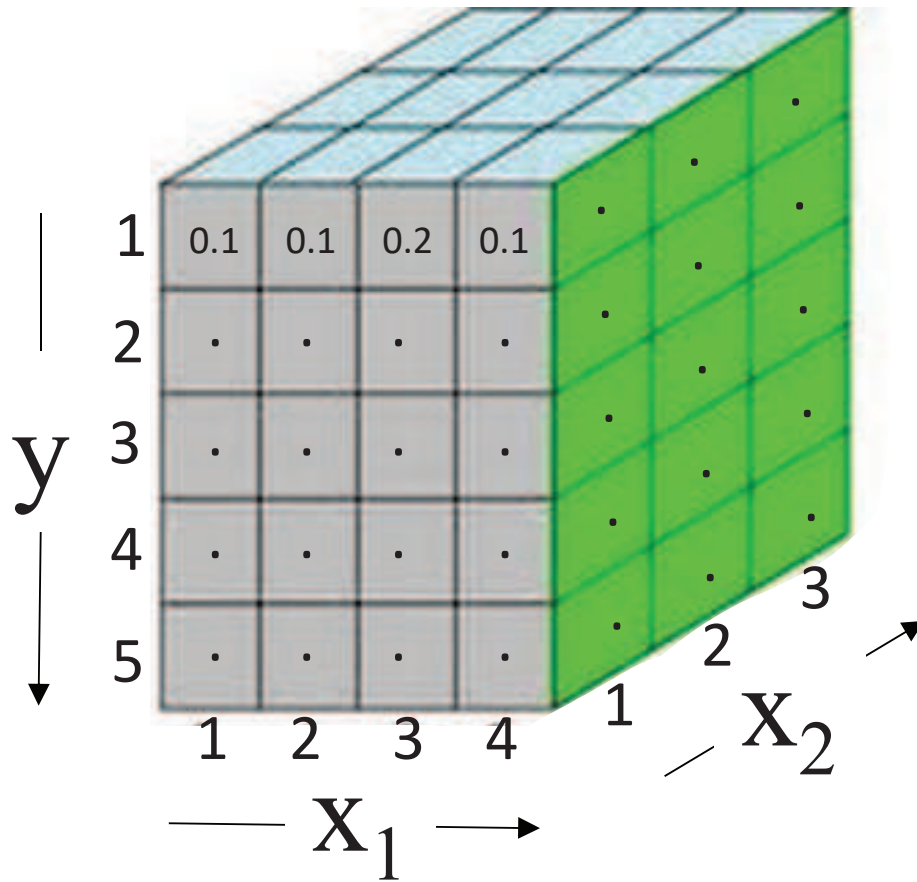


Figure 28.2: Conditional probability table (CPT) for  $P(Y|X_1, X_2)$  for  $Y \in \{1, 2, 3, 4, 5\}$ ,  $X_1 \in \{1, 2, 3, 4\}$ , and  $X_2 \in \{1, 2, 3\}$ . There are  $N_y \times N_{x1} \times N_{x2} = 5 \times 4 \times 3 = 60$  conditional probability entries for  $P(Y|X_1, X_2)$  in this table. Each conditional probability value might be estimated with 25 realizations, so the total number of realizations is equal to  $25 \times 60 = 1500$ .

Similarly, the marginal probability  $P_Y(y)$  can be obtained in a similar manner except the summation of  $P(X, Y)$  is over the  $X$  values:

$$P_X(x_i) = \sum_j P(x_i, y_j); \quad P_Y(y_j) = \sum_i P(x_i, y_j), \quad (28.23)$$

which is called the sum rule by marginalization<sup>5</sup>. By definition of conditional probability, the joint distributions in equation 28.23 can be replaced by conditional probabilities

$$P_X(x_i) = \sum_j P(x_i|y_j)P(y_j); \quad (28.24)$$

$$P_Y(y_j) = \sum_i P(y_j|x_i)P(x_i), \quad (28.25)$$

which is called the Law of Total Probability. This law can be interpreted intuitively by noting that the union of  $A_X \cap A_Y$  is the purple area in Figure 28.1 and  $A_{\tilde{X}} \cap A_Y$  is the red area devoid of the purple region. Therefore, the union of  $(A_X \cap A_Y)/A = P(X, Y)$  and  $(A_{\tilde{X}} \cap A_Y)/A = P(\tilde{X}, Y)$  is the red circular region  $P(Y) = P(X, Y) + P(\tilde{X}, Y)$ . By definition of conditional probabilities, it follows that  $P(Y) = P(Y|X)P(X) + P(Y|\tilde{X})P(\tilde{X})$ , which is the Law of Total Probability.

Another example is in Table 28.3, which depicts the values of the joint probability function  $P(X, Y)$  for  $X \in \{1, 2, 3\}$  and  $Y \in \{1, 2, 3, 4\}$ . Notice how the summations over the elements of the  $i^{th}$  row of  $\sum_{j=1}^4 P(x_i, y_j)$  are written in the right-hand margin of the  $i^{th}$  row. Hence, the name *marginal probability*.

If  $X$  and  $Y$  are independent RVs, then the marginal probabilities must satisfy the independence test

$$P(x_i, y_j) = P_X(x_i)P_Y(y_j). \quad (28.26)$$

For example, from the table the marginal probabilities are  $P_Y(1) = 0.3$  and  $P_X(1) = 0.2$ , so their product is  $P_Y(1)P_X(1) = 0.06$ . This agrees with the joint probability function  $P(X, Y)|_{X=1, Y=1} = P(1, 1) = 0.06$  in the table. Testing all the joint probabilities with equation 28.26 shows independence to be true with the values listed in Table 28.3.

We must also determine if the marginal probabilities sum to 1. Clearly, the sum of the marginal probabilities  $P_X(x_i)$  over  $x_i$  in the 6th column and 5th row is equal to 1. Similarly, the sum of the marginal probabilities  $P_Y(y_i)$  over  $y_i$  in the 6th column and 5th row is also 1. Hence,  $X$  and  $Y$  are independent RVs (DeGroot, 1975).

Another example is for picking refraction traveltimes in seismic traces by a class of students. You divided the class into two groups: students in group  $X$  wear glasses so that the probability of picking a correct traveltime event  $Z$  is  $P(Z|X) = 0.95$ , while the other group  $Y$  of students do not wear glasses so  $P(Z|Y) = 0.99$ . Group  $X$  is given  $P(X) = 40\%$  of the data and Group  $Y$  is given the remaining traces, i.e.  $P(Y) = 60\%$  of the data. What is the overall probability  $P(Z)$  of a picked traveltime being correct? Picking traveltimes by students in groups  $X$  and  $Y$  are exclusive events because they constitute picking different portions of the seismic data. Hence the Law of Total Probability is appropriate so that

$$\begin{aligned} P(Z) &= P(Z|X)P(X) + P(Z|Y)P(Y), \\ &= 0.95 * 0.4 + 0.99 * 0.6 = 0.974. \end{aligned} \quad (28.27)$$

Therefore, there is more than a 97% chance that the traveltime in a trace will be accurately picked.

---

<sup>5</sup>The subscript in  $P_Y(y)$  (or  $P_X(x)$ ) will often be silent so the marginal probability is implied by the notation  $P(y)$  (or  $P(x)$ ).

Table 28.3: Joint Probability Mass Function  $P(X, Y)$  and Marginal Probabilities  $P_X(X)$  and  $P_Y(Y)$  (DeGroot, 1975).

$P(X, Y)$	$Y=1$	$Y=2$	$Y=3$	$Y=4$	$P_X(X) = \sum_{j=1}^4 P(X, y_j)$ $\downarrow$
$X=1$	0.06	0.02	0.04	0.08	$P_X(1)=0.20$
$X=2$	0.15	0.05	0.10	0.20	$P_X(2)=0.50$
$X=3$	0.09	0.03	0.06	0.12	$P_X(3)=0.30$
$P_Y(Y) = \Rightarrow$ $\sum_{i=1}^3 P(x_i, Y)$	$P_Y(1)=0.30$	$P_Y(2)=0.10$	$P_Y(3)=0.20$	$P_Y(4)=0.40$	$\sum_{i=1}^3 P_X(x_i) = 1$ $\sum_{i=1}^4 P_Y(y_i) = 1$

## 28.2 Bayes' Theorem

Equating equations 28.13 and 28.14 gives Bayes' Theorem:

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}, \quad (28.28)$$

where  $P(Y) \neq 0$ ,  $P(X)$  is the prior probability for  $X$ ,  $P(Y)$  is the evidence,  $P(X|Y)$  is the posterior probability computed after the prior  $P(X)$  is known and  $P(Y|X)$  is the likelihood. These terms are discussed in more detail in Chapter 20.

### 28.2.1 Bayes' Theorem and Cancer Example

Bayes' Theorem shows the relation between the conditional probability  $P(X|Y)$  and its reverse form  $P(Y|X)$ . *This is useful when one only has access to the probabilities  $P(X)$ ,  $P(Y)$ , and  $P(Y|X)$  from a table of conditional data, but one really needs the reverse conditional probability  $P(X|Y)$ .*

Let's apply this Bayes' Theorem to a cancer example, where we replace  $Y$  by a positive cancer test denoted  $T$  and a negative test by  $\bar{T}$ . We also replace  $X$  by  $C$ , which indicates we actually have cancer and  $\bar{C}$  says we do not have cancer. For example, if the cancer test is positive, then the worried patient is desperate to know the rate of false positives  $P(\bar{C}|T)$ . All cancer tests have errors and it would be great for the patient if  $P(\bar{C}|T) \approx 0.3$  where the chances of a false positive is very high. This motivates one to undergo retesting before starting chemotherapy.

In this example, clinical tests will have supplied historical records that 1) gives the probability  $P(T|C)$  that actually having cancer will give you a positive test, 2) the fraction of the population with cancer  $P(C)$  and 3) the fraction of the population  $P(T)$  that tested for true positives or false positives. The Law of Total Probability says  $P(T) = P(T|C)P(C) + P(T|\bar{C})P(\bar{C})$ . Therefore, this formula can be computed because historical records give the rates for true positives  $P(T|C)$  and false positives  $P(T|\bar{C})$  as well as the fraction of the population with cancer  $P(C)$  and without cancer  $P(\bar{C})$ .

Therefore Bayes' Theorem in equation 28.28 will give the rates for false positives  $P(\bar{C}|T)$  and true positives  $P(C|T)$ , which are the uncertainty rates for testing. This is exactly what the worried patients needs, start calculating!

Bayes' Theorem can also be used to get the uncertainty in predicting rock properties and types from geophysical measurements of well logs (Xu et al., 2016). As discussed in section 20.6, the uncertainty in predicting rock types, oil or gas from well logs can be used to decide whether or not to drill an expensive well.

### 28.2.2 Bayes Theorem for $K$ Disjoint Events

If  $P(Y)$  is unknown, then it can be computed using the Law of Total Probability  $P(Y) = P(Y|X)P(X) + P(Y|\tilde{X})P(\tilde{X})$ . If  $P(X)$  is known then  $P(\tilde{X}) = 1 - P(X)$ . More generally, if there are  $K$  disjoint events  $\{X_1, X_2, \dots, X_K\}$  and the union of these events defines the sample space (DeGroot, 1975), then the Law of Total Probability in equation 28.25 says that Bayes' Theorem becomes

$$P(X_k|Y) = \frac{P(Y|X_k)P(X_k)}{\sum_{k=1}^K P(Y|X_k)P(X_k)}. \quad (28.29)$$

Here,  $P(Y) = \sum_{k=1}^K P(Y|X_k)P(X_k)$  from equation 28.25.

### 28.2.3 Bayesian Terminology

Bayes theorem is often described in terms of evidential data  $D$  and hypothesis  $H$  being true, so that equation 28.28 can be re-expressed by setting  $X = H$  and  $Y = D$ :

$$\overbrace{P(H|D)}^{\text{posterior}} = \frac{\overbrace{P(D|H)P(H)}^{\text{likelihood prior}}}{\underbrace{P(D)}_{\text{evidence}}}. \quad (28.30)$$

The above terminology in the over- and under-bars is now explained and translated to some geo-physical examples.

- **Posterior Probability  $P(H|D)$ .**  $P(H|D)$  is the conditional posterior probability that the hypothesis  $H$ , i.e. you have cancer, is true given that you observed the positive test data  $D$ .

Similarly, seismologists are interested in finding the hypothetical Earth model  $H \rightarrow \mathbf{m}$  that explains the recorded seismic data  $D \rightarrow \mathbf{d} = \mathbf{L}\mathbf{m}$ , where  $\mathbf{L}$  represents the seismic forward modeling operator. Therefore,  $P(\mathbf{m}|\mathbf{d})$  represents the uncertainty of the model parameters inverted from the data  $\mathbf{d}$ .

- **Likelihood  $P(D|H)$ .** The reversed conditional probability  $P(D|H)$  is called the likelihood, and is the probability, i.e. likelihood, that your hypothetical Earth model can be used to generate the recorded data  $D$ .

Seismic forward modelers assume a velocity model  $H \rightarrow \mathbf{m}$  and then numerically solve the wave equation in the computer to get the predicted data  $\mathbf{d}$ . Assigning a probability distribution  $P(H)$  to the earth model  $H$  allows one to perform many forward modeling experiments, i.e. realizations of  $H$ , to get many data examples  $\mathbf{d}$ . The sample mean and sample variance<sup>6</sup> of these simulated data  $\mathbf{d}$  can then be computed, which can be used to get an analytical Gaussian probability density function for the data.

- **Prior  $P(H)$  and Evidence  $P(D)$ .** The prior belief that the probability  $P(H)$  of the hypothesis being true is specified by the user, and the probability  $P(D)$  for the evidence being true is also supplied. In the cancer example,  $P(H)$  is the fraction of the population with cancer and  $P(D)$  is the fraction of the tested population that received either a false positive or a true positive test. The prior belief for  $P(H)$  and evidence  $P(D)$  can be obtained by a combination of acquired knowledge, experiments, and/or biased intuition!

---

<sup>6</sup>In practice, the data covariance matrix of the data is rarely computed so data RVs are assumed to be uncorrelated.

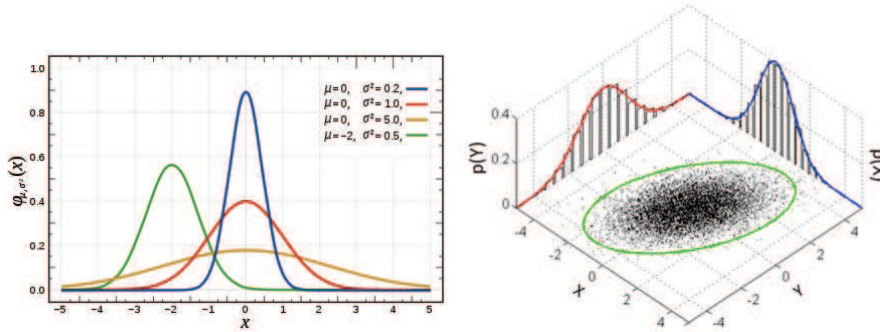


Figure 28.3: Gaussian probability distribution on the left for the real-valued RV  $x$  with various values of  $\mu$  and  $\sigma$ . The right plot is that for a multivariate normal distribution function for the RVs  $X$  and  $Y$ . Figure from [https://en.wikipedia.org/wiki/Normal\\_distribution](https://en.wikipedia.org/wiki/Normal_distribution).

The exploration geophysicist can use readings from well logs to estimate the probability  $P(H)$  of, as an example, rock velocities associated with shales. As shown in section 20.5, this prior information can be used to regularize the inversion of the earth model from new data.

In summary, equation 28.30 says that the prior belief of the hypothesis  $P(H)$  can be updated with new data evidence  $D$  by multiplying  $P(H)$  by the normalized likelihood  $P(D|H)/P(D)$  to get the new belief  $P(H|D)$ . This update is also known as the posterior probability.

## 28.3 Gaussian Probability Distribution

In general, the Gaussian continuous probability distribution  $f(x)$  for a continuous univariate RV  $x$  is

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left[\frac{x-\mu}{\sigma}\right]^2\right), \quad (28.31)$$

where  $\mu$  is the mean and  $\sigma$  is the standard deviation. For a continuous RV,  $f(x)$  is also known as a Gaussian probability density function.

The Gaussian distribution is plotted in Figure 28.3 for various values of the variance  $\sigma^2$  and mean  $\mu$ . A RV taken from a Gaussian distribution with mean  $\mu$  and variance  $\sigma^2$  is described by

$$X \sim \mathcal{N}(\mu, \sigma^2). \quad (28.32)$$

and is referred to as a unit normal Gaussian if  $\mu = 0$  and  $\sigma = 1$ .

The Gaussian function is a conjugate function, which means that the multiplication of two Gaussians is also a Gaussian<sup>7</sup>, as shown in Figure 28.4. The product of the green  $P(Y|X)$  and blue  $P(Y)$  Gaussians gives the unnormalized posterior Gaussian  $P(Y|X)P(X)$  plotted as the red dashed curve. This product is peaked in between the means of the green and blue curves and has a smaller variance than either one. This gives a more refined estimate of the likely possibilities associated with the data and the prior distributions in equation 28.28. Moreover, if the likelihood  $P(Y|X)$

<sup>7</sup>According to [https://en.wikipedia.org/wiki/Conjugate\\_prior#cite\\_note-beta\\_interp-4](https://en.wikipedia.org/wiki/Conjugate_prior#cite_note-beta_interp-4) if the posterior distributions  $p(y|x)$  are in the same probability distribution family as the prior probability distribution  $p(x)$ , then the prior and posterior are then called conjugate distributions, and the prior is called a conjugate prior for the likelihood function  $p(x|y)$ . In other words, the posterior distribution is in the same family as the prior and likelihood distributions.

is computed from noisy input data or too few data then this could lead to erroneous or overfitted estimates of the parameters of interest. The Bayesian estimate mitigates this problem by including the prior in the computation of the posterior distribution  $P(X|Y) \propto P(X|Y)P(Y)$ . In other words, the green curve  $P(Y|X)$  in Figure 28.4 might be too far to the right because of erroneous data or overfitting, the blue curve  $P(X)$  is a sanity check, so the compromise is the red curve  $P(Y|X)P(X)$  that is between the sensible prior and the error-prone green likelihood. We can include weighting to the prior to overemphasize it so the posterior more resembles the prior than the likelihood<sup>8</sup>. As will be discussed in section 20.5, this is known as regularization in the context of parameter estimation.

### 28.3.1 Central Limit Theorem

The Central Limit Theorem says that if many samples of independent RVs with finite mean and variance are averaged, then their properly normalized sum tends toward a normal distribution even if the original variables are not normally distributed. The theorem is a key concept in probability theory because it implies that probabilistic and statistical methods that work for normal distributions can be applicable to many problems involving other types of distributions ([https://en.wikipedia.org/wiki/Central\\_limit\\_theorem](https://en.wikipedia.org/wiki/Central_limit_theorem)). For example, the sum of two normal RVs is also a normal RV governed by a Gaussian distribution with mean  $\mu_1 + \mu_2$  and variance  $\sigma_1^2 + \sigma_2^2$ .

### 28.3.2 Multivariate Normal Distribution

The multivariate normal distribution, multivariate Gaussian distribution, or joint normal distribution is a generalization of the one-dimensional (univariate) normal distribution to higher dimensions. For correlated real-valued variables  $\mathbf{x} = (x_1, x_2, \dots, x_N)$  with mean  $\boldsymbol{\mu} = (\mu_1, \mu_2, \dots, \mu_N)$ , we have the multivariate normal distribution plotted in Figure 28.3 and described by

$$f(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^N |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right), \quad (28.33)$$

where  $N$  is the dimension of the data vector  $\mathbf{x}$ ,  $\boldsymbol{\Sigma}$  is the covariance matrix with the  $ij$  components

$$[\boldsymbol{\Sigma}]_{ij} = \text{Cov}(x_i, x_j), \quad (28.34)$$

where

$$\text{Cov}(x_i, x_j) = \langle x_i - \mu_i, x_j - \mu_j \rangle. \quad (28.35)$$

If the data RVs are uncorrelated, zero mean, and have the same variance  $\sigma_d^2$  then the data covariance matrix is a scaled identity matrix so that  $[\boldsymbol{\Sigma}]_{ij} = \sigma_d^2 \delta_{ij}$ .

A real random vector  $\mathbf{X} = (X_1, \dots, X_k)^T$  is defined as a standard normal random vector if all of its components  $X_n$  are independent and each is a zero-mean unit-variance normally distributed random variable. The standard normal RV can be compactly described as  $X_n \sim \mathcal{N}(0, 1)$  for all  $n$ .

Equation 28.33 is a function of the term  $\sqrt{(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})}$ , which is known as the Mahalanobis distance that represents the distance of the test point  $\mathbf{x}$  from the mean  $\boldsymbol{\mu}$ . For the special case  $k = 1$ , the distribution reduces to a univariate normal distribution and the Mahalanobis distance reduces to the absolute value of the standard score. In summary, the multivariate normal distribution is often used to describe, at least approximately, any set of (possibly) correlated real-valued random variables each of which clusters around a mean value.

<sup>8</sup><https://towardsdatascience.com/the-truth-about-bayesian-priors-and-overfitting-84e24d3a1153>



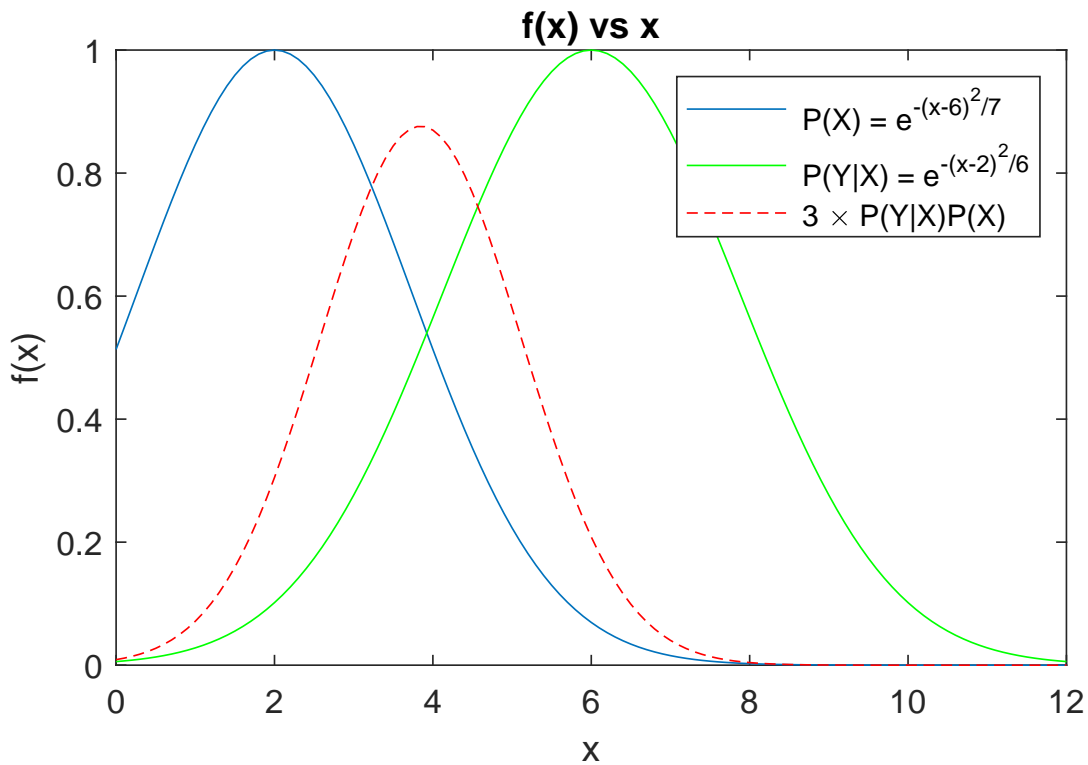


Figure 28.4: Red dashed curve depicts the posterior distribution computed by the scaled product of the unnormalized blue prior  $P(X)$  and green likelihood  $P(Y|X)$  distributions. Here, the scaling factor is equal to 3. The blue prior curve acts as a sanity check to the erroneous green curve, which is an estimate of the likelihood distribution possibly corrupted by data errors and/or overfitting .

## 28.4 Sampling a Probability Distribution

It is often the case that a probability distribution is given in analytic, e.g. Gaussian density function, or numerical form. The goal is to draw samples from this distribution such that the frequency of drawn samples around some point honors the occurrence probability from the given distribution. For example, flipping an unfair coin, i.e.  $p(\text{heads}) = 0.7$  and  $p(\text{tails}) = 0.3$ , and sampling from the probability distribution will tend to give the correct probability distribution as the number of coin flips increases. This illustrated in the Figure 19.1 example where the normalized histogram of coin flips tends to the correct probability with an increase in the number of trials. In this case, a uniform distribution  $U(0, 1)$  is approximated by a random number generator in MATLAB, and any number drawn between 0.0 and 0.7 is assigned a heads, otherwise it is a tails. In this case, 20 random numbers are sequentially generated to give Figure 19.1a, which is an unacceptable accuracy in drawing numbers that approximate the actual  $p(x)$ . If the experiment is rolling an unfair dice, then the values of  $x$  are assigned to one of 6 unique intervals between 0 and 1, with the length of each non-overlapping interval equal to the probability  $p(x)$ . For more complicated multivariate PDFs, this sampling strategy cannot be easily employed so we use the more sophisticated sampling methods discussed in Chapter 21.

## 28.5 Maximum Likelihood Estimate and Neural Network Models

Learning the matrix coefficients of a supervised neural network (NN) is functionally equivalent to learning the probability distribution  $P(Y|X, \mathbf{w})$ , where  $\mathbf{w}$  are the parameters of the NN model. Why? If a NN model is well trained as a binary classifier for any given input  $\mathbf{x}^{(n)}$ , it can accurately predict the most likely outputs  $\mathbf{y}^{(n)}$  for all  $N$  training pairs. We can assign a probability to this prediction as well, as explained in Chapter 4. This approximates the capability of knowing the probability distribution  $P(Y|X, \mathbf{w})$  of an input RV and predicting its output RV  $Y$ .

For example, if  $\mathbf{x}^{(n)}$  is the  $n^{\text{th}}$  input image vector and  $y^{(n)}$  denotes the class of that input, then the linear model that predicts this class is  $y^{(n)} \approx \mathbf{w}^T \mathbf{x}^{(n)}$ . These observed data have random errors so that the probability distribution for  $y^{(n)}$  might be assumed to be governed by the Gaussian distribution:

$$P(y^{(1)}, \dots, y^{(N)} | \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}, \mathbf{w}) \propto e^{-\frac{1}{2} \sum_{n=1}^N (\mathbf{w}^T \mathbf{x}^{(n)} - y^{(n)})^2 / \sigma^2}, \quad (28.36)$$

where the RVs  $\mathbf{x}^{(n)}$  are uncorrelated,  $\mathbf{w}$  is the model vector with unknown coefficients and  $\sigma^2$  is the variance for any of the components of  $\mathbf{x}^{(n)}$ . In this case the optimal coefficients in  $\mathbf{w}$  that predict  $y^{(n)}$  from  $\mathbf{x}^{(n)}$  are given by the least squares solution  $\mathbf{w}^*$  that minimizes the sum of the squared errors in the argument of the exponential. This is equivalent to maximizing the likelihood function w/r to the parameters  $\mathbf{w}$

$$P(y^{(1)}, \dots, y^{(N)} | \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}, \mathbf{w}) = \prod_{n=1}^N P(y^{(n)} | \mathbf{x}^{(n)}, \mathbf{w}), \quad (28.37)$$

which is also known as the maximum likelihood estimate (MLE). Here, the pairs  $(\mathbf{x}^{(n)}, y^{(n)})$  are assumed to be independent of the other pairs of RVs for  $n = \{1, 2, \dots, N\}$ .

From the frequentist point of view, there are no statistical representations for the model variable  $\mathbf{w}$  (Press et al., 2007, p. 777). Instead, frequentists interpret  $P(y^{(1)}, \dots, y^{(N)} | \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}, \mathbf{w})$  in equation 28.37 as the probability of the data  $y^{(1)}, \dots, y^{(N)}$  given the model  $\mathbf{w}$ . This function  $P(y^{(1)}, \dots, y^{(N)} | \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}, \mathbf{w})$  is known as the *likelihood* function.

The optimal model  $\mathbf{w}^*$  that maximizes the likelihood function in equation 28.37 is known as the maximum likelihood estimate, where  $P(y^{(1)}, \dots, y^{(N)} | \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}, \mathbf{w}^*)$  can be used as a discriminative classifier for new input data  $\mathbf{x}$ . A logarithm can be applied to equation 28.37 to give the

log-likelihood function  $\ln P(y^{(1)}, \dots, y^{(N)} | \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}, \mathbf{w})$ , whose formula for the gradients can be found with simpler algebraic formulas.

The observed data  $\mathbf{y}$  have random errors, and so these errors increase the uncertainty in the model  $\mathbf{w}$ . How does the frequentist account for these model errors when they consider  $\mathbf{w}$  to be a deterministic variable? The propagation of errors from the data leaking into the model is accounted for by assuming, for example, a linear model such that  $\mathbf{y} = \mathbf{L}\mathbf{w} + \boldsymbol{\epsilon}$ , where  $\boldsymbol{\epsilon}$  is a random noise vector with the same dimension as  $\mathbf{y}$ ,  $\mathbf{L}$  is an  $M \times N$  matrix<sup>9</sup> that connects the  $M \times 1$  data vector  $\mathbf{y}$  with the  $N \times 1$  model vector  $\mathbf{w}$ . For a zero-mean model  $\boldsymbol{\mu} = 0$ , the  $N \times N$  model covariance matrix (Menke, 1984) is

$$\begin{aligned}
 \text{Cov } \mathbf{w} &= \langle \delta \mathbf{w} \delta \mathbf{w}^T \rangle, \\
 &= \langle \overbrace{[\mathbf{L}^T \mathbf{L}]^{-1} \mathbf{L}^T}^{\delta \mathbf{w}} \overbrace{\mathbf{y} \mathbf{y}^T \mathbf{L} [\mathbf{L}^T \mathbf{L}]^{-1}}^{\delta \mathbf{w}^T} \rangle, \\
 &= [\mathbf{L}^T \mathbf{L}]^{-1} \mathbf{L}^T \underbrace{\langle \mathbf{y} \mathbf{y}^T \rangle}_{\sigma_{data}^2 \mathbf{I}} \mathbf{L} [\mathbf{L}^T \mathbf{L}]^{-1}, \\
 &= \sigma_{data}^2 [\mathbf{L}^T \mathbf{L}]^{-1},
 \end{aligned} \tag{28.38}$$

where  $\delta \mathbf{w} = \mathbf{w} - \bar{\mathbf{w}}$ , the least squares solution is  $\delta \mathbf{w} = [\mathbf{L}^T \mathbf{L}]^{-1} \mathbf{L}^T \mathbf{y}$  and zero-mean data with constant variance  $\sigma_{data}^2$  are assumed such that  $\text{Cov } \mathbf{y} = \langle \mathbf{y} \mathbf{y}^T \rangle = \sigma_{data}^2 \mathbf{I}$ . Here,  $\mathbf{I}$  is the  $N \times N$  identity matrix and the data are assumed to be uncorrelated.

In contrast to the frequentist approach, Bayesian inversion in section 20.5 computes both the optimal model and the uncertainty associated with it. It also includes prior beliefs in the form of probability functions that are similar to the role played by the model penalty function in regularized inversion. The choice between using the frequentist or the Bayesian approaches to inversion often appear to be a matter of taste for the user, where the Bayesian approach is mathematically more rigorous while the frequentist approach with regularization is much more flexible in accounting for prior beliefs about the model with many types of smoothness or sparsity regularizers.

#### Key Idea 28.5.1. Bayesian and Frequentist Philosophies

- **Bayesian perspective:** both data and parameters are RVs so they can be assigned a probability distribution function (PDF). Therefore, Bayes' Theorem can be used to learn about probabilities, i.e. uncertainties, of both unobservable model parameters as well as observable events.
- **Frequentist perspective:** Only the observable data are RVs governed by a probability distribution, so that the model parameters are deterministic and are not characterized by a PDF (Press et al., 2007). However, the uncertainties associated with the model parameters are caused by errors in the data, which can be obtained by computing the, e.g., model covariance matrix  $\mathbf{C}_{model}$ . Here,  $\mathbf{C}_{model}$  contains the data variance information as seen in equation 28.38.

<sup>9</sup>We use an approximation in the equation  $\mathbf{y} \approx \mathbf{L}\mathbf{w}$  because many geophysical data sets have noise in the recorded data  $\mathbf{y}$  so  $\mathbf{y} \neq \mathbf{L}\mathbf{w}$  denotes an inconsistent set of equations.

## 28.6 Summary

We reviewed the important aspects of probability and statistics theory in machine learning. These concepts include the Law of Total Probability, as well as joint, marginal, and conditional probabilities. These theories are used extensively on VAEs, NNs, GANs, maximum likelihood optimization, recurrent NNs, generative models and discriminative networks. Bayes' Theorem is derived, and many examples are used in Chapter 20 to illuminate its meaning. The Gaussian probability distribution is defined and its connection with maximum likelihood optimization is established. If the probability distribution is defined as the Gaussian in equation 28.36 then the maximum likelihood estimate is the same as the least squares solution where  $\mathbf{w}$  is treated as a deterministic variable. The leakage of data errors into the model can be quantified by forming the model covariance matrix, which describes the uncertainty, i.e. variance, of each estimated model parameter.

Therefore, the MLE frequentist approach can give both the optimal model estimate and the uncertainty associated with the estimated model parameters. As will be shown in section 20.5, prior beliefs about the model can be introduced into the solution by incorporating a model penalty term into the objective function, which is known as regularization. This is similar to the Bayesian approach that updates the posterior prediction of the model's conditional probability distribution with the model's prior probability distribution. The choice between using the frequentist or the Bayesian approaches for inversion often appears to be a matter of taste and experience level for the user.

## 28.7 Exercises

1. Show that equation 28.29 is equal to  $P(Y)$  after applying  $\sum_{k=1}^K$  to both sides of the equation. Explain why the denominator on the right-side of equation 28.29 is known as a normalization factor.
2. Why can the expectation operator  $\langle \rangle$  in equation 28.38 pass through the modeling operators  $\mathbf{L}$  and only be applied to the statistical data term  $\mathbf{y}\mathbf{y}^T$  in equation 28.38?
3. For  $\mathbf{y} = (y_1, y_2, y_3)^T$ , show that the data covariance matrix  $\langle \mathbf{y}\mathbf{y}^T \rangle$  reduces to the  $3 \times 3$  identity matrix  $\mathbf{I}$  for zero-mean and uncorrelated data with unit variance.
4. Show that the MLE  $\mathbf{w}^*$  of equation 28.36 is the same as the least squares solution  $\mathbf{w}^* = [\mathbf{L}^T \mathbf{L}]^{-1} \mathbf{L}^T \mathbf{y}$ .

# References

- Aarre, V., 2016, Understanding spectral decomposition by victor aarre: <https://youtu.be/1nDyMHs8zuw>. (Accessed: 2019-04-19).
- Abadi, M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, and M. Devin, 2016, Tensorflow: Large-scale machine learning on heterogeneous distributed systems: arXiv:1603.04467.
- Abma, R., and J. Claerbout, 1995, Lateral prediction for noise attenuation by t-x and f-x techniques: *Geophysics*, **60**, 1887–1896, <https://library.seg.org/doi/abs/10.1190/1.1443920>
- Achanta, R., A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Sasstrunk, 2012, SLIC superpixels compared to state-of-the-art superpixel methods: *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **34**, 2274–2282.
- Affholder, A., Guyot, F., Sauterey, B., R. Ferriere and S. Mazevet, 2021, Bayesian analysis of Enceladus's plume data to assess methanogenesis: *Nat. Astron.*, doi.org/10.1038/s41550-021-01372-6.
- Ahmad, R., H. Xue, S. Giri, Y. Ding, J. Craft, and O. P. Simonetti, 2015, Variable density incoherent spatiotemporal acquisition (VISTA) for highly accelerated cardiac MRI: *Magnetic Resonance in Medicine*, 74, no. 5, 1266–1278.
- Akaike, H., 1973, Information theory and an extension of the maximum likelihood principle: in B. PetrovF. Caaki, eds., *Second International Symposium on Information Theory*: Budapest Akademiai Kiado, 267–281.
- Aki, K. and P. Richards, 2002, *Quantitative Seismology*: University Science Books, [https://www.ldeo.columbia.edu/richards/Aki\\_Richards.html](https://www.ldeo.columbia.edu/richards/Aki_Richards.html).
- Akram, J. and D. Eaton, 2016, A review and appraisal of arrival-time picking methods for downhole microseismic data: *Geophysics*, **81**, KS67–KS87.
- Akter, S., L. Nahar, S. Akter, and T. Huda, 2015, Travel time prediction using Support Vector Machine (SVM) and weighted moving average (WMA): *Int. J. of Eng. Research and Tech.*, **04**, 496–502.
- Al-Anazi, A. and I. Gates, 2010a, Support vector regression for porosity prediction in a heterogeneous reservoir: a comparative study: *Comput. Geosci.*, **36**, 1494–503.
- Allan, C., 2008, Can adaptive management help us? In *Adaptive and Integrated Water Management. Coping with Complexity and Uncertainty*, New York, Springer, 61–73.
- Al-Anazi, A. and I. Gates, 2010b A support vector machine algorithm to classify lithofacies and model permeability in heterogeneous reservoirs: *Engin. Geol.*, **114**, 267–77.
- Alfarraj, M. and G. AlRegib, 2018, Petrophysical property estimation from seismic data using recurrent neural networks: *SEG Technical Program Expanded Abstracts*, SEG, 2141–2146.

- Al-Madani, M., U. b. Waheed, and M. Masood, 2019, Fast and accurate dictionary learning for seismic data denoising using convolutional sparse coding: 89th Annual International Meeting, SEG Technical Program, Expanded Abstracts, 4645–4649.
- Al-Mojel, A., P. Razin, and G. Dera, 2020, High-resolution sedimentology and sequence stratigraphy of the Oxfordian-Kimmeridgian, Hanifa, Jubaila and Arab outcrops along Jabal Tuwaiq, Central Saudi Arabia: *Journal of African Earth Sciences*, <https://doi.org/10.1016/j.jafrearsci.2020.103803>.
- Almuheidib, A. M. and N. Toksöz, 2015, Imaging of near-surface heterogeneities by scattered elastic waves: *Geophysics*, **80**, A83–A88.
- AlRegib, G., M. Deriche, Z. Long, H. Di, Z. Wang, Y. Alaudah, M. A. Shaq, and M. Alfarraj, 2018, Subsurface structure analysis using computational interpretation and learning: A visual signal processing perspective: *IEEE Signal Processing Magazine*, **35**, 82–98, <https://arxiv.org/abs/1812.08756>
- Altman, N., 1992, An introduction to kernel and nearest-neighbor nonparametric regression: *The American Statistician*, **46**, 175–185.
- Alqahtani, H., M. Kavakli-Thorne, and G. Kumar, 2021, Applications of Generative Adversarial Networks (GANs): An updated review: *Arch. Computat. Methods Eng.*, **28**, 525–552. <https://doi.org/10.1007/s11831-019-09388-y>.
- Alyousuf, T. and Yaoguo Li, 2021, Inversion using adaptive physics-based neural network: Application to magnetotelluric inversion: 89th Annual International Meeting, SEG Technical Program, Expanded Abstracts, 1455–1459.
- Anka, A., 2020, YOLO v4: Optimal speed and accuracy for object detection: Towards Data Science, <https://towardsdatascience.com/yolo-v4-optimal-speed-accuracy-for-object-detection-79896ed47b50>.
- Antoniou, A., A. Storkey and H. Edwards, 2017, Data augmentation generative adversarial networks: *arXiv:1711.04340*.
- Aoki, N. and G. T. Schuster, 2009, Fast least-squares migration with a deblurring filter: *Geophysics*, **74**, WCA83–WCA93.
- Apolloni, B., 2009, Support vector machines and MLP for automatic classification of seismic signals at Stromboli volcano: Paper presented at the Neural Nets WIRN09: Proceedings of the 19th Italian Workshop on Neural Nets, Vietri Sul Mare, Salerno, Italy.
- Aravkin, A., S. Becker, V. Cevher, and P. Olsen, P., 2014, A variational approach to stable principal component pursuit: *Conference on Uncertainty in Artificial Intelligence (UAI)*, *arXiv: 1406.1089*.
- Araya-Polo, M., T. Dahlke, C. Frogner, C. Zhang, T. Poggio, and D. Hohl, 2017, Automated fault detection without seismic processing: *The Leading Edge*, **36**, 208–214, doi: 10.1190/tle36030208.1.
- Araya-Polo, M., J. Jennings, A. Adler, and T. Dahlke, 2018, Deep-learning tomography: *The Leading Edge*, **37**, 58–66.
- Arjovsky, M., S. Chintala, and L. Bottou, 2017, Wasserstein GAN: *arXiv:1701.07875*.
- Arpit, D., S. Jastrzebski, N. Ballas, D. Krueger, E. Bengio, M. Kanwal, T. Maharaj, A. Fischer, A. Courville, Y. Bengio and S. Lacoste-Julien, 2017, A closer look at memorization in deep networks: *arXiv:1706.05394*, <https://arxiv.org/abs/1706.05394>

- Ars, J., P. Tarits, S. Hautot, M. Bellanger, and M. Maiä, 2020, Geophysical integration of joint inverted models using principal component analysis: application to geothermal exploration: European Geophysical Union Annual meeting (poster).
- Ashby, D., 2005, Bayesian statistics in medicine: A 25 year review: *Statistics in Medicine*, **25**, 3589–631, DOI:10.1002/sim.2672.
- Astic T. and D. Oldenburg, 2018, Petrophysically guided geophysical inversion using a dynamic Gaussian mixture model prior: 88th Annual International Meeting, SEG Technical Program, Expanded Abstracts, <https://doi.org/10.1190/segam2018-2995155.1>.
- Arora, S., R. Ge, B. Neyshabur and Y. Zhang, 2018, Stronger generalization bounds for deep nets via a compression approach: arXiv:1802.05296.
- Avseth, P., T. Mukerji, and G. Mavko, 2005, *Quantitative Seismic Interpretation: Applying Rock Physics Tools to Reduce Interpretation Risk*: Cambridge University Press.
- Ba, J. L., J. Kiros and G. Hinton, 2016, Layer normalization, arXiv:1607.06450.
- Bai, T. and P. Tahmasebi, 2022, Attention-based LSTM-FCN for earthquake detection and location: *Geophysical Journal International*, **228**, 1568–1576, <https://doi.org/10.1093/gji/ggab401>.
- Bachrach, R., 2006, Joint estimation of porosity and saturation using stochastic rock-physics modeling: *Geophysics*, **71**, 053–063, doi: 10.1190/1.2235991.GPYSA70016–8033.
- Bader, S., X. Wu, and S. Fomel, 2019, Missing log data interpolation and semiautomatic seismic well ties using data matching techniques: *Interpretation*, 7, no. 2, T347–T361.
- Baechle, G., A. Colpaert, G. Eberli, and R. Weger, 2008, Effects of microporosity on sonic velocity in carbonate rocks: *Geophysics*, **27**, 1012–1018.
- Bahorich, M. and S. Farmer, 1995, 3-D seismic discontinuity for faults and stratigraphic features: The coherence cube: <https://doi.org/10.1190/1.1887523>.
- Bakir, G.H., A. Zien, and K. Tsuda, 2004, Learning to find graph pre-images: in Rasmussen, C.E., H. Bülthoff, B. Schölkopf, and M. Giese (eds), *Pattern Recognition, DAGM, Lecture Notes in Computer Science*, **3175**, Springer, Berlin, Heidelberg. [https://doi.org/10.1007/978-3-540-28649-3\\_31](https://doi.org/10.1007/978-3-540-28649-3_31).
- Ball, G. and Hall, D., 1965, ISODATA, a novel method of data analysis and pattern classification: Technical report NTIS, AD 699616. Stanford Research Institute, Stanford, CA, <https://www.worldcat.org/title/isodata-a-novel-method-of-data-analysis-and-pattern-classification/oclc/23236650>
- Barala, S. and P. Mohanty, 2016, Enhancement of geologic interpretation using a new poststack seismic attribute: 86th Annual International Meeting, SEG Technical Program, Expanded Abstracts, 1677–1681.  
<https://library.seg.org/doi/abs/10.1190/segam2016-13959241.1>
- Barrash, W. and R. Morin, 1997, Recognition of units in coarse, unconsolidated braided-stream deposits from geophysical log data with principal components analysis: *Geology*, **25**, 687–690.
- Barghout, L., 2015, *Spatial-Taxon information granules as used in iterative fuzzy-decision-making for image segmentation: Granular Computing and Decision-Making*, Springer International Publishing, 285–318.  
[https://link.springer.com/chapter/10.1007/978-3-319-16829-6\\_12](https://link.springer.com/chapter/10.1007/978-3-319-16829-6_12)

- Baydin, A., B. Pearlmutter, A. Radul, J. Siskind, 2018, Automatic differentiation in machine learning: a survey: *Journal of Machine Learning Research*, 18: 1–43.  
<https://dl.acm.org/doi/abs/10.5555/3122009.3242010>
- Beck, A. and M. Teboulle, 2009, A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. Imag. Sci.*, 2(1), 183–202.
- Beck, A., and M. Teboulle, 2009, A fast iterative shrinkage-thresholding algorithm for linear inverse problems: *SIAM Journal on Imaging Sciences*, 2, no. 1, 183–202.
- Beckouche, S., and J. Ma, 2014, Simultaneous dictionary learning and denoising for seismic data: *Geophysics*, **79**, no. 3, A27–A31.
- Bekara, M. and M. Van der Baan, 2007, Local singular value decomposition for signal enhancement of seismic data: *GEOPHYSICS*, **72**, V59–V65.
- Bekara, M., and M. Van der Baan, 2009, Random and coherent noise attenuation by empirical mode decomposition: *Geophysics*, **74**, no. 5, V89–V98.
- Bengio, Y., P. Simard, and P. Frasconi, 1994, Learning long-term dependencies with gradient descent is difficult: *Neural Networks, IEEE Transactions*, 5(2), 157–166.
- Bengio, Y., 2009, Learning deep architectures for AI. *Found: Trends Mach. Learn.*, 2(1), 1–127
- Bengio, Y., N. Boulanger-Lewandowski, and R. Pascanu, 2013, Advances in optimizing recurrent networks: *IEEE International Conference on Acoustics, Speech and Signal Processing*, 8624–8628.
- Ben-Hur, A., D. Horn, H. Siegelmann, and V. Vapnik, 2001, Support vector clustering: *Journal of Machine Learning Research*, 2(12), 125–137 DOI: 10.1162/15324430260185565
- Ben-Hur, A. and J. Weston, 2010, A user’s guide to Support Vector Machines: *Methods in molecular biology*, (Clifton, N.J.), **609**, 223–239, DOI: 10.1007/978-1-60327-241-4\_13
- Beniaguev, D., I. Segev, and M. London, 2021, Single cortical neurons as deep artificial neural networks: *Neuron*, **109**, 2727–2739.
- Berahas, A., M. Jahani and M. Takac, 2019, Quasi-Newton methods for deep learning: forget the past, just sample: *arXiv*: 1901.09997v3.
- Beresford, G. and G. Janex, 2005, A practical approach to OBC summation and geophone calibration in areas of shallow water and hard seafloor: 9th International Congress of the Brazilian Geophysical Society, 1466–1470.
- Bergen, K. J., P. A. Johnson, M. V. de Hoop, and G. C. Beroza, 2019, Machine learning for data-driven discovery in solid earth geoscience: *Science*, **363**, eaau0323, doi: 10.1126 / science.aau0323.
- Berthelot, D., C. Raffel, A. Roy, and I. Goodfellow, 2018, Understanding and improving interpolation in autoencoders via an adversarial regularizer: *arXiv*:1807.07543v2.
- Beyer, K., J. Goldstein, R. Ramakrishnan, and U. Shaft, 1999, When Is “Nearest Neighbor” meaningful?: *Proc. Int. Conf. Database Theory*, 217–235.
- Bezdek, J., 2013, *Pattern Recognition with Fuzzy Objective Function Algorithms*: Springer Science & Business Media, ISBN 147570450X, 9781475704501.
- Bharadwaj, P., L. Demanet, and A. Fournier, 2018, Focused blind deconvolution of interferometric Green’s functions: 88th Annual International Meeting, SEG Technical Program, Expanded Abstracts, 4085–4090.



- Bin, H., C. Weihai, W. Xingming, and L. Chun-Liang, 2017, High-quality face image SR using conditional Generative Adversarial Networks: arXiv:1707.00737.
- Bishop, T., K. Bube, R. Cutler, R. Langan, P. Love, J. Resnick, R. Shuey, D. Spindler, and H. Wyld, 1985, Tomographic determination of velocity and depth in laterally varying media: *Geophysics*, **50**, 903–923.
- Bishop, C., 1995a, Training with noise is equivalent to Tikhonov regularization: *Neural Computation*, **7**, 108–116.
- Bishop, C., 1995b, *Neural Networks for Pattern Recognition*: Oxford University Press.
- Bishop, C., 2006, *Pattern Recognition and Machine Learning*: Springer Press.
- Biswas, M., V. Kuppili, L. Saba, D. Edla, H. Suri, E. Cuadrado-Godia, J. Laird, R. Marinho, J. Sanches, A. Nicolaides, and J. Suri, 2019, State-of-the-art review on deep learning in medical imaging: *Frontiers in Bioscience*, **24**, 380–406.
- Bleistein, N., 1984, *Mathematical Methods for Wave Phenomena*: Academic Press Inc.
- Bochkovskiy, A., C. Wang, and H. Liao, 2020, YOLOv4: optimal speed and accuracy of object detection: arXiv:2004.10934.
- Boiero, D., E. Wiarda, and P. Vermeer, 2013, Surface-and guided-wave inversion for near-surface modeling in land and shallow marine seismic data: *The Leading Edge*, **32**, 638–646.
- Bonar, D., and M. Sacchi, 2012, Denoising seismic data using the nonlocal means algorithm: *Geophysics*, **77**, no. 1, A5–A8.
- Boonyasiriwat, C., P. Valasek, P. Routh, B. Macy, W. Cao, and G. T. Schuster, 2009, A multiscale method for time-domain waveform tomography: *Geophysics* **74**, WCC59–WCC68.
- Boonyasiriwat, C., and G.T. Schuster, 2010, 3D multisource full-waveform inversion using dynamic random phase encoding: 80th Annual International Meeting, SEG Technical Program, Expanded Abstracts, DOI: 10.1190/1.3513025.
- Borsuk, M., 2004, Predictive assessment of fish health and fish kills in the Neuse River Estuary using elicited expert judgment: *Hum. Ecol. Risk Assess*, **10**, 415–434.
- Bowman, A. P., 2004, Sequence stratigraphy and reservoir characterization in the Columbus Basin, Trinidad: Ph.D. dissertation, Imperial College.
- Boyd, S., N. Parikh, E. Chu, B. Peleato, and J. Eckstein, 2011, Distributed optimization and statistical learning via the alternating direction method of multipliers: *Foundations and Trends in Machine learning*, **3**, no. 1, 1–122.
- Boyd, S. and L. Vandenberghe, 2004, *Convex Optimization*, Cambridge University Press. p. 216.
- Bradley, W. and R. Hardisty, 2019, Unsupervised seismic facies using Gaussian mixture models: *Interpretation*, **7**, SE93–SE111.
- Bristow, H., A. Eriksson, and S. Lucey, 2013, Fast convolutional sparse coding: *Computer Vision and Pattern Recognition (CVPR)*, 2013 IEEE Conference on, IEEE, 391–398.
- Bromley, J., N. Jackson, O. Clymer, A. Giacomello, and F. Jensen, 2005, The use of Hugin to develop Bayesian networks as an aid to integrated water resource planning: *Environ Model Software*, **20**, 231–242.

- Brownlee, J., 2017, What is the difference between test and validation datasets?  
<https://machinelearningmastery.com/difference-test-validation-datasets/> :  
*text = ValidationdatasetTestfit*
- Brownlee, J., 2019a, A gentle introduction to the Rectified Linear Unit (ReLU): <https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/>.
- Brownlee, J., 2019b, 18 impressive applications of Generative Adversarial Networks (GANs): <https://machinelearningmastery.com/impressive-applications-of-generative-adversarial-networks/>.
- Bube, K. and R. Langan, 1997, Hybrid l1/l2 minimization with applications to tomography: *Geophysics*, **62**, 1183–1195.
- Bube, K. and R. Langan, 2008, Resolution of slowness and reflectors in crosswell tomography with transmission and reflection traveltimes: *Geophysics*, **73**, VE321–VE335.
- Buland, A., and Omre, H., 2003, Bayesian linearized AVO inversion: *Geophysics*, **68**, 185–198.
- Buland, A., and Y. El Ouair, 2006, Bayesian time-lapse inversion: *Geophysics*, **71**, R43–R48, doi: 10.1190/1.2196874.GPYSA70016–8033.
- Buland, A., O. Kolbjørnsen, R. Hauge, O. Skjaeveland, and K. Duffaut, 2008, Bayesian lithology and fluid prediction from seismic prestack data: *Geophysics*, **73**, C13–C21, doi: 10.1190/1.2842150.GPYSA70016–8033.
- Buland, A., O. Kolbjørnsen and A. J. Carter, 2011, Bayesian Dix inversion: *Geophysics*, **76**, R15–R22, doi: 10.1190/1.3552596.GPYSA70016–8033.
- Buland, A., and O. Kolbjørnsen, 2012, Bayesian inversion of CSEM and magnetotelluric data: *Geophysics*, **77**, E33–E42, doi: 10.1190/geo2010-0298.1.GPYSA70016–8033.
- Bunks, C., F. Saleck, S. Zaleski, and G. Chavent, 1995, Multiscale seismic waveform inversion: *Geophysics*, **60**, 1457–1473.
- Bushaev, V., 2018, Adam-latest trends in deep learning optimization:  
<https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c>
- Candès, E. and T. Tao, 2005, Decoding by linear programming: *IEEE Trans. Inf. Theory*, **51**, 4203–4215.
- Candès, E. and J. Romberg, 2007, Sparsity and incoherence in compressive sampling: *Inverse Problems*, **23**, 969–983.
- Cano, E., J. Akram and D. Peter, 2021, Automatic seismic phase picking based on unsupervised machine-learning classification and content information analysis: *Geophysics*, **86**, V299–V315.
- Cantrell, D. L., and R. M. Hagerty, 1999, Microporosity in Arab Formation carbonates, Saudi Arabia: *GeoArabia*, **4**, 129–154.
- Cao, J., and B. Roy, 2017, Time-lapse reservoir property change estimation from seismic using machine learning: *The Leading Edge*, **36**, 234–238.
- Carion, N., F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, 2020, End-to-end object detection with Transformers: In *Proceedings of ECCV*, 213–229, [https://doi.org/10.1007/978-3-030-58452-8\\_13](https://doi.org/10.1007/978-3-030-58452-8_13).

- Carreira, J. and C. Sminchisescu, 2010, Constrained parametric mincuts for automatic object segmentation: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, DOI: 10.1109/CVPR.2010.5540063
- Cary, P., and Chapman, C., 1988, Automatic 1-D waveform inversion of marine seismic refraction data: Geophysical Journal, **93**, 527–546.
- Cavalca, M., Fletcher, R. P. and Nichols, D. [2018] Image-domain least-squares migration in rapidly-varying media: Practical considerations. *EAGE and SBGf workshop on LSM in Rio de Janeiro*, Expanded abstracts, Nov. 28–29.
- Cervantes, J., F. Garcia-Lamont, L. Rodriguez-Mazahua, and A. Lopez, 2020, A comprehensive survey on support vector machine classification: Applications, challenges and trends: Neurocomputing, **408**, 189–215.
- Chang, D., 2020, Effect of batch size on neural net training: <https://medium.com/deep-learning-experiments/effect-of-batch-size-on-neural-net-training-c5ae8516e57>.
- Chanerley, A., and N. Alexander, 2002, An approach to seismic correction which includes wavelet de-noising: Proceedings of the sixth conference on Computational structures technology, Civil-Comp Press, 107–108.
- Chapelle, O., 2007, Training a support vector machine in the primal: Neural Computation, 1155–1178.
- Chau, L., X. Li, W. Yu and J. Cervantes, 2010, Support vector candidates pre-selection strategy based on non convex hulls: Program and Abstract Book, 7th International Conference on Electrical Engineering, Computing Science and Automatic Control, 345–350.
- Chau, L., X. Li, and W. Yu, 2013, Convex and concave hulls for classification with support vector machine: Neurocomputing, **122**, 198–209.
- Chavent, G., and R. E. Plessix, 1999, An optimal true-amplitude least-squares prestack depth-migration operator: Geophysics, **64**, 508–515.
- Chawla, N., K. Bowyer, L. Hall, and W. Kegelmeyer, 2002, SMOTE: synthetic minority over-sampling technique: J. Artif. Intell. Res., **16**, 321–357.
- Che, T., Y. Li, A. Jacob, Y. Bengio, and W. Li, 2016, Mode regularized generative adversarial networks: arXiv:1612.02136.
- Chemingui, N., Liu, F. and Lu, S., 2018, Least-squares migration beyond primaries: *EAGE and SBGf workshop on LSM in Rio de Janeiro*, Expanded abstracts, Nov. 28–29.
- Chen, N., Zhu, J., Chen, J. and T. Chen, 2018, Dropout training for SVMs with data augmentation: Front. Comput. Sci. 12, 694–713, doi.org/10.1007/s11704-018-7314-7.
- Chen, Q., and S. Sidney, 1997, Seismic attribute technology for reservoir forecasting and monitoring: The Leading Edge, **16**, 445–448.
- Chen, Y., 2016, Dip-separated structural filtering using seislet transform and adaptive empirical mode decomposition based dip filter: Geophysical Journal International, **206**, no. 1, 457–469.
- Chen, Y., J. Ma, and S. Fomel, 2016, Double-sparsity dictionary for seismic noise attenuation: Geophysics, **81**, no. 2, V103–V116.
- Chen, Y., G. Dutta, W. Dai, and G. T. Schuster, 2017, Q-least-squares reverse time migration with viscoacoustic deblurring filters: Geophysics, **82**, S425–S438.

- Chen, Y., 2018a, K-means clustering for picking stacking velocity curves in semblance panels: CSIM midyear report.
- Chen, Y., 2018b, Automatic semblance picking by a bottom-up clustering method: SEG workshop on Maximizing Asset Value through Artificial Intelligence and Machine Learning Workshop in Beijing, doi.org/10.1190/AIML2018-12.1.
- Chen, Y., B. Guo, and G. T. Schuster, 2019, Migration of viscoacoustic data using acoustic reverse time migration with hybrid deblurring filters: *Geophysics*, **84**, S127–S136.
- 
- Chen, Y., 2020, Automatic microseismic event picking via unsupervised machine learning: *Geophysical Journal International*, **222** 1750–1764, doi: 10.1093/gji/ggaa186.GJINEA0956-540X.
- Chen, Y. and G.T. Schuster, 2020, Seismic inversion by Newtonian Machine Learning: *Geophysics*, **85**, 1JA–Z18.
- Chen, Y., Huang Y., and Huang L., 2020a, Suppressing migration image artifacts using a support vector machine method: *Geophysics*, **85**, S255–S268.
- Chen, Y., S. Erdinc and G.T. Schuster, 2020b, Seismic inversion by multi-dimensional Newtonian machine learning: 90th Annual International Meeting, SEG Technical Program, Expanded Abstracts, 1691–1694.
- Cho, K., B. van Merriënboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio, 2014, Learning phrase representations using RNN encoder- decoder for statistical machine translation: arXiv: 1406.1078.
- Cho, K., 2015, Introduction to neural machine translation with GPUs (part 1): devblogs.nvidia.com/introduction-neural-machine-translation-with-gpus/.
- Chollet, F., 2016, Xception: Deep learning with depthwise separable convolutions: arXiv:1610.02357.
- Chopra, S., and K. Marfurt, 2006, Seismic attributes-a promising aid for geologic prediction: *CSEG Recorder*, **31**, 110–120.
- Chopra, S., and K. J. Marfurt, 2014, Churning seismic attributes with principal component analysis: 84th Annual International Meeting, SEG Technical Program, Expanded Abstracts, 2672–2676, doi: 10.1190/segam2014-0235.1.
- Choquette, P. W. and L. C. Pray, 1970, Geologic Nomenclature and Classification of Porosity in Sedimentary Carbonates: *AAPG Bulletin*, **54**, 207–250.
- Choudhury, B., R. Swanson, F. Heide, G. Wetzstein, and W. Heidrich, 2017, Consensus convolutional sparse coding: *Proceedings of the IEEE International Conference on Computer Vision*, 4280–4288.
- Christian, L., L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi, 2017, Photo-realistic single image super-resolution using a Generative Adversarial Network: *IEEE Conference on Computer Vision and Pattern Recognition*, 105–114, doi:10.1109 /CVPR.2017.19.
- Chung, J., C. Gulcehre, K. Cho, and Y. Bengio, 2014, Empirical evaluation of gated recurrent neural networks on sequence modeling: arXiv:1412.3555 [cs].
- Ciresan, D. C., A. Giusti, L. M. Gambardella, and J. Schmidhuber, 2013, Mitosis detection in breast cancer histology images with deep neural networks: *International Conference on Medical Image Computing and Computer-assisted Intervention*, Springer, 411–418.

- Claerbout, J. F., 1992, *Earth Soundings Analysis: Processing vs Inversion*: Blackwell Scientific Inc.
- Coates, A., H. Lee, and A. Ng, 2011, An analysis of single-layer networks in unsupervised feature learning: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, PMLR, 215–223.
- Colombo, D. and D. Rovetta, 2018, Coupling strategies in multiparameter geophysical joint inversion: *Geophysical Journal International*, **215**, 1171–1184, <https://doi.org/10.1093/gji/ggy341>
- Colombo, D., E. Turkoglu, W. Li, E. Sandoval-Curiel, and D. Rovetta, 2021a, Physics-driven deep learning inversion with application to transient electromagnetics: *Geophysics*, **86**, E209–E224.
- Colombo, D., E. Turkoglu, W. Li, and D. Rovetta, 2021b, A framework for coupled physics-deep learning inversion and multi-parameter joint inversion: *SEG Technical Program Expanded Abstracts*, 1706-1710. <https://doi.org/10.1190/segam2021-3583272.1>
- Coomans, D. and D. Massart, 1982, Alternative k-nearest neighbor rules in supervised pattern recognition : Part 1. k-Nearest neighbor classification by using alternative voting rules: *Analytica Chimica Acta*. **136**, 15-27, doi:10.1016/S0003-2670(01)95359-0.
- Cortes, C., and V. Vapnik, 1995, Support-vector networks: *Machine Learning*, 20, 273–297.
- Crammer, K. and Y. Singer, 2001, On the algorithmic implementation of multiclass kernel-based vector machines: *J. Mach. Learn. Res.*, **2**, 265–92, [doi.org/10.1162/15324430260185628](https://doi.org/10.1162/15324430260185628).
- Cui, Z., R. Ke, and Y. Wang, 2018, Deep stacked bidirectional and unidirectional LSTM recurrent neural network for network-wide traffic speed prediction: *arXiv:1801.02143 [cs.LG]*.
- Culurciello, E., 2018, The fall of RNN/LSTM: TDS, <https://towardsdatascience.com/the-fall-of-rnn-lstm-2d1594c74ce0>
- Dadi, S., R. Gibson, and W. Sager, 2018, Understanding the late-stage evolution of Shatsky rise using transdimensional acoustic impedance inversion: *Journal of Geophysical Research: Solid Earth*, **123**, 1576–1590.
- Dai, A. and Q. Le, 2015, Semi-supervised sequence learning: *arXiv.org*, arXiv:1511.01432.
- Dai, B., Y. Wang, J. Aston, G. Hua, and D. Wipf, 2018, Connections with robust PCA and the role of emergent sparsity in variational autoencoder models: *Journal of Machine Learning Research*, **18**, 1-42.
- Daubechies, I., M. Defriese, and C. De Mol, 2004, An iterative thresholding algorithm for linear inverse problems with a sparsity constraint: *Commun. Pure Appl. Math*, LVII, 1413–1457.
- Dauphin, Y., A. Fan, M. Auli, and D. Grangier, 2017, Language modeling with gated convolutional networks: *arXiv*: 1612.08083.
- DeCoste, D., 2002, Training invariant support vector machines: *Machine Learning*. 46, 161–190.
- de Figueiredo, L., D. Grana, M. Roisenberg, and B. Rodrigues, 2019, Gaussian mixture Markov chain Monte Carlo method for linear seismic inversion: *Geophysics*, **84**, R463–R476.
- DeGroot, M., 1975, *Probability and Statistics*: Addison-Wesley Publishing Co..

- de Lima, R. and K. Marfurt, 2018, Principal component analysis and K-means analysis of airborne gamma-ray spectrometry surveys: 88th Annual International Meeting, SEG Technical Program, Expanded Abstracts, 2277–2281.
- Delprat-Jannaud, F. and P. Lailly, 1993, Ill-posed and well-posed formulations of the reflection travel time tomography problem: *J. Geophys. Res.*, **98**, 6589–6605.
- De Luca, G., 2020, SVM Vs Neural Network: <https://www.baeldung.com/cs/svm-vs-neural-network>.
- de Matos, M., K. Marfurt, and P. Johann, 2010, Seismic interpretation of self-organizing maps using 2D color displays: *Rev. Bras. Geof.*, **28**, [dx.doi.org/10.1590/S0102-261X2010000400008](https://doi.org/10.1590/S0102-261X2010000400008).
- Dempster, A., N. Laird, and D. Rubin, 1977, Maximum likelihood from incomplete data via the EM algorithm: *Journal of the Royal Statistical Society*, **39**, 1–38.
- Deng, C., H. Pan, S. Fang, A. Konaté, and R. Qin, 2017, Support vector machine as an alternative method for lithology classification of crystalline rocks: *Journal of Geophysics and Engineering*, **14**, 341–349.
- Denton, E., S. Chintala, A. Szlam, and R. Fergus, 2015, Deep generative image models using a Laplacian pyramid of adversarial networks: *arXiv:1506.05751*.
- de Roeck, Y. H., 2002, Sparse linear algebra and geophysical migration: A review of direct and iterative methods: *Numerical Algorithms*, **29**, 283–322.
- Dertat, A., 2017, Applied deep learning-Part 4: Convolutional neural networks: <https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2>
- Dettmer, J. and S. Dosso, 2009, Model selection and Bayesian inference for high-resolution seabed reflection inversion: *The Journal of the Acoustical Society of America*, **125**, 706, [doi.org/10.1121/1.3056553](https://doi.org/10.1121/1.3056553).
- Dettmer, J., S. Dosso, and C. Holland, 2010, Trans-dimensional geoacoustic inversion: *The Journal of the Acoustical Society of America*, **128**, 3393–3405, [doi: 10.1121/1.3500674](https://doi.org/10.1121/1.3500674).
- Deville de Periere, M., C. Durllet, E. Vennin, L. Lambert, R. Bourillot, B. Caline, and E. Poli, 2011, Morphometry of micrite particles in cretaceous microporous limestones of the Middle East: Influence on reservoir properties: *Marine Petroleum and Geology*, **28**, 1727–1750.
- Dharmi, D., 2019, Object detection-localization algorithms: Medium, <https://medium.com/@dhartidhami/object-detection-localization-algorithms-e55b19259cbf>
- Di, H., M. Shafiq and G. AlRegib, 2017, Seismic-fault detection based on multiattribute support vector machine analysis: 87th Annual International Meeting, SEG Technical Program, Expanded Abstracts, 1949–4645.
- Di, H., Z. Li, H. Maniar, and A. Abubakar, 2020, Seismic stratigraphy interpretation by deep convolutional neural networks: A semi-supervised workflow: *Geophysics*, **85**, 1–41.
- Diamantaras, K. and S. Kung, 1996, Principal component neural networks: theory and applications: New York, NY: Wiley.
- Dinh, L., R. Pascanu, S. Bengio, and Y. Bengio, 2017, Sharp minima can generalize for deep nets: *arXiv:1703.04933v2*.
- Doersch, C., 2016, Tutorial on variational autoencoders: *arXiv:1606.05908*.
- Donoho, D. L., 2019, Deepnet spectra and the two cultures of data science: Presented at the Al-Kindi Distinguished Statistics Lectures, King Abdullah University of Science and Technology.

- Donahue, J., L. Hendricks, M. Rohrbach, S. Venugopalan, S. Guadarrama, K. Saenko, and T. Darrell, 2014, Long-term Recurrent Convolutional Networks for Visual Recognition and Description: arXiv:1411.4389.
- Donahue J, P. Krahenbuhl, and T. Darrell, 2016, Adversarial feature learning: arXiv:1605.09782.
- Doshi, C., 2019, Why Relu? Tips for using Relu. Comparison between Relu, Leaky Relu, and Relu-6:  
<https://medium.com/chinesh4/why-relu-tips-for-using-relu-comparison-between-relu-leaky-relu-and-relu-6-969359e48310>.
- Dosso, S. , J. Dettmer, G. Steininger, and C. Holland, 2014, Efficient trans-dimensional Bayesian inversion for geoacoustic profile estimation: *Inverse Problems*, **30**, 114018, doi: 10.1088/0266-5611/30/11/114018.INPEEY0266-5611
- Dumoulin, V., I. Belghazi, B. Poole, O. Mastropietro, A. Lamb, M. Arjovsky, and A. Courville, 2016, Adversarially learned inference: arXiv:1606.00704
- Drucker, H., C. Burges, L. Kaufman, A. Smola and V. Vapnik, 1997, Support vector regression machines: in *Advances in Neural Information Processing Systems*, **9**, 155–161, MIT Press.
- Duarte, L. T., E. Z. Nadalin, K. Nose Filho, R. Zanetti, J. M. Romano, and M. Tygel, 2012, Seismic wave separation by means of robust principal component analysis: *Proceedings of the 20th European Signal Processing Conference, IEEE*, 1494–1498.
- Duchi, J., E. Hazan, and Y. Singer, 2011, Adaptive subgradient methods for online learning and stochastic optimization: *Journal of Machine Learning Research*, **12**, 2121–2159.
- Duda, R. O. and P. E. Hart, 1973, *Pattern Classification and Scene Analysis*, Wiley.
- Duda, R., P. Hart, and D. Stork, 2000, *Pattern Classification*, 2nd Edition, ISBN: 978-0-471-05669-0.
- Duijndam, A., 1988, Bayesian estimation in seismic inversion, part I: Principles: *Geophysical Prospecting*, **36**, 878–898.
- Dumoulin, V. and F. Visinm, 2016, A guide to convolution arithmetic for deep learning: ArXiv, 1603.07285.
- Dunn, J., 1973, A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters: *Journal of Cybernetics*, **3**, 32–57.
- Duquet, B., K. J. Marfurt, and J. A. Dellinger, 2000, Kirchhoff modeling, inversion for reflectivity, and subsurface illumination: *Geophysics*, **65**, 1195–1209.
- Durrant, R. and A. Kabán, 2009, When is ‘nearest neighbour’ meaningful: A converse theorem and implications: *Journal of Complexity*, **25**, 385–397, ISSN 0885-064X, <https://doi.org/10.1016/j.jco.2009.02.011>.
- Dutta, G., 2017, Sparse least-squares reverse time migration using seislets: *Journal of Applied Geophysics*, **136**, 142–155.
- Efron, B., 1979, Bootstrap methods: another look at the jackknife. *Annals of Statistics*, **7**, 1–26.
- Ehrenberg, S. N., A. M. Aqrabi, and P. H. Nadeau, 2008, An overview of reservoir quality in producing Cretaceous strata of the Middle East: *Petroleum Geoscience*, **14**, 307–318.
- Elad, M., and M. Aharon, 2006, Image denoising via sparse and redundant representations over learned dictionaries: *IEEE Transactions on Image processing*, **15**, no. 12, 3736–3745.

- Elbayad, M., L. Besacier, and J. Verbeek, 2018, Pervasive attention: 2D convolutional neural networks for sequence-to-sequence prediction: arXiv:1808.03867.
- El Mohtar, S., B. Ait-El-Fquih, O. Knio, I. Lakkis, and I. Hoteit, 2021, Bayesian identification of oil spill source parameters from image contours: *Marine Pollution Bulletin*, **169**, 112514, ISSN 0025-326X.
- Embry, A. F. and J. E. Klován, 1971, A late Devonian reef tract on north-eastern Banks Island, Northwest Territories: *Bulletin of Canadian Petroleum Geology*, **19**, 730-781.
- Ester, M., H. Kriegel, J. Sander, and X. Xu, 1996, A density-based algorithm for discovering clusters in large spatial databases with noise: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*. AAAI Press., 226-231, 10.1.1.121.9220, ISBN 1-57735-004-9.
- Estrela, V., H. Magalhaes, O. Saotome, 2016, Total variation applications in computer vision: arXiv:1603.09599.
- Ezukwoke, K. and S. Zareian, 2019, Kernel methods for PCA: A comparative study of classical and kernel PCA: DOI: 10.13140/RG.2.2.17763.09760.
- Farnia and Ozdaglar, 2020, GANs may have no Nash equilibria: acXiv: 2002.09124v1.
- Fei, B., 2020, Hyperspectral imaging in medical applications: in *Hyperspectral Imaging Edited by José Manuel Amigo*, Elsevier Publ., Vol. 32, pp. 523-565, doi.org/10.1016/B978-0-444-63977-6.00021-3.
- Felzenszwalb, P.F. and D. Huttenlocher, 2004, Efficient graph-based image segmentation.: *International Journal of Computer Vision*, **59**: 167.  
<https://doi.org/10.1023/B:VISI.0000022288.19776.77>.
- Feng, Z., and G. T. Schuster, 2017, Elastic least-squares reverse time migration: *Geophysics*, **82**, S143-S157.
- Feng, S., L. Passone and G. T. Schuster, 2021a, Superpixel-based convolutional neural network for georeferencing the drone images: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, doi: 10.1109/JSTARS.2021.3065398.
- Feng, S., Zhang, X., Wohlberg, B., Symons, N. and Lin, Y., 2021b, Connect the dots: In situ 4D seismic monitoring of  $CO_2$  storage with spatio-temporal CNNs, arXiv preprint arXiv:2105.11622.
- Finkbeiner, T., V. Chandra, V. Vahrenkamp, B. Yalcin, A. Ramdani, A. Perbawa, A., 2018, Petrophysical and geomechanical properties of Late Jurassic carbonates outcropping in central Saudi Arabia: Correlation with depositional sequences and diagenetic overprints: 13th GEO Middle East Geosciences Conference and Exhibition 2018, Bahrain.
- Fletcher, R., Nichols, D., Bloor, R. and Coates, R. T., 2016, Least-squares migration: Data domain versus image domain using point spread functions: *The Leading Edge*, 35(2), 157-162.
- Fomel, S., J. Berryman, R. Clapp, M. Prucha, 2002, Iterative resolution estimation in least-squares Kirchhoff migration: *Geophysical Prospecting*, **50**, 577-588.
- Fomel, S., 2003, Seismic reflection data interpolation with differential offset and shot continuation: *Geophysics*, **68**, 733-744, <https://doi.org/10.1190/1.1567243>.
- Foster, D., 2019, *Generative Deep Learning*: O'Reilly Media Inc., Sebastopol, CA.
- Frankle, J. and M. Carbin, 2019, The lottery ticket hypothesis: Finding sparse, trainable neural networks: arXiv:1803.03635.



- Frid-Adar, M., I. Diamant, E. Klang, M. Amitai, J. Goldberger and H. Greenspan, 2018, GAN-based synthetic medical image augmentation for increased CNN performance in liver lesion classification: *Neurocomputing*, **321**, 321–331.
- Fukushima, K., 1980, Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position: *Biol. Cybernetics*, **36**, 193–202.
- Fung, G. and O. Mangasarian, 2001, Proximal support vector machine classifiers: 7th ACM Sigkdd international conference on knowledge discovery and data mining, doi.org/10.1145/502512.502527.
- Gaiser, J. and I. Vasconcelos, 2008, Elastic interferometry for OBC data: theory and examples: 70th EAGE Conference and Exhibition, 1073–1077.
- Gan, S., S. Wang, Y. Chen, Y. Zhang, and Z. Jin, 2015, Dealiased seismic data interpolation using seislet transform with low-frequency constraint: *IEEE Geoscience and Remote Sensing Letters*, **12**, 2150–2154, <https://doi.org/10.1109/LGRS.2015.2453119>.
- Gao, H., X. Wu and G. Liu, 2021, ChannelSeg3D: Channel simulation and deep learning for channel interpretation in 3D seismic images: *Geophysics*, **86**, IM73–IM83.
- Gaonkar, B. and Davatzikos, C., 2013, Analytic estimation of statistical significance maps for support vector machine based multi-variate image analysis and classification: *Neuroimage*, **78**, 270–283.
- Garcia, S., 2018, L0 norm, L1 norm, L2 norm and  $L^\infty$  norm: <https://medium.com/montjoile/l0-norm-l1-norm-l2-norm-l-infinity-norm-7a7d18a4f40c>.
- Gautam, T., Y. Zhou, S. Feng, and G.T. Schuster, 2021, Tomographic deconvolution of reflection tomograms: 91st Annual International Meeting, SEG Expanded Abstracts, <https://doi.org/10.1190/segam2021-3595023.1>.
- Gazley, M., K. Collins, J. Roberston<sup>1</sup>, B. Hines, L. Fisher and A. McFarlane, 2015, Application of principal component analysis and cluster analysis to mineral exploration and mine geology: AusIMM New Zealand Branch Annual Conference, 131–139.
- Gebraad, L., C. Boehm, and A. Fichtner, 2020, Bayesian elastic full-waveform inversion using Hamiltonian Monte Carlo: *Journal of Geophysical Research: Solid Earth*, **125**, e2019JB018428.
- Gehring, J., M. Auli, D. Grangier, D. Yarats, and Y. Dauphin, 2017, Convolutional sequence to sequence learning: arXiv:1705.03122v2.
- Gelfand, A. and S. Banerjee, 2017, Bayesian modeling and analysis of Geostatistical data: *Annual Review of Statistics and Its Application*, **4**, 245–266.
- Gelman A (2008) Objections to Bayesian statistics Rejoinder: *Bayesian Anal.*, **3**, 467–477.
- Geng, Z., X. Wu, Y. Shi, and S. Fomel, 2019, Relative geologic time estimation using a deep convolutional neural network: 89th Annual International Meeting, SEG Technical Program, Expanded Abstracts, 2238–2242, DOI: 10.1190 /segam2019-3214459.1. .
- Geoffrion, A. M., 1971, Duality in nonlinear programming: A simplified applications-oriented development: *SIAM Review*, **13**, 1–37.
- Gers, F., J. Schmidhuber, and F. Cummins, 1999, Learning to forget: Continual prediction with LSTM: *Artificial Neural Networks, ICANN 99*, Ninth International Conference on (Conf. Publ. No. 470), pp. 850–855.
- Ghosh, A V. Kulharia, V. Namboodiri, P. Torr, and P. Dokania, 2018, Multi-agent diverse generative adversarial networks: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp 8513–8521.

- Ghosh, P., M. Sajjadi, A. Vergari, M. Black, and B. Scholkopf, 2019, From variational to deterministic autoencoders: ICLR 2020 Conference, arXiv:1903.12436.
- Gill, P., W. Murray, and M. Wright, 1981, Practical Optimization: Academic Press Inc.
- Gillespie, D., 1992, Markov Processes: An Introduction for Physical Scientists: Academic Press Inc., San Diego, CA.
- Giraud, J., E. Pakyuz-Charrier, M. Jessell, M. Lindsay, R. Martin, and V. Ogarko, 2017, Uncertainty reduction through geologically conditioned petrophysical constraints in joint inversion: Geophysics, **82**, ID19–ID34.
- Girshick, R., J. Donahue, T. Darrell, and J. Malik, 2015, Rich feature hierarchies for accurate object detection and semantic segmentation: arXiv:1311.2524v5.
- Girshick, R., 2015, Fast R-CNN: arXiv:1504.08083v2.
- Girshick, R., J. Donahue, T. Darrell and J. Malik, 2016, Region-based convolutional networks for accurate object detection and segmentation: IEEE Transactions on Pattern Analysis and Pattern Recognition, **38**, 142-158.
- Glorot, X. and Y. Bengio, 2010, Understanding the difficulty of training deep feedforward neural networks: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, 249–256.
- Goled, S., 2021, Why Transformers are increasingly becoming as important as RNN And CNN: <https://analyticsindiamag.com/why-transformers-are-increasingly-becoming-as-important-as-rnn-and-cnn/>
- Golovko, V., A. Brich, and A. Sachenko, 2017, A shallow convolutional neural network for accurate handwritten digits classification: Communications in Computer and Information Science, **673**, 77-85 DOI: 10.1007/978-3-319-54220-1\_8.
- Golub, G. and U. von Matt, 1997, Generalized cross-validation for large-scale problems: Journal of Computational and Graphical Statistics, **6**, 1–34.
- Goncalves, C.A., 1998, Lithologic interpretation of downhole logging data from the Cote d'Ivoire-Ghana Transform Margin: A Statistical Approach: Mascle, J., Lohmann, G.P. and Moullade, M. (eds.), Proceedings of the Ocean Drilling Program, Scientific Results, **159**, 157–170.
- Gonzalez, E. F., T. Mukerji, and G. Mavko, 2001, AVO signatures of Eastern Venezuela gas sands: Feasibility and uncertainty estimation: 71st Annual International Meeting, SEG Technical Program, Expanded Abstracts, 215-218.
- Gonzalez, E. F., S. Gesbert, and R. Hofmann, 2012a, Importance of geology and rock physics in quantitative seismic interpretation: A turbidite case study: Geophysical Society of Houston Journal, **3**, 12-15.
- Gonzalez, E. F., R. Hofmann, and S. Gesbert, 2012b, Importance of coupled geological and rock physics prior information in quantitative interpretation of seismic data: Presented at the Workshop on Applications and Challenges of Rock Physics for Quantitative Geophysical Interpretation, EAGE.
- Gonzalez, E., T. Mukerji and G. Mavko, 2008, Seismic inversion combining rock physics and multiple-point geostatistics: Geophysics, **73**, R11-R21.
- Gonzalez, E., S. Gesbert, and R. Hofmann, 2016, Adding geologic prior knowledge to Bayesian lithofluid facies estimation from seismic data: Interpretation, **4**, SL1-SL8.

- Goodfellow, I., 2014, On distinguishability criteria for estimating generative models: arXiv:1412.6515.
- Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, 2014, Generative adversarial networks: arXiv:1406.2661.
- Goodfellow, I., Y. Benigo, and A. Courville, 2016, Deep Learning, MIT Press.
- Goodfellow, I., 2016, Tutorial: Generative Adversarial Networks: arXiv:1701.00160v4.
- Google, 2017, Tensorflow tutorials: convolutional neural networks, <https://www.tensorflow.org/tutorials/images/cnn>, August 2017.
- Goutte, C., P. Toft, E. Rostrup, F. Arup Nielsen, L.Hansen, 1999, On Clustering fMRI time series: NeuroImage, **9-3**, 298-310.
- Grana, D. and A. del Monte, 2010, A probabilistic approach to 3D joint estimation of reservoir properties based on Gaussian Mixture Models: 80th Annual International Meeting, SEG Technical Program, Expanded Abstracts, 2351-2355.
- Grana, G., 2016, Bayesian linearized rock-physics inversion: Geophysics, **81**, D625–D641. <https://doi.org/10.1190/geo2016-0161.1>.
- Graves, A. and J. Schmidhuber, 2005, Framewise phoneme classification with bidirectional LSTM and other neural network architectures: Neural Networks, **18**, 602–610.
- Graves, A., N. Jaitly, and A.-r. Mohamed, 2013, Hybrid speech recognition with deep bidirectional LSTM: in Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop, 273–278.
- Greff, K., R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber, 2015, LSTM: A search space odyssey: IEEE Transactions on Neural Networks and Learning Systems. 28 (10): 2222–2232.
- Gu, S., W. Zuo, Q. Xie, D. Meng, X. Feng, and L. Zhang, 2015, Convolutional sparse coding for image super-resolution: Proceedings of the IEEE International Conference on Computer Vision, 1823–1831.
- Guerrero-Pena, F., P. Fernandez, T. Ren, M. Yui, E. Rothenberg, and A. Cunha, 2018, Multiclass weighted loss for instance segmentation of cluttered cells: arXiv:1802.07465v1.
- Guest, Y., 2011, Your gateway to the Bayesian realm: <https://astrobites.org/2011/11/26/your-gateway-to-the-bayesian-realm/>
- Guitton, A., 2002, Coherent noise attenuation using inverse problems and prediction-error filters: First Break, **20**, no. 3.
- Guitton, A., 2004, Amplitude and kinematic corrections of migrated images for nonunitary imaging operators: Geophysics, **69**, 1017–1024.
- Guitton, A., 2018, 3D convolutional neural networks for fault interpretation: 80th Annual International Conference and Exhibition, EAGE, Extended Abstracts, 1–5.
- Gulati, A., J. Qin, C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu, and R. Pang, 2020, Conformer: Convolution-augmented Transformer for speech recognition: In Proceedings of Interspeech, 5036-5040, <https://doi.org/10.21437/Interspeech.2020-3015>.
- Guo, B., and G. T. Schuster, 2017, Wave-equation migration velocity analysis using plane-wave common image gathers: Geophysics, **82**, doi: 10.1190/geo2016-0653.1.GPYSA70016-8033

- Guo, G., J. Zhang, D. Liu, Y. Zhang, and D. Zhang, 2019, Application of bi-directional long short-term memory recurrent neural network for seismic impedance inversion: 81st EAGE Conference and Exhibition, Extended Abstract, DOI: 10.3997/2214-4609.201901386.
- Guo, L., S. Ye, J. Han, H. Zheng, H. Gao, D. Z. Chen, J. Wang, and C. Wang, 2020, SSR-VFD: Spatial super-resolution for vector field data analysis and visualization: In Proc. 2020 IEEE Pacific Visualization Symposium, 71-80.
- Han, H., W. Wang, and B. Mao, 2005, Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning: Advances in Intelligent Computing, Lecture Notes in Computer Science, **3644**, Berlin-Springer, 878–887, 10.1007/11538059\_91.
- Han, S., J. Pool, J. Tran, and W. Dally, 2015, Learning both weights and connections for efficient neural network: in C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, Advances in Neural Information Processing Systems, **28**, 1135–1143, Curran Associates, Inc..
- Han, J. and C. Wang, 2020, SSR-TVD: Spatial super-resolution for time-varying data analysis and visualization: IEEE Transactions on Visualization and Computer Graphics.
- Hanafy, S., S. Jonsson and Y. Klinger, 2014, Imaging normal faults in alluvial fans using geophysical techniques: Field example from the coast of Gulf of Aqaba, Saudi Arabia: 84th Annual International Meeting, SEG Technical Program, Expanded Abstracts, 4670–4674.
- Haney, M. M. and H. Douma, 2012, Rayleigh-wave tomography at coronation field, Canada: The topography effect: The Leading Edge, 31, 54–61.
- Hasiuk, F., S. Kaczmarek, and F. Shawn, 2016, Diagenetic origins of the calcite microcrystals that host microporosity in limestone reservoirs: Journal of Sedimentary Research, **86**, 1163–1178.
- Hastie, T. and R. Tibshirani, 1996, Discriminant analysis by Gaussian Mixtures: Royal Statistical Society, <https://doi.org/10.1111/j.2517-6161.1996.tb02073.x>.
- Hastie, T., R. Tibshirani, and J. Friedman, 2001, The elements of statistical learning: data mining, inference, and prediction: New York, Springer, ISBN 0-387-95284-5. OCLC 46809224.
- Hawkins, R., R. C. Brodie, and M. Sambridge, 2018, Trans-dimensional Bayesian inversion of airborne electromagnetic data for 2D conductivity profiles: Exploration Geophysics, **49**, 134–147, doi: 10.1071/EG16139.
- He, K., X. Zhang, S. Ren and J. Sun, 2015, Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), arXiv:1502.01852.
- He, K., X. Zhang, S. Ren and J. Sun, 2016, Deep residual learning for image recognition: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), DOI: 10.1109/CVPR.2016.90.
- He, K., G. Gkioxari, P. Dollar, and R. Girshick, 2017, Mask R-CNN: IEEE International Conference in Computer Vision (ICCV), 2980–2988.
- Heide, F., W. Heidrich, and G. Wetzstein, 2015, Fast and flexible convolutional sparse coding: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 5135–5143.
- Hennenfent, G., and F. J. Herrmann, 2006, Seismic denoising with nonuniformly sampled curvelets: Computing in Science Engineering, **8**, no. 3, 16–25.

- Henriksen, H.J., P. Zorrilla-Miras, A. de la Hera, and M. Brugnach, 2012, Use of Bayesian belief networks for dealing with ambiguity in integrated groundwater management: *Integr. Environ. Assess. Manag.*, **8**, 430–444.
- Herrmann, F. J., and G. Hennenfent, 2008, Non-parametric seismic data recovery with curvelet frames: *Geophysical Journal International*, **173**, 233–248, <https://doi.org/10.1111/j.1365-246X.2007.03698.x>.
- Herrmann, F. J., C. R. Brown, Y. A. Erlangga, and P. P. Moghaddam, 2009, Curvelet-based migration preconditioning and scaling: *Geophysics*, **74**, A41–A46.
- Hinton, G. and R. Salakhutdinov, 2006, Reducing the dimensionality of data with neural networks: *Science*, **313**, 504–507.
- Hinton, G., 2013, Non-linear dimensionality reduction: University of Toronto Computer Science, CSC 2535: Lecture 11 Non-linear dimensionality reduction. <https://www.cs.toronto.edu/hinton/csc2535/notes/lec11new.pdf>.
- Hitawala, S., 2018, Comparative study on generative adversarial networks: arXiv:1801.04271.
- Hochreiter, S., Y. Bengio, P. Frasconi, and J. Schmidhuber, 2001, Gradient flow in recurrent nets: the difficulty of learning long-term dependencies: *A Field Guide to Dynamical Recurrent Neural Networks*, IEEE Press. bibtex/279df6721c014a00bfac62abd7d5a9968/schaul.
- Hole, J. and C. Zelt, 1995, 3-D finite-difference reflection traveltimes: *Geophys. J. Internat.*, **121**, 427–434.
- Hong, Y., 2019, Comparison of Generative Adversarial Networks architectures which reduce mode collapse: arXiv:1910.04636.
- Hochreiter, S. and Schmidhuber, J., 1997, Long short-term memory: *Neural computation*, **9**, 1735–1780.
- Horgan, J., 2016, Bayes's Theorem: What's the big deal?: *Scientific American: Blog* at <https://blogs.scientificamerican.com/cross-check/bayes-s-theorem-what-s-the-big-deal/>
- Horn, B.K. and Schunck, B.G., 1981. Determining optical flow: *Artificial intelligence*, **17**, 185–203.
- Hornik, K., 1991, Approximation capabilities of multilayer feedforward networks: *Neural Networks*, **4**(2), 251–257.
- Hornik, K., M. Stinchcombe, and H. White, 1989, Multilayer feedforward networks are universal approximators: *Neural Networks*, **2**(5), 359–366.
- Hotelling, H., 1933, Analysis of a complex of statistical variables into principal components: *Journal of educational psychology*, **24**, 417.
- Hotelling, H., 1936, Relations between two sets of variates: *Biometrika*, **28**, 321–377.
- Hou, Y., 2019, Pixel level pavement crack detection using deep convolutional neural network with residual blocks: MS Thesis, Univ. of Missouri Computer Science Department, <https://hdl.handle.net/10355/70143>.
- Hsu, C., C. Chang and C. Lin, 2003, A practical guide to Support Vector Classification: Technical Report, Department of Computer Science and Information Engineering, University of National Taiwan, Taipei, 1–12, [www.ee.columbia.edu/~sfchang/course/svia/papers/svm-practical-guide.pdf](http://www.ee.columbia.edu/~sfchang/course/svia/papers/svm-practical-guide.pdf).
- Hu, J. and Schuster G. T., 1998, Migration deconvolution: *Proceedings of SPIE*, **3453**, 118–124.

- Hu, J., G. T. Schuster, and P. Valasek, 2001, Migration deconvolution: Poststack data: Geophysics, **66**, 939–952.
- Hu, S., W. Zhao, Z. Xu, H. Zeng, Q. Fu, L. Jiang, S. Shi, Z. Wang, and W. Liu, 2017, Applying principal component analysis to seismic attributes for interpretation of evaporite facies: Lower Triassic Jialingjiang Formation, Sichuan Basin, China: Interpretation, **5** T461–T475.
- Hu, L., X. Zheng, Y. Duan, X. Yan, Y. Hu, and X. Zhang, 2019, First-arrival picking with a U-net convolutional network: Geophysics, **84**, U45–U57.
- Hu, Y., Q. Zhang, W. Zhao, and H. Wang, 2021, TransQuake: A transformer-based deep learning approach for seismic P-wave detection: Earthquake Research Advances, **1**, ISSN 2772-4670, <https://doi.org/10.1016/j.eqrea.2021.100004>.
- Huang, Y. and G.T. Schuster, 2014, Resolution limits for wave equation imaging: Journal of Applied Geophysics, **107**, 137–148.
- Huang, W., R. Wang, Y. Zhou, Y. Chen, and R. Yang, 2016, Improved principal component analysis for 3D seismic data simultaneous reconstruction and denoising: 86th Annual International Meeting, SEG Technical Program, Expanded Abstracts, 4102–4106, [doi.org/10.1190/segam2016-13858769.1](https://doi.org/10.1190/segam2016-13858769.1).
- Huang, L., X. Dong, and T. E. Clee, 2017, A scalable deep learning platform for identifying geologic features from seismic attributes: The Leading Edge, **36**, 249–256.
- Hubel, D. and T. Wiesel, 1959, Receptive fields of single neurones in the cat’s striate cortex: The Journal of Physiology, **124**, 574–591.
- Hubel, D. and T. Wiesel, 1962, Receptive fields, binocular interaction and functional architecture in Cat’s visual cortex: J. Physiol., **160**, 106–154.
- Hughes, T., I. Williamson, M. Minkov, and S. Fan, 2019, Wave physics as an analog recurrent neural network: Science Advances, **5**, eaay6946, DOI : 10.1126/sciadv.aay6946.
- Hughes, G.F., 1968, On the mean accuracy of statistical pattern recognizers: IEEE Transactions on Information Theory, **14**, 55–63.
- Hussain, M., 2020, GANs-what and where? <https://medium.com/the-cyph /gans-what-and-where-b377672283c5>.
- Hussein, M., R. Stewart, and J. Wu, 2020, Which seismic attributes are best for subtle fault detection? Interpretation, pp. 1-5, <https://doi.org/10.1190/int-2020-0068.1>
- Hussein, M., R. Stewart, D. Sacrey, D. Johnston, and J. Wu, 2021, Unsupervised machine learning for time-lapse seismic studies and reservoir monitoring: Interpretation, T791-T807.
- Hyndman, R.J. and G. Athanasopoulos, 2018, Forecasting: Principles and Practice: 2nd edition, OTexts: Melbourne, Australia, <https://otexts.com/fpp2/>.
- Ibrahim, A., and M. Sacchi, 2014, Eliminating blending noise using fast apex shifted hyperbolic radon transform: Presented at the 76th EAGE Conference and Exhibition 2014.
- Infante-Pacheco, V., J. Montalvo-Arrieta, I. de León, and F. Velasco-Tapia, 2020, Improvement of shallow seismic characterization using the Singular Value Decomposition (SVD) method in seismic data inversion: A case study of a site in Northeast Mexico, Journal of Environmental and Engineering Geophysics, **25**, 447–462.
- Ioffe, S., and C. Szegedy, 2015, Batch normalization: Accelerating deep network training by reducing internal covariate shift: arXiv:1502.03167.

- Isola, P., J. Zhu, T. Zhou and A. Efros, 2018, Image-to-Image translation with conditional adversarial networks: arXiv: 1611.07004.
- Jain, A., 2010, Data clustering: 50 years beyond K-means: *Pattern Recognition*, **31**, 651–666.
- Jackson, D., 1979, The use of a priori data to resolve non-uniqueness in linear inversion: *Geophysical Journal of the Royal Astronomical Society*, **57**, 137–157.
- Jackson, D. D., and Matsuura, M., 1985, A Bayesian approach to nonlinear inversion: *Journal of Geophysical Research*, **90**, 581–591.
- Jacquemyn, C., M. D. Jackson, G. J. Hampson, C. M. John, D. L. Cantrell, D. L., R. Zühlke, A. J. AbuBshait, R. F. Lindsay, and R. Monsen, 2018, Geometry, spatial arrangement and origin of carbonate grain-dominated, scour-fill and event-bed deposits: Late Jurassic Jubaila Formation and Arab-D Member, Saudi Arabia. *Sedimentology*, **65**, 1043-1066.
- Jaques, N., S. Taylor, A. Sano, and R. Picard, 2017, Multimodal autoencoder: A deep learning approach to filling in missing sensor data and enabling better mood prediction: 2017 Seventh International Conference on Affective Computing and Intelligent Interaction (ACII), October 2017, Texas, USA.
- Jas, M., T. D. La Tour, U. Simsekli, and A. Gramfort, 2017, Learning the morphology of brain signals using alpha-stable convolutional sparse coding: *Advances in Neural Information Processing Systems*, 1099–1108.
- Jeeva, M., 2018, The scuffle between two algorithms: Neural network vs Support Vector Machine: <https://medium.com/analytics-vidhya/the-scuffle-between-two-algorithms-neural-network-vs-support-vector-machine-16abe0eb4181>.
- Jenseit, G. and B. McGillivray, 2014, Multivariate analyses of affix productivity in translated English: in *Quantitative Methods in Corpus-Based Translation Studies: A practical guide to descriptive translation research*, editors: Michael P. Oakes and M. Ji, pp.301–324. ISBN: 978-90-272-0356-4.
- Jia, Y., S. Yu, and J. Ma, 2018, Intelligent interpolation by Monte Carlo machine learning: *Geophysics*, **83**, V83–V97.
- Jiang, Y. and J. Ning, 2019, Automatic identification of seismic body-wave phases and picking of their arrival times based on support vector machine: *Chinese Journal of Geophysics*, **62**, 361–373.
- Jin, Y., X. Wu, J. Chen, and Y. Huang, 2019, Using a physics-driven deep neural network to solve inverse problems for LWD azimuthal resistivity measurements: SPWLA 60th Annual Logging Symposium, doi: <https://doi.org/10.30632/T60ALS-2019.IIII>.
- Joachims, T., 1998, Text categorization with support vector machines: Learning with many relevant features: *European Conference on Machine Learning*, 137-142.
- Johnson, J., A. Alahi, and L. Fei-Fei, 2016, Perceptual losses for real-time style transfer and super-resolution: *European conference on computer vision*, 694-711. S
- Jolliffe, I., 2002, *Principal Component Analysis*: 2nd edition, New York, NY: Springer-Verlag.
- Jolliffe, I. and J. Cadima, 2016, Principal component analysis: a review and recent developments: *Philos Trans A Math Phys Eng Sci.*, 374(2065), doi: 10.1098/rsta.2015.0202.
- Jordan, J., 2018, Introduction to autoencoders: Data Science Blog, <https://www.jeremyjordan.me/autoencoders/>.

- Kaaresen, K. F., and T. Taxt, 1998, Multichannel blind deconvolution of seismic signals: *Geophysics*, **63**, 2093–2107.
- Kaczmarek, S. E., S. M. Fullmer, and F. Hasiuk, 2015, A universal classification scheme for the microcrystals that host limestone microporosity: *Journal of Sedimentary Research*, **85**, 1197–1212.
- Kang, B. and Y. Lee, 2020, High-resolution neural network for driver visual attention prediction: *Sensors*, **20**, 2030, doi:10.3390/s20072030.
- Kaplan, S. T., M. D. Sacchi, T. J. Ulrych, et al., 2009, Sparse coding for data-driven coherent and incoherent noise attenuation: 79th Annual International Meeting, SEG Technical Program, Expanded Abstracts, DOI: 10.1190/1.3255551.
- Karimpouli, S., N. Fathianpour, and J. Roohi, 2010, A new approach to improve neural networks' algorithm in permeability prediction of petroleum reservoirs using supervised committee machine neural network (SCMNN): *Journal of Petroleum Science and Engineering*, **73**, 227–232.
- Karlik, B. and A. Vehbi, 2011, Performance analysis of various activation functions in generalized MLP architectures of neural networks: *International Journal of Artificial Intelligence and Expert Systems (IJAE)*, **1**, 111–122. <https://www.cscjournals.org/library/manuscriptinfo.php?mc=IJAE-26>.
- Károly, Artúr, R. Fullér, and P. Galambos, 2018, Unsupervised clustering for deep learning: A tutorial survey: *Acta Polytechnica Hungarica*, **15**, 29–53.
- Karniadakis, G., I. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, 2021, Physics-informed machine learning: *Nature Reviews Physics*, **IF31.068**, DOI: 10.1038/s42254-021-00314-5.
- Karpathy, A., 2019, The unreasonable effectiveness of recurrent neural networks: Karpathy blog at <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>
- Karpatne, A., G. Atluri, J. H. Faghmous, M. Steinbach, A. Banerjee, A. Ganguly, S. Shekhar, N. Samatova, and V. Kumar, 2017, Theory-guided datascience: A new paradigm for scientific discovery from data: *IEEE Transactions on Knowledge and Data Engineering*, **29**, 2318–233.
- Karras, T., T. Aila, S. Laine, and J. Lehtinen, 2017, Progressive growing of GANS for improved quality, stability, and variation: arXiv: 1710.10196.
- Karras, T., S. Laine and T. Aila, 2018, A style-based generator architecture for generative adversarial networks: arXiv:1812.04948.
- Kassenaar, J. D., 1991, An application of principal components analysis to borehole geophysical data: 4th International MGLS/KEGS Symposium on Borehole Geophysics for Minerals, Geotechnical and Groundwater Applications, Proceedings.
- Kausar, B., S. Brahim, A. Abdullah, and I. Ahmad, 2011, A review of classification approaches using Support Vector Machine in intrusion detection: *Communications in Computer and Information Science, Conference: International Conference on Informatics Engineering and Information Science*, **253**, DOI: 10.1007/978-3-642-25462-8\_3.
- Kaur, H., N. Pham, and S. Fomel, 2020, Improving resolution of migrated images by approximating the inverse hessian using deep learning: *Geophysics*, doi.org/10.1190/geo2019-0315.1.
- Kawaguchi, K., L. P. Kaelbling, and Y. Bengio, 2017, Generalization in deep learning: arXiv:1710.05468.



- Kazemnejad, A., 2019, Transformer architecture: The positional encoding: [https://kazemnejad.com/blog/transformer\\_architecture\\_positional\\_encoding/](https://kazemnejad.com/blog/transformer_architecture_positional_encoding/)
- Keskar, N. and R. Socher, 2017, Improving generalization performance by switching from Adam to SGD: arXiv:1712.07628v1.
- Keskar, N., D. Mudigere, J. Nocedal, M. Smelyanskiy and P. Tang, 2017, On large-batch training for deep learning: generalization gap and sharp minima: CoRR, abs /1609.04836, <http://arxiv.org/abs/1609.04836>.
- Khandelwal, R., 2018, Bias and variance in machine learning: <https://medium.com/datadriveninvestor/bias-and-variance-in-machine-learning-51fdd38d1f86>.
- Kim, Y., A. Rush, L. Yu, A. Kuncoro, C. Dyer, and G. Melis, 2019, Unsupervised recurrent neural network grammars: arXiv:1904.03746 [cs.CL].
- Kingma, D. P., and J. Ba, 2014, Adam: A method for stochastic optimization: arXiv:1412.6980.
- Kingma, J. and M. Welling, 2014, Auto-encoding variational Bayes: arXiv:1312.6114v9.
- Kivinen, J., C. Williams, and N. Heess, 2014, Visual boundary prediction: A deep neural prediction network and quality dissection: Artificial Intelligence and Statistics, 512-521.
- Klir, G., 1994, On the alleged superiority of probabilistic representation of uncertainty: IEEE Trans. Fuzzy Syst., **2**, 27-31.
- Kocić, J., N. Jovivić and V. Drndarević, 2019, An end-to-end deep neural network for autonomous driving designed for embedded automotive platforms: Sensors, **19**, 2064, <https://doi.org/10.3390/s19092064>
- Kohonen, T., 1982, Self-organized formation of topologically correct feature maps: Biological Cybernetics, **43**, 59-69.
- Kohonen, T., 2005, Intro to SOM; SOM Toolbox: <http://www.cis.hut.fi/projects/somtoolbox/theory/somalgorithm.shtml>.
- Koster, K., Gabriels, P., Hartung, M., Verbeek, J., Deinum, G. and Staples, R., 2000, Time-lapse seismic surveys in the North Sea and their business impact: The Leading Edge, **19**, pp.286-293.
- Körtström, J., M. Uski, and T. Tiira, 2016, Automatic classification of seismic events within a regional seismograph network: Computers and Geosciences, **87**, 22-30, <https://doi.org/10.1016/j.cageo.2015.11.006>.
- Kotsi, M., J. Edgar, A. Malcolm and S. de Ridder, 2019, Combining reflection and transmission information in time-lapse velocity inversion: A new hybrid approach: Geophysics, **84**, R601-R611.
- Koutroumbas, K. and S. Theodoridis, 2008, Pattern Recognition (4th ed.): Burlington. ISBN 978-1-59749-272-0.
- Krebs, J.R., J. Anderson, D. Hinkley, R. Neelamani, S. Lee, A. Baumstein, and M.D. Lacasse, 2009, Fast full-wavefield seismic inversion using encoded sources: Geophysics, **74**, WCC177-WCC188.
- Kriegel, H., P. Kröger, E. Schubert, and A. Zimek, 2008, A general framework for increasing the robustness of PCA-based correlation clustering algorithms. Scientific and Statistical Database Management. Lecture Notes in Computer Science, 5069, 418-435. CiteSeerX 10.1.1.144.4864. doi:10.1007/978-3-540-69497-7\_27. ISBN 978-3-540-69476-2.

- Krizhevsky, A., and G. Hinton, 2009, Learning multiple layers of features from tiny images: M.S. thesis, University of Toronto.
- Krizhevsky, A., I. Sutskever, and G. E. Hinton, 2012, Imagenet classification with deep convolutional neural networks: *Advances in neural information processing systems*, 25, no. 2, 1097–1105.
- Krogh, A. and Vedelsby, J., 1994, Neural network ensembles, cross validation, and active learning: *NIPS*, 231-238.
- Krumbein, W. and L. Sloss, 1963, *Stratigraphy and Sedimentation*: W.H. Freeman and Co. San Francisco, USA.
- Kühl, H., and M. D. Sacchi, 2003, Least-squares wave-equation migration for AVP/AVA inversion: *Geophysics*, **68**, 262–273.
- Kullback, S., and R. A. Leibler, 1951, On information and sufficiency: *The Annals of Mathematical Statistics*, **22**, 79–86.
- Kuzma, H., 1999, A support vector machine for AVO interpretation: 69th Annual International Meeting, SEG Technical Program, Expanded Abstracts, DOI: 10.1190 /1.1817668.
- Kuzma, H. and J. Rector, 2004, Non-linear AVO inversion using support vector machines: 64th Annual International Meeting, SEG Technical Program, Expanded Abstracts, <https://doi.org/10.1190/1.1843305>.
- Kuzma, H., J. Rector, and D. Bremer, 2006, Seismic interpretation using Support Vector Machines implemented on Graphics Processing Units: presented at The IAMG 2006 Annual Conference on Quantitative Geology from Multiple Sources, Liege, Belgium, DOE Contract UCRL-CONF-222359 TRN: US200721 //872.
- Ladjal, S., A. Newson and C. Pham, 2019, A PCA-like autoencoder: [arXiv:1904.01277v1](https://arxiv.org/abs/1904.01277).
- Lailly, P., 1983, The seismic inverse problem as a sequence of before stack migrations: *Conference on inverse scattering: theory and application*, SIAM Philadelphia, PA, 206–220.
- Langan, R., I. Lerche and R. T. Cutler, 1985, Tracing of rays through heterogeneous media: An accurate and efficient procedure: *Geophysics*, **50**, 1456–1465.
- Langan, R., S. Lazaratos, J. Harris, A. Vassiliou, T. Jensen, and J. Fairborn, 1996, Imaging of a stratigraphically complex carbonate reservoir with crosswell seismic data: in *Graphically Complex Carbonate Reservoirs with Crosswell Seismic Data*: in Palaz, I., Ed., *Carbonate Seismology*: Soc. Expl. Geophys., 417–424.
- LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner, 1998, Gradient-based learning applied to document recognition: *Proceedings of the IEEE*, **86**, 2278-2324, doi: 10.1109 /5.726791.
- LeCun, Y., Y. Bengio, and G. Hinton, 2015, Deep learning: *Nature*, **521**, 436.
- Ledig, C., L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi, 2017, Photo-realistic single image super-resolution using a GANs: [arXiv:1609.04802](https://arxiv.org/abs/1609.04802).
- Lee, J., M. Simchowitz, M. Jordan, and B. Recht, 2016, Gradient descent only converges to minimizers: *Conference on learning theory*, 1246–1257.
- Li, J., Z. Feng, G. Schuster, 2017a, Wave equation dispersion inversion: *Geophys. J. Int.*, doi: 10.1093 /gji /ggw465.

- Li, J., G. Dutta, and G. T. Schuster, 2017b, Wave-equation Qs inversion of skeletonized surface waves: *Geophysical Journal International*, 209, 979–991, doi: 10.1093/gji/ggx051.GJINEA0956-540X.
- Li, F., and J. Gao, 2018, Sparse least-squares reverse time migration using 2-D undecimated wavelet transform: *International Geophysical Conference*, Beijing, China, 24–27 April 2018, Society of Exploration Geophysicists and Chinese Petroleum Society, 616–619.
- Li, H., Z. Xu, G. Taylor, C. Studer, and T. Goldstein, 2018a, Visualizing the loss landscape of neural nets: arXiv:1712.09913.
- Li, C., Y. Zhang, and C. C. Mosher, 2018b, An unsupervised learning method for residual seismic signal recovery: 88th Annual International Meeting, SEG Technical Program, Expanded Abstracts, 1996–2000.
- Li, J., S. Hanafy, and G.T. Schuster, 2018c, Wave equation dispersion inversion of guided P waves in a waveguide of arbitrary geometry: *J. Geophys. Research*, <https://doi.org/10.1029/2018JB016127>.
- Li, J., S. Hanafy, Z. Liu, and G. T. Schuster, 2019a, Wave equation dispersion inversion of Love waves: *Geophysics*, **84**, no. 5, 1–45.
- Li, J., Y. Chen, and G.T. Schuster, 2019b, Separation of multi-mode surface waves by supervised machine learning methods: *Geophysical Prospecting*, *Geophysical Prospecting*, <https://doi.org/10.1111/1365-2478.12927>.
- Li, S., X. Yang, H. Liu, Y. Cai, and Z. Peng, 2020, Seismic data denoising based on sparse and low-rank regularization: *Energies*, **13**, no. 2, 372.
- Li, J., X. Wu, and Z. Hu, 2020, Deep learning for simultaneous seismic image super-resolution and denoising: 90th Annual International Meeting, SEG Technical Program Expanded Abstracts, 1661–1665.
- Li, Y., Y. Ni, R. Croft, T. Di Matteo, S. Bird, and Y. Feng, 2021, AI-assisted superresolution cosmological simulations: *Proceedings of the National Academy of Sciences*, **118**, e2022038118 DOI: 10.1073/pnas.2022038118
- Lim, J.S., J. Kang, and J. Kim, 1998, Artificial intelligence approach for well-to-well log correlation: *SPE India Oil and Gas Conference and Exhibition*, SPE-39541–MS.
- Lin, F.-C., D. Li, R. W. Clayton, and D. Hollis, 2013, High-resolution 3d shallow crustal structure in Long Beach, California: Application of ambient noise tomography on a dense seismic array: *Geophysics*, **78**, 45–56.
- Lin, T., P. Goyal, R. Girshick, K. He, and P. Dollar, 2018, Focal loss for dense object detection: arXiv:1708.02002v2.
- Lin, T., Y. Wang, X. Liu, and X. Qiu, 2021, A survey of Transformers: arXiv:2106.04554.
- Linnainmaa, S., 1976, Taylor expansion of the accumulated rounding error: *BIT Numerical Mathematics*, **16**, 146–160.
- Liu, G.-C., X.-H. Chen, J.-Y. Li, J. Du, and J.-W. Song, 2011, Seismic noise attenuation using nonstationary polynomial fitting: *Applied Geophysics*, **8**, no. 1, 18–26.
- Liu, L., S. Z. Sun, and H. Wang, 2011, A new reservoir prediction method: PCA value-weighted attribute optimization: 81st Annual International Meeting, SEG Technical Program, Expanded Abstracts, 2024–2028.
- Liu, Q., and D. Wang, 2016, Stein variational gradient descent: A general purpose Bayesian inference algorithm: *Advances In Neural Information Processing Systems*, 2378–2386.

- Liu, C., C. Song, and Q., Lu, 2017a, Random noise de-noising and direct wave eliminating based on SVD method for ground penetrating radar signals: *Journal of Applied Geophysics*, **144**, 125–133.
- Liu, C., W. Wang, M. Wang, F. Lv, and M. Konan, 2017b, An efficient instance selection algorithm to reconstruct training set for support vector machine: *Knowledge-Based Systems*, **116**, 58–73.
- Liu, Z., and G. Schuster, 2018, Neural network least squares migration: First EAGE/SBGf Workshop on Least-Squares Migration, 1–5.
- Liu, Z., K. Lu, and X. Ge, 2018, Convolutional sparse coding for noise attenuation of seismic data: SEG 2018 Workshop: SEG Maximizing Asset Value Through Artificial Intelligence and Machine Learning, Beijing, China, 17–19 September 2018, 5–9.
- Liu, Z., and G. Schuster, 2019, Multilayer sparse LSM = deep neural network: 89th Annual International Meeting, SEG Technical Program, Expanded Abstracts, 2324–2327.
- Liu, Z., J. Cao, Y. Lu, S. Chen, and J. Liu, 2019a, A seismic facies classification method based on the convolutional neural network and the probabilistic framework for seismic attributes and spatial classification: *Interpretation*, **7**, SE225–SE236.
- Liu, Z., J. Li, S. M. Hanafy, and G. T. Schuster, 2019b, 3D wave-equation dispersion inversion of Rayleigh waves: *Geophysics*, **84**, no. 5, 1–127.
- Liu, X., X. Chen, J. Li, and Y. Chen, 2020a, Nonlocal weighted robust principal component analysis for seismic noise attenuation: *IEEE Transactions on Geoscience and Remote Sensing*, [https://doi: 10.1109/TGRS.2020.2996686](https://doi.org/10.1109/TGRS.2020.2996686).
- Liu, Z., Y. Chen, and G. Schuster, 2020b, Deep convolutional neural network and sparse least squares migration: *Geophysics*, [doi.org/10.1190/geo2019-0412.1](https://doi.org/10.1190/geo2019-0412.1).
- Liu, X., L. Zhou, X. Chen, and J. Li, 2020c, Lithofacies identification using support vector machine based on local deep multi-kernel learning: *Pet. Sci.* **17**, 954–966, <https://doi.org/10.1007/s12182-020-00474-6>.
- Lizarralde, D. and S. Swift, 1999, Smooth inversion of VSP traveltimes data: *Geophysics*, **64**, 659–661.
- Lloyd, S., 1982. Least squares quantization in PCM: *IEEE Trans. Inform. Theory* **28**, 129–137. Originally as an unpublished Bell laboratories Technical Note (1957).
- Loiseau, J., 2020, Binary cross-entropy and logistic regression: <https://towardsdatascience.com/binary-cross-entropy-and-logistic-regression-bf7098e75559>.
- Long, J., E. Shelhamer, and T. Darrell, 2015, Fully convolutional networks for semantic segmentation: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 3431–3440.
- Lonoy, A., 2006, Making sense of carbonate pore systems: *AAPG Bulletin*, **90**, 1381–1405.
- Lorenz, R., 2019, Bayesian approach to biosignature detection on ocean worlds: *Nat. Astron.*, **3**, 466–467.
- Löising, J. Ebbing and W. Szwillus, 2020, Geothermal heat flux in Antarctica: Assessing models and observations by Bayesian Inversion: *Front. Earth Sci.*, **21**, <https://doi.org/10.3389/feart.2020.00105>.
- Lu, G. and B. Fei, 2014, Medical hyperspectral imaging: a review: *J. of Biomedical Optics*, **19**(1), 010901-1–010901-23, [doi.org/10.1117/1.JBO.19.1.010901](https://doi.org/10.1117/1.JBO.19.1.010901)

- Lu, K., J. Li, B. Guo, L. Fu, and G. Schuster, 2017, Tutorial for wave-equation inversion of skeletonized data: Interpretation, SO1–SO10.
- Lu, K., and S. Feng, 2018, Auto-windowed super-virtual interferometry via machine learning: A strategy of first-arrival traveltimes automatic picking for noisy seismic data: SEG 2018 Workshop: SEG Maximizing Asset Value Through Artificial Intelligence and Machine Learning, Beijing, China, 10–14.
- Lubo, D., K. Marfurt, and V. Jayaram, 2014, Statistical characterization and geological correlation of wells using automatic learning Gaussian Mixture Models: 84th Annual International Meeting, SEG Technical Program, Expanded Abstracts, 774–783.
- Lucia, F. J., 2007, Carbonate Reservoir Characterization; An Integrated Approach (Second Edition): Springer Berlin Heidelberg, 336 p.
- Luo, Y., 1991, Calculation of wavepaths for band-limited seismic waves: 61st Annual International Meeting, SEG Technical Program, Expanded Abstracts, 1509–1512.
- Luo, Y., and G. T. Schuster, 1991a, Wave equation inversion of skeletonized geophysical data: Geophysical Journal International, 105, 289–294.
- Luo, Y. and G. T. Schuster, 1991b, Wave equation traveltimes inversion: Geophysics, **56**, 645–653.
- Luo, Z., 2017, Different tasks in computer vision: <https://luozm.github.io/cv-tasks>.
- Luthi, S. M., 2001, Nuclear spectroscopy logging, geological well logs: their use in reservoir modelling: Springer Berlin, Heidelberg. 183–215.
- Ma, Y. and M. Zhai, 2018, Random noise suppression algorithm for seismic signals based on Principal Component Analysis. Wireless Pers. Commun., **102**, 653–665, doi.org/10.1007/s11277-017-5081-7.
- Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C., 2011, Learning word vectors for sentiment analysis: In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1 (pp. 142–150). Association for Computational Linguistics.
- Maas, A., A. Hannun and A. Ng, 2013, Rectifier nonlinearities improve neural network acoustic models: Proc. ICML 30 (1), 3, 2990. <https://www.semanticscholar.org/paper/Rectifier-Nonlinearities-Improve-Neural-Network-Maas/367f2c63a6f6a10b3b64b8729d601e69337ee3cc>.
- Ma, Y. and Y. Luo, 2018, Automatic first-arrival picking with Reinforcement Learning: SEG Global Meeting Abstracts, 493–497. <https://doi.org/10.1190/IGC2018-121>
- MacQueen, J., 1967. Some methods for classification and analysis of multivariate observations: In *Fifth Berkeley Symposium on Mathematics. Statistics and Probability*, University of California Press, 281–297.
- Makhzani, A., J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, 2016, Adversarial autoencoders: arXiv:1511.05644v2.
- Malinverno, A., and Leaney, W. S., 2000, A Monte Carlo method to quantify uncertainty in the inversion of zero-offset VSP data: 70th Annual International Meeting, SEG Technical Program, Expanded Abstracts, 2393–2396.
- Malinverno, A. and V. Briggs, 2004, Expanded uncertainty quantification in inverse problems: Hierarchical Bayes and empirical Bayes: Geophysics, **69**, 1005–1016.

- Mandelli, S., F. Borra, V. Lipari, P. Bestagini, A. Sarti and S. Tubaro, 2018, Seismic data interpolation through convolutional autoencoder: 88th Annual International Meeting, SEG Technical Program, Expanded Abstracts, doi.org/10.1190/segam2018-2995428.1.
- Mandelli, S., V. Lipari, P. Bestagini, and S. Tubaro, 2019, Interpolation and denoising of seismic data using convolutional neural networks: arXiv preprint arXiv:1901.07927.
- Mao, X., Q. Li, H. Xie, R. Lau, Z. Wang, and S. Smolley, 2017, Least squares generative adversarial networks: In IEEE International Conference on Computer Vision.  
Mao, X., J.K. Chow, P.S. Tan, K.F. Liu, J. Wu, Z. Su, Y.H. Cheong, G.L. Ooi, C.C. Pang, and Y.H. Wang, 2021, Domain randomization-enhanced deep learning models for bird detection: Sci Rep., **11**:639, doi: 10.1038/s41598-020-80101-x. PMID: 33436851; PMCID: PMC7803967.
- Margossian, C., 2018, A review of automatic differentiation and its efficient implementation: arXiv:1811.05031.
- McCulloch, W. S. and W. A. Pitts, 1943, Logical calculus of the ideas immanent in nervous activity: Bull. Math. Biophysics, 115–133.
- Menke, W., 1984, Geophysical Data Analysis: Discrete Inverse Theory: Academic Press Inc.
- Mescheder, L., S. Nowozin, and A. Geiger, 2017, Adversarial variational Bayes: Unifying variational autoencoders and generative adversarial networks: arXiv:1701.04722.
- Miller, M., 2020, Unsupervised learning: clustering algorithms: towardsdatascience.com, [https://towardsdatascience.com/unsupervised-learning-clustering-algorithms-5b290967f746#:~:text=One of the more common, means clustering and agglomerative clustering.](https://towardsdatascience.com/unsupervised-learning-clustering-algorithms-5b290967f746#:~:text=One%20of%20the%20more%20common,means%20clustering%20and%20agglomerative%20clustering.)
- Mirzaei, A., 2019, Adversarial autoencoders on MNIST dataset Python Keras implementation: <https://medium.com/a-mirzaei69/adversarial-autoencoders-on-mnist-dataset-python-keras-implementation-5eeafd52ab21>.
- Mishra, R., 2020, Transposed convolution demystified: <https://towardsdatascience.com/transposed-convolution-demystified-84ca81b4baba>.
- Mnih, V., N. Heess, A. Graves, and K. Kavukcuoglu, 2014, Recurrent models of visual attention: arxiv.org/abs/1406.6247.
- Moline, G., J. Bahr, P. Drzewjecki, and L. Sheperd, 1992, Identification and characterization of pressure seals through the use of wireline logs: A multivariate statistical approach: The Log Analyst, **34**, 362–372.
- Morin, R.H., 2006, Negative correlation between porosity and hydraulic conductivity in sand-and-gravel aquifers at Cape Cod, Massachusetts, USA: Journal of Hydrology, **316**, 43–52.
- Mosegaard, K., and Tarantola, A., 1995, Monte Carlo sampling of solutions to inverse problems: Journal of Geophysical Research: **100**, 12431–12447.
- Mousavi, S., W. Ellsworth, W. Zhu, L. Chuang, and G. Beroza, 2020, Earthquake transformer-an attentive deep-learning model for simultaneous earthquake detection and phase picking: Nature Communications, **11**, Article number 3952, <https://www.nature.com/articles/s41467-020-17591-w#citeas>.
- Mountrakis, G., J. Im, and C. Ogole, 2011, Support vector machines in remote sensing: A review: Journal of Photogrammetry and Remote Sensing, **66**, Issue 3, pp. 247–259, doi.org/10.1016/j.isprsjprs.2010.11.001.
- Moya-Albor, E., H. Ponce and J. Brieva, 2017, An edge detection method using a fuzzy ensemble approach: Acta Polytechnica Hungarica, 14(3):20.

- Mukerji, T., P. Avseth, G. Mavko, I. Takahashi, and E. F. Gonzalez, 2001, Statistical rock physics: Combining rock physics, information theory, and geostatistics to reduce uncertainty in seismic reservoir characterization: *The Leading Edge*, 20, 313-319, doi: 10.1190/1.1438938.
- Nasien, D., H. Haron, A. Azmi, N. Aini and Y. Siti, 2014, Support Vector Machine (SVM) in handwritten character recognition using Freeman Chain Code (FCC). *Advanced Science Letters*. 20. 10.1166/asl.2014.5657.
- Navarro, J., P. Silva, D. Pan, and K. Hester, 2020, Real-time seismic attributes computation with conditional GANs: 90th Annual International Meeting, SEG Technical Program Expanded Abstracts, 1611–1615.
- Nemeth, T., C. Wu, and G. T. Schuster, 1999, Least-squares migration of incomplete reflection data: *Geophysics*, **64**, 208–221.
- Ng, A., 2013, <https://web.stanford.edu/class/cs294a/sparseAutoencoder.pdf>.
- Ng, A. and M. Jordan, 2013, On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes: *NIPS 2001: Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, 841–848.
- Ni, Y., Y. Li, P. Lachance, R. Croft, T. Di Matteo, S. Bird, and Y. Feng, 2021, AI-assisted super-resolution cosmological simulations II: Halo substructures, velocities and higher order statistics: *arXiv:2105.01016*.
- Niculescu, B. and G. Andrei, 2016, Principal Component Analysis as a tool for enhanced well log interpretation: *Revue Roumaine de Géophysique, Romanian Geophysical Journal*, **60**, 49–61.
- Nitish, N., C. G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, 2014, Dropout: A Simple Way to Prevent Neural Networks from overfitting: *Journal of Machine Learning Research*, **15**, 1929–1958.
- Nocedal, J. and S. Wright, 1999, *Numerical Optimization*: Springer Verlag Inc.
- Nwankpa, C., W. Ijomah, A. Gachagan, and S. Marshall, 2018, Activation functions: Comparison of trends in practice and research for deep learning: *arXiv:1811.03378v1*.
- O'Hagan, A. and J. Forster, 2004, *Advanced Theory of Statistics, Volume 2B: Bayesian Inference*: Oxford University Press, NY, NY. ISBN 978-0-340-80752-1.
- Oropeza, V., and M. Sacchi, 2011, Simultaneous seismic data denoising and reconstruction via multichannel singular spectrum analysis: *Geophysics*, 76, no. 3, V25–V32, <https://doi.org/10.1190/1.3552706>.
- Odena, A., V. Dumoulin, and C. Olah, 2016, Deconvolution and checkerboard artifacts: *Distill*, [doi.org/10.23915/distill.00003](https://doi.org/10.23915/distill.00003).
- Oh, K., and K. Jung, 2004, GPU implementation of neural networks: *Pattern Recognition*, **37**, 1311–1314.
- Oja, E., 1982, Simplified neuron model as a principal component analyzer: *Journal of Mathematical Biology*, **15**(3), 267-273.
- Oja E. and J. Karhunen, 1985, On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix: *J. Math. Anal. Appl.*, **106**, -69–84.
- Olshausen, B. and D. Field, 1996, Emergence of simple-cell receptive field properties by learning a sparse code for natural images: *Letters to Nature*, **381**, 607-609.

- Oppenheim, A., A. Willsky and S. Hamid Nawab, 1996, Signals and Systems (2nd Edition), Prentice-Hall, New Jersey.
- Oring, A., Z. Yakhini and Y. Hel-Or, 2020, Faithful autoencoder interpolation by shaping the latent space: arXiv:2008.01487.
- Paige, C. C. and M. A. Saunders, 1982, LSQR: an algorithm for sparse linear equations and sparse least squares: ACM Trans. Mathematical Software, 8, 43-71.
- Papayan, V., Y. Romano, and M. Elad, 2016, Convolutional neural networks analyzed via convolutional sparse coding: arXiv:1607.08194.
- Papayan, V., Y. Romano, and M. Elad, 2017a, Convolutional neural networks analyzed via convolutional sparse coding: The Journal of Machine Learning Research, 18, no. 1, 2887–2938.
- Papayan, V., Y. Romano, J. Sulam, and M. Elad, 2017b, Convolutional dictionary learning via local processing: Proceedings of the IEEE International Conference on Computer Vision, 5296–5304.
- Park, C. B., R. D. Miller, and J. Xia, 1998, Imaging dispersion curves of surface waves on multi-channel record: 68th Annual International Meeting, SEG Technical Program, Expanded Abstracts, 1377–1380.
- Park, M. J. and M. D. Sacchi, 2020, Automatic velocity analysis using convolutional neural network and transfer learning: Geophysics, 85, V33-V43.
- Parker, R., 1994, Geophysical Inverse Theory: Princeton University Press.
- Pascanu, R., C. Gulcehre, K. Cho, and Y. Bengio, 2014, How to construct deep recurrent neural networks: arXiv:1312.6026v2.
- Patel, A. B., T. Nguyen, R. Baraniuk, 2016, A probabilistic framework for deep learning: 2016, arXiv:1612.01936P.
- Patel, A. and S. Chatterjee, 2016, Computer vision-based limestone rock-type classification using probabilistic neural network: Geoscience Frontiers, 7, 53–60.
- Pathak, D., P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros, 2016, Context encoders: Feature learning by inpainting: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2536-2544, <https://doi.org/10.1109/CVPR.2016.278>.
- Paul, S., 2020, Reparamterization trick in variational autoencoders: <https://towardsdatascience.com/reparameterization-trick-126062cfd3c3>
- Paulsson, B. and M. Kelley, 2021, Deployment of optical sensor arrays to characterize mature oil fields and understanding subsurface CO2 movement to enhance oil recovery: AAPG CCUS Conference, March 23, 2021.
- Pearson, K., 1901, On lines and planes of closest fit to systems of points in space: Philosophical Magazine, 2 (11), 559–572.
- Pei, S., Y. Richard, D. Xu, S. Xiang and G. Meng, 2021, Alleviating mode collapse in GAN via diversity penalty module: arXiv:2108.02353.
- Peizhuang, W., 1983, Pattern recognition with fuzzy objective function algorithms: SIAM Review, 25(3):442.
- Perez, D. O., D. R. Velis, and M. D. Sacchi, 2013, Estimating sparse-spike attributes from avo data using a fast iterative shrinkage-thresholding algorithm and least squares: 83rd Annual International Meeting, SEG Technical Program, Expanded Abstracts, 3062–3067.



- Perez, C., 2013, Regularization of neural networks using DropConnect: ICML and JMLR, W and CP., jmlr.org.
- Perez, L. and J. Wang, 2017, The effectiveness of data augmentation in image classification using deep learning: arXiv:1712.04621.
- Perkins, H., M. Xu, J. Zhu and B. Zhang, 2015, Fast parallel SVM using data augmentation: arXiv: 1512.07716.
- Peter, S., E. Kirschbaum, M. Both, L. Campbell, B. Harvey, C. Heins, D. Durstewitz, F. Diego, and F. A. Hamprecht, 2017, Sparse convolutional coding for neuronal assembly detection: Advances in Neural Information Processing Systems, 3675–3685.
- Peters, B., J. Granek, and E. Haber, 2019a, Multiresolution neural networks for tracking seismic horizons from few training images: Interpretation, **7**, SE201–SE213.
- Peters, B., E. Haber, and J. Granek, 2019b, Neural networks for geophysicists and their application to seismic data interpretation: The Leading Edge, **38**, 534–540.
- Pham, N. and E. Naeini, 2019, Missing well log prediction using deep recurrent neural networks: 81st EAGE Conference and Exhibition, Expanded Abstract, DOI: 10.3997/2214-4609.201901612.
- Pham, N., X. Wu, and E. Zabihi Naeini, 2020, Missing well log prediction using convolutional long short-term memory network: Geophysics, **85**, 1–55.
- Pirayonesi, S. and T. El-Diraby, 2020, Role of data analytics in infrastructure asset management: Overcoming data size and quality problems: Journal of Transportation Engineering, Part B: Pavements. **146**, 04020022, doi:10.1061/JPEODX.0000175.
- Platt, J., 1998, Sequential minimal optimization: A fast algorithm for training Support Vector Machines: Technical Report MSR-TR-98-14.
- Plessix, R., 2007, A Helmholtz iterative solver for 3D seismic-imaging problems: Geophysics, **72**, SM185–SM194.
- Press, W., S. Teukolsky, W. Vetterling, and B. Flannery, 2007, Numerical Recipes: The Art of Scientific Computing: Cambridge University Press Inc.
- Prechelt, L. and G. Orr, 2012, Early stopping but when?: In neural networks: Tricks of the trade, Lecture Notes in Computer Science by G. Montavon and Klaus-Robert Müller (eds.). Springer Berlin Heidelberg. pp. 53–67.
- Qi, J., G. Machado, and K. Marfurt, 2017, A workflow to skeletonize faults and stratigraphic features: Geophysics, **82**, no. 4, O57–O70.
- Qi, J., B. Lyu, X. Wu, and K. Marfurt, 2020, Comparing convolutional neural networking and image processing seismic fault detection methods: 90th Annual International Meeting, SEG Technical Program Expanded Abstracts, 1111–1115.
- Qian, F., M. Yin, X. Y. Liu, Y. J. Wang, C. Lu, and G. M. Hu, 2018, Unsupervised seismic facies analysis via deep convolutional autoencoders: Geophysics, **83**, A39–A43.
- Qiao, T., Y. Yang, G. Turkiyah, and G.T. Schuster, 2022, Efficient physics-informed Bayesian inversion of seismic data: SEG Research Workshop: Data Analytics and Machine Learning for Exploration and Production, pp. 1-4.
- Quan, L. and Y. Kim, 2019, Iterative application of Autoencoders for video inpainting and fingerprint denoising: in Escalera, S., S. Ayache, J. Wan, M. Madadi, U. Guclu, X. and Baro, (eds) Inpainting and Denoising Challenges. The Springer Series on Challenges in Machine Learning. Springer, Cham.

- Radford, A., L. Metz, and S. Chintala, 2015, Unsupervised representation learning with deep convolutional generative adversarial networks: arXiv:1511.06434.
- Radhakrishnan, A., K. Yang, M. Belkin, and C. Uhler, 2019, Memorization in overparameterized autoencoders: arXiv:1810.10333.
- Rahman, A. S. and A. Rahman, 2020, Application of principal component analysis and cluster analysis in regional flood frequency analysis: a case study in New South Wales, Australia: *Water*, **12**, <https://doi.org/10.3390/w12030781>
- Ramachandran, P., B. Zoph, Q. V. Le, 2017, Searching for activation functions: *arXiv:1710.05941v2v2*.
- Ramdani, A., 2022, High-fidelity outcrop-based reservoir and seismic model of the Hanifa reservoir: King Abdullah University of Science and Technology, PhD Dissertation.
- Ranjan, C., 2019a, Build the right autoencoder—Tune and optimize using PCA principles. Part I: <https://towardsdatascience.com/build-the-right-autoencoder-tune-and-optimize-using-pca-principles-part-i-1f01f821999b>.
- Ranjan, C., 2019b, Build the right autoencoder—Tune and optimize using PCA principles. Part II: <https://towardsdatascience.com/build-the-right-autoencoder-tune-and-optimize-using-pca-principles-part-ii-24b9cca69bd6>.
- Ranjan, C., 2019c, Understanding dropout with the simplified math behind it: <https://towardsdatascience.com/simplified-math-behind-dropout-in-deep-learning-6d50f3f47275>.
- Ray, A., S. Kaplan, J. Washbourne, and U. Albertin, 2017, Low frequency full waveform seismic inversion within a tree based Bayesian framework: *Geophysical Journal International*, **212**, 522–542, doi: 10.1093/gji/ggx428.GJINEA0956–540X.
- Redmon, J., S. Divvala, R. Girshick, and A. Farhadi, 2015, You Only Look Once: Unified, real-time object detection: arXiv, 1506.02640.
- Redmon, J. and A. Farhadi, 2016, YOLO9000: Better, faster, stronger: arXiv:1612.08242v1.
- Reed, S., Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, 2016, Generative adversarial text to image synthesis: arXiv:1605.05396.
- .
- Ren, R., K. He, R. Girshick, and J. Sun, 2016, Faster R-CNN: Towards real-time object detection with region proposal networks: arXiv:1506.01497.
- Reynolds, D. S. Li, and A. Jain, 2009, Gaussian mixture models: *Encyclopedia of Biometrics*, Springer US, Boston, MA, pp. 659-663, doi:10.1007/978-0-387-73003-5\_196.
- E. Ricci and R. Perfetti, 2007, Retinal blood vessel segmentation using line operators and support vector classification: in *IEEE Transactions on Medical Imaging*, **26**, no. 10, 1357–1365, doi: 10.1109/TMI.2007.898551.
- Richardson, A., 2018, Seismic full-waveform inversion using deep learning tools and techniques: arXiv preprint arXiv:1801.07232.
- Rickett, J., 2003, Illumination-based normalization for wave-equation depth migration: *Geophysics*, **68**, 1371–1379.
- Rifai, S., X. Glorot, Y. Bengio, and P. Vincent. 2011, Adding noise to the input of a model trained with a regularized objective: arXiv:1104.3250.
- Rifkin, R. and A. Klautau, 2004, In defense of one-vs-all classification: *J. of Machine Learning*, **5**, 101–141.

- Ripley, B. D., 1996, Pattern Recognition and Neural Networks: Cambridge University Press.
- Alexander Rives, A., J. Meier, T. Sercu, S. Goyal, Z. Lin, J. Liu, D. Guo, M. Ott, C. Zitnick, J. Ma, and R. Fergus, 2021, Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences: Proceedings of the National Academy of Sciences 118, **15**, <https://doi.org/10.1073/pnas.2016239118>.
- Rolon, L., S. D. Mohaghegh, S. Ameri, R. Gaskari, and B. McDaniel, 2009, Using artificial neural networks to generate synthetic well logs: Journal of Natural Gas Science and Engineering, **1**, 118–133.
- Roman, V., 2020, Most popular convolutional neural networks architectures: Towards Data Science, <https://towardsdatascience.com/convolutional-neural-networks-most-common-architectures-6a2b5d22479d>.
- Ronneberger, O., P. Fischer, and T. Brox, 2015, U-Net: Convolutional networks for biomedical image segmentation: arXiv.org, arXiv:1505.04597.
- Rosenblatt, F., 1958, The Perceptron: A probabilistic model for information storage and organization in the brain: Cornell Aeronautical Laboratory, Psychological Review, v65, No. 6, 386–408. doi:10.1037/h0042519.
- Rosenblatt, F., 1962, Principles of Neurodynamics: Washington, DC:Spartan Books.
- Ross, C. P., and D. M. Cole, 2017, A comparison of popular neural network facies-classification schemes: The Leading Edge, **36**, 340–349.
- Ross, Z., M. Meier and E. Hauksson, 2018, P wave arrival picking and first-motion polarity determination with deep learning: Journal of Geophysical Research, Solid Earth, **123**, 5120–5129.
- Roth, H. R., L. Lu, J. Liu, J. Yao, A. Seff, K. Cherry, L. Kim, and R. M. Summers, 2016, Improving computer- aided detection using convolutional neural networks and random view aggregation: IEEE transactions on Medical Imaging, **35**, 1170–1181.
- Roy, A., M. M. C. De Matos, and K. J. Marfurt, 2011, Application of 3D clustering analysis for deep marine seismic facies classification-An example from deep-water northern Gulf of Mexico: Gulf Coast Section SEPM 31st Annual Bob. F. Perkins Research Conference, 410–439.
- Roy, A., B. L. Dowell, and K. J. Marfurt, 2013, Characterizing a Mississippian Tripolitic Chert reservoir using 3D unsupervised seismic facies analysis and well logs: An example from Osage County, Oklahoma: Interpretation, 1, no. 2, SB109–SB124, doi: 10.1190/INT-2013-0023.1.
- Roy, A., A. Vaswani, A. Neelakantan, and N. Parmar, 2018, Theory and experiments on vector quantized autoencoders: arXiv:1805.11063.
- Rudner, S., 2017, An overview of gradient descent optimization algorithms: arXiv: 1609.04747.
- Russel, B., 2019, Machine learning and geophysical inversion– A numerical study: The Leading Edge, **38**, 498–576, doi: <https://doi.org/10.1190/tle38070512.1>.
- Saggar, M., and E. L. Nebrija, 2003, Estimation of missing logs by regularized neural networks: AAPG bulletin, **87**, 1377–1389.
- Sacchi, M. D., and T. J. Ulrych, 1995, High-resolution velocity gathers and offset space reconstruction: Geophysics, **60**, 1169–1177.
- Salehi, M. M., M. Rahmati, M. Karimnezhad, and P. Omidvar, 2017, Estimation of the non records logs from existing logs using artificial neural networks: Egyptian Journal of Petroleum, **26**, 957–968.

- Salimans, T., I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, 2016, Improved techniques for training GANs: arXiv:1606.03498.
- Sambridge, M., P. Rickwood, N. Rawlinson, and S. Sommacal, 2007, Automatic differentiation in geophysical inverse problems: *Geophysical Journal International*, **170**, 1–8.
- Samuel, A., 1959, Some studies in Machine Learning using the game of checkers: *IBM Journal*, **3**, 211–229.
- Samuel, A., 1967, Some studies in Machine Learning using the game of checkers. II-Recent progress: in *IBM Journal of Research and Development*, **11**, 601–617, doi: 10.1147/rd.116.0601.
- Sanger, T., 1993, Two iterative algorithms for computing the singular value decomposition from input/output samples: in *Proc. Advances in Neural Information Processing Systems 6*, J. D. Cowan, G. Tesauro, and J. Alspector, eds., 144–151.
- Saygin, E., P. Cummins, A. Cipta, R. Hawkins, R. Pandhu, J. Murjaya, M. Irsyam, S. Widiyantoro, and B. Kennett, 2015, Imaging architecture of the Jakarta basin, Indonesia with transdimensional inversion of seismic noise: *Geophysical Journal International*, **204**, 918–931, doi: 10.1093/gji/ggv466.GJINEA0956–540X.
- Scales, J., 1987, Tomographic inversion via the conjugate gradient method: *Geophysics*, **52**, 179–185.
- Scales, J., P. Docherty and A. Gersztenkorn, 1990, Regularization of nonlinear inverse problems: imaging the near-surface weathering layer: *Inverse Problems*, **6**, 115–131.
- Scales, J. and L. Tenorio, 2001, Prior information and uncertainty in inverse problems: *Geophysics*, **66**, 389–397.
- Scheidt, C., L. Li, J. Caers, 2018, *Quantifying Uncertainty in Subsurface Systems*: AGU Geophysical Monograph Series, ISBN:9781119325888.
- Schölkopf, B., 1998, Nonlinear component analysis as a kernel eigenvalue problem: *Neural Computation*, **10**, 1299–1319.
- Schölkopf, B., R. Williamson, A. Smola, J. Shawe-Taylor and J. Platt, 1999, Support vector method for novelty detection: *Proceedings of the 12th International Conference on Neural Information Processing Systems*, 582–588.
- Schölkopf, B., J. Platt, J. Shawe-Taylor, A. Smola and R. Williamson, 2001, Estimating the support of a high-dimensional distribution: *Neural Computation*, **13**, 1443–1471.
- Schroot, B., and R. Scüttenhelm, 2003, Expressions of shallow gas in the Netherlands North Sea: *Netherlands Journal of Geosciences–Geologie en Mijnbouw*, **82**, 91–105.
- Schuster, G. T., 1988, An analytic generalized inverse for common depth-point and vertical-seismic profile traveltime equations: *Geophysics*, **53**, 314–325.
- Schuster, G.T., 1989, Analytic generalized inverse for transmission+reflection tomography: *Geophysics*, **54**, 1046–1049.
- Schuster, G. T., 1993, Least-squares cross-well migration: 63rd Annual International Meeting, SEG Technical Program, Expanded Abstracts, 110–113.
- Schuster, G. T. and J. Hu, 2000, Green’s functions for migration: continuous recording geometry: *Geophysics*, **65**, 167–175.
- Schuster, 2017, *Seismic Inversion*: SEG, Tulsa, Oklahoma.

- Schuster, G., 2018, Machine learning and wave equation inversion of skeletonized data: 80th EAGE Conference and Exhibition, Expanded Abstract, 10.3997//2214-4609.201801882.
- Schwaller, P., T. Laino, T. Gaudin, P. Bolgar, C. Hunter, C. Bekas, and A. Lee, 2019, Molecular Transformer: A model for uncertainty-calibrated chemical reaction prediction: ACS Central Science, **9**, 1572-1583, <https://doi.org/10.1021/acscentsci.9b00576>.
- Schwarz, G., 1978, Estimating the dimension of a model: Annals of Statistics, **6**, 461–464, doi: 10.1214/aos/1176344136.
- Selesnick, I., 2018, Sparse signal restoration:  
Link: [eeweb.poly.edu/iselesni/lecture\\_notes/sparse\\_signal\\_restoration.pdf](http://eeweb.poly.edu/iselesni/lecture_notes/sparse_signal_restoration.pdf).
- Sen, M. and Stoffa, P., 1995, Global Optimization Methods in Geophysical Inversion: Elsevier Press.
- Sen, M. and R. Biswas, 2017, Transdimensional seismic inversion using the reversible jump Hamiltonian Monte Carlo algorithm: Geophysics, **82**, R119–R134, doi: 10.1190/geo2016-0010.1.GPYSA70016–8033
- Serrano, A., F. Heide, D. Gutierrez, G. Wetzstein, and B. Masia, 2016, Convolutional sparse coding for high dynamic range imaging: Computer Graphics Forum, Wiley Online Library, 153–163.
- Sebtosheikh, M., R. Motafakkerfard, M. Riahi, S. Moradi and N. Sabety, 2015 Support vector machine method, a new technique for lithology prediction in an Iranian heterogeneous carbonate reservoir using petrophysical well logs: Carbonates and Evaporites, **30**, 59–68.
- Sebtosheikh, M. and A. Salehi, 2015, Lithology prediction by support vector classifiers using inverted seismic attributes data and petrophysical logs as a new approach and investigation of training data set size effect on its performance in a heterogeneous carbonate reservoir: J. Petrol. Sci. Eng., **134**, 143–149.
- Setyorini, D., P. Novianti, and U. Rafflesia, 2017, K-Means cluster analysis in earthquake epicenter clustering: International Journal of Advances in Intelligent Informatics, **3**, 81-89.
- Sha, N., M. Yan, and Y. Lin, 2019, Efficient seismic denoising techniques using robust principal component analysis: 89th Annual International Meeting, SEG Technical Program, Expanded Abstracts, 2543–2547.
- Sharma, V., 2020, An AI game of thrones: <https://medium.com/tri-pi-media/fei-fei-li-geoffrey-hinton-and-the-ai-game-of-thrones-afcb76d54f5e>
- Shahriari, M., D. Pardo, A. Picon, A. Galdran, J. Del Ser, and C. Torres-Verdin, 2020, A deep learning approach to the inversion of borehole resistivity: Computational Geosciences, **24**, 971-994.
- Shamshad, A., M.A. Bawadi, W.M.A. Wan Hussin, T.A. Majid, S.A.M. Sanusi, 2005, First and second order Markov chain models for synthetic generation of wind speed time series: Energy, **30**, 693–708.
- Shcherbakov, O. and V. Batishcheva, 2014, Image inpainting based on stacked autoencoders: Journal of Physics: Conference Series, **536**, doi:10.1088/1742-6596/536/1/012020.
- Shi, Y., L. Huang, X. Dong, T. Liu, and J. Ning, 2018, Application of fully convolutional neural network on fault detection: AGU meeting, San Francisco.
- Shi, Y., X. Wu, and S. Fomel, 2019, Saltseg: Automatic 3D salt segmentation using a deep convolutional neural network: Interpretation, **7**, SE113–SE122.

- Shi, Y., X. Wu, and S. Fomel, 2020, Waveform embedding: automatic horizon picking with unsupervised deep learning: *Geophysics*, **85**, 1–48.
- Shi, Y., M. Ballesio, K. Johansen, D. Trentman, Y. Huang, M. McCabe, R. Bruhn, and G.T. Schuster, 2021a, Semi-universal geo-crack detection by Machine Learning: Second EAGE Workshop on Unmanned Aerial Vehicles.
- Shi, Y., X. Wu, and S. Fomel, 2021b, Interactively tracking seismic geobodies with a deep-learning flood-filling network: *Geophysics*, **86**, A1–A5.
- Shihab, K., 2005, Modeling groundwater quality with Bayesian techniques. *Intelligent Systems Design and Applications: Proceedings of the 5th International Conference on Intelligent Systems Design and Applications*, IEEE Computer Society, 73–78.
- Shima, Y., 2018, Image augmentation for object image classification based on combination of pre-trained CNN and SVM: 2nd International Conference on Machine Vision and Information Technology, *J. Phys.: Conf. Ser.*, 1004 012001.
- Shibuya, N., 2016, Up-sampling with transposed convolution: <https://medium.com/activating-robotic-minds/up-sampling-with-transposed-convolution-9ae4f2df52d0>.
- Shorten, C. and T.M. Khoshgoftaar, 2019, A survey on image data augmentation for deep learning: *Journal of Big Data*, **6**, pp.1-48.
- Siahkoohi, A., R. Kumar, and F. Herrmann, 2018, Seismic data reconstruction with generative adversarial networks: 80th EAGE Conference and Exhibition 2018, European Association of Geoscientists and Engineers, 1–5.
- Siahsar, M., S. Gholtashi, E. O. Torshizi, W. Chen, and Y. Chen, 2017, Simultaneous denoising and interpolation of 3-D seismic data via damped data-driven optimal singular value shrinkage: *IEEE Geosci. Remote Sens. Lett.*, **14**, 1086-1090.
- Silipo, R., K. Melcher and S. Schmid, 2019, Neural machine translation with sequence to sequence RNN: [www.dataversity.net /neural-machine-translation-with-sequence-to-sequence-rnn /#](http://www.dataversity.net/neural-machine-translation-with-sequence-to-sequence-rnn/#).
- Silva, R., L. Baroni, R. Ferreira, D. Civitarese, D. Szwarcman, E. Vital, 2019, Brazil Netherlands dataset: A new public dataset for machine learning in seismic interpretation: *arXiv:1904.00770*.
- Simard, P.Y., D. Steinkraus and J.C. Platt, 2003, Best practices for convolutional neural networks applied to visual document analysis: *Seventh International Conference on Document Analysis and Recognition Proceedings*, 958-963.
- Simonyan, K. and A. Zisserman, 2014, Very deep convolutional networks for large-scale image recognition: *arXiv:1409.1556*.
- Simonyan, K., A. Veldaldi and A. Zisserman, 2014, Deep inside convolutional networks: Visualising image classification models and saliency maps: *arXiv:1312.6034v2*.
- Singh, A., 2018, Fulfillment lies in the creating something: <https://towardsdatascience.com/fulfillment-lies-in-the-creating-something-d118db307405>.
- Singer, Z., 2019, Softmax and uncertainty: <https://towardsdatascience.com/softmax-and-uncertainty-c8450ea7e064>.
- Singh, S., I. Tsvankin, and E. Naeini, 2021, Facies prediction with Bayesian inference using supervised and semisupervised deep learning: 91st Annual International Meeting, SEG Technical Program, Expanded Abstracts, 1510-1514.

- Sisson, S., Y. Fan, and M. Beaumont, 2018, Handbook of Approximate Bayesian Computation: Chapman and Hall/CRC.
- Sitzmann, V., J. Martel, A. Bergman, D. Lindell and G. Wetzstein, 2019, Implicit neural representations with periodic activation functions: arXiv:2006.09661v1.
- Sloan, S. D., M. Ralston, R. H. Stevens, and J. T. Schwenk, 2015, An example of the effects of near-surface variability on shallow seismic reflection data: 85th Annual International Meeting, SEG Technical Program, Expanded Abstracts, 2348–2352.
- Socco, L. V., S. Foti, and D. Boiero, 2010, Surface-wave analysis for building near-surface velocity models established approaches and new perspectives: *Geophysics*, **75**, 83–102.
- Spitz, S., 1991, Seismic trace interpolation in the f-x domain: *Geophysics*, **56**, 785–794, <https://doi.org/10.1190/1.1443096>.
- Springenberg, J., 2015, Unsupervised and semi-supervised learning with categorical generative adversarial networks: arXiv preprint arXiv:1511.06390.
- Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, 2014, Dropout: A simple way to prevent neural networks from overfitting: *The Journal of Machine Learning Research*, **15**, 1929–1958.
- Srivastava, R., K. Greff, and J. Schmidhuber, 2015, Highway networks: arXiv:1505.00387.
- Srivastava, A., L. Valkov, C. Russell, M. Gutmann, and C. Sutton, 2017, VEEGAN: Reducing mode collapse in GANs using implicit variational learning: arXiv:1705.07761.
- Statnikov, A., C. Aliferis, D. Hardin and I. Guyon, 2011, A Gentle Introduction to support vector machines in biomedicine Volume 1: Theory and Methods: publ. by World Scientific, <https://doi.org/10.1142/7922>.
- Steinhaus, H., 1956, Sur la division des corp materiels en parties: *Bull. Acad. Polon. Sci. IV (C1.III)*, 801–804.
- Stefani, J., 1995, Turning-ray tomography: *Geophysics*, **60**, 1917–1929.
- Stepanov, A., V. Chernykh, A. Kononov, 2021, The Seismo-Performer: A novel machine learning approach for general and efficient seismic phase recognition from local earthquakes in real time: *Sensors*, **21**, <https://doi.org/10.3390/s21186290>.
- Stevens, J. L. and S. M. Day, 1986, Shear velocity logging in slow formations using the Stoneley wave: *Geophysics*, **51**, 137–147.
- Stewart, M., 2019, Comprehensive introduction to autoencoders: <https://towardsdatascience.com/generating-images-with-autoencoders-77fd3a8dd368>.
- Stolt, R. H., 2002, Seismic data mapping and reconstruction: *Geophysics*, **67**, 890–908, <https://doi.org/10.1190/1.1484532>.
- Sulam, J., V. Pappayan, Y. Romano, and M. Elad, 2018, Multilayer convolutional sparse modeling: Pursuit and dictionary learning: *IEEE Transactions on Signal Processing*, **66**, 4090–4104.
- Sun, Y. and G. T. Schuster, 1992, Hierarchic optimizations for smoothing and cooperative inversion: 62nd Annual International Meeting, SEG Technical Program, Expanded Abstracts, 745–748.
- Sun, J., Z. Niu, K. Innanen, J. Li, and D. Trad, 2020, A theory-guided deep-learning formulation and optimization of seismic waveform inversion: *Geophysics*, **85**, R87–R99.

- Sun, Y., M. Araya-Polo, and P. Williamson, 2021, Data characterization and transfer learning for DL-driven velocity model building: 91st Annual International Meeting, SEG Technical Program, Expanded Abstracts, 1475–1479.
- Sutskever, I., J. Martens, and G. Hinton, 2011, Generating text with recurrent neural networks: Proceedings of the 28th International Conference on Machine Learning, Bellevue, WA. ([proceedings.mlr.press/v32/graves14.pdf](http://proceedings.mlr.press/v32/graves14.pdf)).
- Sutskever, I., O. Vinyals, and Q. Le, 2014, Sequence to sequence learning with neural networks: [arxiv.org/abs/1409.3215](http://arxiv.org/abs/1409.3215).
- Sutton, R., 1986, Two problems with backpropagation and other steepest-descent learning procedures for networks: Proceedings of the 8th Annual Conference on the Cognitive Science Society, 823–831.
- Sutton, R. and A. Barto, 2018, Reinforcement Learning: An Introduction: MIT Press, Cambridge, MA, <http://incompleteideas.net/book/the-book-2nd.html>
- Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, 2014, Going deeper with convolutions: [arXiv:1409.4842](https://arxiv.org/abs/1409.4842).
- Takashimizu, Y. and M. Iiyoshi, 2016, New parameter of roundness R: Circularity corrected by aspect ratio: Progress in Earth and Planetary Science, **3**, <https://doi.org/10.1186/s40645-015-0078-x>.
- Taner, M. T., 2001, Seismic attributes: CSEG Recorder, **26**, 49–56.
- Tang, C. and C. Monteleoni, 2018, Demystifying overcomplete nonlinear auto-encoders: fast S-GD convergence towards sparse representation from random initialization: <https://openreview.net/forum?id=HyiRazbRb>.
- Tang, L., M. Zhang, and L. Wen, 2019, Support Vector Machine classification of seismic events in the Tianshan Orogenic Belt: Journal of Geophysical Research: Solid Earth Volume 125, <https://doi.org/10.1029/2019JB018132>.
- Tarantola, A. and Valette, B., 1982, Generalized nonlinear inverse problems solved using the least squares criterion: Review of Geophysics and Space Physics, **20**, 219–232.
- Tarantola, A., 1987, Inverse Problem Theory Methods for Data Fitting and Model Parameter Estimation: Elsevier Science Publication.
- Tax, D. and R. Duin, 1999, Support vector domain description: Pattern Recognition Letters, **20**, 1991–1999.
- Theiling, B. P., L. B. Railsback, S. M. Holland, and D. E. Crowe, 2007. Heterogeneity in geochemical expression of subaerial exposure in limestones, and its implications for sampling to detect exposure surfaces: Journal of Sedimentary Research **77**, 159–169.
- Thorndike, R., 1953, Who belongs in the family? Psychometrika, **18** (4), 267–276.
- Thrane, E. and C. Talbot, 2019, An introduction to Bayesian inference in gravitational-wave astronomy: Parameter estimation, model selection, and hierarchical models: Publications of the Astronomical Society of Australia, **36**, E010, [doi:10.1017/pasa.2019.2](https://doi.org/10.1017/pasa.2019.2).
- Tikhonov, A. and V. Arsenin, 1977. Solutions of ill-posed problems: W. H. Winston and Sons.
- Toosi, A., A. Bottino, B. Saboury, E. Siegel, and A. Rahmim, 2021, A brief history of AI: how to prevent another winter (a critical review): [arXiv:2109.01517v2](https://arxiv.org/abs/2109.01517v2).



- Torres, A. and J. Reveron, 2013, Lithofacies discrimination using support vector machines, rock physics and simultaneous seismic inversion in clastic reservoirs in the Orinoco Oil Belt, Venezuela: 83rd Annual International Meeting, SEG Technical Program, Expanded Abstracts, 2578–2582, <https://doi.org/10.1190/segam2013-0039.1>.
- Trunk, G. V., 1979, A Problem of Dimensionality: A Simple Example: IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-1 (3), 306–307.
- Tschannen, M., O. Bachem, M. Lucic, 2018, Recent advances in autoencoder-based representation learning: arXiv:1812.05069.
- Uijlings, J., K. van de Sande, T. Gevers, and A. Smeulders, 2013, Selective search for object recognition: Journal of Computer Vision: 104(2):154–171.
- Ulrych, T., Sacchi, M., and Woodbury, A., 2001, A Baye’s tour of inversion: A tutorial: Geophysics, 66, 55–69.
- Ultsch, A., 2007, Emergence in self-organizing feature maps: in Ritter and Haschke (eds.), Proceedings of the 6th International Workshop on Self-Organizing Maps (WSOM 2007). Bielefeld, Germany: Neuroinformatics Group. ISBN 978-3-00-022473-7.
- van de Sande, K., T. Gevers, and C. G. M. Snoek, 2010, Evaluating color descriptors for object and scene recognition: TPAMI, 32: 1582–1596.
- van de Sande, K., J. Uijlings, T. Gevers, and A. Smeulders, 2011, Segmentation as selective search for object recognition: IEEE Xplore, DOI: 10.1109/ICCV.2011.6126456
- Ulrych, T. J., S. L. M. Freire, and P. Siston, 1988, Eigenimages processing of seismic section: 58th Annual International Meeting, SEG Technical Program, Expanded Abstracts, 1261–1265.
- van Herwaarden, D., C. Boehm, M. Afanasiev, S. Thrastarson, L. Krischer, J. Trampert, and A. Fichtner, 2020, Accelerated full-waveform inversion using dynamic mini-batches: **221**, 1427–1438.
- Vahrenkamp V., J. Barata J., P.J. Van Laer, P. Swart, and S. Murray, 2014, Micro-rhombic calcite of a giant Barremian (Thamama B) reservoir onshore Abu Dhabi-clumped isotope analyses fix temperature, water composition and timing of burial diagenesis: Paper presented at the Abu Dhabi International Petroleum Exhibition and Conference, 10-13 November 2014, Abu Dhabi, UAE.
- Vanorio, T. and G. Mavko, 2011, Laboratory measurements of the acoustic and transport properties of carbonate rocks and their link with the amount of microcrystalline matrix: Geophysics, **76**, 105–115.
- Vapnik, V. N., 1995, The Nature of Statistical Learning Theory: New York, NY, USA: Springer-Verlag New York, Inc..
- Vapnik, V. N., 1998, Statistical Learning Theory: Wiley, New York, Inc..
- Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, and I. Polosukhin, 2017, Attention is all you need: arXiv:1706.03762
- Verma, V., A. Lamb, C. Beckham, A. Courville, I. Mitliagkis, and Y. Bengio, 2018, Manifold mixup: Encouraging meaningful on-manifold interpolation as a regularizer: arXiv:1806.05236.
- Vincent, P., H. Larochelle, Y. Bengio, and P. Manzagol, 2008, Extracting and composing robust features with denoising autoencoders: Proceeding ICML 2008 Proceedings of the 25th international conference on Machine learning 1096–1103, doi:10.1145/1390156.1390294.

- Vinyals, O., A. Toshev, S. Bengio, and D. Erhan, 2014, Show and tell: A neural image caption generator: arXiv:1411.4555.
- Virieux, J. and S. Operto, 2009, An overview of full-waveform inversion in exploration: Geophysics, **74**, WCC1–WCC26.
- Visser, G., P. Guo, and E.c Saygin, 2019, Bayesian transdimensional seismic full-waveform inversion with a dipping layer parameterization, Geophysics, **84**, R845–R858.
- Wager, S., S. Wang, and P. Liang, 2013, Dropout training as adaptive regularization: arXiv:1307.1493.
- Waldeland, A. U., A. C. Jensen, L. J. Gelius, and A. H. S. Solberg, 2018, Convolutional neural networks for automated seismic interpretation: The Leading Edge, **37**, 529–537.
- Wang, P., 2004, The limitation of Bayesianism: Artif. Intell., **158**, 97–106.
- Wang, J., and M. D. Sacchi, 2005, Sparse regularization for least-squares AVP migration: CSEG National Convention, Expanded Abstracts, 117–120.
- Wang, J., and M. D. Sacchi, 2007, High-resolution wave-equation amplitude-variation-with-ray-parameter (AVP) imaging with sparseness constraints: Geophysics, **72**, S11–S18.
- Wang, B., R.-S.Wu, Y. Geng, and X. Chen, 2014, Dreamlet-based interpolation using POCS method: Journal of Applied Geophysics, **109**, 256–265, <https://doi.org/10.1016/j.jappgeo.2014.08.008>.
- Wang, P., Gomes, A., Zhang, Z. and Wang, M., 2016, Least-squares RTM: Reality and possibilities for subsalt imaging: *86th Annual International Meeting, SEG Expanded Abstracts*, 4204–4209.
- 
- Wang, T., M. Liu, J. Zhu, A. Tao, J. Kautz, and B. Catanzaro, 2017, High-resolution image synthesis and semantic manipulation with conditional GANs: CoRR, abs /1711.11585.
- Wang, T., M. Liu, J. Zhu, A. Tao, J. Kautz, and B. Catanzaro, 2018a, High-Resolution image synthesis and semantic manipulation with conditional GANs: arXiv:1711.11585.
- Wang, Y., F. Perazzi, B. McWilliams, A. Sorkine-Hornung, O. Sorkine-Hornung, and C. Schroers, 2018b, A fully progressive approach to single-image super-resolution: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, 864–873.
- Wang, B., N. Zhang, W. Lu, and J. Wang, 2018c, Deep-learning-based seismic data interpolation: A preliminary result: Geophysics, **84**, V11–V20, DOI:10.1190 /geo2017-0495.1.
- Wang, B., N. Zhang, W. Lu, and J. Wang, 2019, Deep-learning-based seismic data interpolation: A preliminary result: Geophysics, **84**, V11–V20.
- Wang, Y., B. Wang, N. Tu, and J. Geng, 2020, Seismic trace interpolation for irregularly spatial sampled data using convolutional autoencoder: Geophysics, **85**, V119–V130.
- Wang, Z., J. Chen and S. C. H. Hoi, 2021, Deep learning for image super-resolution: A survey: IEEE Transactions on Pattern Analysis and Machine Intelligence, **43**, 3365–3387.
- Wang, Y., Y. Li, Y. Song, and X. Rong, 2020, The influence of the activation function in a convolution neural network model of facial expression recognition: Appl. Sci., 10, 1897, doi:10.3390/app10051897.
- Wang, K. and D. Yang, 2021, A universal inversion framework for arbitrary regularizations based on the alternating direction method of multipliers: 91st Annual International Meeting, SEG Technical Program Expanded Abstracts, segam2021-3583950.1.

- Li, W., G. Rümpker, H. Stöcker, M. Chakraborty, D. Fenner, J. Faber, K. Zhou, J. Steinheimer, N. Srivastava, 2021, MCA-Unet: Multi-class attention-aware U-net for seismic phase picking: EGU21, 23rd EGU General Assembly, id.EGU21-15841
- White, D., 1989, Two-dimensional seismic refraction tomography: *Geophy. J.*, **97**, 223–245.
- Wiener, N., 1948, *Cybernetics, or Control and Communication in the Animal and the Machine*. Cambridge: MIT Press.
- Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., and Recht, B., 2017, The marginal value of adaptive gradient methods in machine learning: arXiv:1705.08292.
- Wisdom, S., T. Powers, J. Pitton, and L. Atlas, 2016, Interpretable recurrent neural networks using sequential sparse recovery: arXiv:1611.07252.
- Witte, P., M. Yang, and F. Herrmann, 2017, Sparsity-promoting least-squares migration with the linearized inverse scattering imaging condition: 79th EAGE Conference and Exhibition 2017, European Association of Geoscientists and Engineers, 1–5.
- Wohlberg, B., 2016, Boundary handling for convolutional sparse representations: 2016 IEEE International Conference on Image Processing (ICIP), IEEE, 1833–1837.
- Wolf, T., L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush, 2020, Transformers: State-of-the-Art Natural Language Processing: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, 38–45, doi/10.18653/v1/2020.emnlp-demos.6
- Wolpert, R., 2004, A conversation with James O. Berger: *Statistical Science*, **19**, 205–218.
- Woodward, M., 1989, *Wave-equation Tomography*: PhD dissertation, Stanford University, 68 pp.
- Woodward, M., 1992, Wave equation tomography: *Geophysics*, **57**, 15–26.
- Wrona, T., I. Pan, R. Bell, R. Gawthorpe, H. Fossen, and S. Brune, 2021, 3D seismic interpretation with deep learning: A brief introduction: *The Leading Edge*, **40**, 524–532.
- Wu, G. and E. Chang, 2003, Class-boundary alignment for imbalanced dataset learning: ICML 2003 Workshop on Learning From Imbalanced Data Sets, Washington DC, 49–56.
- Wu, C., J. Ho and D. Lee, 2004, Travel-time prediction with support vector regression: *IEEE Transactions on Intelligent Transportation Systems*, **5**, 276–281, doi: 10.1109/TIT-S.2004.837813.
- Wu, S., H. Wai, L. Li and A. Scaglione, 2018, A review of distributed algorithms for Principal Component Analysis: Proceedings of the IEEE, **106**, 1321–1340, doi: 10.1109/JPROC.2018.2846568.
- Wu, X., L. Liang, Y. Shi, Z. Geng, and S. Fomel, 2019a, Multitask learning for local seismic image processing: fault detection, structure-oriented smoothing with edge-preserving, and seismic normal estimation by using a single convolutional neural network: *Geophysical Journal International*, **219**, 2097–2109.
- Wu, X., Y. Shi, S. Fomel, L. Liang, Q. Zhang, and A. Yusifov, 2019b, Faultnet3D: Predicting fault probabilities, strikes, and dips with a single convolutional neural network: *IEEE Transactions on Geoscience and Remote Sensing*, **57**, 9138–9155.
- Wu, X., S. Yan, J. Qi, and H. Zeng, 2020a, Deep learning for characterizing paleokarst features in 3D seismic images: 90th Annual International Meeting, SEG Technical Program Expanded Abstracts, 454–459.

- Wu, X., Z. Geng, Y. Shi, N. Pham, S. Fomel, and G. Caumon, 2020b, Building realistic structure models to train convolutional neural networks for seismic structural interpretation: *Geophysics*, **85**, WA27–WA39.
- Wurster, S., H. Shen, H. Guo, T. Peterka, M. Raj, and J. Xu, 2021, Deep hierarchical super-resolution for scientific data reduction and visualization: *arXiv:2107.00462v1*.
- Xia, J., R. D. Miller, and C. B. Park, 1999, Estimation of near-surface shear-wave velocity by inversion of rayleigh waves: *Geophysics*, **64**, 691–700.
- Xie, J., L. Xu, and E. Chen, 2012, Image denoising and inpainting with deep neural networks: *Proceedings of the 25th International Conference on Neural Information Processing Systems Volume 1*, Curran Associates Inc., 341–349.
- Xiong, W., X. Ji, Y. Ma, Y. Wang, N. Ben-Hassan and Y. Luo, 2018, Seismic fault detection with convolutional neural network: *Geophysics*, *doi.org/10.1190/geo2017-0666.1*.
- Xiong, J., P. Richtárik, and W. Heidrich, 2019, Stochastic convolutional sparse coding: *arXiv preprint arXiv:1909.00145*.
- Xu, K., J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. Zemel, and Y. Bengio, 2015, Show, attend and tell: Neural image caption generation with visual attention: *arXiv:1502.03044*.
- Xu, C., Q. Yang, and C. Torres-Verdin, 2016, Bayesian rock classification and petrophysical uncertainty characterization with fast well-log forward modeling in thin-bed reservoirs: *Interpretation*, **4**, SF19-SF29.
- Xiong, W., X. Ji, Y. Ma, Y. Wang, N. Al BinHassan, M. Ali, and Y. Luo, 2018, Seismic fault detection with convolutional neural network: *Geophysics*, **83**, no. 5, O97–O103.
- Xu, C., Q. Yang, and C. Torres-Verdin, 2016, Bayesian rock classification and petrophysical uncertainty characterization with fast well-log forward modeling in thin-bed reservoirs: *Interpretation*, **4**, SF19-SF29.
- Xu, Y., J. Li, and X. Chen, 2019, Physics informed neural networks for velocity inversion: 89th Annual International Meeting, SEG Technical Program, Expanded Abstracts, 2584–2588.
- Xu, S., 2020, Frequentist vs. Bayesian approaches to machine learning: in *Towards Data Science*, <https://towardsdatascience.com/frequentist-vs-bayesian-approaches-in-machine-learning-86ece21e820e>
- Yadov, S., 2019, Understanding Vector Quantized Variational Autoencoders (VQ-VAE): <http://blog.usejournal.com/understanding-vector-quantized-variational-autoencoders-vq-vae-323d710a888a>.
- Yang, Q., and C. Torres-Verdin, 2011, Efficient 2D Bayesian inversion of borehole resistivity measurements: 82nd Annual International Meeting, SEG Technical Program, Expanded Abstracts, 0427-0430.
- Yang, H. and B. Wu, 2017, Full waveform inversion based on mini-batch gradient descent optimization with geological constrain: SEG Global Meeting Abstracts, 403–406.
- Yang, F., and J. Ma, 2019, Deep-learning inversion: A next generation seismic velocity model building method: *Geophysics*, **84**, R583–R599.
- Yang, Y., 2021, Automatic microseismic event location using deep neural networks: King Abdullah University of Science and Technology MS Thesis.
- Yang, C., B. Xu, J. Khan, G. Uddin, D. Han, Z. Yang, and D. Lo, 2022, Aspect-based API review classification: How far can pre-trained Transformer model go? *ArXiv2201.11327*.

- Yarnell, C., J. Granton, and G. Tomlinson, 2020, Bayesian analysis in critical care medicine: *American Journal of Respiratory and Critical Care Medicine*, **201**, 396–398, doi: 10.1164 / rccm.201910-2019ED.
- Yau, C. and K. Campbell, 2019, Bayesian statistical learning for big data biology: *Biophysical Reviews*, **11**, 95–102, DOI:10.1007 /s12551-019-00499-1.
- Yeung, S., A. Kannan, Y. Dauphin, and L. Fei-Fei, 2018, Tackling over-pruning in variational autoencoders: *CoRR*, abs /1706.03643.
- Yilmaz, Ö., 2001, *Seismic Data Analysis*: SEG Press Book.
- Young, M., N. Rawlinson, and T. Bodin, 2013, Transdimensional inversion of ambient seismic noise for 3D shear velocity structure of the Tasmanian crust: *Geophysics*, **78**, no. 3, WB49–WB62, doi: 10.1190/geo2012–0356.1.GPYSA70016–8033.
- Yu, J., J. Hu, G. T. Schuster, and R. Estill, 2006, Prestack migration deconvolution: *Geophysics*, **71**, S53–S62.
- Yu, S., J. Ma, and W. Wang, 2019, Deep learning for denoising: *Geophysics*, **84**, no. 6, V333–V350.
- Yu, H., Y. Chen, S. M. Hanafy and G. T. Schuster, 2020, Skeletonized wave-equation refraction inversion with autoencoded waveforms: *IEEE Transactions on Geoscience and Remote Sensing*, DOI: 10.1109/TGRS.2020.3046093.
- Yuan, S., J. Liu, S. Wang, T. Wang, and P. Shi, 2018, Seismic waveform classification and first-break picking using convolution neural networks: *IEEE Geoscience and Remote Sensing Letters*, **15**, 272–276.
- Zeiler, M., 2012, ADADELTA: An adaptive learning rate method: *arXiv*:1212.5701.
- Zeiler, M., M. Ranzato, R. Monga, M. Mao, K. Yang, Q.V. Le, P. Nguyen, Senior, A., V. Vanhoucke, J. Dean, and G.E. Hinton, 2013, On Rectified Linear Units for speech processing: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, 3517–3521.
- Zeiler, M. and R. Fergus, 2013, Visualizing and understanding convolutional neural networks: *arXiv*:1311.2901.
- Zhan, G., W. Dai, M. Zhou, Y. Luo, and G. T. Schuster, 2014, Generalized diffraction-stack migration and filtering of coherent noise: *Geophysical Prospecting*, **62**, 427–442.
- Zhang, W., 1988, Shift-invariant pattern recognition neural network and its optical architecture: *Proceedings of Annual Conference of the Japan Society of Applied Physics*.
- Zhang, W., 1990, Parallel distributed processing model with local space-invariant interconnections and its optical architecture: *Applied Optics*, **29**, 4790–4797.
- Zhang, J. and N. Toksöz, 1998, Nonlinear refraction traveltime tomography: *Geophysics*, **63**, 1726–1737.
- Zhang, G., G. Kong, and J. Zheng, 2009, Seismic attribute optimization based on kernel principal component analysis: Presented at the CPS/SEG International Geophysical Conference and Exposition.
- Zhang, C., C. Frogner, M. Araya-Polo, and D. Hohl, 2014, Machine-learning based automated fault detection in seismic traces: 76th Conference and Exhibition, EAGE, Extended Abstract. doi: 10.3997/2214–4609.20141500.

- Zhang, Y., K. Sohn, R. Villegas, G. Pan, and H. Lee, 2015, Improving object detection with deep convolutional networks via Bayesian optimization and structured prediction: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 249-258.
- Zhang, L., F. Yang, Y. D. Zhang, and Y. J. Zhu, 2016, Road crack detection using deep convolutional neural network: Image Processing (ICIP), 2016 IEEE International Conference on Image Processing (ICIP), 3708-3712.
- Zhang, D., Y. Chen, and J. Meng, 2018, Synthetic well logs generation via recurrent neural networks: Petroleum Exploration and Development, **45**, 630-639.
- Zhang, H., W. Wang, X. Wang, W. Chen, Y. Zhou, C. Wang, and Z. Zhao, 2019, An implementation of the seismic resolution enhancing network based on GAN: 89th Annual International Meeting, SEG Technical Program, Expanded Abstracts, 2478-2482, DOI: 10.1190 /segam 2019-3216229.1.
- Zhang, T., B. Xu, F. Thung, S. Haryono, D. Lo, and L. Jiang, 2020, Sentiment analysis for software engineering: How far can pre-trained transformer models go?: IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, pp. 70-80.
- Zhang, X. and A. Curtis, 2021, Bayesian full-waveform inversion with realistic priors: Geophysics, 1-20.
- Zhao, T., V. Jayaram, and K. Marfurt KJ, 2014, Lithofacies classification in Barnett Shale using proximal support vector machines: 84th Annual International Meeting, SEG Technical Program, Expanded Abstracts, 1491-1495, doi.org/10.1190/segam2014-1210.1.
- Zhang, H., T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. Metaxas, 2016, StackGAN: Text to photo-realistic image synthesis with stacked generative adversarial networks: arXiv: 1612.03242.
- Zhang, H., W. Wang, X. Wang, W. Chen, Y. Zhou, C. Wang, and Z. Zhao, 2019, An implementation of the seismic resolution enhancing network based on GAN: 89th Annual International Meeting, SEG Technical Program, Expanded Abstracts, 2478-2482, doi: DOI: 10.1190 /segam2019-3216229.1.
- Zhao, T., V. Jayaram, A. Roy, and K. Marfurt KJ, 2015, A comparison of classification techniques for seismic facies recognition: Interpretation, 29-58, doi.org/10.1190/int-2015-0044.1.
- Zhao, T., J. Zhang, F. Li, and K. J. Marfurt, 2016, Characterizing a turbidite system in Canterbury Basin, New Zealand, using seismic attributes and distance-preserving self-organizing maps: Interpretation, **4**, SB79- SB89, doi: 10.1190/INT-2015-0094.1.
- Zheng, H., 2018, Performance comparison of SVM and ANN for handwritten character recognition: [https://medium.com/\(at\)kiaorahao/performance-comparison-of-svm-and-ann-for-handwritten-character-recognition-8043260c9202](https://medium.com/(at)kiaorahao/performance-comparison-of-svm-and-ann-for-handwritten-character-recognition-8043260c9202).
- Zheng, Y., Q. Zhang, A. Yusifov, and Y. Shi, 2019, Applications of supervised deep learning for seismic interpretation and inversion: The Leading Edge, **38**, 526-533.
- Zhu, X. and McMechan, G., 1989, Estimation of two-dimensional seismic compressional-wave velocity distribution by iterative tomographic imaging: Internat. J. Imag. Sys. Tech., **1**, 13-17.
- Zhou, S., 2018, What happens in sparse autoencoder: <https://medium.com/syoya/what-happens-in-sparse-autencoder-b9a5a69da5c6>.
- Zhu, L., E. Liu, and J. McClellan, 2015, Seismic data denoising through multiscale and sparsity-promoting dictionary learning: Geophysics, **80**, no. 6, WD45-WD57.

- Zhu, L., C. Zhang, Y. Wei, X. Zhou, Y. Huang, and C. Zhang, 2017, Inversion of the permeability of a tight gas reservoir with the combination of a deep Boltzmann kernel extreme learning machine and nuclear magnetic resonance logging transverse relaxation time spectrum data: Interpretation, **5**, T341–T350.
- Zisselman, E., J. Sulam, and M. Elad, 2019, A local block coordinate descent algorithm for the CSC model: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 8208–8217.

# Index

- =, 343
- Activation function, 3, 4, 34, 78, 87, 88, 91, 95, 97, 101, 108, 114, 115, 139, 140, 156, 162, 247, 248, 252, 258, 261, 290, 294, 340, 391, 419
- Adadelta, 35, 49, 58–62, 70
- Adagrad, 35, 49, 56–58, 60, 70, 121
- Adam, 49, 59–63, 70, 72, 121, 239, 294, 297, 320, 332, 407, 553
- AdaMax, 63
- Adversarial autoencoder, 389, 550, 564
- Adversarially constrained autoencoder interpolation (ACAI), 404
- Adversarially constrained autoencoder interpolation (ACAI), 395, 406
- AlexNet, 227, 247, 257, 258, 272–274, 276, 277, 308, 309, 314
- Alternating Direction Method of Multipliers (ADMM), 707
- Angle-domain common image gather (ADCIG), 141, 143
- Attention, 357, 367, 370, 371, 373–376, 378–380, 382–384
- Autoencoder (AE), 387–395, 400, 402, 406–412, 414–416, 729, 733
- Automatic differentiation (AD), 687, 747, 753, 756
- Batch normalization (BN), 240, 244, 252
- Bayes’ Rule, 591, 593
- Bayes’ Theorem, 591, 594, 595, 599, 602, 620, 624, 626, 630, 632, 634, 793, 794, 800
- Bayesian inversion (BI), 591, 612
- Bias, 34, 42, 79, 88, 91, 115, 126, 127, 139, 156, 172, 178, 233, 361, 364, 390, 794
- Bias-variance, 134
- Blind deconvolution (BD), 720
- Cell state, 348, 357, 361, 363–365
- Chain rule, 359, 789, 790
- Common midpoint gather (CMP), 770
- Common midpoint, 771
- Common midpoint gather (CMP), 771
- Common shot-point gather (CSG), 145, 737, 740, 742, 770–772, 777
- Conditional probability, 343, 595, 599, 601, 794
- Confusion matrix, 332
- Conjugate gradient (CG), 65–70, 72
- Contractive autoencoder, 398, 399
- Convolution, 207, 210, 212–214, 220
- Convolution neural network (CNN), 223, 224, 226–228, 233, 238, 240
- Convolution: 1D, 210, 214, 781
- Convolution: 2D, 212, 250, 357, 407, 425
- Convolution: 3D, 215, 230
- Convolutional neural network (CNN), 195, 257, 258, 261, 272–274, 276, 278, 283, 286, 294, 298, 302, 303, 306–308, 323, 326, 333, 389, 404, 545, 564, 686, 687, 703, 704, 707, 721
- Correlation, 207, 212, 214, 215, 220
- Cross-entropy, 104–107, 111, 113, 117, 124, 236, 273, 281, 310, 341, 563
- Cross-validation, 33, 41, 127, 134, 195, 197, 601
- Curse of dimensionality, 169, 573, 685
- Data augmentation, 301, 309, 310, 404, 561, 564
- DCGAN, 547, 548, 550, 553, 554, 557–559
- Deblurring, 714, 716
- Deconvolution, 283, 287, 289, 308, 406, 615, 720
- Deep convolutional GAN, 547
- Denoising autoencoder, 397, 398, 406
- Density-based spatial clustering (DBSCAN), 479, 487, 502–506, 508, 516, 522, 524–529, 532, 541
- Diffraction migration, 765
- Diffraction stack migration (DSM), 772
- Discriminator, 545–549, 551–553, 555, 557–559, 562
- Dropout, 397, 414, 419, 420
- Dual problem, 172, 174, 175, 186, 194, 197, 198, 200
- Encoder: Attention, 378, 379



- Encoder: Autoencoder, 286, 290, 310, 390–392, 394, 398, 399, 403, 404, 411, 415, 417, 566, 735
- Encoder: RNN, 345
- Encoder: Transformer, 377, 378, 382, 383
- False positive, 328
- Faster region-based CNN, 258, 263, 266, 298, 310
- Forget gate, 348, 361
- Full waveform inversion (FWI), 308, 407, 594, 683, 729, 730, 732, 733, 736, 738, 743, 755, 756
- Fully-connected neural network (FCNN), 137, 139
- Fuzzy C-means Clustering method (FCM), 493, 665, 672
- GAN, 545, 547–551, 553, 555, 557, 559, 561
- Gated recurrent units (GRUs), 337, 348, 356, 361
- Gauss-Newton, 26–28, 34, 35, 42
- Gaussian kernel, 178, 181, 195
- Generative adversarial network (GAN), 333
- Generative Adversarial Networks, 545
- Generator, 545, 548, 549, 551–557, 559, 561, 562, 564–566
- Gibbs sampling, 583–587, 845
- Gibbs sampling, 580, 587
- GIMP, 324, 333
- Gradient descent (GD), 28, 41, 42, 46, 52, 56, 63, 69–71, 86, 91, 119, 172, 233, 236, 246, 252, 267, 309, 320, 402, 403, 553, 555, 563, 601, 684, 686, 687, 723, 732
- Hessian, 26, 36, 38, 44, 54, 56, 58, 68–70, 131, 563, 704, 705, 709
- Hinge loss, 183, 184, 186
- Ill-conditioned equations, 15, 16, 23, 29, 41, 49, 69, 129, 249, 704
- Inconsistent equations, 15, 16, 29, 32, 41, 128, 799
- Interpolation, 388, 392, 394, 395, 404–406, 411, 414, 415
- Iterative-soft-threshold-algorithm, 722, 723
- Joint probability, 343, 605, 608, 638, 787, 789, 790, 792
- Karush-Kuhn-Tucker (KKT), 197
- Kernel method, 175, 196
- Kernel trick, 177
- KL Regularization, 421
- Kullback-Leibler (KL), 404, 421, 562, 594, 602, 628, 635–637
- Lagrangian, 174, 175, 177, 180, 186, 196–203, 426
- Laplacian GAN, 564
- Laplacian kernel, 178
- Law of Total Probability, 592, 631, 633, 790, 792, 794
- Least squares migration (LSM), 703–705, 714, 718, 721
- Linear autoencoder (LAE), 393
- Logistic regression, 147, 148, 150, 151
- Long short-term memory (LSTM), 337, 344, 348, 352
- Marginal probability, 624, 643, 789, 790, 792
- Markov Chain Monte Carlo (MCMC), 578, 583, 587, 603, 605, 613–615, 619
- Matched filter, 216, 217
- Metropolis sampling, 578–581, 584, 587
- migration, 770, 772
- Migration Green's function (MGF), 721, 723, 724
- Migration image, 137–140, 143, 147, 150, 181, 182, 197, 251, 253, 272, 273, 278, 283, 312, 446, 447, 450, 491, 615, 616, 686, 689, 703–706, 710, 712, 714, 716, 718, 723, 730, 770, 772, 775
- Minimax, 548–551
- Momentum, 49, 51, 52, 54, 56, 58–61, 70, 71, 121
- Multiclass classification, 112, 114, 132, 235
- Multilabel classification, 111, 118
- Nash equilibrium, 405, 551, 552, 563
- Nesterov (NAG), 59
- Neural network (NN), 84, 88, 91, 95, 100, 101, 105, 107, 112, 114, 115, 118, 121, 122, 127, 132, 135, 137, 139, 141, 143, 145–147, 165, 181, 193, 195, 196, 207, 233, 301, 388, 402, 555, 600, 635, 684, 685, 687, 735, 748, 750, 798
- Neural network least squares migration (NNLSM), 703, 705, 707–710, 712, 714, 720, 721
- Newtonian machine learning (NML), 729, 733, 735, 737, 742, 745
- Non-linear optimization, 485
- Overdetermined equations, 5, 15, 16, 35, 41, 67, 68, 610
- Overfitting, 30, 41, 118, 133, 189, 193, 195, 242, 243, 250, 395, 396, 400, 557, 558, 593, 626, 635, 695, 796, 797
- overfitting, 128, 177, 397, 399

- Physics-informed ML, 683, 686, 687
- Pooling, 224, 226, 233, 234, 253
- Poststack, 721, 722, 765, 774, 775
- Preconditioned, 51, 52, 55, 56, 58, 70
- Prestack migration, 151, 251, 446, 723, 770, 774
- Primal problem, 174, 175, 185, 186
- Primal solution, 194
- Rayleigh waves, 144, 145, 776, 777
- Rectified linear unit (ReLU), 97, 98, 108, 115, 226, 231, 247, 252, 273, 281, 390, 394, 407, 553, 706, 718, 722
- Recurrent neural network (RNN), 337, 338, 342, 351, 352, 354
- Recursive back-propagation, 93
- Recursive backpropagation, 751
- Recursive feed-forward, 91
- Reduced Lagrangian, 175, 196, 197, 199, 200
- Region-based CNN (R-CNN), 257, 258, 263, 264, 266
- Regularization, 15, 35, 41, 42, 99, 134, 177, 182, 184, 185, 192, 196, 242–244, 250, 255, 358, 387, 389, 393, 395, 397, 399, 404, 405, 411, 415, 418, 420, 423, 445, 540, 545, 552, 561, 563, 671, 689, 690, 704–706, 721, 731, 744, 799
- Regularized AE, 400, 401, 406
- Regularized least squares, 24, 41, 194, 419, 594, 602, 610, 612, 683
- Reinforcement Learning (RL), 7, 8, 10
- ResNet, 248, 249
- RMSprop, 49, 58, 61
- Rosenbrock function, 38, 43, 44, 51, 52, 54, 58, 70–72
- Self-organizing map (SOM), 508, 510–512, 514, 515, 533–540, 648, 665, 666
- Semantic segmentation, 254, 257, 270, 272, 273, 276, 290, 310, 323
- Sequential autoencoder+physics inversion, 687
- Simultaneous ML+Physics inversion, 687
- Skeletonized, 6, 138, 145, 146, 149, 150, 327, 390, 407, 410, 508, 555, 565, 685–687, 729–733, 742
- Skip connection, 248–250, 345, 348, 379, 415, 694
- Soft-margin regularization, 185, 187, 189
- Sparse AE, 405
- Sparse autoencoder, 399
- Sparse least squares migration, 704–707
- Statics, 770
- Steepest descent (SD), 26, 27, 34, 35, 37, 42, 48, 49, 52, 56, 59, 65, 67, 69, 71, 78, 86, 99, 108, 119, 127, 162, 163, 198, 343, 549, 550, 746, 753, 755
- Step length, 35, 36, 46, 51, 52, 63, 70, 104, 121, 124, 236, 239, 246, 294, 309
- Stochastic gradient descent (SGD), 61, 63, 70, 86, 118, 119, 122, 235, 236, 309, 405, 561
- Super-resolution (SR), 307, 308, 318, 565, 703, 714, 721
- Super-resolution GANs, 565
- Support vector machine (SVM), 137, 141, 143, 147, 150, 165–167, 169, 171, 172, 175–177, 179–181, 183–188, 192, 193, 195–197
- Surface waves, 767, 776, 777
- Tomography, 68, 407, 415, 581, 686, 693, 694, 700, 701, 732
- Transfer learning, 333, 335
- Transfer training, 332
- Transformer, 357, 367, 377–380, 383, 384
- Transpose, 233, 234, 283, 289, 290, 308, 392, 407
- U-Net, 254, 257, 258, 286, 289, 290, 293, 298, 323–326, 328, 332, 333
- Underfit, 32, 134, 135
- Vanilla GAN, 548, 562
- Vanilla recurrent neural network (VRNN), 338, 343–345, 348, 358
- Variational autoencoder (VAE), 333, 388, 389, 402, 404, 415, 516, 522, 524, 562
- Vertical seismic profile (VSP), 454, 591, 593, 602, 605, 609, 612, 613, 628, 637, 643, 683
- VGGNet, 248
- Workflow: Attention encoder and decoder, 378
- Workflow: Autoencoder (AE), 408
- Workflow: Bayesian prediction of rock types, 451, 457, 617, 624, 626
- Workflow: Cano traveltime picking, 495
- Workflow: Computation of weights in a NN, 91, 132
- Workflow: Computing posterior for point-source locations, 615
- Workflow: Conjugate gradient, 66
- Workflow: Convolutional neural network (CNN), 227
- Workflow: CSC for noise attenuation., 426
- Workflow: FWI, 684, 685
- Workflow: Metropolis MCMC, 579
- Workflow: Principal component analysis (PCA), 450

- Workflow: Region-based CNN (R-CNN), 264–266
- Workflow: Rock classification by a NN, 731
- Workflow: Seismic noise identification by a NN, 147
- Workflow: Seismic noise identification by a NN, 137
- Workflow: Simple linear iterative clustering (SLIC), 316, 322
- Workflow: skeletonized NN, 685
- Workflow: Superpixel-CNN, 302, 303
- Workflow: Untrained self-attention, 368
- Workflow: VAE, 402
- Workflow: VAE+clustering, 524
- Workflow: Vanilla recurrent neural network (VRN-N), 338
- YOLO, 263, 266, 267, 270, 302