

Machine Learning Note

Hao Zhang

haomoodzhang@gmail.com

December 25, 2015

Preface

This note was written when I was taking the following courses (and some books):

- Andrea Vedaldi, Andrew Zisserman. *VGG Convolutional Neural Networks Practical*. Oxford Visual Geometry Group
- Andrew Ng. *Machine Learning*. Coursera
- Andrew Ng. *cs229 Machine Learning*. Stanford
- Andrew Ng et al. *Deep Learning Tutorial*. Stanford
- Geoffrey Hinton. *Neural Networks for Machine Learning*. Coursera
- Hsuan-Tien Lin. *Machine Learning Foundations*. Coursera
- Hsuan-Tien Lin. *Machine Learning Techniques*. Coursera
- Ian Goodfellow, Aaron Courville and Yoshua Bengio. *Deep Learning*. Book in preparation for MIT Press. 2015
- Jianxin Wu. *CNN for Dummies*. Nanjing University

This note was intended to be my personal reference.

If you have any suggestions or advice, or you find this material helps you a little bit, please be free to contact with me.

Contents

I Mathematical Background	1
1 Linear Algebra	3
1.1 Matrix Multiplication	3
1.1.1 Vector-Vector Products	3
1.1.2 Matrix-Vector Products	4
1.1.3 Matrix-Matrix Products	5
1.2 Operations and Properties	6
1.2.1 Diagonal Matrix	6
1.2.2 Symmetric Matrix	7
1.2.3 Trace	7
1.2.4 Norms	8
1.2.5 Linear Independence and Rank	11
1.2.6 Orthogonal Matrices	12
1.2.7 Projection and Four Foundamental Subspaces	12
1.2.8 Determinant	15
1.2.9 Quadratic Forms and Positive Semidefinite Matrices .	17
1.2.10 Eigenvalues and Eigenvectors	18
1.2.11 Eigenvalues and Eigenvectors of Symmetric Matrices .	19
1.2.12 Linear Transformation	20
1.2.13 Singular Value Decomposition (SVD)	22
1.2.14 Nonsingular Matrix and Singular Matrix	25
1.3 Matrix Calculus	25
1.3.1 Gradient	25
1.3.2 Derivatives of Matrices	26
1.3.3 Hessian	27
1.3.4 Chain Rule of Matrix Calculus	27
1.3.5 Gradients and Hessians of Quadratic and Linear Func- tions	29
1.3.6 Gradients of the Determinant	30
1.3.7 Gradients of the Trace	30
1.3.8 Gradients of the Transpose	30
1.3.9 Derivative of the Inverse	30
1.3.10 Derivative of the Linear Functions	30

1.3.11	Second Order Taylor Expansion	31
1.4	Tensor and Vectorization	31
1.4.1	Tensor	31
1.4.2	Vectorization	31
2	Probability, Statistics and Information Theory	33
2.1	Why Probability	33
2.1.1	Why Need Probability Theory	33
2.1.2	Two Kinds of Probability	34
2.2	Elements of probability	34
2.2.1	Properties	34
2.2.2	Conditional Probability	35
2.2.3	Independence	35
2.3	Random Variables	35
2.3.1	Cumulative Distribution Functions (CDF)	36
2.3.2	Probability Mass Functions (PMF)	37
2.3.3	Probability Density Functions (PDF)	37
2.3.4	Transformations	38
2.3.5	Conditional Distributions	39
2.3.6	Chain Rule	39
2.3.7	Structured Probabilistic Models	40
2.3.8	Bayes's Rule	41
2.3.9	Independence	41
2.3.10	Expectation	42
2.3.11	Variance and Covariance	43
2.3.12	Moment Generating Function (MGF)	44
2.4	Information Theory	44
2.4.1	Intuition	44
2.4.2	Several Quantity Measures	45
2.5	Some Common Distributions	47
2.5.1	Discrete Distributions	47
2.5.2	Continuous Distributions	49
2.5.3	Mixtures of Distributions	52
2.5.4	Properties of Gaussian Distribution	52
2.6	Estimation	55
2.6.1	Estimators, Bias and Variance	55
2.6.2	Trading off Bias and Variance and the Mean Squared Error	59
2.6.3	Maximum Likelihood Estimation	60
2.6.4	Bayesian Statistics	61
2.6.5	The EM Algorithm	64
2.7	Probability Inequalities	66
2.8	Useful Properties of Common Functions	67
2.9	Gaussian Process	68

2.9.1	Probability Distributions Over Functions with Finite Domains	69
2.9.2	Probability Distributions Over Functions with Infinite Domains	69
2.9.3	The Squared Exponential Kernel	70
2.10	Markov Process	71
2.10.1	Markov Models	71
2.10.2	Hidden Markov Model	73
3	Numerical Computation	79
3.1	Underflow and Overflow	79
3.1.1	Underflow	79
3.1.2	Overflow	79
3.2	Poor Conditioning	80
3.3	Gradient-Based Optimization	81
3.3.1	Gradient-Based Method Intuition	81
3.3.2	Gradient Descent	82
3.3.3	Stochastic Gradient Descent	85
3.3.4	Mini-batch Gradient Descent	88
3.3.5	Map-reduce and Data Parallelism	89
3.3.6	Online Gradient Descent	89
3.3.7	Advanced Optimization	89
3.4	Hessian and Newton's Method	90
3.4.1	Second Derivative Test	90
3.4.2	Understand the Performance of GD	91
3.4.3	Newton's Method	91
3.5	Convex Optimization	94
3.5.1	Introduction	94
3.5.2	Convex Sets	95
3.5.3	Convex Functions	97
3.5.4	Convex Optimization Problems	101
3.5.5	Lagrange duality	105
II	Machine Learning Foundation	109
4	When Can Machines Learn	111
4.1	The Learning Problem	111
4.1.1	What is Machine Learning	111
4.1.2	Components of Machine Learning	112
4.1.3	Machine Learning and Other Fields	112
4.2	Learning to Answer Yes/ No	112
4.2.1	Perceptron Hypothesis Set	112
4.2.2	Perceptron Learning Alg. (PLA)	113

4.2.3	Guarantee of PLA	113
4.2.4	Non-Sep. Data	115
4.3	Types of Learning	116
4.3.1	Learning with Diff. Ouput Space \mathcal{Y}	116
4.3.2	Learning with Diff. Data Label $y^{(i)}$	117
4.3.3	Learning with Diff. Protocol $f \mapsto \{\vec{x}^{(i)}, y^{(i)}\}$	118
4.3.4	Learing with Diff. Input Space \mathcal{X}	118
4.4	Feasiblty of Learning	120
4.4.1	Learning is Impossible?	120
4.4.2	Prob. to the Rescue	120
4.4.3	Connection to Learning	121
4.4.4	Connection to Real Learning	123
5	Why Can Machines Learn	125
5.1	Training vs. Testing	125
5.1.1	Recap and Preview	125
5.1.2	Effective # of Lines	126
5.1.3	Effective # of Hypotheses	129
5.1.4	Break Point	132
5.2	Theory of Generalization	133
5.2.1	Restriction of Break Point	133
5.2.2	Bounding Fcn.: Basic Cases	133
5.2.3	Bounding Fcn.: Inductive Cases	135
5.2.4	A Pictorial Proof	139
5.3	The VC Dim.	140
5.3.1	Def. of VC Dim.	140
5.3.2	VC Dim. of Perceptrons	142
5.3.3	Physical Intuition of VC Dim.	144
5.3.4	Interpreting VC Dim.	145
5.4	Noise and Error	147
5.4.1	Noise and Probabilistic Target	147
5.4.2	Err. Measure	149
5.4.3	Alg. Err. Measure	150
5.4.4	Weighted Classification	150
6	How Can Machine Learn	153
6.1	Linear Regression	153
6.1.1	Linear Regression Problem	153
6.1.2	Linear Regression Algorithm	153
6.1.3	Probabilistic Interpretation	154
6.1.4	Generalization Issue	156
6.1.5	Locally Weighted Linear Regression	158
6.1.6	Bayesian Learning Regression	159
6.1.7	Gaussian Process Regression	160

6.1.8	Linear Regression for Binary Classification	161
6.2	Logistic Regression	162
6.2.1	Logistic Regression Problem	162
6.2.2	Logistic Regression Error	163
6.2.3	Gradient of Logistic Regression Error	164
6.2.4	Gradient Descent	166
6.3	Generalized Linear Models	167
6.3.1	the Exponential Family	167
6.3.2	Constructing GLMs	169
6.3.3	Stochastic Gradient Ascent on GLM	173
6.4	Generative Learning Algorithms	174
6.4.1	Discriminant and Generative Learning Algorithms . .	174
6.4.2	Gaussian Discriminant Analysis (GDA)	175
6.4.3	Naive Bayes	177
6.5	Linear Models for Classification	182
6.5.1	Linear Models for Binary Classification	182
6.5.2	Stochastic Gradient Descent	184
6.5.3	Multiclass via Logistic Regression	184
6.5.4	Multiclass via Binary Classification	186
6.6	Nonlinear Transformation	188
6.6.1	Quadratic Hypotheses	188
6.6.2	Nonlinear Transform	188
6.6.3	Price of Nonlinear Transform	188
6.6.4	Structured Hypothesis Sets	190
7	How Can Machine Learn Better	191
7.1	Hazard of Overfitting	191
7.1.1	What is Overfitting?	191
7.1.2	The Role of Noise and Data Size	191
7.1.3	Deterministic Noise	193
7.1.4	Dealing with Overfitting	194
7.2	Regularization	196
7.2.1	Regularized Hypothesis Set	196
7.2.2	Weight Decay Regularization	197
7.2.3	Gradient Descent with Regularization	199
7.2.4	Effect of Regularization	199
7.2.5	Regularization and VC Theory	201
7.2.6	General Regularizers	202
7.2.7	Regularization as Constrained Optimization	204
7.2.8	Regularization and Under-Constrained Problems .	204
7.3	Validation	204
7.3.1	Model Selection Problem	204
7.3.2	Validation	205
7.3.3	Leave-One-Out Cross Validation	205

7.3.4	<i>K</i> -Fold Cross Validation	206
7.3.5	Use Validation to Diagnosing Underfitting and Overfitting	207
7.3.6	Feature Selection	207
7.4	Noise Injection	208
7.4.1	Injecting Noise at the Input	209
7.4.2	Injecting Noise at the Weights	209
7.5	Decide What to Do Next	210
7.5.1	Do More Data Really works?	210
7.5.2	Optimization Algorithm Diagnostics	211
7.5.3	Error Analysis	213
7.5.4	Ceiling Analysis	213
7.5.5	Ablative Analysis	214
7.5.6	Getting Started on a Problem	215
7.6	Three Learning Principles	215
7.6.1	Occams Razor	215
7.6.2	Sampling Bias	216
7.6.3	Data Snooping	216
7.6.4	Power of Three	216
8	Embedding Numerous Features: Kernel Models	219
8.1	Three Major Techniques Surrounding feature transform	219
8.2	Linear Support Vector Machine	219
8.2.1	Large-Margin Separating Hyperplane	219
8.2.2	Standard Large-Margin Problem	221
8.2.3	SVM	223
8.2.4	Reasons behind Large-Margin Hyperplane	224
8.3	Dual SVM	225
8.3.1	Motivation of Dual SVM	225
8.3.2	Lagrange Dual SVM	225
8.3.3	Solving Dual SVM	227
8.3.4	Messages behind Dual SVM	228
8.4	Kernel Support Vector Machine	230
8.4.1	Kernel Trick	230
8.4.2	Polynomial Kernel	231
8.4.3	Gaussian Kernel	232
8.4.4	Comparison of Kernels	233
8.5	Soft-Margin Support Vector Machine	235
8.5.1	Motivation and Primal Problem	235
8.5.2	Dual Problem	237
8.5.3	SMO Algorithm	238
8.5.4	Messages behind Soft-Margin SVM	240
8.5.5	Model Selection	242
8.6	Kernel Logistic Regression	243

8.6.1	Soft-Margin SVM as Regularized Model	243
8.6.2	SVM vs. Logistic Regression	243
8.6.3	SVM for Soft Binary Classification	244
8.6.4	Kernel Logistic Regression	245
8.7	Support Vector Regression	247
8.7.1	Kernel Ridge Regression	247
8.7.2	Support Vector Regression Primal	248
8.7.3	Support Vector Regression Dual	251
8.7.4	Summary of Kernel Models	252
9	Combining Predictive Features: Aggregation Models	255
9.1	Blending and Bagging	255
9.1.1	Motivation of Aggregation	255
9.1.2	Uniform Blending	256
9.1.3	Linear and Any Blending	258
9.1.4	Bagging (Bootstrap Aggregation)	259
9.2	Adaptive Boosting	261
9.2.1	Motivation of Boosting	261
9.2.2	Diversity by Re-weighting	261
9.2.3	Adaptive Boosting Algorithm	263
9.2.4	Adaptive Boosting in Action	265
9.3	Decision Tree	265
9.3.1	Decision Tree Hypothesis	265
9.3.2	Decision Tree Algorithm	266
9.3.3	Decision Tree Heuristics in CART	268
9.3.4	Decision Tree in Action	269
9.4	Random Forest	270
9.4.1	Random Forest Algorithm	270
9.4.2	Out-Of-Bag Estimate	271
9.4.3	Feature Selection	273
9.4.4	Random Forest in Action	274
9.5	Gradient Boosted Decision Tree	275
9.5.1	Adaptive Boosted Decision Tree	275
9.5.2	Optimization View of AdaBoost	276
9.5.3	Gradient Boosting	280
9.5.4	Summary of Aggregation Models	282
10	Distilling Implicit Features: Extraction Models	285
10.1	<i>k</i> -means Clustering Algorithm	285
10.1.1	Clustering Problem	285
10.1.2	Partition Optimization	285
10.1.3	<i>k</i> -means Algorithm	286
10.1.4	How to Choose <i>K</i>	286
10.2	Mixtures of Gaussians	287

10.2.1 Mixtures of Gaussians Model	287
10.2.2 Parameter Evaluation	287
10.2.3 Partition Optimization	288
10.2.4 EM for Mixtures of Gaussians	288
10.2.5 Proof it As a Result of General EM	288
10.3 Factor Analysis	290
10.3.1 Single Gaussian	290
10.3.2 Restrictions on Σ	291
10.3.3 The Factor Analysis Model	293
10.3.4 EM for Factor Analysis	293
10.3.5 Application: Anomaly Detection	294
10.4 Principle Component Analysis	295
10.4.1 PCA Problem	295
10.4.2 Solving PCA	295
10.4.3 Number of components to retain	298
10.4.4 Implementing PCA Whitening	299
10.5 Spectral Clustering	299
10.5.1 Spectral Clustering Algorithm	299
10.5.2 Graph Construction	300
10.5.3 One-Dimensional Case	300
10.5.4 k -Dimensional Case	301
10.6 Non-Negative Matrix Factorization (NMF)	301
10.6.1 Non-Negative Matrix Factorization	301
10.6.2 Interpretation of NMF	301
10.6.3 Optimization in NMF	302
10.7 Independent Components Analysis	302
10.7.1 ICA Problem	302
10.7.2 ICA ambiguities	303
10.7.3 ICA Algorithm	304
10.8 Recommender Systems	306
10.8.1 Content-Based Recommendations	306
10.8.2 Collaborative Filtering	306
10.8.3 Low Rank Matrix Factorization	307
10.8.4 Implementational Detail	308
10.8.5 Summary of Extraction Models	308
III Neural Network and Deep Learning	309
11 Introduction	311
11.1 AI, Machine Learning and Deep Learning	311
11.1.1 Rule-Based Systems	311
11.1.2 Classic Machine Learning	312
11.1.3 Representation Learning	312

11.1.4 Deep Learning	313
11.2 Feature / Representation Learning	314
11.2.1 Linear Models	314
11.2.2 Kernel Methods	314
11.2.3 Manually Engineer the Representation or Features . .	315
11.2.4 Learn the Features / Representation	315
11.3 Historical Trends in Deep Learning	316
11.3.1 The Many Names and Changing Fortunes of Neural Networks	316
11.3.2 Increasing Dataset Sizes	316
11.3.3 Increasing Model Sizes	317
11.3.4 Increasing Accuracy, Application Complexity and Real-World Impact	317
12 Feedforward Deep Network	319
12.1 MLPs from 1980's	319
12.2 Parametrizing a Learned Predictor	323
12.2.1 Family of Functions	323
12.2.2 Piecewise Linear Hidden Units	326
12.2.3 Error Function and Conditional Log-Likelihood . . .	327
12.3 Flow Graphs and Back-Propagation	329
12.3.1 Neural Network Hypothesis	329
12.3.2 Neural Network Learning	329
12.3.3 Learning in a General Flow Graph	332
12.3.4 Implantation Notes	333
12.4 Back-Propagation Through Random Operations and Graphical Models	334
12.4.1 Back-Propagating Through Discrete Stochastic Operations	334
12.5 Radial Basis Function Network	336
12.5.1 RBF Network Hypothesis	336
12.5.2 RBF Network Learning	338
12.6 Universal Approximation Properties and Depth	339
13 Regularization of Deep Models	341
13.1 Ways to Reduce Overfitting	341
13.2 Weight Decay	341
13.3 Early Stopping as a Form of Regularization	342
13.3.1 Early Stopping	342
13.3.2 Early Stopping and the Use of Surrogate Error Function	343
13.3.3 Early Stopping Acts as a Regularizer	343
13.4 Parameter Tying and Parameter Sharing	344
13.5 Sparse Representations	344
13.6 Dropout	345

13.6.1	Bagging and Other Ensemble Methods	345
13.6.2	Dropout	345
13.6.3	Testing	345
13.6.4	Inverted Dropout	346
13.6.5	Dropout For Other Units	346
13.7	Multi-Task Learning	347
13.8	Adversarial Learning	347
14 Optimization for Training Deep Models		351
14.1	Challenges in Neural Network Optimization	351
14.1.1	Ill-Conditioning	351
14.1.2	Local Minima	352
14.1.3	Plateaus, Saddle Points and Other Flat Regions	352
14.1.4	Cliffs and Exploding Gradients	353
14.1.5	Inexact Gradients	353
14.1.6	Theoretical Limits of Optimization	354
14.2	Optimization Algorithm I: Baisc Algorithm	354
14.2.1	Momentum	354
14.2.2	Nesterov Momentum	356
14.3	Optimization Algorithm II: Adaptive Learning	356
14.3.1	AdaGrad	357
14.3.2	RMSprop	357
14.3.3	Adam	358
14.3.4	AdaDelta	358
14.3.5	Choosing the Right Optimization Algorithm	358
14.4	Optimization Algorithm III: Approximate Second-Order Meth- ods	359
14.4.1	Newton's Method	359
14.4.2	Conjugate Gradients	359
14.4.3	BFGS	362
14.4.4	Limited Memory BFGS (L-BFGS)	362
14.5	Optimization Algorithm IV: Natural Gradient Methods	363
14.6	Optimization Strategies and Meta-Algorithms	364
14.6.1	Coordinate Descent	364
14.6.2	Initialization Strategies	364
14.6.3	Greedy Supervised Pre-training	366
14.6.4	Designing Models to Aid Optimization	367
14.6.5	Continuation Methods and Curriculum Learning	367
15 Convolutional Neural Networks		371
15.1	The Convolution Operation	371
15.2	Motivation	372
15.2.1	Sparse Interactions	372
15.2.2	Parameter Sharing	373

15.2.3	Equivariant Representations	374
15.3	Pooling	375
15.4	Convolution and Pooling as an Infinitely Strong Prior	377
15.5	Variants of the Basic Convolutional Functions	378
15.5.1	Parallel Convolution	378
15.5.2	Tensor Convolution	378
15.5.3	Locally Connected Layers	380
15.5.4	Tiled Convolution	381
15.6	Back Propagation on Convolutional Layers	381
15.7	Structured Outputs	381
15.8	Data Types	382
15.9	Efficient Convolutional Algorithms	382
15.9.1	Fourier Transform	382
15.9.2	Separable Kernels	382
15.10	Random or Unsupervised Features	382
16	Practical Methodology	383
16.1	Default Baseline Models	383
16.2	Data Preprocessing	384
16.2.1	Mean Subtraction	384
16.2.2	Normalization	384
16.2.3	PCA and Whitening	385
16.3	Selecting Hyperparameters	387
16.3.1	Manual Hyperparameter Tuning	387
16.3.2	Automatic Hyperparameter Optimization algorithms	388
16.4	Debugging Strategies	390
16.4.1	Visualize the Model in Action	390
16.4.2	Visualize the Worst Mistakes	390
16.4.3	Compare Train and Test error	390
16.4.4	Fit a Tiny Dataset	391
16.4.5	Numerical Gradient	392
16.4.6	Monitor Histograms of Activations and Gradient	392
17	Applications	395
17.1	Large Scale Deep Learning	395
17.1.1	Fast CPU Implementations	395
17.1.2	GPU Implementations	395
17.1.3	Large Scale Distributed Implementations	396
17.1.4	Model Compression	396
17.1.5	Dynamic Structure	397
17.1.6	Specialized Hardware Implementations of Deep Networks	397
17.2	Computer Vision	397
17.2.1	Preprocessing	397

17.2.2 Contrast Normalization	398
IV Finale	401
18 Finale	403
18.1 Feature Exploitation Techniques	403
18.1.1 Exploiting Numerous Features via Kernel	403
18.1.2 Exploiting Predictive Features via Aggregation	403
18.1.3 Exploiting Hidden Features via Extraction	404
18.1.4 Exploiting Low-Dimensional Features via Compression	405
18.2 Error Optimization Techniques	405
18.2.1 Numerical Optimization via Gradient Descent	405
18.2.2 Indirect Optimization via Equivalent Solution	405
18.2.3 Complicated Optimization via Multiple Steps	405
18.3 Overfitting Elimination Techniques	406
18.3.1 Overfitting Elimination via Regularization	406
18.3.2 Overfitting Elimination via Validation	406
18.4 Machine Learning in Practice	407
18.4.1 NTU KDDCup 2010	407
18.4.2 NTU KDDCup 2011	408
18.4.3 NTU KDDCup 2012	408
18.4.4 NTU KDDCup 2013	408
18.4.5 ICDM 2006 Top 10 Data Mining Algorithms	408

Part I

Mathematical Backgroud

Chapter 1

Linear Algebra

Note: some of the material here (including some of the figures) is based on [?], [?]

1.1 Matrix Multiplication

1.1.1 Vector-Vector Products

Inner Product (Dot Product)

$$\vec{x}, \vec{y} \in \mathbb{R}^n, \vec{x}^T \vec{y} = [x_1 \ x_2 \ \cdots \ x_n] \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \sum_{k=1}^n x_k y_k = \|\vec{x}\|_2 \|\vec{y}\|_2 \cos(\vec{x}, \vec{y}) = \vec{y}^T \vec{x} \in \mathbb{R} \quad (1.1)$$

Outer Product

$$\vec{x} \in \mathbb{R}^m, \vec{y} \in \mathbb{R}^n, \vec{x} \vec{y}^T = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} [y_1 \ y_2 \ \cdots \ y_n] \quad (1.2)$$

$$= \begin{bmatrix} x_1 y_1 & x_1 y_2 & \cdots & x_1 y_n \\ x_2 y_1 & x_2 y_2 & \cdots & x_2 y_n \\ \vdots & \vdots & \ddots & \vdots \\ x_m y_1 & x_m y_2 & \cdots & x_m y_n \end{bmatrix} \in \mathbb{R}^{m \times n} \quad (1.3)$$

$$(\vec{x} \vec{y}^T)_{ij} = x_i y_j \quad (1.4)$$

1.1.2 Matrix-Vector Products

$$A \in \mathbb{R}^{m \times n}, \vec{x} \in \mathbb{R}^n, \vec{y} = A\vec{x} \in \mathbb{R}^m \quad (1.5)$$

Matrix-Column Vector Products

Write A in Columns

$$\vec{y} = A\vec{x} = [\vec{a}_1 \ \vec{a}_2 \ \cdots \ \vec{a}_n] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = x_1\vec{a}_1 + x_2\vec{a}_2 + \cdots + x_n\vec{a}_n \quad (1.6)$$

\vec{y} is a linear combination of the columns of A , where the coefficients of the linear combination are given by the entries of \vec{x}

Write A in Rows

$$\vec{y} = A\vec{x} = \begin{bmatrix} \vec{a}_1^T \\ \vec{a}_2^T \\ \vdots \\ \vec{a}_m^T \end{bmatrix} \vec{x} = \begin{bmatrix} \vec{a}_1^T \vec{x} \\ \vec{a}_2^T \vec{x} \\ \vdots \\ \vec{a}_m^T \vec{x} \end{bmatrix} \quad (1.7)$$

$$y_k = \vec{a}_k^T \vec{x} \quad (1.8)$$

Row Vector-Matrix Products

$$\vec{y}^T = \vec{x}^T A \quad (1.9)$$

Write A in Columns

$$\vec{y}^T = \vec{x}^T A = \vec{x}^T [\vec{a}_1 \ \vec{a}_2 \ \cdots \ \vec{a}_n] = [\vec{x}^T \vec{a}_1 \ \vec{x}^T \vec{a}_2 \ \cdots \ \vec{x}^T \vec{a}_n] \quad (1.10)$$

$$y_k^T = \vec{x}^T \vec{a}_k \quad (1.11)$$

Write A in Rows

$$\vec{y}^T = \vec{x}^T A = [x_1 \ x_2 \ \cdots \ x_n] \begin{bmatrix} \vec{a}_1^T \\ \vec{a}_2^T \\ \vdots \\ \vec{a}_m^T \end{bmatrix} = x_1\vec{a}_1^T + x_2\vec{a}_2^T + \cdots + x_n\vec{a}_n^T \quad (1.12)$$

\vec{y}^T is a linear combination of the rows of A , where the coefficients for the linear combination are given by the entries of \vec{x} .

1.1.3 Matrix-Matrix Products

$$A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{n \times p}, C = AB \in \mathbb{R}^{m \times p} \quad (1.13)$$

$$C_{ij} = \sum_{k=1}^n A_{ik} B_{kj} \quad (1.14)$$

View as Vector-Vector Inner Products

$$C = AB = \begin{bmatrix} \vec{a}_1^T \\ \vec{a}_2^T \\ \vdots \\ \vec{a}_m^T \end{bmatrix} [\vec{b}_1 \ \vec{b}_2 \ \cdots \ \vec{b}_p] = \begin{bmatrix} \vec{a}_1^T \vec{b}_1 & \vec{a}_1^T \vec{b}_2 & \cdots & \vec{a}_1^T \vec{b}_p \\ \vec{a}_2^T \vec{b}_1 & \vec{a}_2^T \vec{b}_2 & \cdots & \vec{a}_2^T \vec{b}_p \\ \vdots & \vdots & \ddots & \vdots \\ \vec{a}_m^T \vec{b}_1 & \vec{a}_m^T \vec{b}_2 & \cdots & \vec{a}_m^T \vec{b}_p \end{bmatrix} \quad (1.15)$$

$$C_{ij} = \vec{a}_i^T \vec{b}_j, \vec{a}_i \in \mathbb{R}^n, \vec{b}_j \in \mathbb{R}^n \quad (1.16)$$

View as Vector-Vector Outer Products

$$C = AB = [\vec{a}_1 \ \vec{a}_2 \ \cdots \ \vec{a}_n] \begin{bmatrix} \vec{b}_1^T \\ \vec{b}_2^T \\ \vdots \\ \vec{b}_n^T \end{bmatrix} = \sum_{k=1}^n \vec{a}_k \vec{b}_k^T, \vec{a}_k \in \mathbb{R}^m, \vec{b}_k \in \mathbb{R}^p \quad (1.17)$$

View as Matrix-Column Vector Products

$$C = AB = A[\vec{b}_1 \ \vec{b}_2 \ \cdots \ \vec{b}_p] = [A\vec{b}_1 \ A\vec{b}_2 \ \cdots \ A\vec{b}_p] = [\vec{c}_1 \ \vec{c}_2 \ \cdots \ \vec{c}_p] \quad (1.18)$$

$$\vec{c}_k = A\vec{b}_k, \vec{c}_k \in \mathbb{R}^m \quad (1.19)$$

View as Row Vector-Matrix Products

$$C = AB = \begin{bmatrix} \vec{a}_1^T \\ \vec{a}_2^T \\ \vdots \\ \vec{a}_m^T \end{bmatrix} B = \begin{bmatrix} \vec{a}_1^T B \\ \vec{a}_2^T B \\ \vdots \\ \vec{a}_m^T B \end{bmatrix} = \begin{bmatrix} \vec{c}_1^T \\ \vec{c}_2^T \\ \vdots \\ \vec{c}_m^T \end{bmatrix} \quad (1.20)$$

$$\vec{c}_k^T = \vec{a}_k^T B, \vec{c} \in \mathbb{R}^p \quad (1.21)$$

Ex. Show that matrix multiplication is associative

$$AB(C) = A(BC) \quad (1.22)$$

Proof. suppose

$$A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{n \times p}, C \in \mathbb{R}^{p \times q} \quad (1.23)$$

$$AB \in \mathbb{R}^{m \times p}, C \in \mathbb{R}^{p \times q}, (AB)C \in \mathbb{R}^{m \times q} \quad (1.24)$$

$$BC \in \mathbb{R}^{n \times q}, A \in \mathbb{R}^{m \times n}, A(BC) \in \mathbb{R}^{m \times q} \quad (1.25)$$

Thus, the dimensions of the resulting matrices agree.

To show that matrix multiplication is associative, it suffices to check that the (i, j) -th entry of $(AB)C$ is equal to the (i, j) -th entry of $A(BC)$.

$$((AB)C)_{ij} = \sum_{k=1}^p (AB)_{ik} C_{kj} \quad (1.26)$$

$$= \sum_{k=1}^p \left(\sum_{l=1}^n A_{il} B_{lk} \right) C_{kj} \quad (1.27)$$

$$= \sum_{k=1}^p \sum_{l=1}^n A_{il} B_{lk} C_{kj} \quad (1.28)$$

$$= \sum_{l=1}^n \sum_{k=1}^p A_{il} B_{lk} C_{kj} \quad (1.29)$$

$$= \sum_{l=1}^n A_{il} \left(\sum_{k=1}^p B_{lk} C_{kj} \right) \quad (1.30)$$

$$= \sum_{l=1}^n A_{il} (BC)_{lj} \quad (1.31)$$

$$= (A(BC))_{ij} \quad (1.32)$$

□

1.2 Operations and Properties

1.2.1 Diagonal Matrix

$$\text{diag}(\vec{x})\vec{y} = \vec{x} \cdot * \vec{y} \text{ for } \vec{x}, \vec{y} \in \mathbb{R}^n \quad (1.33)$$

$$\text{diag}(\vec{x})^{-1} = \text{diag}([x_1^{-1} \ x_2^{-1} \ \cdots \ x_n^{-1}]) \text{ for } \vec{x} \in \mathbb{R}^n, \vec{x} \neq 0 \quad (1.34)$$

1.2.2 Symmetric Matrix

symmetric:

$$A = A^T \in \mathbb{R}^{n \times n} \quad (1.35)$$

denote as

$$A \in \mathbb{S}^n \quad (1.36)$$

anti-symmetric:

$$A = -A^T \quad (1.37)$$

any mat. $A \in \mathbb{R}^{n \times n}$ can be repr. as a sum of a sym. mat. and a anty-sym. mat.

$$A = \frac{1}{2}(A + A^T) + \frac{1}{2}(A - A^T) \quad (1.38)$$

$$A^T = \frac{1}{2}(A^T + A) + \frac{1}{2}(A^T - A) \quad (1.39)$$

$$= \frac{1}{2}(A + A^T) - \frac{1}{2}(A - A^T) \quad (1.40)$$

1.2.3 Trace

$$\text{tr } A^T = \text{tr } A, \text{ for } A \in \mathbb{R}^{n \times n} \quad (1.41)$$

$$\text{tr } cA = c \text{tr } A, \text{ for } A \in \mathbb{R}^{n \times n}, c \in \mathbb{R} \quad (1.42)$$

$$\text{tr}(A + B) = \text{tr } A + \text{tr } B, \text{ for } A, B \in \mathbb{R}^{n \times n} \quad (1.43)$$

$$\text{tr } AB = \text{tr } BA, \text{ for } A, B \text{ st. } AB \in \mathbb{R}^{m \times m} \quad (1.44)$$

$$\text{tr } ABC = \text{tr } BCA = \text{tr } CAB, \text{ for } A, B, C \text{ st. } ABC \in \mathbb{R}^{m \times m} \quad (1.45)$$

$$\vec{x}^T \vec{y} = \text{tr } \vec{x}^T \vec{y} = \text{tr } \vec{x} \vec{y}^T = \text{tr } \vec{y} \vec{x}^T \text{ for } \vec{x}, \vec{y} \in \mathbb{R}^n \quad (1.46)$$

$$(1.47)$$

non-negative
definiteness
homogeneity
triangle inequality

Ex. Show that

$$\text{tr } AB = \text{tr } BA, \text{ for } A, B \text{ st. } AB \in \mathbb{R}^{m \times m} \quad (1.48)$$

Proof. suppose

$$A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{n \times m} \quad (1.49)$$

$$AB \in \mathbb{R}^{m \times m}, BA \in \mathbb{R}^{n \times n} \quad (1.50)$$

both are square mat.

$$\text{tr } AB = \sum_{i=1}^m (AB)_{ii} \quad (1.51)$$

$$= \sum_{i=1}^m \sum_{j=1}^n A_{ij} B_{ji} \quad (1.52)$$

$$= \sum_{j=1}^n \sum_{i=1}^m B_{ji} A_{ij} \quad (1.53)$$

$$= \sum_{j=1}^n (BA)_{jj} \quad (1.54)$$

$$= \text{tr } BA \quad (1.55)$$

□

1.2.4 Norms

a norm is any fcn. $\| \cdot \| : \mathbb{R}^n \mapsto \mathbb{R}$ that satisfies 4 properties:

- **non-negative** : $\forall \vec{x} \in \mathbb{R}^n, \|\vec{x}\| \geq 0$
- **definiteness** : $\|\vec{x}\| = 0 \iff \vec{x} = \vec{0}$
- **homogeneity** : $\forall \vec{x} \in \mathbb{R}^n, c \in \mathbb{R}, \|c\vec{x}\| = |c| \cdot \|\vec{x}\|$
- **triangle inequality** : $\forall \vec{x}, \vec{y} \in \mathbb{R}^n, \|\vec{x} + \vec{y}\| \leq \|\vec{x}\| + \|\vec{y}\|$

L_2 norm

$$\|\vec{x}\|_2 = \sqrt{\sum_{k=1}^n x_k^2} = \sqrt{\vec{x}^T \vec{x}} \quad (1.56)$$

- in many contexts, the squared L_2 norm may be undesirable because it increases very slowly near the origin

- In several machine learning applications, it is important to discriminate between elements that are exactly zero and elements that are small but nonzero

L_1 norm

$$\|\vec{x}\|_1 = \sum_{k=1}^n |x_k| \quad (1.57)$$

- grows at the same rate in all locations
- retains mathematical simplicity
- commonly used in machine learning when the difference between zero and nonzero elements is very important

L_∞ norm

$$\|\vec{x}\|_\infty = \max_k |x_k| \quad (1.58)$$

- also known as the **max norm**
- absolute value of the element with the largest magnitude in the vector

L_p norm

in general,

$$\|\vec{x}\|_p = \left(\sum_{k=1}^n |x_k|^p \right)^{\frac{1}{p}}, p \geq 1 \quad (1.59)$$

L_0 norm

$$\|\vec{x}\|_0 = \sum_{k=1}^n \mathbf{1}\{x_k \neq 0\} \quad (1.60)$$

- counting its number of nonzero elements
- often use L_1 norm as a proxy to L_0
- this is not a true norm because scaling the vec. by c does not change the # nonzero entries
- non-convex

Mahalanobis Distance

$$Maha(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})^T \Sigma^{-1} (\vec{x} - \vec{y})} \quad (1.61)$$

$$= \sqrt{(\vec{x} - \vec{y})^T U \Lambda^{-1} U^T (\vec{x} - \vec{y})} // \Sigma = U \Lambda U^T \quad (1.62)$$

$$= \sqrt{(\vec{x} - \vec{y})^T \sum_{j=1}^n \frac{1}{\lambda_j} \vec{u}_j \vec{u}_j^T (\vec{x} - \vec{y})} \quad (1.63)$$

$$= \sqrt{\sum_{j=1}^n \frac{((\vec{x} - \vec{y})^T \vec{u}_j)^2}{\lambda_j}} \quad (1.64)$$

norms can also be def. for mat.,

$$A = U \Sigma V^T \quad (1.65)$$

Nuclear Norm

$$\|A\|_* = \sum_{k=1}^r \sigma_k = \|\sigma_1 \ \sigma_2 \ \cdots \ \sigma_r\|_1 \quad (1.66)$$

Frobenius norm

$$\|A\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n A_{ij}^2} \quad (1.67)$$

$$= \sqrt{\sum_{j=1}^n \vec{a}_j^T \vec{a}_j} // \vec{a}_j = A_{*j} \quad (1.68)$$

$$= \sqrt{\sum_{j=1}^n \text{tr} \vec{a}_j \vec{a}_j^T} // \vec{a}^T \vec{a} = \text{tr} \vec{a} \vec{a}^T \quad (1.69)$$

$$= \sqrt{\text{tr} \sum_{j=1}^n \vec{a}_j \vec{a}_j^T} \quad (1.70)$$

$$= \sqrt{\text{tr} A A^T} \quad (1.71)$$

$$= \sqrt{\text{tr} U \Sigma^2 U^T} \quad (1.72)$$

$$= \sqrt{\text{tr} \sum_{k=1}^r \sigma_k^2 \vec{u}_k \vec{u}_k^T} \quad (1.73)$$

$$= \sqrt{\sum_{k=1}^r \sigma_k^2 \vec{u}_k^T \vec{u}_k} \quad \begin{array}{l} \text{column rank} \\ (1.74) \quad \text{row rank} \\ \text{full rank} \end{array}$$

$$= \sqrt{\sum_{k=1}^r \sigma_k^2} \quad (1.75)$$

$$= \|\sigma_1 \ \sigma_2 \ \cdots \ \sigma_r\|_2 \quad (1.76)$$

Spectral Norm

$$\|A\|_2 = \max_k \sigma_k = \|\sigma_1 \ \sigma_2 \ \cdots \ \sigma_r\|_\infty \quad (1.77)$$

rank

$$\text{rank } A = \sum_{k=1}^r 1\{\sigma_k \neq 0\} \|\sigma_1 \ \sigma_2 \ \cdots \ \sigma_r\|_0 = // \text{ non-convex} \quad (1.78)$$

Cauchy's inequality

$$|\vec{x}^T \vec{y}| \leq \|\vec{x}\|_2 \cdot \|\vec{y}\|_2 \quad (1.79)$$

“=” holds when $\vec{x} = c\vec{y}$, for $c \in \mathbb{R}$

1.2.5 Linear Independence and Rank

column rank of a mat. $A \in \mathbb{R}^{m \times n}$ is the largest number of cols. of A that constitute a linearly independent set.

row rank of a mat. A is the largest number of rows of A that constitute a linearly independent set.

column rank of A is equal to the row rank of A

$$\text{rank } A = \dim C(A) = \dim C(A^T) \quad (1.80)$$

$$\leq \min(m, n), \text{ for } A \in \mathbb{R}^{m \times n} \quad (1.81)$$

A is said to be **full rank** when “=” holds

$$\text{rank } A = \text{rank } A^T, \text{ for } A \in \mathbb{R}^{m \times n} \quad (1.82)$$

$$\text{rank } AB \leq \min(\text{rank } A, \text{rank } B), \text{ for } A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{n \times p} \quad (1.83)$$

$$\text{rank}(A + B) \leq \text{rank } A + \text{rank } B, \text{ for } A, B \in \mathbb{R}^{m \times n} \quad (1.84)$$

orthogonal
orthogonal matrix
orthonormal
projection
span

1.2.6 Orthogonal Matrices

two vectors are **orthogonal** :

$$\vec{x}^T \vec{y} = 0, \text{ for } \vec{x}, \vec{y} \in \mathbb{R}^n \quad (1.85)$$

orthogonal matrix : all its columns \vec{u}_j 's are **orthonormal**

$$U^T U = \begin{bmatrix} \vec{u}_1^T \\ \vec{u}_2^T \\ \vdots \\ \vec{u}_n^T \end{bmatrix} [\vec{u}_1 \ \vec{u}_2 \ \cdots \ \vec{u}_n] \quad (1.86)$$

$$= \begin{bmatrix} \vec{u}_1^T \vec{u}_1 & \vec{u}_1^T \vec{u}_2 & \cdots & \vec{u}_1^T \vec{u}_n \\ \vec{u}_2^T \vec{u}_1 & \vec{u}_2^T \vec{u}_2 & \cdots & \vec{u}_2^T \vec{u}_n \\ \vdots & \vdots & \ddots & \vdots \\ \vec{u}_n^T \vec{u}_1 & \vec{u}_n^T \vec{u}_2 & \cdots & \vec{u}_n^T \vec{u}_n \end{bmatrix} \quad (1.87)$$

$$= I \quad (1.88)$$

$$= UU^T, \text{ for } U \in \mathbb{R}^{n \times n} \quad (1.89)$$

$$U^{-1} = U^T, \text{ for } U \in \mathbb{R}^{n \times n} \quad (1.90)$$

$$\|U\vec{x}\|_2 = \|\vec{x}\|_2, \text{ for } U \in \mathbb{R}^{n \times n}, \vec{x} \in \mathbb{R}^n \quad (1.91)$$

if

$$U \in \mathbb{R}^{m \times n}, m > n \quad (1.92)$$

but its columns are still orthonormal, then

$$U^T U = I \quad (1.93)$$

$$UU^T \neq I \quad (1.94)$$

1.2.7 Projection and Four Fundamental Subspaces

Projection

the **projection** of a vector $\vec{b} \in \mathbb{R}^m$ onto the **span** of $\{\vec{a}_1, \vec{a}_2, \dots, \vec{a}_n\}$, $\vec{a}_j \in \mathbb{R}^m$ is the vector

$$\hat{\vec{b}} \in \text{span}(\vec{a}_1, \vec{a}_2, \dots, \vec{a}_n) \subseteq \mathbb{R}^m \quad (1.95)$$

st. $\hat{\vec{b}}$ is as close as possible to \vec{b} , as measured by the Euclidean norm $\|\vec{b} - \hat{\vec{b}}\|_2$, denote

$$\mathcal{A} = \text{span}(\vec{a}_1, \vec{a}_2, \dots, \vec{a}_n) \quad (1.96)$$

then,

$$\hat{\vec{b}} = \text{proj}_{\mathcal{A}} \vec{b} = \arg \min_{\hat{\vec{b}} \in \mathcal{A}} \|\vec{b} - \hat{\vec{b}}\|_2^2 \quad (1.97)$$

Four Fundamental Subspaces

Proof

column space of $A \in \mathbb{R}^{m \times n}$, is the span of the columns of A

$$C(A) = \text{span}(\vec{a}_1, \vec{a}_2, \dots, \vec{a}_n) = \{b \in \mathbb{R}^m : b = A\vec{x}, \forall \vec{x} \in \mathbb{R}^n\} \subseteq \mathbb{R}^m \quad (1.98)$$

$$\dim C(A) = \text{rank } A = r \quad (1.99)$$

if

$$m > n, \text{rank } A = n \quad (1.100)$$

which means all \vec{a}_j 's are lin. indep.

then

$$\hat{\vec{b}} = \text{proj}_{C(A)} \vec{b} = A(A^T A)^{-1} A^T \vec{b} \quad (1.101)$$

Proof : suppose

$$\hat{\vec{b}} = A\hat{\vec{x}} \quad (1.102)$$

the error term

$$\vec{e} = \vec{b} - \hat{\vec{b}} = \vec{b} - A\hat{\vec{x}} \perp C(A) \quad (1.103)$$

$$\because C(A) = N(A^T)^\perp \quad (1.104)$$

$$\therefore \vec{e} \in N(A^T) \quad (1.105)$$

$$\therefore A^T \vec{e} = A^T(\vec{b} - A\hat{\vec{x}}) = 0 \quad (1.106)$$

$$\therefore A^T A\hat{\vec{x}} = A^T \vec{b} \quad (1.107)$$

$$\therefore \hat{\vec{x}} = (A^T A)^{-1} A^T \vec{b} \quad (1.108)$$

$$\therefore \hat{\vec{b}} = A\hat{\vec{x}} = A(A^T A)^{-1} A^T \vec{b} \quad \square \quad (1.109)$$

when

$$A = \vec{a} \in \mathbb{R}^m \quad (1.110)$$

this is to proj. a vec. on to a line

$$\hat{\vec{b}} = \frac{\vec{a}\vec{a}^T}{\vec{a}^T \vec{a}} \vec{b} \quad (1.111)$$

Nullspace

$$N(A) = \{\vec{x} \in \mathbb{R}^n : A\vec{x} = \vec{0}\} \subseteq \mathbb{R}^n \quad (1.112)$$

$$\dim N(A) = n - \text{rank } A = n - r \quad (1.113)$$

Row Space

$$\dim C(A^T) = \text{rank } A = r \quad (1.114)$$

orthogonal
complements
Proof

Left Nullspace

$$\dim N(A^T) = m - \text{rank } A = m - r \quad (1.115)$$

when do

```
1 R = rref(A);
```

$$R = EA \quad (1.116)$$

- $C(A^T) = C(R^T)$, $\dim C(A^T) = \dim C(R^T) = r$, and same basis, we can choose first r rows of R as a basis of $C(A^T)$ or r rows of A corresponding to the pivot rows in R (elimination change rows, but not row space)
- $C(A) \neq C(R)$, $\dim C(A) = \dim C(R) = r$, the r pivot cols. of A are a basis of $C(A)$
- $N(A) = N(R)$, $\dim N(A) = \dim N(R) = n - r$, and same basis, the special sol. \vec{x}_n are a basis of $N(A)$
- $N(A^T) \neq C(R)$, $\dim N(A^T) = \dim N(R^T) = m - r$, the last $m - r$ rows of E are a basis for the $N(A^T)$

$C(A^T)$ and $N(A)$ are **orthogonal complements** in \mathbb{R}^n : $C(A^T)$ contains every vector that is perpendicular to $N(A)$, see Fig. 1.1

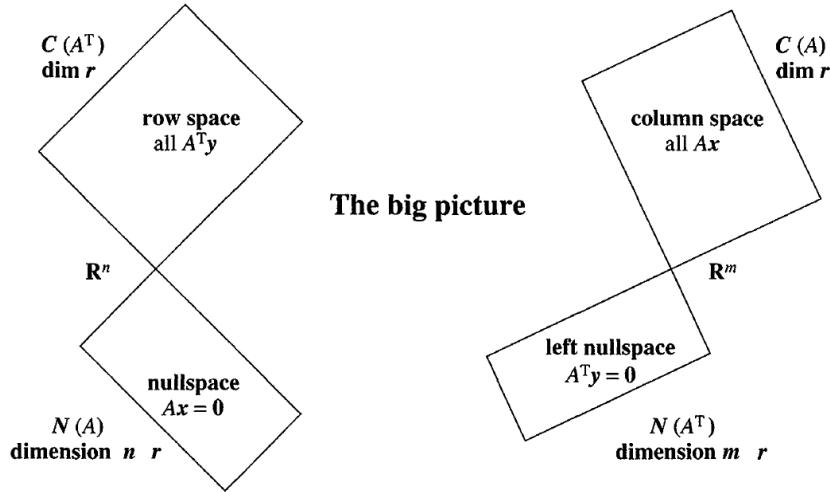


Figure 1.1: the Four Fundamental Subspaces

$$C(A^T) = N(A)^\perp \quad (1.117)$$

Proof : suppose

$$\vec{x} \in N(A) \quad (1.118)$$

$$\therefore A\vec{x} = \begin{bmatrix} \vec{a}_1^T \\ \vec{a}_2^T \\ \vdots \\ \vec{a}_m^T \end{bmatrix} \vec{x} = \vec{0} \quad \begin{array}{l} \text{orthogonal} \\ \text{complements} \\ \text{Proof} \end{array} \quad (1.119)$$

$$\therefore \vec{a}_i^T \vec{x} = 0, \forall i \in \{1, 2, \dots, m\} \quad (1.120)$$

$$\therefore \left(\sum_{i=1}^m \alpha_i \vec{a}_i \right)^T \vec{x} = 0, \forall i, \alpha_i \quad (1.121)$$

$$\therefore \vec{x} \perp C(A^T) \quad \square \quad (1.122)$$

$C(A)$ and $N(A^T)$ are **orthogonal complements** in \mathbb{R}^m : $C(A)$ contains every vector that is perpendicular to $N(A^T)$

$$C(A) = N(A^T)^\perp \quad (1.123)$$

Proof : suppose

$$\vec{y} \in N(A^T) \quad (1.124)$$

$$\therefore A^T \vec{y} = [\vec{a}_1 \ \vec{a}_2 \ \dots \ \vec{a}_n]^T \vec{y} = \begin{bmatrix} \vec{a}_1^T \\ \vec{a}_2^T \\ \vdots \\ \vec{a}_n^T \end{bmatrix} \vec{y} = \vec{0} \quad (1.125)$$

$$\therefore \vec{a}_j^T \vec{y} = 0, \forall j \in \{1, 2, \dots, n\} \quad (1.126)$$

$$\therefore \left(\sum_{i=1}^m \alpha_i \vec{a}_i \right)^T \vec{y} = 0, \forall j, \alpha_j \quad (1.127)$$

$$\therefore \vec{y} \perp C(A) \quad \square \quad (1.128)$$

1.2.8 Determinant

for

$$A = \begin{bmatrix} \vec{a}_1^T \\ \vec{a}_2^T \\ \vdots \\ \vec{a}_n^T \end{bmatrix} \in \mathbb{R}^{n \times n}, \vec{a}_j \in \mathbb{R}^n \quad (1.129)$$

consider

$$\mathcal{A} = \{\vec{c} \in \mathbb{R}^n : \vec{c} = \sum_{j=1}^n \alpha_j \vec{a}_j, 0 \leq \alpha_j \leq 1\} \quad (1.130)$$

is formed by taking all possible linear combinations of the row vecs. of A , where the coefficients of the linear combination are all between 0 and 1; that is, \mathcal{A} is a restricted span($\vec{a}_1, \vec{a}_2, \dots, \vec{a}_n$), then

$$|\det A| = \text{vol}\mathcal{A} \quad (1.131)$$

determinant

that is, The absolute value of the determinant of A , it turns out, is a measure of the “volume” of the \mathcal{A} .

Ex. Compute the absolute value of the determinant of A , where

$$A = \begin{bmatrix} 1 & 3 \\ 3 & 2 \end{bmatrix} \quad (1.132)$$

Sol.:

$$\vec{a}_1 = \begin{bmatrix} 1 \\ 3 \end{bmatrix}, \vec{a}_2 = \begin{bmatrix} 3 \\ 2 \end{bmatrix} \quad (1.133)$$

then, \mathcal{A} corresponds to the shaded region (i.e., the parallelogram) in Fig. 1.2.

$$|\det A| = \text{area}(\mathcal{A}) = 7 \quad (1.134)$$

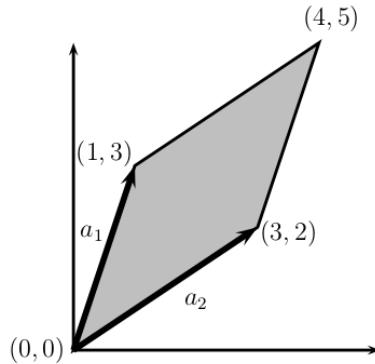


Figure 1.2: the parallelogram corresponds to A

In three dimensions, \mathcal{A} corresponds to an object known as a parallelepiped (a three-dimensional box with skewed sides, such that every face has the shape of a parallelogram). In even higher dimensions, the set S is an object known as an n -dimensional parallelotope.

the **determinant** satisfies the following three properties

- The determinant of the identity is 1 (the volume of a unithypercube is 1)

$$|I| = 1 \quad (1.135)$$

- If we multiply a single row in A by a scalar c , then the det. of the new mat. is $c \det A$ (multiplying one of the sides of \mathcal{A} by a factor c causes

the vol. to increase by a factor c)

adjoint
Proof

$$\det \begin{bmatrix} c\vec{a}_1^T \\ \vec{a}_2^T \\ \vdots \\ \vec{a}_n^T \end{bmatrix} = c|A|, \text{ for } A \in \mathbb{R}^{n \times n}, c \in \mathbb{R} \quad (1.136)$$

- If we exchange any two rows of A , then the determinant of the new matrix is $-\det A$

$$\det \begin{bmatrix} \vec{a}_2^T \\ \vec{a}_1^T \\ \vdots \\ \vec{a}_n^T \end{bmatrix} = -|A| \quad (1.137)$$

Several properties that follow from the three properties above include

$$|A^T| = |A|, \text{ for } A \in \mathbb{R}^{n \times n} \quad (1.138)$$

$$|AB| = |A||B|, \text{ for } A, B \in \mathbb{R}^{n \times n} \quad (1.139)$$

$$|cA| = c^n|A|, \text{ for } A \in \mathbb{R}^{n \times n}, c \in \mathbb{R} \quad (1.140)$$

$$|A^{-1}| = \frac{1}{|A|}, \text{ if } A \in \mathbb{R}^{n \times n} \text{ is not singular} \quad (1.141)$$

$$A^{-1} = \frac{1}{|A|} C^T \quad (1.142)$$

C is **adjoint** mat. of A

1.2.9 Quadratic Forms and Positive Semidefinite Matrices

$$\vec{x}^T A \vec{x} = \sum_{i=1}^m \sum_{j=1}^n A_{ij} x_i x_j, \text{ for } A \in \mathbb{S}^n, \vec{x} \in \mathbb{R}^n \quad (1.143)$$

pd. and nd. mat. has full rank, is invertible

Proof : by contradiction

suppose $A \in \mathbb{R}^{n \times n}$ is not full rank, wlog., let

$$\vec{a}_n = \sum_{j=1}^{n-1} x_j \vec{a}_j \quad (1.144)$$

by setting $x_n = -1$, which means $\vec{x} \neq \vec{0}$

$$A\vec{x} = \sum_{j=1}^{n-1} x_j \vec{a}_j - x_n \vec{a}_n = \vec{0} \quad (1.145)$$

Gram matrix
diagonalizable

$$\therefore \vec{x}^T A \vec{x} = 0 \quad (1.146)$$

it is a contradiction \square

for any mat. $A \in \mathbb{R}^{m \times n}$ (not necessarily symmetric or even square), the **Gram matrix**

$$G = A^T A \in \mathbb{S}_+^n \quad (1.147)$$

is always psd. if $m \geq n$, $\text{rank } A = n$, then $G \in \mathbb{S}_{++}^n$ is pd.

when a mat. $A \in \mathbb{S}^n$ has one of these properties, it has them all

- $\vec{x}^T A \vec{x} > 0$ for $\vec{x} \neq \vec{0}$
- all n pivots > 0
- all n upper left determinant > 0
- all n eig.-vals. > 0
- $A = R^T R$ for a mat. R with indep. cols.

1.2.10 Eigenvalues and Eigenvectors

$$\text{tr } A = \sum_{i=1}^m \lambda_i \quad (1.148)$$

$$|A| = \prod_{i=1}^m \lambda_i \quad (1.149)$$

$$\text{rank } A = \sum_{i=1}^m 1\{\lambda_i \neq 0\} \quad (1.150)$$

$$A^{-1} \vec{x} = \frac{1}{\lambda} \vec{x}, \text{ if } A \text{ is non-singular} \quad (1.151)$$

the eig.-vals. of a diag. mat.

$$D = \text{diag}(d_1, d_2, \dots, d_n) \quad (1.152)$$

are just the diag. entries d_1, d_2, \dots, d_n

we can write all

$$A \vec{x}_i = \lambda_i \vec{x}_i \quad (1.153)$$

as

$$AX = A[\vec{x}_1 \ \vec{x}_2 \ \cdots \ \vec{x}_n] = [\lambda_1 \vec{x}_1 \ \lambda_2 \vec{x}_2 \ \cdots \ \lambda_n \vec{x}_n] = X \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix} = X \Lambda \quad (1.154)$$

if the eigenvectors of A are linearly independent, then the matrix X will be invertible, so A is **diagonalizable**

$$A = X \Lambda X^{-1} \quad (1.155)$$

1.2.11 Eigenvalues and Eigenvectors of Symmetric Matrices

spectral decomposition

when $A \in \mathbb{S}^n$

- any A is guaranteed to have an eigendecomposition
- all the eigenvalue λ_i 's are real
- the eigenvector \vec{x}_i 's are orthonormal, ie., X is an orthogonal matrix, often written as $U = X$
- the eigendecomposition is not unique
 - if any two or more eigenvectors share the same eigenvalue, then any set of orthogonal vectors lying in their span are also eigenvectors with that eigenvalue, and we could equivalently choose a U using those eigenvectors instead
 - by convention, we usually sort the entries of Λ in descending order
 - under this convention, the eigendecomposition is unique only if all of the eigenvalues are unique

$$A = X\Lambda X^{-1} \quad (1.156)$$

$$= U\Lambda U^T \quad (1.157)$$

$$= [\vec{u}_1 \ \vec{u}_2 \ \cdots \ \vec{u}_n] \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix} \begin{bmatrix} \vec{u}_1^T \\ \vec{u}_2^T \\ \vdots \\ \vec{u}_n^T \end{bmatrix} \quad (1.158)$$

$$= \sum_{j=1}^n \lambda_j \vec{u}_j \vec{u}_j^T, \text{ for } A \in \mathbb{S}^n \quad (1.159)$$

this is **spectral decomposition**

$$\vec{c}^T A \vec{c} = \vec{c}^T U \Lambda U^T \vec{c} = \vec{b}^T \Lambda \vec{b} = \sum_{i=1}^m \lambda_i b_i^2 \quad (1.160)$$

- since U is full rank, any vec. $\vec{b} \in \mathbb{R}^n$ can be repr.

$$\vec{b} = U^T \vec{x} \quad (1.161)$$

- the sign of this expression depends entirely on the λ_i 's, it can be pd., psd., nd., nsd., or indefinite (if some of λ_i 's are pos., and some of λ_i 's are neg.)

linear

Ex. Suppose $A \in \mathbb{S}^n$

(1). sol

$$\max_{\vec{x} \in \mathbb{R}^n} \vec{x}^T A \vec{x} \quad (1.162)$$

$$\text{st. } \|\vec{x}\|_2^2 = 1 \quad (1.163)$$

(2). sol

$$\max_{\vec{x} \in \mathbb{R}^n} \vec{x}^T A \vec{x} \quad (1.164)$$

$$\text{st. } \|\vec{x}\|_2^2 = 1 \quad (1.165)$$

Sol.: assume eigenvalues are ordered as

$$\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n \quad (1.166)$$

(1). the opt. for this problem is \vec{x}_1 , the eigenvector corresponding to λ_1
the max. val. of the quad. form is λ_1 (2). the opt. for this problem is \vec{x}_n , the eigenvector corresponding to λ_n
the max. val. of the quad. form is λ_n

1.2.12 Linear Transformation

Linear Transformation

A transformation \vec{T} is **linear** if

$$\vec{T}(c_1 \vec{v}_1 + c_2 \vec{v}_2) = c_1 \vec{T}(\vec{v}_1) + c_2 \vec{T}(\vec{v}_2), \forall \vec{v}_1, \vec{v}_2 \in \mathbb{R}^n, c_1, c_2 \in \mathbb{R} \quad (1.167)$$

$$\therefore \vec{T}(\vec{0}) = \vec{T}(\vec{v} - \vec{v}) = \vec{T}(\vec{v}) - \vec{T}(\vec{v}) = \vec{0} \quad (1.168)$$

linear transformations are an abstract description of multiplication by a matrix

$$\vec{T}(\vec{v}) = B \vec{v}, \text{ for } B \in \mathbb{R}^{m \times n} \quad (1.169)$$

$$B(c_1 \vec{v}_1 + c_2 \vec{v}_2) = c_1 B \vec{v}_1 + c_2 B \vec{v}_2, \forall \vec{v}_1, \vec{v}_2 \in \mathbb{R}^n, c_1, c_2 \in \mathbb{R} \quad (1.170)$$

if we wish to know $\vec{T}(\vec{v}), \forall \vec{v} \in \mathbb{R}^n$, we just need to know $\vec{T}(\vec{v}_1), \vec{T}(\vec{v}_2), \dots, \vec{T}(\vec{v}_n)$
for any basis $\vec{v}_1, \vec{v}_2, \dots, \vec{v}_n$ of the input space

$$\vec{v} = x_1 \vec{v}_1 + x_2 \vec{v}_2 + \dots + x_n \vec{v}_n = V \vec{x} \in \mathbb{R}^n \quad (1.171)$$

$$\vec{T}(\vec{v}) = x_1 \vec{T}(\vec{v}_1) + x_2 \vec{T}(\vec{v}_2) + \dots + x_n \vec{T}(\vec{v}_n) = A \vec{x} \in \mathbb{R}^m \quad (1.172)$$

This is how we get from a (coordinate-free) linear transformation to a
(coordinate based) matrix, x_j 's are our coordinates.

Once we've chosen a basis, every vector \vec{v} in the space can be written as a combination of basis vectors in exactly one way. The coefficients of those vectors are the coordinates of \vec{v} in that basis.

suppose $\vec{w}_1, \vec{w}_2, \dots, \vec{w}_m \in \mathbb{R}^m$ is the basis of the output space, then the entries of j -col. of mat. A is determined by

$$\vec{T}(\vec{v}_j) = A_{1j}\vec{w}_1 + A_{2j}\vec{w}_2 + \dots + A_{mj}\vec{w}_m \in \mathbb{R}^m \quad (1.173)$$

Change of Basis

let

$$W = [\vec{w}_1 \ \vec{w}_2 \ \dots \ \vec{w}_m] \in \mathbb{R}^{m \times m} \quad (1.174)$$

be the basis vectors of the new basis. Then if \vec{x} is a vec. in the old basis, we can convert it to a vec. \vec{a} in the new basis using

$$\vec{x} = W\vec{a} \quad (1.175)$$

Wavelet Transform = Change to Wavelet Basis

Haar basis

$$\vec{w}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \vec{w}_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}, \vec{w}_3 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0 \end{bmatrix}, \vec{w}_4 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ -1 \end{bmatrix} \quad (1.176)$$

$$\vec{w}_i^T \vec{w}_j = 0, \forall i, j \quad (1.177)$$

DWT finds the coefficients \vec{a}

$$\vec{a} = W^{-1}\vec{x} \quad (1.178)$$

\vec{x} is expressed in the wavelet basis

$$\vec{x} = W\vec{a} = a_1\vec{w}_1 + a_2\vec{w}_2 + \dots + a_n\vec{w}_n \quad (1.179)$$

compress that long signal, by keeping only the largest 5% of the coefficients in \vec{a} , we get $\hat{\vec{a}}$

reconstruct step:

$$\hat{\vec{x}} = W\hat{\vec{a}} \quad (1.180)$$

Fourier Transform = Change to Fourier Basis

when $n = 4$

$$F = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & j^2 & j^3 \\ 1 & j^2 & j^4 & j^6 \\ 1 & j^3 & j^6 & j^9 \end{bmatrix} \quad (1.181)$$

DFT decomposes the signal into waves at equally spaced frequencies

$$\vec{a} = F^{-1}\vec{x} \quad (1.182)$$

$$\vec{x} = F\vec{a} \quad (1.183)$$

Transform Matrix

Suppose we have a linear transformation \vec{T} . If \vec{T} has the matrix A when working with the input basis $\vec{v}_1, \dots, \vec{v}_n$ and \vec{T} has the matrix B when working with the input basis $\vec{w}_1, \dots, \vec{w}_n$, it turns out that A and B must be similar matrices. In other words,

$$B = M^{-1}AM \quad (1.184)$$

for some change of basis matrix M .

If our basis consists of eigenvectors of our transformation

$$\vec{T}(\vec{v}_j) = \lambda_j \vec{v}_j \quad (1.185)$$

then

$$A = V\Lambda \quad (1.186)$$

the (diagonal) matrix of eigenvalues. It would be wonderful to use a basis of eigenvectors for image processing, but finding such a basis requires far more computation than simply using a Fourier or wavelet basis.

1.2.13 Singular Value Decomposition (SVD)

$$A \in \mathbb{R}^{m \times n}, \text{rank } A = r \quad (1.187)$$

$$A = U\Sigma V^T = \sum_{k=1}^r \sigma_k \vec{u}_k \vec{v}_k^T \quad (1.188)$$

$$AV = U\Sigma \quad (1.189)$$

$$A\vec{v}_k = \sigma_k \vec{u}_k, \forall k \in \{1, 2, \dots, r\} \quad (1.190)$$

$$A\vec{v}_k = 0, \forall k \in \{r+1, r+2, \dots, n\} \quad (1.191)$$

$$A^T = V\Sigma U^T \quad (1.192)$$

$$A^T U = V\Sigma \quad (1.193)$$

$$A^T \vec{u}_k = \sigma_k \vec{v}_k, \forall k \in \{1, 2, \dots, r\} \quad (1.194)$$

$$A^T \vec{u}_k = 0, \forall k \in \{r+1, r+2, \dots, m\} \quad (1.195) \quad \text{Moore-Penrose Pseudoinverse}$$

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 & \vec{0}^T \\ 0 & \sigma_2 & \cdots & 0 & \vec{0}^T \\ \vdots & \vdots & \ddots & \vdots & \vec{0}^T \\ 0 & 0 & \cdots & \sigma_r & \vec{0}^T \\ \vec{0} & \vec{0} & \vec{0} & \vec{0} & 0_{(m-r) \times (n-r)} \end{bmatrix} \in \mathbb{R}^{m \times n} \quad (1.196)$$

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0 \quad (1.197)$$

$$U = [\vec{u}_1 \ \vec{u}_2 \ \cdots \ \vec{u}_m] \in \mathbb{R}^{m \times m} \quad (1.198)$$

\vec{u}_i 's are eig.-vecs. of $AA^T \in \mathbb{S}^m$, σ_i^2 is the corresponding eig.-vals., \vec{u}_i 's are orthonormal

$$U^T U = UU^T = I \quad (1.199)$$

$$\vec{u}_k \in C(A), \forall k \in \{1, 2, \dots, r\} \quad (1.200)$$

$$\vec{u}_k \in N(A^T), \forall k \in \{r+1, r+2, \dots, m\} \quad (1.201)$$

$$V = [\vec{v}_1 \ \vec{v}_2 \ \cdots \ \vec{v}_n] \in \mathbb{R}^{n \times n} \quad (1.202)$$

\vec{v}_j 's are eig.-vecs. of $A^T A \in \mathbb{S}^n$, σ_i^2 is the corresponding eig.-vals., \vec{v}_j 's are orthonormal

$$V^T V = VV^T = I \quad (1.203)$$

$$\vec{v}_k \in C(A^T), \forall k \in \{1, 2, \dots, r\} \quad (1.204)$$

$$\vec{v}_k \in N(A), \forall k \in \{r+1, r+2, \dots, n\} \quad (1.205)$$

SVD and Linear Transformation

$$\therefore A = U\Sigma V^T \quad (1.206)$$

$$\therefore A\vec{x} = U\Sigma V^T \vec{x} \quad (1.207)$$

- A uses the std. basis I for $\vec{x} \in \mathbb{R}^n$
- Σ uses input basis V and output basis U
- V change from std. basis I to V 's basis
- U^T change from U 's basis to std. basis I

Pseudoinverse Moore-Penrose Pseudoinverse

$$A\vec{x} = \vec{y} \text{ for } A \in \mathbb{R}^{m \times n} \quad (1.208)$$

$$A^+ = V\Sigma^+U^T \quad (1.209)$$

$$\Sigma^+ = \begin{bmatrix} \frac{1}{\sigma_1} & & \\ & \ddots & \\ & & \frac{1}{\sigma_k} \end{bmatrix} \quad (1.210)$$

$$\hat{\vec{x}} = A^+\vec{y} \quad (1.211)$$

- when $m \leq n$, there could be multiple possible inverse, A^+ provides one sol. with min. $\|\hat{\vec{x}}\|_2$
- when $m > n$, there may be no inverse, A^+ provides one sol. with min. $\|A\hat{\vec{x}} - \vec{y}\|_2$
- if $\vec{x} \in C(A^T)$, $\vec{x} = A^+A\vec{x}$

low rank approximation a rank 5 approx.

$$A \approx \sigma_1 \vec{u}_1 \vec{v}_1^T + \sigma_2 \vec{u}_2 \vec{v}_2^T + \cdots + \sigma_5 \vec{u}_5 \vec{v}_5^T \quad (1.212)$$

PCA

$$X = [\vec{x}^{(1)} \ \vec{x}^{(2)} \ \cdots \ \vec{x}^{(m)}] = U\Sigma V^T \in \mathbb{R}^{n \times m} \quad (1.213)$$

with zero mean in each row

top k cols. of U give the PCs of X

$$U = [\vec{u}_1 \ \vec{u}_2 \ \cdots \ \vec{u}_n] \in \mathbb{R}^{n \times n} \quad (1.214)$$

$$U_{red} = [\vec{u}_1 \ \vec{u}_2 \ \cdots \ \vec{u}_k] \in \mathbb{R}^{n \times k} \quad (1.215)$$

the new coordinates of \vec{x} is

$$\vec{a} = U_{red}^T \vec{x} \in \mathbb{R}^k \quad (1.216)$$

the new coordinates of X is

$$A = U_{red}^T X \quad (1.217)$$

$$= U_{red}^T U \Sigma V^T \quad (1.218)$$

$$= \begin{bmatrix} \vec{u}_1^T \\ \vec{u}_2^T \\ \vdots \\ \vec{u}_k^T \end{bmatrix} [\vec{u}_1 \ \vec{u}_2 \ \cdots \ \vec{u}_n] \Sigma V^T \quad (1.219)$$

$$= \begin{bmatrix} \vec{u}_1^T \vec{u}_1 & \vec{u}_1^T \vec{u}_2 & \cdots & \vec{u}_1^T \vec{u}_n \\ \vec{u}_2^T \vec{u}_1 & \vec{u}_2^T \vec{u}_2 & \cdots & \vec{u}_2^T \vec{u}_n \\ \vdots & \vdots & \ddots & \vdots \\ \vec{u}_k^T \vec{u}_1 & \vec{u}_k^T \vec{u}_2 & \cdots & \vec{u}_k^T \vec{u}_n \end{bmatrix} \Sigma V^T \quad (1.220)$$

$$= [I_{k \times k} \ 0_{k \times (n-k)}] \begin{bmatrix} \sigma_1 & & & \\ & \ddots & & \\ & & \sigma_k & \\ & & & \ddots \\ & & & & \sigma_r \end{bmatrix} V^T \quad (1.221)$$

$$= \begin{bmatrix} \sigma_1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & \sigma_k \end{bmatrix} V^T \quad (1.222)$$

$$= \Sigma_k V^T \quad (1.223)$$

1.2.14 Nonsingular Matrix and Singular Matrix

Table 1.1: Nonsingular Matrix and Singular Matrix

Nonsingular	Singular
A is invertible	A is not invertible
cols. of A are indep.	cols. of A are dep.
rows. of A are indep.	rows. of A are dep.
$ A \neq 0$	$ A = 0$
$A\vec{x} = \vec{0}$ has only one sol. $\vec{x} = \vec{0}$	$A\vec{x} = \vec{0}$ has infinitely many sols.
$A\vec{x} = \vec{b}$ has only one sol. $\vec{x} = A^{-1}\vec{b}$	$A\vec{x} = \vec{b}$ has no sol. or infinitely many sols.
A has n (non-zero) pivots	A has $r < n$ pivots
rank $A = n$	rank $A = r < n$
$R = \text{rref}(A) = I$	R has ≥ 1 zero rows
$C(A) = \mathbb{R}^n$	$\dim C(A) = r < n$
$C(A^T) = \mathbb{R}^n$	$\dim C(A^T) = r < n$
all eig.-vals. are > 0	0 is an eig.-val. of A
$A^T A$ is pd.	$A^T A$ is always psd.
A has n singular vals.	A has $r < n$ singular vals.

suppose $A \in \mathbb{R}^{n \times n}$, see Tab. 1.1

1.3 Matrix Calculus

1.3.1 Gradient

$$\nabla_A f(A) = \begin{bmatrix} \frac{\partial f(A)}{\partial A_{11}} & \frac{\partial f(A)}{\partial A_{12}} & \cdots & \frac{\partial f(A)}{\partial A_{1n}} \\ \frac{\partial f(A)}{\partial A_{21}} & \frac{\partial f(A)}{\partial A_{22}} & \cdots & \frac{\partial f(A)}{\partial A_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f(A)}{\partial A_{m1}} & \frac{\partial f(A)}{\partial A_{m2}} & \cdots & \frac{\partial f(A)}{\partial A_{mn}} \end{bmatrix}, \text{ for } A \in \mathbb{R}^{m \times n} \quad (1.224)$$

$$(\nabla_A f(A))_{ij} = \frac{\partial f(A)}{\partial A_{ij}} \quad (1.225)$$

$$\nabla_{\vec{x}} f(\vec{x}) = \begin{bmatrix} \frac{\partial f(\vec{x})}{\partial x_1} \\ \frac{\partial f(\vec{x})}{\partial x_2} \\ \vdots \\ \frac{\partial f(\vec{x})}{\partial x_n} \end{bmatrix} \quad (1.226)$$

$$(\nabla_{\vec{x}} f(\vec{x}))_j = \frac{\partial f(\vec{x})}{\partial x_j} \quad (1.227)$$

the gradient of a function is only defined if f is real-valued, that is, if it returns a scalar value. We can not take the gradient of $A\vec{x}$, $A \in \mathbb{R}^{n \times n}$ wrt. \vec{x} , since this quantity is vector-valued.

$$\nabla_{\vec{x}}(f(\vec{x}) + g(\vec{x})) = \nabla_{\vec{x}}f(\vec{x}) + \nabla_{\vec{x}}g(\vec{x}) \quad (1.228)$$

$$\nabla_{\vec{x}} c f(\vec{x}) = c \nabla_{\vec{x}} f(\vec{x}), \text{ for } c \in \mathbb{R} \quad (1.229)$$

1.3.2 Derivatives of Matrices

$$\frac{\partial f(\vec{x})}{\partial \vec{x}} = \nabla_{\vec{x}} f(\vec{x}) = \begin{bmatrix} \frac{\partial f(\vec{x})}{\partial x_1} \\ \frac{\partial f(\vec{x})}{\partial x_2} \\ \vdots \\ \frac{\partial f(\vec{x})}{\partial x_n} \end{bmatrix} \in \mathbb{R}^n, \text{ for } \vec{x} \in \mathbb{R}^n \quad (1.230)$$

$$(\frac{\partial f(\vec{x})}{\partial \vec{x}})_i = \frac{\partial f(\vec{x})}{\partial x_i} \quad (1.231)$$

$$\frac{\partial \vec{f}(\vec{x})}{\partial \vec{x}} = \nabla_{\vec{x}} \vec{f}(\vec{x}) = \begin{bmatrix} \frac{\partial f(\vec{x})_1}{\partial x} \\ \frac{\partial f(\vec{x})_2}{\partial x} \\ \vdots \\ \frac{\partial f(\vec{x})_m}{\partial x} \end{bmatrix} \in \mathbb{R}^m, \text{ for } \vec{f} \in \mathbb{R}^m \quad (1.232)$$

$$(\frac{\partial \vec{f}(\vec{x})}{\partial \vec{x}})_i = \frac{\partial f(\vec{x})_i}{\partial x} \quad (1.233)$$

$$\frac{\partial \vec{f}(\vec{x})}{\partial \vec{x}} = \nabla_{\vec{x}^T} \vec{f}(\vec{x}) \quad (1.234)$$

$$= \begin{bmatrix} \nabla_{\vec{x}^T} f(\vec{x})_1 \\ \nabla_{\vec{x}^T} f(\vec{x})_2 \\ \vdots \\ \nabla_{\vec{x}^T} f(\vec{x})_m \end{bmatrix} \quad (1.235)$$

$$= (\nabla_{\vec{x}} \vec{f}(\vec{x})^T)^T \quad (1.236)$$

$$= [\nabla_{\vec{x}} f(\vec{x})_1 \ \nabla_{\vec{x}} f(\vec{x})_2 \ \dots \ \nabla_{\vec{x}} f(\vec{x})_m]^T \quad (1.237)$$

$$= \begin{bmatrix} \frac{\partial f(\vec{x})_1}{\partial x_1} & \frac{\partial f(\vec{x})_1}{\partial x_2} & \dots & \frac{\partial f(\vec{x})_1}{\partial x_n} \\ \frac{\partial f(\vec{x})_2}{\partial x_1} & \frac{\partial f(\vec{x})_2}{\partial x_2} & \dots & \frac{\partial f(\vec{x})_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f(\vec{x})_m}{\partial x_1} & \frac{\partial f(\vec{x})_m}{\partial x_2} & \dots & \frac{\partial f(\vec{x})_m}{\partial x_n} \end{bmatrix} \quad (1.238)$$

$$\in \mathbb{R}^{m \times n}, \text{ for } \vec{f}(\vec{x}) \in \mathbb{R}^m, \vec{x} \in \mathbb{R}^n \quad (1.239)$$

$$(\frac{\partial \vec{f}(\vec{x})}{\partial \vec{x}})_{ij} = \frac{\partial f(\vec{x})_i}{\partial x_j} \quad (1.240)$$

this really means taking the gradient of each entry of \vec{f} , not the gradient of the whole vector

1.3.3 Hessian

$$H = \nabla_{\vec{x}}^2 f(\vec{x}) \quad (1.241)$$

$$= \frac{\partial}{\partial \vec{x}} \nabla_{\vec{x}} f(\vec{x}) \quad (1.242)$$

$$= \nabla_{\vec{x}^T} \nabla_{\vec{x}} f(\vec{x}) \quad (1.243)$$

$$= (\nabla_{\vec{x}} (\nabla_{\vec{x}} f(\vec{x}))^T)^T \quad (1.244)$$

$$= \nabla_{\vec{x}} (\nabla_{\vec{x}} f(\vec{x}))^T \quad (1.245)$$

$$= \begin{bmatrix} \frac{\partial^2 f(\vec{x})}{\partial x_1 \partial x_1} & \frac{\partial^2 f(\vec{x})}{\partial x_1 \partial x_2} & \dots & \frac{\partial^2 f(\vec{x})}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(\vec{x})}{\partial x_2 \partial x_1} & \frac{\partial^2 f(\vec{x})}{\partial x_2 \partial x_2} & \dots & \frac{\partial^2 f(\vec{x})}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\vec{x})}{\partial x_n \partial x_1} & \frac{\partial^2 f(\vec{x})}{\partial x_n \partial x_2} & \dots & \frac{\partial^2 f(\vec{x})}{\partial x_n \partial x_n} \end{bmatrix} \quad (1.246)$$

$$\in \mathbb{S}^n, \text{ for } \vec{x} \in \mathbb{R}^n \quad (1.247)$$

$$(\nabla_{\vec{x}}^2 f(\vec{x}))_{ij} = \frac{\partial^2 f(\vec{x})}{\partial x_i \partial x_j} \quad (1.248)$$

Similar to the gradient, the Hessian is defined only when f is real-valued.

1.3.4 Chain Rule of Matrix Calculus

suppose $B = g(A) \in \mathbb{R}^{m \times n}$, $f(B) \in \mathbb{R}$, we want to find

$$\frac{\partial}{\partial A} f(B) = \nabla_A f(B) \quad (1.249)$$

$$\frac{\partial f(B)}{\partial A_{kl}} = \sum_{i=1}^m \sum_{j=1}^n \frac{\partial f(B)}{\partial B_{ij}} \frac{\partial B_{ij}}{\partial A_{kl}} \quad (1.250)$$

$$= \text{tr}\left(\frac{\partial f(B)}{\partial B}\right)^T \frac{\partial B}{\partial A_{kl}} \quad (1.251)$$

$$= \text{tr}(\nabla_B f(B))^T \nabla_{A_{kl}} g(A) \quad (1.252)$$

suppose $\vec{y} = \vec{g}(\vec{x}) \in \mathbb{R}^m$, $f(\vec{y}) \in \mathbb{R}$, we want to find

$$\frac{\partial}{\partial \vec{x}} f(\vec{y}) = \nabla_{\vec{x}} f(\vec{y}) \quad (1.253)$$

$$\nabla_{\vec{x}} f(\vec{y}) = \sum_{i=1}^m \frac{\partial f(\vec{y})}{\partial y_i} \frac{\partial y_i}{\partial \vec{x}} \quad (1.254)$$

$$= \left(\frac{\partial f(\vec{y})}{\partial \vec{y}} \frac{\partial \vec{y}}{\partial \vec{x}} \right)^T \quad (1.255)$$

$$= ((\nabla_{\vec{y}} f(\vec{y}))^T \nabla_{\vec{x}^T} \vec{g}(\vec{x}))^T \quad (1.256)$$

$$= \nabla_{\vec{x}} \vec{g}(\vec{x})^T \nabla_{\vec{y}} f(\vec{y}) \quad (1.257)$$

Proof.

$$(\nabla_{\vec{x}} f(\vec{y}))_j = \frac{\partial f(\vec{y})}{\partial x_j} = \sum_{i=1}^m \frac{\partial f(\vec{y})}{\partial y_i} \frac{\partial y_i}{\partial x_j} \quad (1.258)$$

$$\therefore \nabla_{\vec{x}} f(\vec{y}) = \sum_{i=1}^m \frac{\partial f(\vec{y})}{\partial y_i} \begin{bmatrix} \frac{\partial y_i}{\partial x_1} \\ \frac{\partial y_i}{\partial x_2} \\ \vdots \\ \frac{\partial y_i}{\partial x_n} \end{bmatrix} \quad (1.259)$$

$$= \sum_{i=1}^m \frac{\partial f(\vec{y})}{\partial y_i} \nabla_{\vec{x}} y_i \quad (1.260)$$

$$= \sum_{i=1}^m \nabla_{\vec{x}} y_i \frac{\partial f(\vec{y})}{\partial y_i} \quad (1.261)$$

$$= [\nabla_{\vec{x}} y_1 \ \nabla_{\vec{x}} y_2 \ \cdots \ \nabla_{\vec{x}} y_m] \begin{bmatrix} \frac{\partial f(\vec{y})}{\partial y_1} \\ \frac{\partial f(\vec{y})}{\partial y_2} \\ \vdots \\ \frac{\partial f(\vec{y})}{\partial y_m} \end{bmatrix} \quad (1.262)$$

$$= \nabla_{\vec{x}} \vec{y}^T \nabla_{\vec{y}} f(\vec{y}) \quad (1.263)$$

□

1.3.5 Gradients and Hessians of Quadratic and Linear Functions

$$\nabla_{\vec{x}} \vec{a}^T \vec{x} = \vec{a}, \text{ for } \vec{x}, \vec{a} \in \mathbb{R}^n \quad (1.264)$$

Proof

$$\frac{\partial}{\partial x_j} \vec{a}^T \vec{x} = \frac{\partial}{\partial x_j} \sum_{k=1}^n a_k x_k = a_j \quad \square \quad (1.265)$$

$$\nabla_{\vec{x}}^2 \vec{a}^T \vec{x} = 0, \text{ for } \vec{x}, \vec{a} \in \mathbb{R}^n \quad (1.266)$$

$$\nabla_{\vec{x}} \vec{x}^T A \vec{x} = 2A \vec{x}, \text{ for } A \in \mathbb{S}^n, \vec{x} \in \mathbb{R}^n \quad (1.267)$$

Proof

$$\frac{\partial}{\partial x_k} \vec{x}^T A \vec{x} = \frac{\partial}{\partial x_k} \sum_{i=1}^n \sum_{j=1}^n A_{ij} x_i x_j \quad (1.268)$$

$$= \frac{\partial}{\partial x_k} \left(\sum_{i \neq k} \sum_{j \neq k} A_{ij} x_i x_j + \sum_{i \neq k} A_{ik} x_i x_k + \sum_{j \neq k} A_{kj} x_k x_j + A_{kk} x_k x_k \right) \quad (1.269)$$

$$= \sum_{i \neq k} A_{ik} x_i + \sum_{j \neq k} A_{kj} x_j + 2A_{kk} x_k \quad (1.270)$$

$$= \sum_{i \neq k} A_{ik} x_i + A_{kk} x_k + \sum_{j \neq k} A_{kj} x_j + A_{kk} x_k \quad (1.271)$$

$$= \sum_{i=1}^n A_{ik} x_i + \sum_{j=1}^n A_{kj} x_j \quad (1.272)$$

$$= \sum_{i=1}^n A_{ki} x_i + \sum_{j=1}^n A_{kj} x_j // A \in \mathbb{S}^n \quad (1.273)$$

$$= 2 \sum_{j=1}^n A_{kj} x_j \quad (1.274)$$

$$= 2A_{kk} x_k \quad \square \quad (1.275)$$

$$\nabla_{\vec{x}}^2 \vec{x}^T A \vec{x} = 2A, \text{ for } A \in \mathbb{S}^n, \vec{x} \in \mathbb{R}^n \quad (1.276)$$

Proof

$$\frac{\partial^2}{\partial x_k \partial x_l} \vec{x}^T A \vec{x} = \frac{\partial}{\partial x_l} \left(\frac{\partial}{\partial x_k} \vec{x}^T A \vec{x} \right) = \frac{\partial}{\partial x_l} 2 \sum_{j=1}^n A_{kj} x_j = 2A_{kl} \quad \square \quad (1.277)$$

1.3.6 Gradients of the Determinant

$$\nabla_A |A| = C = |A| A^{-T}, \text{ for } A \in \mathbb{R}^{n \times n} \quad (1.278)$$

Proof

$$\frac{\partial}{\partial A_{kl}} |A| = \frac{\partial}{\partial A_{kl}} \sum_{j=1}^n A_{ij} C_{ij}, \forall i \in \{1, 2, \dots, n\} \quad (1.279)$$

$$= \frac{\partial}{\partial A_{kl}} \sum_{l=1}^n A_{il} C_{il} \quad (1.280)$$

$$= C_{kl} \quad \square \quad (1.281)$$

$$\nabla_A \log |A| = \frac{1}{|A|} \nabla_A |A| = \frac{1}{|A|} |A| A^{-T} = A^{-1}, \text{ for } A \in \mathbb{S}_{++}^n \quad (1.282)$$

Proof

$$\frac{\partial}{\partial A_{kl}} \log |A| = \frac{1}{|A|} \frac{\partial}{\partial A_{kl}} |A| = \frac{1}{|A|} C_{kl} \quad \square \quad (1.283)$$

1.3.7 Gradients of the Trace

$$\nabla_A \operatorname{tr} AB = \nabla_A \operatorname{tr} BA = B^T \quad (1.284)$$

$$\nabla_A \operatorname{tr} ABA^T C = CAB + C^T AB^T \quad (1.285)$$

1.3.8 Gradients of the Transpose

$$\nabla_{A^T} f(A) = (\nabla_A f(A))^T \quad (1.286)$$

1.3.9 Derivative of the Inverse

$$\frac{\partial}{\partial x} A^{-1} = -A^{-1} \frac{\partial A}{\partial x} A^{-1} \quad (1.287)$$

1.3.10 Derivative of the Linear Functions

$$\frac{\partial}{\partial \vec{x}} A \vec{x} = A \quad (1.288)$$

1.3.11 Second Order Taylor Expansion

scalar case

$$f(x) = f(x_0) + \frac{df(x_0)}{dx}(x - x_0) + \frac{1}{2} \frac{d^2f(x_0)}{dx^2}(x - x_0)^2 \quad (1.289)$$

n -dim. case

$$f(\vec{x}) = f(\vec{x}_0) + \left(\frac{\partial f(\vec{x}_0)}{\partial \vec{x}} \right)^T (\vec{x} - \vec{x}_0) + \frac{1}{2} (\vec{x} - \vec{x}_0)^T \frac{\partial^2 f(\vec{x}_0)}{\partial \vec{x}^2} (\vec{x} - \vec{x}_0) \quad (1.290)$$

$$= f(\vec{x}_0) + (\nabla_{\vec{x}} f(\vec{x}_0))^T (\vec{x} - \vec{x}_0) + \frac{1}{2} (\vec{x} - \vec{x}_0)^T H \Big|_{\vec{x}=\vec{x}_0} (\vec{x} - \vec{x}_0) \quad (1.291)$$

1.4 Tensor and Vectorization

1.4.1 Tensor

$$x \in \mathbb{R}^{r \times c \times n} \quad (1.292)$$

treat it as containing n slices of matrices

every slice is a matrix with size $r \times c$

The first slice contains all the numbers in the tensor that can be indexed by $(i, j, 1), 1 \leq i \leq r, 1 \leq j \leq c$

when $n = 1$, a 3D tensor reduces to a matrix

1.4.2 Vectorization

eg.,

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad (1.293)$$

in Matlab,

```
A_vec = A(:);
```

converts a matrix into a column vector in the column-first order

$$A_{vec} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} \quad (1.294)$$

In order to vectorize a 3D tensor, we could vectorize its first slice (which is a matrix), then the second slice, ..., till all slices are vectorized. The vectorization of the 3D tensor is then the concatenation of all the slices. This recursive process can be applied to vectorize a 4D (or even higher order) tensor.

Chapter 2

Probability, Statistics and Information Theory

2.1 Why Probability

2.1.1 Why Need Probability Theory

3 sources of uncertainty

- inherent stochasticity in the system being modeled: eg., in a hypothetical card game where we assume that the cards are truly shuffled into a random order.
- incomplete observability: eg., Monty Hall problem, a contestant is asked to choose between three doors, two doors lead to a goat while the third leads to a car. Given the contestant's choice, the outcome is deterministic, but from the contestant's point of view, the outcome is uncertain.
- incomplete modeling: eg., a robot that can observe the location of object around it, discretization of space makes the robot uncertain about the precise position of objects: each object could be anywhere within the discrete cell that it was observed to occupy.

In many cases, it is more practical to use a simple but uncertain rule rather than a complex but certain one

- eg., the simple rule “Most birds fly” is cheap to develop and is broadly useful
- a rule “Birds fly, except for very young birds that have not yet learned to fly, sick or injured birds that have lost the ability to fly . . .” is expensive to develop, maintain and communicate, and after all of this effort is still very brittle and prone to failure

machine learning must always deal with uncertain quantities, and sometimes may also need to deal with stochastic (non-deterministic) quantities

Frequentist Probability

Bayesian Probability

degree of belief

Sample Space Ω

outcome

Event Space \mathcal{F}

events

Probability Measure \Pr

Axioms of Probability

Union Bound

2.1.2 Two Kinds of Probability

Define 2.1 (Frequentist Probability). *If events are repeatable, when we say that an outcome has a probability p of occurring, it means that if we repeated the experiment infinitely many times, then proportion p of the repetitions would result in that outcome.*

Define 2.2 (Bayesian Probability). *We use probability to represent a **degree of belief**, with 1 indicating absolute certainty and 0 indicating absolute uncertainty.*

2.2 Elements of probability

Define 2.3 (Sample Space Ω). *The set of all the outcomes of a random experiment. Here, each **outcome** $\omega \in \Omega$ can be thought of as a complete description of the state of the real world at the end of the experiment.*

eg., consider the event of tossing a six-sided die, $\Omega = \{1, 2, 3, 4, 5, 6\} = \{\omega_1, \omega_2, \omega_3, \omega_4, \omega_5, \omega_6\}$

Define 2.4 (Event Space \mathcal{F}). *A set whose elements $A \in \mathcal{F}$ (called **events**) are subsets of Ω (ie., $A \subseteq \Omega$ is a collection of possible outcomes of an experiment).*

eg., one possible $\mathcal{F} = \{\{1, 3, 5\}, \{2, 4, 6\}\} = \{A_1, A_2\}$

Define 2.5 (Probability Measure \Pr). *A function $\Pr : \mathcal{F} \mapsto \mathbb{R}$ that satisfies the following properties (often called the **Axioms of Probability**)*

- $\Pr(A) \geq 0, \forall A \in \mathcal{F}$
- $\Pr(\Omega) = 1$
- if A_1, A_2, \dots are disjoint events (ie., $A_k \cap A_j = \emptyset, \forall k \neq j$), then $\Pr(\bigcup_k A_k) = \sum_k \Pr(A_k)$

eg., $\Pr(A_1) = \Pr(\{1, 3, 5\}) = \frac{3}{6} = \frac{1}{2}$

2.2.1 Properties

$$A \subseteq B \rightarrow \Pr(A) \leq \Pr(B) \quad (2.1)$$

$$\Pr(A \cap B) \leq \min(\Pr(A), \Pr(B)) \quad (2.2)$$

$$\Pr(\Omega - A) = 1 - \Pr(A) \quad (2.3)$$

Theorem 2.1 (Union Bound).

$$\Pr(A \cup B) \leq \Pr(A) + \Pr(B) \quad (2.4)$$

Theorem 2.2 (Law of Total Probability). *if A_1, \dots, A_K are a set of disjoint events st. $\bigcup_{k=1}^K A_k = \Omega$*

$$\sum_{k=1}^K \Pr(A_k) = 1 \quad (2.5)$$

*Law of Total Probability
Conditional Probability
Independent
Mutually Independent
Random Variable
Discrete Random Variable*

2.2.2 Conditional Probability

Define 2.6 (Conditional Probability).

$$\Pr(A|B) = \frac{\Pr(A \cap B)}{\Pr(B)} \quad (2.6)$$

every statement remains true if we add conditions (“|.”) consistently on all the p, \Pr in that statement

2.2.3 Independence

Define 2.7 (Independent). *Two events are called independent iff.*

$$\Pr(A \cap B) = \Pr(A) \Pr(B) \quad (2.7)$$

or equivalently,

$$\Pr(A|B) = \Pr(A) \quad (2.8)$$

Therefore, independence is equivalent to saying that observing B does not have any effect on the probability of A

Define 2.8 (Mutually Independent). *For multiple events, A_1, \dots, A_K , we say they are mutually independent if for any subset $S \subseteq \{1, 2, \dots, K\}$, we have*

$$\Pr(\bigcap_{k \in S} A_k) = \prod_{k \in S} \Pr(A_k). \quad (2.9)$$

2.3 Random Variables

Define 2.9 (Random Variable). *A rv. X is a function $X : \Omega \mapsto \mathbb{R}$, denote as $X(\omega)$ or X .*

$$\Pr(X \in A) = \Pr(\{\omega \in \Omega : X(\omega) \in A\}) \quad (2.10)$$

Define 2.10 (Discrete Random Variable). *$X(\omega)$ can take only a finite number of values.*

$$\Pr(X = k) = \Pr(\{\omega : X(\omega) = k\}) \quad (2.11)$$

eg., consider an experiment in which we flip 10 coins, each ω_i is a length-10 sequence of heads and tails, say, $\omega_1 = \{HHTHTHHHTT\} \in \Omega$, suppose $X(\omega)$ is # heads in the seq. of tosses ω , $X(\omega_1) = 5$.

Continuous Random Variable

Random Vector

Cumulative Distribution Functions (CDF)

Joint Cumulative Distribution Function

Marginal Cumulative Distribution Functions is

Define 2.11 (Continuous Random Variable). $X(\omega)$ takes on a infinite number of possible values.

$$\Pr(a \leq X \leq b) = \Pr(\{\omega : a \leq X(\omega) \leq b\}) \quad (2.12)$$

Define 2.12 (Random Vector). Consider n rv., random vector $\vec{X} : \Omega \mapsto \mathbb{R}^n$

$$\vec{X} = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \end{bmatrix} \quad (2.13)$$

2.3.1 Cumulative Distribution Functions (CDF)

Define 2.13 (Cumulative Distribution Functions (CDF)). Consider rv. $X, F_X : \mathbb{R} \mapsto [0, 1]$

$$F_X(x) = \Pr(X \leq x) \quad (2.14)$$

$$0 \leq F_X(x) \leq 1 \quad (2.15)$$

$$\lim_{x \rightarrow -\infty} F_X(x) = 0 \quad (2.16)$$

$$\lim_{x \rightarrow \infty} F_X(x) = 1 \quad (2.17)$$

$$x \leq y \rightarrow F_X(x) \leq F_X(y) \quad (2.18)$$

Define 2.14 (Joint Cumulative Distribution Function). Consider two rv. $X, Y, F_{XY} : \mathbb{R} \times \mathbb{R} \mapsto [0, 1]$

$$F_{XY}(x, y) = \Pr(X \leq x, Y \leq y) \quad (2.19)$$

$$0 \leq F_{XY}(x, y) \leq 1 \quad (2.20)$$

$$\lim_{x, y \rightarrow \infty} F_{XY}(x, y) = 1 \quad (2.21)$$

$$\lim_{x, y \rightarrow -\infty} F_{XY}(x, y) = 0 \quad (2.22)$$

Define 2.15 (Marginal Cumulative Distribution Functions).

$$F_X(x) = \lim_{y \rightarrow \infty} F_{XY}(x, y) \quad (2.23)$$

$$F_Y(y) = \lim_{x \rightarrow \infty} F_{XY}(x, y) \quad (2.24)$$

2.3.2 Probability Mass Functions (PMF)

Define 2.16 (Probability Mass Functions (PMF)). *When X is a discrete rv., $p_X : \mathbb{R} \mapsto [0, 1]$*

$$p_X(x) = \Pr(X = x) \quad (2.25)$$

$$0 \leq p_X(x) \leq 1 \quad (2.26)$$

$$\sum_x p_X(x) = 1 \quad (2.27)$$

$$\Pr(X \in A) = \sum_{x \in A} p_X(x) \quad (2.28)$$

Define 2.17 (Joint Probability Mass Function). *When X, Y are a discrete rv., $p_{XY} : \mathbb{R} \times \mathbb{R} \mapsto [0, 1]$*

$$p_{XY}(x, y) = \Pr(X = x, Y = y) \quad (2.29)$$

$$\sum_x \sum_y p_{XY}(x, y) = 1 \quad (2.30)$$

Define 2.18 (Marginal Probability Mass Function).

$$p_X(x) = \sum_y p_{XY}(x, y) \quad (2.31)$$

$$p_Y(y) = \sum_x p_{XY}(x, y) \quad (2.32)$$

2.3.3 Probability Density Functions (PDF)

Define 2.19 (Probability Density Functions (PDF)). *When X is a continuous rv.,*

$$p_X(x) = \frac{dF_X(x)}{dx} \quad (2.33)$$

$$\Pr(x \leq X \leq x + \Delta x) \approx p_X(x)\Delta x \quad (2.34)$$

$$p_X(x) \geq 0 \quad (2.35)$$

$$\int_{-\infty}^{\infty} p_X(x) dx = 1 \quad (2.36)$$

$$\Pr(X \in A) = \int_{x \in A} p_X(x) dx \quad (2.37)$$

Define 2.20 (Joint Probability Density Function). *When X, Y are continuous rv.,*

$$p_{XY}(x, y) = \frac{\partial^2 F_{XY}(x, y)}{\partial x \partial y} \quad (2.38)$$

Marginal Probability Density Function

$$p_{XY}(x, y) \geq 0, \forall x, y \quad (2.39)$$

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} p_{XY}(x, y) dx dy = 1 \quad (2.40)$$

$$\Pr((X, Y) \in A) = \iint_{(x,y) \in A} p_{XY}(x, y) dx dy \quad (2.41)$$

Define 2.21 (Marginal Probability Density Function).

$$p_X(x) = \int_{-\infty}^{\infty} p_{XY}(x, y) dy \quad (2.42)$$

$$p_Y(y) = \int_{-\infty}^{\infty} p_{XY}(x, y) dx \quad (2.43)$$

2.3.4 Transformations

let $Y = g(X)$, then

$$F_Y(y) = \Pr(Y \leq y) = \Pr(g(X) \leq y) = \int_{\{x: g(x) \leq y\}} p_X(x) dx \quad (2.44)$$

if g is monotonic, then

$$p_Y(y) = p_X(h(y)) \left| \frac{dh(y)}{dy} \right| \quad (2.45)$$

where $h = g^{-1}$

Proof.

$$\because \Pr(y \leq Y \leq y + dy) = p_Y(y) dy \quad (2.46)$$

$$\therefore \Pr(g(x) \leq Y \leq g(x) + dy) = p_Y(g(x)) dy \quad (2.47)$$

$$\therefore \Pr(x \leq X \leq x + dx) = p_X(x) dx \quad (2.48)$$

$$\therefore |p_Y(g(x)) dy| = |p_X(x) dx| \quad (2.49)$$

$$\therefore |p_Y(y) dy| = |p_X(h(y)) dx| \quad (2.50)$$

$$\therefore p_Y(y) = p_X(h(y)) \left| \frac{dx}{dy} \right| \quad (2.51)$$

□

Ex. $p_X(x) = \exp(-x), x > 0, Y = \log X$, sol. $F_Y(y), p_Y(y)$

Sol.

$$F_Y(y) = \Pr(Y \leq y) = \Pr(\log X \leq y) = \int_0^{\exp(y)} \exp(-x) dx = 1 - \exp(-\exp(y)) \quad (2.52)$$

$$p_Y(y) = \exp(-\exp(y)) \exp(y) \quad (2.53)$$

Ex. $X_j \sim Unif(0, 1), \forall j, \vec{Y} = U\vec{X}$, sol. $p(\vec{y})$	<i>Conditional Distribution</i>
<i>Sol.</i> eg., consider one dim. case, $Y = 2X$	
$X \sim Unif(0, 1) \quad (2.54)$	
$p_X(x) = 1\{0 \leq x \leq 1\} \quad (2.55)$	
Y is distr. uniformly in the int. $[0, 2]$	
$p_Y(y) = 1\{0 \leq y \leq 2\} \frac{1}{2} \quad (2.56)$	
n -dim. case <ul style="list-style-type: none"> • let $C_{\vec{x}} = [0, 1]^n$ be the n-dim. hypercube, $p(\vec{x}) = 1\{\vec{x} \in C_{\vec{x}}\}$ • let $C_{\vec{y}} = \{U\vec{x} : \vec{x} \in C_{\vec{x}}\} \subseteq \mathbb{R}^n$ to be the img. of the $C_{\vec{x}}$ under the mapping by U • then, the vol. of $C_{\vec{y}}$ is U, \vec{y} will be uniformly distr. in $C_{\vec{y}}$ • since it must integrate over $C_{\vec{y}}$ to 1 	
$p_{\vec{X}}(\vec{y}) = \frac{1\{\vec{y} \in C_{\vec{y}}\}}{ U } \quad (2.57)$	
$= 1\{\vec{y} \in C_{\vec{y}}\} U^{-1} \quad (2.58)$	
$= 1\{\vec{y} \in C_{\vec{y}}\} W \quad (2.59)$	
$= 1\{W\vec{y} \in C_{\vec{x}}\} W \quad (2.60)$	
$= p_{\vec{X}}(W\vec{y}) W \quad (2.61)$	

2.3.5 Conditional Distributions

Define 2.22 (Conditional Distribution). *What is the probability distribution over Y , when we know that X must take on a certain value x ?*

$$p_{Y|X}(y|x) = \frac{p_{XY}(x,y)}{p_X(x)} \quad (2.62)$$

assuming $p_X(x) \neq 0$

2.3.6 Chain Rule

$$p_{XY}(x,y) = p_{Y|X}(y|x)p_X(x) \quad (2.63)$$

$$\begin{aligned} p(x_1, x_2, \dots, x_n) &= p(x_n|x_1, x_2, \dots, x_{n-1})p(x_1, x_2, \dots, x_{n-1}) \\ &= p(x_n|x_1, x_2, \dots, x_{n-1})p(x_{n-1}|x_1, x_2, \dots, x_{n-2})p(x_1, x_2, \dots, x_{n-2}) \end{aligned} \quad (2.64) \quad (2.65)$$

Structured Probabilistic Model (Graphical Model)
Directed Graphical Model
Undirected Graphical Model clique

$$= \prod_{j=2}^n p(x_j|x_1, \dots, x_{j-1})p(x_1) \quad (2.66)$$

2.3.7 Structured Probabilistic Models

e.g., for three rv., a, b, c , the joint probability distr.

$$p(a, b, c) = p(a)p(b|a)p(c|b, a) \quad (2.67)$$

suppose a and c are independent given b , then

$$p(a, b, c) = p(a)p(b|a)p(c|b) \quad (2.68)$$

these factorizations can greatly reduce the number of parameters needed to describe the distribution

Define 2.23 (Structured Probabilistic Model (Graphical Model)). *We represent the factorization of a probability distribution with a graph, where each node in the graph corresponds to a rv., and an edge connecting two rv. means that the probability distribution is able to represent direct interactions between those two random variables.*

Define 2.24 (Directed Graphical Model). *Use graphs with directed edges, and they represent factorizations into conditional probability distributions, see Fig. 2.1 for an example.*

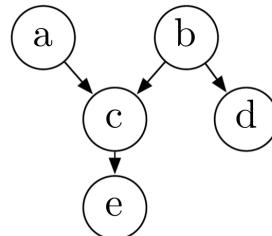


Figure 2.1: A directed graphical model. This corr. to a prob. density $p(a, b, c, d, e) = p(a)p(b)p(c|a, b)p(d|b)p(e|c)$

Define 2.25 (Undirected Graphical Model). *Use graphs with undirected edges. Any set of nodes that are all connected to each other in G is called a **clique**, each clique C_j in an undirected model is associated with a factor $\phi_j(C_j) \geq 0$. These factors are just functions, not probability distributions. The prob. distr. is prop. to the product of all of these factors, we divide by a normalizing const. Z*

$$p(x_1, x_2, \dots, x_n) = \frac{1}{Z} \prod_j \phi_j(C_j) \quad (2.69)$$

see Fig. 2.2 for an example.



Figure 2.2: An undirected graphical model. This corr. to a prob. density $p(a, b, c, d, e) = \frac{1}{Z} \phi_1(a, b, c) \phi_2(b, d) \phi_3(c, e)$

any prob. distr. can be described in both ways

2.3.8 Bayes's Rule

Theorem 2.3 (Bayes's Rule). *in the case of discrete rv. X and Y,*

$$p_{Y|X}(y|x) = \frac{p_{X|Y}(x|y)p_Y(y)}{p_X(x)} = \frac{p_{X|Y}(x|y)p_Y(y)}{\sum_y p_{X|Y}(x|y)p_Y(y)} \quad (2.70)$$

in the case of continuous rv. X and Y,

$$p_{Y|X}(y|x) = \frac{p_{X|Y}(x|y)p_Y(y)}{p_X(x)} = \frac{p_{X|Y}(x|y)p_Y(y)}{\int_{-\infty}^{\infty} p_{X|Y}(x|y)p_Y(y)dy} \quad (2.71)$$

2.3.9 Independence

Define 2.26 (Independence). *When two rv. X, Y have one of them, then have them all*

- X, Y are independent
- $F_{XY}(x, y) = F_X(x)F_Y(y), \forall x, y$
- $p_{XY}(x, y) = p_X(x)p_Y(y), \forall x, y$
- $p_{Y|X}(y|x) = p_Y(y), \forall y, p_X(x) \neq 0$

informally, X and Y are independent if “known” the value of X will never have any effect on Y, that is, you know all the information about (X, Y) by just knowing p(x) and p(y).

Likewise, We say than rv. X_1, X_2, \dots, X_n are independent if

$$p(x_1, x_2, \dots, x_n) = p(x_1)p(x_2) \cdots p(x_n) \quad (2.72)$$

Theorem 2.4. *If X and Y are independent then*

$$\Pr(X \in A, Y \in B) = \Pr(X \in A)\Pr(Y \in B), \forall A, B \subseteq \mathbb{R} \quad (2.73)$$

Theorem 2.5. *If X is independent of Y then any function of X is independent of any function of Y.*

*Expectation
mean*

Conditional Expectation **Define 2.27** (Expectation). Consider an arbitrary function $g : \mathbb{R} \mapsto \mathbb{R}$

$$\mathbb{E}[g(X)] = \sum_x g(x)p_X(x) \text{ discrete case} \quad (2.74)$$

$$\mathbb{E}[g(X)] = \int_{-\infty}^{\infty} g(x)p_X(x)dx \text{ continuous case} \quad (2.75)$$

Intuitively, it can be thought of as a “weighted average” of the values that $g(x)$ can take on for different values of x , where the weights are given by $p_X(x)$.

Especially, $\mathbb{E}[X]$ is also known as the **mean**

$$\mu = \mathbb{E}[X] = \sum_x xp_X(x) \text{ discrete case} \quad (2.76)$$

$$\mu = \mathbb{E}[X] = \int_{-\infty}^{\infty} xp_X(x)dx \text{ continuous case} \quad (2.77)$$

Consider an arbitrary function $g : \mathbb{R} \times \mathbb{R} \mapsto \mathbb{R}$,

$$\mathbb{E}[g(X, Y)] = \sum_x \sum_y g(x, y)p_{XY}(x, y) \text{ discrete case} \quad (2.78)$$

$$\mathbb{E}[g(X, Y)] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x, y)p_{XY}(x, y)dxdy \text{ continuous case} \quad (2.79)$$

Consider an arbitrary function $g : \mathbb{R}^n \mapsto \mathbb{R}$

$$\mathbb{E}[g(\vec{X})] = \int_{\mathbb{R}^n} g(x_1, x_2, \dots, x_n)p(x_1, x_2, \dots, x_n)dx_1 dx_2 \cdots dx_n \quad (2.80)$$

where \mathbb{R}^n is n consecutive integrations from $-\infty$ to ∞ .

Consider an arbitrary function $g : \mathbb{R}^n \mapsto \mathbb{R}^m$

$$\mathbb{E}[\vec{g}(\vec{X})] = \mathbb{E} \begin{bmatrix} g(\vec{X})_1 \\ g(\vec{X})_2 \\ \vdots \\ g(\vec{X})_n \end{bmatrix} = \begin{bmatrix} \mathbb{E}[g(\vec{X})_1] \\ \mathbb{E}[g(\vec{X})_2] \\ \vdots \\ \mathbb{E}[g(\vec{X})_n] \end{bmatrix} \quad (2.81)$$

Define 2.28 (Conditional Expectation).

$$\mathbb{E}[Y|X = x] = \int y p_{Y|X}(y|x)dy \quad (2.82)$$

$$\mathbb{E}[c] = c, \forall c \in \mathbb{R} \quad (2.83)$$

$$\mathbb{E}[1\{X \in A\}] = \Pr(X \in A) \quad (2.84)$$

if X_1, X_2, \dots, X_n are indep.

$$\mathbb{E}\left[\prod_{j=1}^n g_j(X_j)\right] = \prod_{j=1}^n \mathbb{E}[g_j(X_j)] \quad (2.85)$$

Theorem 2.6 (Linearity of Expectation).

$$\mathbb{E}[af(X) + bg(X)] = a\mathbb{E}[f(X)] + b\mathbb{E}[g(X)], \forall a, b \in \mathbb{R} \quad (2.86)$$

$$\mathbb{E}[af(X, Y) + bg(X, Y)] = a\mathbb{E}[f(X, Y)] + b\mathbb{E}[g(X, Y)], \forall a, b \in \mathbb{R} \quad (2.87)$$

Theorem 2.7 (Law of Total Expectation (Law of Iterated Expectation)).

$$\mathbb{E}[Y] = \mathbb{E}[\mathbb{E}[Y|X]] = \int \mathbb{E}[Y|X=x]p_X(x)dx \quad (2.88)$$

*Linearity of Expectation
Law of Total
Expectation (Law of
Iterated Expectation)
Variance
Law of Total Variance
Covariance
Pearson's Correlation
Coefficient*

2.3.11 Variance and Covariance

Define 2.29 (Variance). *It measures how concentrated the distribution of a rv. X is around its mean*

$$\sigma^2 = \text{Var}(X) \quad (2.89)$$

$$= \mathbb{E}[(X - \mu)^2] \quad (2.90)$$

$$= \mathbb{E}[X^2 - 2\mu X + \mu^2] \quad (2.91)$$

$$= \mathbb{E}[X^2] - 2\mu \mathbb{E}[X] + \mu^2 \quad (2.92)$$

$$= \mathbb{E}[X^2] - \mu^2 \quad (2.93)$$

$$\text{Var}(c) = 0, \forall c \in \mathbb{R} \quad (2.94)$$

$$\text{Var}(cg(X)) = c^2 \text{Var}(g(X)), \forall c \in \mathbb{R} \quad (2.95)$$

Theorem 2.8 (Law of Total Variance).

$$\text{Var}(Y) = \text{Var}(\mathbb{E}[Y|X]) + \mathbb{E}[\text{Var}(Y|X)] \quad (2.96)$$

Define 2.30 (Covariance). *It measures the how much two random variables are linearly related with each other*

$$\text{Cov}(X, Y) = \mathbb{E}[(X - \mu_X)(Y - \mu_Y)] \quad (2.97)$$

$$= \mathbb{E}[XY - \mu_X Y - \mu_Y X + \mu_X \mu_Y] \quad (2.98)$$

$$= \mathbb{E}[XY] - \mu_X \mathbb{E}[Y] - \mu_Y \mathbb{E}[X] + \mu_X \mu_Y \quad (2.99)$$

$$= \mathbb{E}[XY] - \mu_X \mu_Y \quad (2.100)$$

Define 2.31 (Pearson's Correlation Coefficient).

$$\rho(X, Y) = \frac{\text{Cov}(X, Y)}{\sigma_X \sigma_Y} \in [-1, 1] \quad (2.101)$$

- high $|\rho(X, Y)| \rightarrow X, Y$ change a lot and are both far from their respective means at the same time
- if $\rho(X, Y) > 0 \rightarrow$ the X, Y tend to change in the same direction

uncorrelated

Covariance Matrix

Moment Generating Function (MGF)

- if $\rho(X, Y) < 0 \rightarrow X, Y$ tend to change in opposite directions

- when $\rho(X, Y) = 0$, it is called **uncorrelated**

X, Y may or may not be independent

but if X, Y are indep \rightarrow they are uncorrelated

$$\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y) + 2 \text{Cov}(X, Y) \quad (2.102)$$

if X and Y are independent

$$\text{Cov}(X, Y) = 0 \rightarrow \text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y) \quad (2.103)$$

if X_1, X_2, \dots, X_n are indep.

$$\text{Var}\left(\sum_{j=1}^n c_j X_j\right) = \sum_{j=1}^n c_j^2 \text{Var}(X_j) \quad (2.104)$$

Define 2.32 (Covariance Matrix).

$$\Sigma_{ij} = \text{Cov}(X_i, X_j) = \mathbb{E}[X_i X_j] - \mu_i \mu_j \quad (2.105)$$

$$\Sigma = \mathbb{E}[\vec{X} \vec{X}^T] - \vec{\mu} \vec{\mu}^T \quad (2.106)$$

$$= \mathbb{E}[\vec{X} \vec{X}^T] - \mathbb{E}[\vec{X}] \mathbb{E}[\vec{X}]^T \quad (2.107)$$

$$= \mathbb{E}[(\vec{X} - \mathbb{E}[\vec{X}])(\vec{X} - \mathbb{E}[\vec{X}])^T] \quad (2.108)$$

$$\in \mathbb{S}_+^n \quad (2.109)$$

X_j 's are uncorrelated $\rightarrow \Sigma$ is a diag. mat.

2.3.12 Moment Generating Function (MGF)

Define 2.33 (Moment Generating Function (MGF)).

$$M_X(s) = \mathbb{E}[\exp(sX)] \quad (2.110)$$

$$M_X^{(n)}(s) \Big|_{s=0} = \mathbb{E}[X^n] \quad (2.111)$$

2.4 Information Theory

2.4.1 Intuition

it quantifies how much information is present in a signal

- a message saying “the sun rose this morning” is so uninformative as to be unnecessary to send
- but a message saying “there was a solar eclipse this morning” is very informative.

we want to formalize this intuition

- quantity of information is prop. to uncertainty of events
- events that are guaranteed to happen should have no information
- likely events should have low information
- less likely events should have higher information
- independent events should have additive information: eg., finding out a coin got tossed heads twice should convey twice as much information as finding out a coin got tossed heads once.

how to quantify the information?

- how many Yes / No questions you need to ask to get the information
- eg, you want to among team 1–32, which team won the championship
- suppose all team are equally likely to win

$$X \sim \text{Unif}(1, 32) \quad (2.112)$$

$$p_X(x) = \frac{1}{32}, 1 \leq x \leq 32 \quad (2.113)$$

Q1 : Is the winner in 1 – 16? Suppose Yes

Q2 : Is the winner in 1 – 8? Suppose No

Q3 : Is the winner in 9 – 12? Suppose Yes

Q4 : Is the winner in 9 – 10? Suppose No

Q5 : Is the winner is 11? Suppose No

→ : the winner is 12

- the quantify of this information is $\lg 32 = 5 = -\lg p(x)$
- when some team are more likely to win
 - we put the more likely teams in a group, other in a group
 - ask whether winner is in the likely group
 - repeat all that steps until find the winner
- the quantify of this information is less than 5

2.4.2 Several Quantity Measures

Define 2.34 (Self-Information).

$$I(x) = -\log p_X(x) \quad (2.114)$$

- it deals only with a single outcome
- with unit of *nats*, one nat is the amount of information gained by observing an event of probability $\frac{1}{e}$
- if use $I_X(x) = -\lg p_X(x)$, the unit is *bits*
- information measured in bits is just a rescaling of information measured in nats.

Define 2.35 (Shannon Entropy). *It is the expected amount of information in an event drawn from that distribution*

$$H(X) = \text{E}[I(x)] = -\text{E}[\log p(x)] = -\sum_x p(x) \log p(x) \quad (2.115)$$

Self-Information
Shannon Entropy

differential entropy
Conditional Entropy

Mutual Information

- when use $H = -E[\lg p(x)]$, it actually gives a lower bound on the number of bits needed in average to encode symbols drawn from this distr.
- distributions that are nearly deterministic (where the outcome is nearly certain) have low entropy; distributions that are closer to uniform (uncertain) have high entropy, see Fig. 2.3
- when X is continuous, the Shannon entropy is known as the **differential entropy**

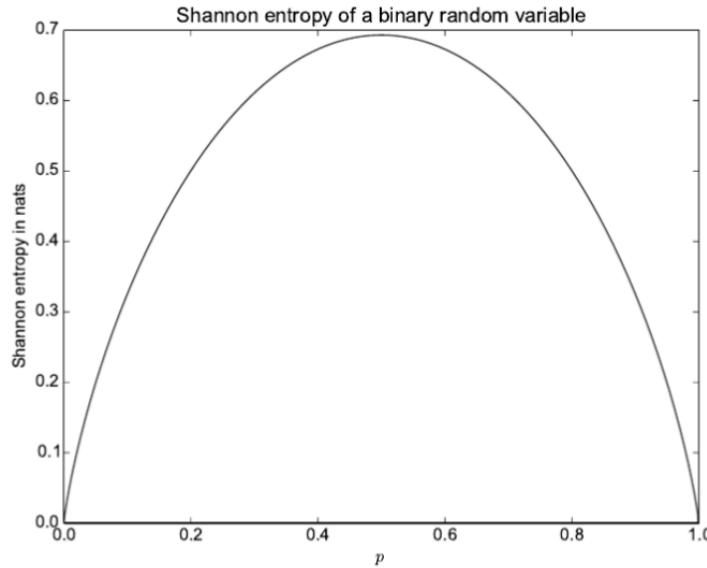


Figure 2.3: Shannon Entropy on a Bernoulli distr. with diff. para. p . When p is near 1 or 0, the rv. is nearly deterministic, when $p = 0.5$, the distribution is uniform over the two outcomes.

Define 2.36 (Conditional Entropy). *When know the information about Y , what is the Shannon Entropy of X*

$$H(X|Y) = -E[\log p(x|y)] = -\sum_x \sum_y p(x,y) \log p(x|y) \quad (2.116)$$

Theorem 2.9. *Entropy decreases when knowing Y , “=” holds when X, Y are independent*

$$H(X) \geq H(X|Y) \quad (2.117)$$

Define 2.37 (Mutual Information). *Amount information (uncertainty) decrease when knowing Y , it measures the correlation betw. two rv.*

$$I(X;Y) = H(X) - H(X|Y) \quad (2.118)$$

$$= \sum_x \sum_y p(x,y) \log p(x|y) - \sum_x p(x) \log p(x) \quad (2.119)$$

$$= \sum_x \sum_y p(x, y) \log p(x|y) - \sum_x \sum_y p(x, y) \log p(x) \quad (2.120)$$

*KL Divergence
Cross Entropy*

$$= \sum_x \sum_y p(x, y) (\log p(x|y) - \log p(x)) \quad (2.121)$$

Bernoulli

$$= \sum_x \sum_y p(x, y) \log \frac{p(x, y)}{p(x)p(y)} \quad (2.122)$$

$$= \mathbb{E}[\log \frac{p(x, y)}{p(x)p(y)}] \quad (2.123)$$

$$I(X; Y) = I(Y; X) \quad (2.124)$$

$$I(X; Y) \leq \min(H(X), H(Y)) \quad (2.125)$$

$$I(X; Y) = 0 \text{ if } X, Y \text{ are uncorrelated} \quad (2.126)$$

Define 2.38 (KL Divergence). *If we have two separate probability distributions $p(x)$ and $q(x)$ over the same rv. X , we can measure how different these two distributions by*

$$KL(p||q) = \mathbb{E}[\log \frac{p(x)}{q(x)}] \quad (2.127)$$

- when use $KL(p||q) = \mathbb{E}[\lg \frac{p(x)}{q(x)}]$, it is the extra amount of information needed to send a message containing symbols drawn from probability distribution p , when we use a code that was designed to minimize the length of messages drawn from probability distribution q
- $KL(p||q) = 0$ iff. p and q are the same distr.
- KL divergence is not symmetric $KL(p||q) \neq KL(q||p)$

Define 2.39 (Cross Entropy).

$$H(p, q) = H(p) + KL(p||q) = \mathbb{E}[\log q(x)] \quad (2.128)$$

see Fig. 2.4

when computing these quantities, often need to compute

$$\lim_{x \rightarrow 0} x \log x = - \lim_{x \rightarrow 0^+} \frac{-\log x}{\frac{1}{x}} = - \lim_{x \rightarrow 0^+} \frac{-\frac{1}{x}}{-\frac{1}{x^2}} = \lim_{x \rightarrow 0^+} x = 0 \quad (2.129)$$

2.5 Some Common Distributions

2.5.1 Discrete Distributions

Define 2.40 (Bernoulli). *One if a coin with heads probability ϕ comes up heads, zero otherwise.*

$$X \sim Ber(\phi), 0 \leq \phi \leq 1 \quad (2.130)$$

*Binomial
Geometric
Poisson*

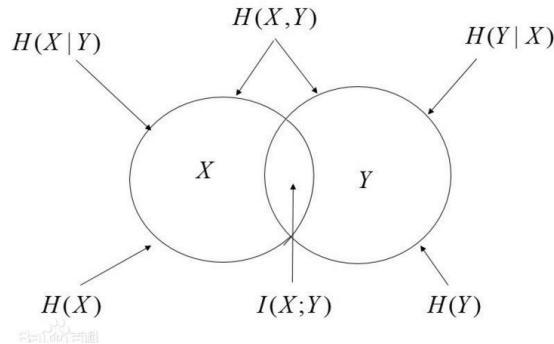


Figure 2.4: Several Quantity Measures

$$p(x) = \begin{cases} \phi & \text{if } x = 1 \\ 1 - \phi & \text{if } x = 0 \end{cases} \quad (2.131)$$

$$\rightarrow p(x) = \phi^x(1 - \phi)^{1-x}, x \in \{0, 1\} \quad (2.132)$$

$$\mu = \phi \quad (2.133)$$

$$\sigma^2 = \phi(1 - \phi) \quad (2.134)$$

$$H(p) = (\phi - 1) \log(1 - \phi) - \phi \log \phi \quad (2.135)$$

Define 2.41 (Binomial). *The number of heads in n independent flips of a coin with heads probability ϕ .*

$$X \sim Bin(n, \phi), 0 \leq \phi \leq 1 \quad (2.136)$$

$$p(x) = \binom{n}{x} \phi^x (1 - \phi)^{n-x}, x \in \{0, 1, \dots, n\} \quad (2.137)$$

$$\mu = n\phi \quad (2.138)$$

$$\sigma^2 = n\phi(1 - \phi) \quad (2.139)$$

Define 2.42 (Geometric). *The number of flips of a coin with heads probability p until the first heads.*

$$X \sim Geo(p), 0 \leq p \leq 1 \quad (2.140)$$

$$p(x) = p(1 - p)^{x-1} \quad (2.141)$$

$$\mu = \frac{1}{p} \quad (2.142)$$

$$\sigma^2 = \frac{1-p}{p^2} \quad (2.143)$$

Define 2.43 (Poisson). *A probability distribution over the nonnegative integers used for modeling the frequency of rare events.*

$$X \sim Poi(\lambda), \lambda > 0 \quad (2.144)$$

$$p(x) = \exp(-\lambda) \frac{\lambda^x}{x!} \quad (2.145) \quad \begin{matrix} \text{Multinomial} \\ \text{Uniform} \end{matrix}$$

$$\mu = \lambda \quad (2.146)$$

$$\sigma^2 = \lambda \quad (2.147)$$

$$M(s) = \exp(\lambda(\exp(s) - 1)) \quad (2.148)$$

if $X_1 \sim Poi(\lambda_1), X_2 \sim Poi(\lambda_2)$

$$Y = X_1 + X_2 \sim Poi(\lambda_1 + \lambda_2) \quad (2.149)$$

Define 2.44 (Multinomial). *This is the multivariate version of a Binomial.*

Consider drawing a ball from an urn with has balls with K different colors.

Let

$$\vec{p} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_K \end{bmatrix}, \sum_{k=1}^K p_k = 1 \quad (2.150)$$

where p_k is the probability of drawing color k with. Draw n balls from the urn (independently and with replacement) and let

$$\vec{X} = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_K \end{bmatrix}, \sum_{j=1}^n X_j = n \quad (2.151)$$

be the count of the number of balls of each color drawn with, then

$$\vec{X} \sim Mult(n, \vec{p}) \quad (2.152)$$

$$p(\vec{x}) = \binom{n}{x_1, \dots, x_K} p_1^{x_1} \cdots p_K^{x_K} \quad (2.153)$$

Multinoulli distributions are often used over categories of objects, so we do not usually assume that object 1 has numerical value 1, etc. For this reason, we do not usually need to compute the expectation or variance.

2.5.2 Continuous Distributions

Define 2.45 (Uniform). *Equal probability density to every value between a and b on the real line.*

$$X \sim Unif(a, b), a < b \quad (2.154)$$

$$p(x) = \frac{1\{a \leq x \leq b\}}{b - a} \quad (2.155)$$

$$\mu = \frac{a + b}{2} \quad (2.156)$$

$$\sigma^2 = \frac{(b - a)^2}{12} \quad (2.157)$$

Exponential Laplace (Double Exponential)

Gamma Gaussian (Normal) precision / inverse variance

Multivariate Gaussian

Define 2.46 (Exponential). *Decaying probability density over the nonnegative reals. It has a sharp density at $x = 0$*

$$X \sim Exp(\beta), \beta > 0 \quad (2.158)$$

$$p(x) = \frac{1}{\beta} \exp(-\frac{x}{\beta}) \text{ if } x \geq 0 \quad (2.159)$$

$$\mu = \beta \quad (2.160)$$

$$\sigma^2 = \beta^2 \quad (2.161)$$

Define 2.47 (Laplace (Double Exponential)). *Place a sharp peak of probability mass at an arbitrary point μ*

$$x \sim Lap(\beta, \mu) \quad (2.162)$$

$$p(x) = \frac{1}{2\beta} \exp(-\frac{|x - \mu|}{\beta}) \quad (2.163)$$

It can be written as the sum of two exponential distributions, with one flipped and both shifted to reposition the mode to $x = \mu$.

Define 2.48 (Gamma).

$$X \sim \Gamma(\alpha, \beta) \quad (2.164)$$

$$p(x) = \frac{1}{\Gamma(\alpha)\beta^\alpha} x^{\alpha-1} \exp(-\frac{x}{\beta}), x > 0 \quad (2.165)$$

$$\Gamma(\alpha) = \int_0^\infty \frac{1}{\beta^\alpha} x^{\alpha-1} \exp(-\frac{x}{\beta}) dx \quad (2.166)$$

$$Exp(\beta) = \Gamma(1, \beta) \quad (2.167)$$

Define 2.49 (Gaussian (Normal)).

$$X \sim N(\mu, \sigma^2) \quad (2.168)$$

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{(x - \mu)^2}{2\sigma^2}) \quad (2.169)$$

*A more computational efficient way is to use a parameter $\beta = \frac{1}{\sigma^2} \in \mathbb{R}_+$ as the **precision / inverse variance** of the distribution*

$$p(x) = \sqrt{\frac{\beta}{2\pi}} \exp(-\frac{1}{2}\beta(x - \mu)^2) \quad (2.170)$$

Define 2.50 (Multivariate Gaussian).

$$\vec{X} \sim N(\vec{\mu}, \Sigma), \Sigma \in \mathbb{S}_{++}^n \quad (2.171)$$

$$p(\vec{x}) = \frac{1}{\sqrt{2\pi^n} \sqrt{|\Sigma|}} \exp\left(-\frac{1}{2}(\vec{x} - \vec{\mu})^T \Sigma^{-1} (\vec{x} - \vec{\mu})\right) \quad (2.172)$$

$$M(\vec{s}) = \exp\left(\vec{\mu}^T \vec{s} + \frac{\vec{s}^T \Sigma \vec{s}}{2}\right) \quad (2.173)$$

precision matrix
 Central Limit Theorem
Chi-Squared
Dirac
Empirical

Compute Σ^{-1} is not efficient, use a **precision matrix** $B = \Sigma^{-1}$ instead

$$p(\vec{x}) = \sqrt{\frac{|B|}{(2\pi)^n}} \exp\left(-\frac{1}{2}(\vec{x} - \vec{\mu})^T B (\vec{x} - \vec{\mu})\right) \quad (2.174)$$

Gaussian rv. are extremely useful in machine learning and statistics

- common used to model noise
 - noise can be considered as the accumulation of a large number of small independent random perturbations
 - by the **Central Limit Theorem**, summations of independent random variables will tend to look Gaussian
- Gaussian distribution makes the fewest assumptions, it has highest entropy, so choosing to use it inserts the least amount of prior knowledge
- Gaussian distribution are convenient in math, often have simple closed form solutions

Define 2.51 (Chi-Squared).

$$X \sim \chi_K^2 \quad (2.175)$$

$$\text{if } X = \sum_{k=1}^K Z_k^2, Z_k \stackrel{\text{iid}}{\sim} N(0, 1) \quad (2.176)$$

some pdf. and cdf. are shown in Fig. 2.5

Define 2.52 (Dirac). All of the density in a probability distribution clusters around a single point

$$x \sim \delta(\mu) \quad (2.177)$$

$$p(x) = \delta(x - \mu) \quad (2.178)$$

The Dirac delta function $\delta(x)$ is defined such that it is zero-valued everywhere but 0, yet integrates to 1. By shifting by μ we obtain an infinitely narrow and infinitely high peak of probability density where $x = \mu$.

Define 2.53 (Empirical). It puts probability density $\frac{1}{n}$ on each of the n points x_1, \dots, x_n forming a given data set or collection of samples.

$$\hat{p}(x) = \frac{1}{n} \sum_{j=1}^n \delta(x - x_j) \quad (2.179)$$

for discrete variables

- an empirical distribution can be conceptualized as a Multinoulli distribution
- with probability associated to each possible input value is the empirical frequency of that value in the training set

*A Mixture Distribution
latent variable
Gaussian Mixture
Model
Level Curves,
Isocountours*

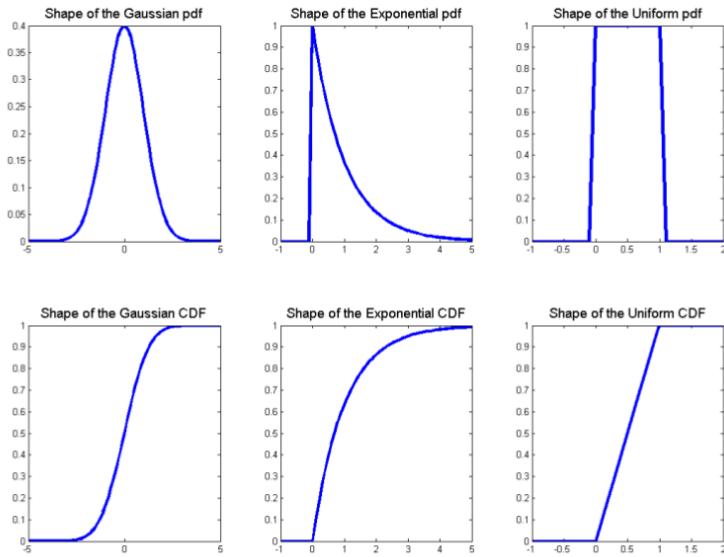


Figure 2.5: PDF and CDF of a couple of random variables.

2.5.3 Mixtures of Distributions

Define 2.54 (A Mixture Distribution). *On each trial, the choice of which component distribution generates the sample is determined by sampling a component identity from a multinoulli distribution. $p(z)$ is a Multinoulli distribution, z is the **latent variable** that we cannot observe directly.*

$$p(x) = \sum_j p(x|z=j)p(z=j) \quad (2.180)$$

Define 2.55 (Gaussian Mixture Model).

$$p(x|z=j) \sim N(\mu_j, \Sigma) \quad (2.181)$$

a Gaussian mixture model is a universal approximator of densities, in the sense that any smooth density can be approximated to a particular precision by a Gaussian mixture model with enough components.

2.5.4 Properties of Gaussian Distribution

Shape of Level Curves

Define 2.56 (Level Curves, Isocountours). *For a function $f : \mathbb{R}^n \mapsto \mathbb{R}$, a c -level curve is a set of form*

$$\{\vec{x} \in \mathbb{R}^n : f(\vec{x}) = c\}, c \in \mathbb{R} \quad (2.182)$$

A Case Study consider $\vec{X} \sim N(\vec{0}, \Sigma)$ case, with Σ is diagonal, $\vec{X} \in \mathbb{R}^2$

$$\frac{1}{2\pi\sigma_1\sigma_2} \exp\left(-\frac{1}{2\sigma_1^2}(x_1 - \mu_1)^2 - \frac{1}{2\sigma_2^2}(x_2 - \mu_2)^2\right) = c \quad (2.183)$$

$$-\frac{1}{2\sigma_1^2}(x_1 - \mu_1)^2 - \frac{1}{2\sigma_2^2}(x_2 - \mu_2)^2 = \log 2\pi c\sigma_1\sigma_2 \quad (2.184)$$

$$\frac{1}{2\sigma_1^2}(x_1 - \mu_1)^2 + \frac{1}{2\sigma_2^2}(x_2 - \mu_2)^2 = \log \frac{1}{\log 2\pi c\sigma_1\sigma_2} \quad (2.185)$$

$$\frac{(x_1 - \mu_1)^2}{2\sigma_1^2 \log \frac{1}{\log 2\pi c\sigma_1\sigma_2}} + \frac{(x_2 - \mu_2)^2}{2\sigma_2^2 \log \frac{1}{\log 2\pi c\sigma_1\sigma_2}} = 1 \quad (2.186)$$

$$\frac{(x_1 - \mu_1)^2}{r_1^2} + \frac{(x_2 - \mu_2)^2}{r_2^2} \stackrel{\text{def}}{=} 1 \quad (2.187)$$

$$(2.188)$$

this is axis-aligned ellipse

- center (μ_1, μ_2)
 - x_1 axis has length $2r_1$, x_2 axis has length $2r_2$
- how is r_1, r_2 ?
the max. of $p(x_1, x_2)$ is $\frac{1}{2\pi\sigma_1\sigma_2}$
let $c = \frac{1}{e} \frac{1}{2\pi\sigma_1\sigma_2}$, we can sol.

$$r_1 = \sqrt{2}\sigma_1 \quad (2.189)$$

$$r_2 = \sqrt{2}\sigma_2 \quad (2.190)$$

the axis length in the j -th dimension is prop. to σ_j

the smaller the variance of some random variable X_j , the more “tightly” peaked the Gaussian distribution in that dimension, and hence the smaller the radius r_j

General Case Instead of being an axis-aligned ellipse, the level curve turn out to be simply rotated ellipses

in the n -dimensional case, the geometrical structures is ellipsoids

- Σ large, Gaussian more spread out
- Σ smaller, Gaussian more compressed

Diagonal Σ

Theorem 2.10. If $\vec{X} \sim N(\vec{\mu}, \sigma^2 I)$, then X_j 's are indep., in Gaussian distr., uncorrelated \iff indep.

$$X_j \sim N(\mu_j, \sigma^2) \quad (2.191)$$

$$\frac{\|\vec{X}\|^2}{\sigma^2} = \frac{\vec{X}^T \vec{X}}{\sigma^2} \sim \chi_n^2\left(\frac{\vec{\mu}^T \vec{\mu}}{\sigma^2}\right) \quad (2.192)$$

General Σ

Theorem 2.11. If $\vec{X} \sim N(\vec{\mu}, \Sigma)$, then

$$\vec{X}^T \Sigma^{-1} \vec{X} \sim \chi_n^2(\vec{\mu}^T \Sigma^{-1} \vec{\mu}) \quad (2.193)$$

$$(\vec{X} - \vec{\mu})^T \Sigma^{-1} (\vec{X} - \vec{\mu}) \sim \chi_n^2(0) \quad (2.194)$$

Linear Transformation is Gaussian

Theorem 2.12. If $\vec{X} \sim N(\vec{\mu}, \Sigma)$, then

$$c\vec{X} \sim N(c\vec{\mu}, c^2\Sigma) \quad (2.195)$$

$$A\vec{X} + \vec{b} \sim N(A\vec{\mu} + \vec{b}, A\Sigma A^T) \quad (2.196)$$

Standard Gaussian Distribution

Theorem 2.13. If $\vec{X} \sim N(\vec{\mu}, \Sigma)$, then there exists matrix $A \in \mathbb{R}^{n \times n}$, such that $A^{-1}(\vec{X} - \vec{\mu}) \sim N(\vec{0}, I)$.

Sum of independent Gaussians is Gaussian

Theorem 2.14. If $\vec{X} \sim N(\vec{\mu}_1, \Sigma_1)$, $\vec{Y} \sim N(\vec{\mu}_2, \Sigma_2)$, and \vec{X}, \vec{Y} are indep., then $\vec{X} + \vec{Y} \sim N(\vec{\mu}_1 + \vec{\mu}_2, \Sigma_1 + \Sigma_2)$

- \vec{X} and \vec{Y} must be indep.
- density of $\vec{X} + \vec{Y}$ is not sum of $p(\vec{X})$ and $p(\vec{Y})$, it is the convolution of $p(\vec{X})$ and $p(\vec{Y})$

Marginal of a Joint Gaussian is Gaussian

Theorem 2.15. If $\vec{X} \sim N(\vec{\mu}, \Sigma)$, let

$$\vec{X} = \begin{bmatrix} \vec{X}_1 \\ \vec{X}_2 \end{bmatrix}, \vec{\mu} = \begin{bmatrix} \vec{\mu}_1 \\ \vec{\mu}_2 \end{bmatrix}, \Sigma = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \quad (2.197)$$

then

$$\vec{X}_1 \sim N(\vec{\mu}_1, \Sigma_{11}) \quad (2.198)$$

$$\vec{X}_2 \sim N(\vec{\mu}_2, \Sigma_{22}) \quad (2.199)$$

Conditional of a joint Gaussian is Gaussian

Theorem 2.16. if

*Delta Method
Point Estimator
sampling distribution*

$$\begin{bmatrix} \vec{X}_1 \\ \vec{X}_2 \end{bmatrix} \sim N\left(\begin{bmatrix} \vec{\mu}_1 \\ \vec{\mu}_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}\right) \quad (2.200)$$

then

$$\vec{X}_1 | \vec{X}_2 \sim N(\vec{\mu}_1 + \Sigma_{12}\Sigma_{22}^{-1}(\vec{X}_2 - \vec{\mu}_2), \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}) \quad (2.201)$$

$$\vec{X}_2 | \vec{X}_1 \sim N(\vec{\mu}_2 + \Sigma_{21}\Sigma_{11}^{-1}(\vec{X}_1 - \vec{\mu}_1), \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12}) \quad (2.202)$$

\vec{X}_1 and \vec{X}_2 are indep. iff. $\Sigma_{12} = 0$

Delta Method

Theorem 2.17 (Delta Method). if $X \sim N(\mu, \sigma^2)$, $Y = g(X)$, σ^2 is small, then approx.

$$Y \sim N(g(\mu), \sigma^2(g'(\mu))^2) \quad (2.203)$$

Proof.

$$Y = g(X) \approx g(\mu) + (X - \mu)g'(\mu) \quad (2.204)$$

$$\mathbb{E}[Y] \approx g(\mu) + (\mathbb{E}[X] - \mu)g'(\mu) = g(\mu) + 0 = g(\mu) \quad (2.205)$$

$$\text{Var}(Y) \approx \text{Var}(X)g'(\mu) = \sigma^2 g'(\mu) \quad (2.206)$$

□

2.6 Estimation

2.6.1 Estimators, Bias and Variance

Define 2.57 (Point Estimator). Let $\{\vec{x}^{(i)}\}_{i=1}^m$ be m iid. data points, a point estimator is any statistic function of the whole data that attempt to provide the single ‘best’ prediction of the parameter in the model

$$\hat{\theta} = g(\{\vec{x}^{(i)}\}_{i=1}^m) \quad (2.207)$$

We assume that the true parameter value $\vec{\theta}$ is fixed but unknown, while the point estimate $\hat{\theta}$ is a function of the data. Since the data is drawn from a **sampling distribution**, any function of the data is random. Therefore $\hat{\theta}$ is a random variable.

Function Estimation

covariates

Bias

unbiased

asymptotically unbiased

Sample Mean

Variance

Define 2.58 (Function Estimation). *Here we are trying to predict a variable \vec{y} given an input vector \vec{x} (also called the **covariates**). We consider that there is a function \vec{f} that describes the relationship betw. \vec{x} and \vec{y}*

$$\vec{y} = \vec{f}(\vec{x}) + \vec{\varepsilon} \quad (2.208)$$

where $\vec{\varepsilon}$ stands for the part of \vec{y} that is not predictable from \vec{x} .

In function estimation, we are interested in approximating \vec{f} with a model or estimate \vec{h} . Like point estimator, \vec{h} is a rv.

Define 2.59 (Bias). *Bias measures the expected deviation from the true value $\vec{\theta}$*

$$\text{Bias}(\hat{\vec{\theta}}) = E[\hat{\vec{\theta}}] - \vec{\theta} \quad (2.209)$$

where the expectation is over the data.

If $\text{Bias}(\hat{\vec{\theta}}) = \vec{0}$, it is said to be **unbiased**, ie., $E[\hat{\vec{\theta}}] = \vec{\theta}$.

If $\lim_{m \rightarrow \infty} \text{Bias}(\hat{\vec{\theta}}) = \vec{0}$, it is said to be **asymptotically unbiased**, ie., $\lim_{m \rightarrow \infty} E[\hat{\vec{\theta}}] = \vec{\theta}$.

Bias of a hypothesis is classifier makes its own modeling assumptions about the decision boundary.

Define 2.60 (Sample Mean). let $x^{(1)}, \dots, x^{(m)}$ be iid. rv.

$$\hat{\mu} = \frac{1}{m} \sum_{i=1}^m x^{(i)} \quad (2.210)$$

Theorem 2.18. *Sample mean is unbiased*

$$E[\hat{\mu}] = \mu \quad (2.211)$$

with $\mu = E[x^{(i)}]$

Proof.

$$E[\hat{\mu}] = \frac{1}{m} \sum_{i=1}^m E[x^{(i)}] \quad (2.212)$$

$$= \frac{1}{m} \sum_{i=1}^m \mu \quad (2.213)$$

$$= \mu \quad (2.214)$$

□

Define 2.61 (Variance). *The variance of an estimator provides a measure of how we would expect the estimate we compute from data to vary as we*

independently resample the dataset from the underlying data generating process.

$$\text{Var}(\hat{\theta}) = \mathbb{E}[\hat{\theta}^2] - (\mathbb{E}[\hat{\theta}])^2 \quad (2.215) \quad \text{Standard Error}$$

Variance of a hypothesis is random variations in the training data will lead to different models. Sample Variance

Theorem 2.19. The variance of the sample mean is

$$\text{Var}(\hat{\mu}) = \frac{\sigma^2}{m} \quad (2.216)$$

Proof.

$$\text{Var}(\hat{\mu}) = \mathbb{E}[\hat{\mu}^2] - \mathbb{E}[\hat{\mu}]^2 \quad (2.217)$$

$$= \mathbb{E}\left[\frac{1}{m^2} \sum_{i=1}^m x^{(i)2} + \frac{1}{m^2} \sum_{i=1}^m \sum_{j \neq i} x^{(i)}x^{(j)}\right] - \mu^2 \quad (2.218)$$

$$= \frac{1}{m}(\mu^2 + \sigma^2) + \frac{m-1}{m}\mu^2 - \mu^2 \quad (2.219)$$

$$= \frac{\sigma^2}{m} \quad (2.220)$$

□

Theorem 2.20.

$$\hat{\mu} \sim N(\mu, \frac{\sigma^2}{m}) \quad (2.221)$$

Proof. By the two prev. thm., and by the central limit theorem, the sample mean (which is summation of many iid. rv.) will be approximately distributed with a normal distribution. □

Define 2.62 (Standard Error).

$$\text{SE}(\hat{\theta}) = \sqrt{\text{Var}(\hat{\theta})} \quad (2.222)$$

Theorem 2.21. The standard error of the sample mean, sometimes simply called the standard error, is

$$\text{SE}(\hat{\mu}) = \frac{\sigma}{\sqrt{m}} \quad (2.223)$$

Define 2.63 (Sample Variance). let $x^{(1)}, \dots, x^{(m)}$ be iid. rv.

$$\hat{\sigma}^2 = \frac{1}{m-1} \sum_{j=1}^n (x^{(i)} - \hat{\mu})^2 \quad (2.224)$$

Theorem 2.22. *Sample variance is unbiased.*

$$\mathbb{E}[\hat{\sigma}^2] = \sigma^2 \quad (2.225)$$

Proof.

$$\mathbb{E}[\hat{\sigma}^2] = \mathbb{E}\left[\frac{1}{m-1} \sum_{i=1}^m (x^{(i)} - \hat{\mu})^2\right] \quad (2.226)$$

$$= \frac{1}{m-1} \sum_{i=1}^m \mathbb{E}[x^{(i)2} - 2\hat{\mu}x^{(i)} + \hat{\mu}^2] \quad (2.227)$$

$$= \frac{1}{m-1} \sum_{i=1}^m \mathbb{E}[x^{(i)2}] - \frac{2}{m} \sum_{j=1}^m \mathbb{E}[x^{(i)}x^{(j)}] + \frac{1}{m^2} \sum_{j=1}^m \sum_{k=1}^m \mathbb{E}[x^{(j)}x^{(k)}] \quad (2.228)$$

$$\begin{aligned} &= \frac{1}{m-1} \sum_{i=1}^m \mathbb{E}[x^{(i)2}] - \left(\frac{2}{m} \sum_{j \neq i} \mathbb{E}[x^{(i)}x^{(j)}] + \frac{2}{m} \mathbb{E}[x^{(i)2}]\right) \\ &\quad + \left(\frac{1}{m^2} \sum_{j=1}^m \sum_{k \neq j} \mathbb{E}[x^{(j)}x^{(k)}] + \frac{1}{m^2} \sum_{j=1}^m \mathbb{E}[x^{(j)2}]\right) \end{aligned} \quad (2.229)$$

$$\begin{aligned} &= \frac{1}{m-1} \sum_{i=1}^m \left(\left(1 - \frac{2}{m}\right) \mathbb{E}[x^{(i)2}] - \frac{2}{m} \sum_{j \neq i} \mathbb{E}[x^{(i)}] \mathbb{E}[x^{(j)}] \right. \\ &\quad \left. + \frac{1}{m^2} \sum_{j=1}^m \sum_{k \neq j} \mathbb{E}[x^{(j)}] \mathbb{E}[x^{(k)}] + \frac{1}{m^2} \sum_{j=1}^m \mathbb{E}[x^{(j)2}]\right) \end{aligned} \quad (2.230)$$

$$\begin{aligned} &= \frac{1}{m-1} \sum_{i=1}^m \left(\left(1 - \frac{2}{m}\right) (\mu^2 + \sigma^2) - \frac{2}{m} \sum_{j \neq i} \mu^2 \right. \\ &\quad \left. + \frac{1}{m^2} \sum_{j=1}^m \sum_{k \neq j} \mu^2 + \frac{1}{m^2} \sum_{j=1}^m (\mu^2 + \sigma^2) \right) \end{aligned} \quad (2.231)$$

$$\begin{aligned} &= \frac{1}{m-1} \sum_{i=1}^m \left(\left(1 - \frac{2}{m}\right) (\mu^2 + \sigma^2) - \frac{2(m-1)}{m} \mu^2 \right. \\ &\quad \left. + \frac{m(m-1)}{m^2} \mu^2 + \frac{m}{m^2} (\mu^2 + \sigma^2) \right) \end{aligned} \quad (2.232)$$

$$\begin{aligned} &= \frac{1}{m-1} \sum_{i=1}^m \left(\frac{m-1}{m} (\mu^2 + \sigma^2) - \frac{m-1}{m} \mu^2 \right) \\ &= \sigma^2 \end{aligned} \quad (2.233) \quad (2.234)$$

□

Ex. Let $x^{(i)} \stackrel{\text{iid}}{\sim} N(\mu, \sigma^2), \forall i \in \{1, 2, \dots, m\}$, compute the Variance of the sample variance $\hat{\sigma}^2$

$$\hat{\sigma}^2 = \frac{1}{m-1} \sum_{i=1}^m (x^{(i)} - \hat{\mu})^2 \quad (2.235)$$

Sol.

$$\therefore \frac{m-1}{\sigma^2} \hat{\sigma}^2 \sim \chi_{m-1}^2 \quad (2.236)$$

$$\therefore \text{Var}\left(\frac{m-1}{\sigma^2} \hat{\sigma}^2\right) = 2(m-1) = \frac{(m-1)^2}{\sigma^4} \text{Var}(\hat{\sigma}^2) \quad (2.237)$$

$$\therefore \text{Var}(\hat{\sigma}^2) = \frac{2\sigma^4}{m-1} \quad (2.238)$$

Note if use biased estimator of variance

$$\tilde{\sigma}^2 = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \hat{\mu})^2 = \frac{m-1}{m} \hat{\sigma}^2 \quad (2.239)$$

$$\therefore \text{Var}(\tilde{\sigma}^2) = \frac{(m-1)^2}{m^2} \text{Var}(\hat{\sigma}^2) = \frac{(m-1)^2}{m^2} \frac{2\sigma^4}{m-1} = \frac{2(m-1)\sigma^4}{m^2} < \text{Var}(\hat{\sigma}^2) \quad (2.240)$$

Mean Square Error

2.6.2 Trading off Bias and Variance and the Mean Squared Error

Define 2.64 (Mean Square Error). *MSE measures the overall expected deviation between the estimator and the true value of the parameter $\vec{\theta}$, it is one case of measuring generalization error.*

$$\text{MSE}(\hat{\vec{\theta}}) = E[(\vec{\theta} - \hat{\vec{\theta}})^2] \quad (2.241)$$

$$= E[\vec{\theta}^2 - 2\vec{\theta}\hat{\vec{\theta}} + \hat{\vec{\theta}}^2] \quad (2.242)$$

$$= \vec{\theta}^2 - 2\vec{\theta}E[\hat{\vec{\theta}}] + E[\hat{\vec{\theta}}^2] \quad (2.243)$$

$$= E[\hat{\vec{\theta}}]^2 - 2\vec{\theta}E[\hat{\vec{\theta}}] + \vec{\theta}^2 + E[\hat{\vec{\theta}}^2] - E[\hat{\vec{\theta}}]^2 \quad (2.244)$$

$$= (\vec{\theta} - E[\hat{\vec{\theta}}])^2 + E[\hat{\vec{\theta}}^2] - E[\hat{\vec{\theta}}]^2 \quad (2.245)$$

$$= \text{Bias}(\hat{\vec{\theta}})^2 + \text{Var}(\hat{\vec{\theta}}) \quad (2.246)$$

Desirable estimators are those with small MSE and these are estimators that manage to keep both their bias and variance somewhat in check.

*Consistency
Maximum Likelihood
Estimation*

Ex. Let $x^{(i)} \stackrel{\text{iid}}{\sim} N(\mu, \sigma^2), \forall i \in \{1, 2, \dots, m\}$, compute the MES of unbiased $\hat{\sigma}^2$ and biased $\tilde{\sigma}^2$

$$\hat{\sigma}^2 = \frac{1}{m-1} \sum_{i=1}^m (x^{(i)} - \hat{\mu})^2 \quad (2.247)$$

$$\tilde{\sigma}^2 = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \hat{\mu})^2 = \frac{m-1}{m} \hat{\sigma}^2 \quad (2.248)$$

Sol.

$$\text{MSE}(\hat{\sigma}^2) = \text{Bias}(\hat{\sigma}^2)^2 + \text{Var}(\hat{\sigma}^2) \quad (2.249)$$

$$= 0 + \frac{2\sigma^4}{m-1} \quad (2.250)$$

$$= \frac{2\sigma^4}{m-1} \quad (2.251)$$

$$\text{MSE}(\tilde{\sigma}^2) = (\mathbb{E}\left[\frac{m-1}{m}\hat{\sigma}^2\right] - \sigma^2)^2 + \frac{2(m-1)\sigma^4}{m^2} \quad (2.252)$$

$$= \left(\frac{m-1}{m} - 1\right)\sigma^2 + \frac{2(m-1)\sigma^4}{m^2} \quad (2.253)$$

$$= \left(\frac{-\sigma^2}{m}\right)^2 + \frac{2(m-1)\sigma^4}{m^2} \quad (2.254)$$

$$= \frac{2m-1}{m^2}\sigma^4 \quad (2.255)$$

Note

$$\text{MSE}(\hat{\sigma}^2) > \text{MSE}(\tilde{\sigma}^2) \quad (2.256)$$

This implies that despite incurring bias in the estimator $\tilde{\sigma}^2$, the resulting reduction in variance more than makes up for the difference, at least in a mean squared sense.

Define 2.65 (Consistency). *We want as the number of data points in our dataset increases, our point estimates converge to the true value of the parameter, thus ensures that the bias induced by the estimator is assured to diminish.*

$$\lim_{m \rightarrow \infty} \Pr(|\hat{\vec{\theta}} - \vec{\theta}| > \varepsilon) = 0, \forall \varepsilon > 0 \quad (2.257)$$

2.6.3 Maximum Likelihood Estimation

Theorem 2.23 (Maximum Likelihood Estimation). *Let $\vec{x}^{(1)}, \dots, \vec{x}^{(m)}$ drawn iid. from the true but unknown data generating distr. p . Let $\hat{p}(D; \vec{\theta})$ be a para. family of prob. distr. estimating the true prob. distr. p . Then maximum likelihood estimator for $\vec{\theta}$ is to identify the $\vec{\theta}$ that best fit the data.*

$$\vec{\theta}_{ML} = \arg \max_{\vec{\theta}} \hat{p}(D; \vec{\theta}) \quad (2.258)$$

$$= \arg \max_{\vec{\theta}} \prod_{i=1}^m \hat{p}(\vec{x}^{(i)}; \vec{\theta}) \quad // \text{ warning: numerical underflow} \quad (2.259)$$

$$= \arg \max_{\vec{\theta}} \sum_{i=1}^m \log \hat{p}(\vec{x}^{(i)}; \vec{\theta}) \quad (2.260)$$

$$= \arg \max_{\vec{\theta}} \frac{1}{m} \sum_{i=1}^m \log \hat{p}(\vec{x}^{(i)}; \vec{\theta}) \quad (2.261)$$

$$= \arg \max_{\vec{\theta}} \mathbb{E}_{\vec{x} \sim p_{Emp}} [\log \hat{p}(\vec{x}; \vec{\theta})] \quad (2.262)$$

One way to interpret MLE is to view it as minimizing the dissimilarity between the **empirical distribution** p_{Emp} and the model distribution \hat{p}

$$\vec{\theta}_{ML} = \arg \min_{\vec{\theta}} KL(p_{Emp} || \hat{p}) \quad (2.263)$$

$$= \arg \min_{\vec{\theta}} \mathbb{E}_{\vec{x} \sim p_{Emp}} [\log \frac{p_{Emp}}{\hat{p}}] \quad (2.264)$$

$$= \arg \min_{\vec{\theta}} \mathbb{E}_{\vec{x} \sim p_{Emp}} [\log p_{Emp}(\vec{x}; \vec{\theta}) - \log \hat{p}(\vec{x}; \vec{\theta})] \quad (2.265)$$

$$= \arg \min_{\vec{\theta}} - \mathbb{E}_{\vec{x} \sim p_{Emp}} [\log \hat{p}(\vec{x}; \vec{\theta})] \quad (2.266)$$

$$= \arg \max_{\vec{\theta}} \mathbb{E}_{\vec{x} \sim p_{Emp}} [\log \hat{p}(\vec{x}; \vec{\theta})] \quad (2.267)$$

$$(2.268)$$

Theorem 2.24 (Maximum Likelihood Estimation on Conditional Probability).

$$\vec{\theta}_{ML} = \arg \max_{\vec{\theta}} p(\vec{y}|D; \vec{\theta}) \quad (2.269)$$

$$= \arg \max_{\vec{\theta}} \prod_{i=1}^m p(y^{(i)} | \vec{x}^{(i)}; \vec{\theta}) \quad (2.270)$$

$$= \arg \max_{\vec{\theta}} \sum_{i=1}^m \log p(y^{(i)} | \vec{x}^{(i)}; \vec{\theta}) \quad (2.271)$$

$$(2.272)$$

2.6.4 Bayesian Statistics

Define 2.66 (Frequentist Statistics). *The frequentist perspective is that the true parameter value $\vec{\theta}$ is fixed but unknown, while the point estimate $\hat{\vec{\theta}}$ is a random variable.*

Define 2.67 (Bayesian Statistics). *The data is directly observed and so is not random, the true parameter $\vec{\theta}$ is unknown and uncertain and thus is*

prior probability distribution
posterior probability distribution
Maximum A Posterior (MAP) Point Estimate

represented as a random variable. The Bayesian uses probability to reflect degrees of certainty of states of knowledge.

Before observing the data, we represent our knowledge of $\vec{\theta}$ using the **prior probability distribution**, $p(\vec{\theta})$. Generally, the prior distribution is quite broad (ie., with high entropy) to reflect a high degree of uncertainty in the value of $\vec{\theta}$ before observing any data.

Now consider that we have a set of data samples $D = \{\vec{x}^{(i)}\}_{i=1}^m$, we can recover the effect of data on our belief about $\vec{\theta}$ by the **posterior probability distribution**

$$p(\vec{\theta}|D) = \frac{p(D|\vec{\theta})p(\vec{\theta})}{p(D)} \quad (2.273)$$

$$= \frac{\prod_{i=1}^m p(\vec{x}^{(i)}, \vec{\theta})p(\vec{\theta})}{\int_{\vec{\theta}} \prod_{i=1}^m p(\vec{x}^{(i)}, \vec{\theta})p(\vec{\theta})d\vec{\theta}} \quad (2.274)$$

If the data is at all informative about the value of $\vec{\theta}$, the posterior distribution $p(\vec{\theta}|D)$ will have less entropy (will be more ‘peaky’) than the prior $p(\vec{\theta})$.

use Bayesian to make (fully) future prediction

$$p(\vec{x}^{(m+1)}|D) = \int_{\vec{\theta}} p(\vec{x}^{(m+1)}, \vec{\theta}|D)d\vec{\theta} \quad (2.275)$$

$$= \int p(\vec{x}^{(m+1)}|\vec{\theta})p(\vec{\theta}|D)d\vec{\theta} \quad (2.276)$$

$$p(y|\vec{x}, D) = \int_{\vec{\theta}} p(y, \vec{\theta}|\vec{x}, D)d\vec{\theta} \quad (2.277)$$

$$= \int p(y|\vec{x}, \vec{\theta})p(\vec{\theta}|D)d\vec{\theta} \quad (2.278)$$

- our prediction is computed by taking an avg. wrt. the posterior $p(\vec{\theta}|D)$ over $\vec{\theta}$
 - in general, it is very difficult to compute since $\vec{\theta}$ is usually high-dim. diff. betw. MLE and Bayesian estimation
 - MLE make point estimate on $\vec{\theta}$, express uncertainty of $\hat{\vec{\theta}}$ by variance. Bayesian express uncertainty by making decision wrt. a full distr. over $\vec{\theta}$, which tends to protect well against overfitting
 - Bayesian prior distribution often expresses a preference on the parameter space, and it reduces the uncertainty (or entropy) in the posterior density, further reduction in overfitting as compared to ML estimation.
- in practice, we will instead approx. the posterior distr. $p(\vec{\theta}|D)$ with a single point estimate, but by allowing the prior to influence the choice of the point estimate.

Theorem 2.25 (Maximum A Posterior (MAP) Point Estimate). *The MAP estimate chooses the point of maximal posterior probability*

$$\vec{\theta}_{MAP} = \arg \max_{\vec{\theta}} p(\vec{\theta}|D) \quad (2.279)$$

$$= \arg \max_{\vec{\theta}} p(D|\vec{\theta})p(\vec{\theta}) \quad (2.280)$$

$$= \arg \max_{\vec{\theta}} \prod_{i=1}^m p(\vec{x}^{(i)}|\vec{\theta})p(\vec{\theta}) \quad (2.281)$$

$$= \arg \max_{\vec{\theta}} \left(\sum_{i=1}^m \log p(\vec{x}^{(i)}|\vec{\theta}) + \log p(\vec{\theta}) \right) \quad (2.282)$$

- the prior distr. $p(\vec{\theta})$ plays the same role as a regularizer as a term added to the objective function
- a common choice for $p(\vec{\theta})$ is to assume $\vec{\theta} \sim N(\vec{0}, \tau^2 I)$
- thus, $\vec{\theta}_{MAP}$ will have a smaller norm than $\vec{\theta}_{ML}$
- in this case, MAP estimate is less susceptible to overfitting than the ML estimate if the paras.
- however, it does so at the price of increased bias
- eg., Bayesian LogReg turns out to be an effective alg. for text classification, even though in text classification we usually have $n \gg m$

Ex. proof that $\|\vec{\theta}_{ML}\| \leq \|\vec{\theta}_{MAP}\|$

Proof. by contradiction, assume that $\|\vec{\theta}_{ML}\| < \|\vec{\theta}_{MAP}\|$

$$p(\vec{\theta}_{MAP}) = \frac{1}{\sqrt{2\pi}^{n+1} \sqrt{|\Sigma|}} \exp\left(-\frac{1}{2\tau^2} \vec{\theta}_{MAP}^T \vec{\theta}_{MAP}\right) \quad (2.283)$$

$$< \frac{1}{\sqrt{2\pi}^{n+1} \sqrt{|\Sigma|}} \exp\left(-\frac{1}{2\tau^2} \vec{\theta}_{ML}^T \vec{\theta}_{ML}\right) \quad (2.284)$$

$$= p(\vec{\theta}_{ML}) \quad (2.285)$$

$$\prod_{i=1}^m p(y^{(i)} | \vec{x}^{(i)}, \vec{\theta}_{MAP}) p(\vec{\theta}_{MAP}) < \prod_{i=1}^m p(y^{(i)} | \vec{x}^{(i)}, \vec{\theta}_{MAP}) p(\vec{\theta}_{ML}) \quad (2.286)$$

$$\leq \prod_{i=1}^m p(y^{(i)} | \vec{x}^{(i)}, \vec{\theta}_{ML}) p(\vec{\theta}_{ML}) \quad (2.287)$$

$$\text{// since } \vec{\theta}_{ML} = \arg \max_{\vec{\theta}} \prod_{i=1}^m p(y^{(i)} | \vec{x}^{(i)}, \vec{\theta}) \quad (2.288)$$

this is a contradiction, since

$$\vec{\theta}_{MAP} = \arg \max_{\vec{\theta}} \prod_{i=1}^m p(y^{(i)} | \vec{x}^{(i)}, \vec{\theta}) p(\vec{\theta}) \quad (2.289)$$

□

2.6.5 The EM Algorithm

suppose we have an estimation problem in which we have a training set $\{\vec{x}^{(i)}\}_{i=1}^m$ consisting of m iid. examples, we wish to fit the parameters $\vec{\theta}$

$$\vec{\theta}^* = \arg \max p(D; \vec{\theta}) \quad (2.290)$$

$$= \arg \max_{\vec{\theta}} \prod_{i=1}^m p(\vec{x}^{(i)}; \vec{\theta}) \quad (2.291)$$

$$= \arg \max_{\vec{\theta}} \sum_{i=1}^m \log p(\vec{x}^{(i)}; \vec{\theta}) \quad (2.292)$$

$$\text{// when explicitly MLE wrt. } \vec{\theta} \text{ is hard} \quad (2.293)$$

$$= \arg \max_{\vec{\theta}} \sum_{i=1}^m \log \sum_{\vec{z}^{(i)}} p(\vec{x}^{(i)}, \vec{z}^{(i)}; \vec{\theta}) \quad (2.294)$$

$$\text{// } \vec{z}^{(i)}: \text{latent random variables} \quad (2.295)$$

Define 2.68 (EM Algorithm). *When maximizing $l(\vec{\theta})$ explicitly might be difficult, the EM strategy will be to instead repeatedly construct a lower-bound on $l(E\text{-step})$, and then optimize that lower-bound (M-step).*

for each i , let q_i be some distribution over the \vec{z} 's

$$q_i(\vec{z}) \geq 0 \quad (2.296)$$

$$\sum_{\vec{z}} q_i(\vec{z}) = 1 \quad (2.297)$$

then

$$\sum_{i=1}^m \log \sum_{\vec{z}^{(i)}} p(\vec{x}^{(i)}, \vec{z}^{(i)}; \vec{\theta}) \quad (2.298)$$

$$= \sum_{i=1}^m \log \sum_{\vec{z}^{(i)}} q_i(\vec{z}^{(i)}) \frac{p(\vec{x}^{(i)}, \vec{z}^{(i)}; \vec{\theta})}{q_i(\vec{z}^{(i)})} \quad (2.299)$$

$$= \sum_{i=1}^m \log \mathbb{E}_{\vec{z}^{(i)} \sim q_i} \left[\frac{p(\vec{x}^{(i)}, \vec{z}^{(i)}; \vec{\theta})}{q_i(\vec{z}^{(i)})} \right] \quad (2.300)$$

$$\geq \sum_{i=1}^m \mathbb{E}_{\vec{z}^{(i)} \sim q_i} \left[\log \frac{p(\vec{x}^{(i)}, \vec{z}^{(i)}; \vec{\theta})}{q_i(\vec{z}^{(i)})} \right] \quad (2.301)$$

$$\text{// } f(s) = \log s \text{ is concave since } f''(s) = -\frac{1}{s^2} < 0 \quad (2.302)$$

$$\text{// by Jensen's Ineq., } f(\mathbb{E}[s]) \geq \mathbb{E}[f(s)] \quad (2.303)$$

$$= \sum_{i=1}^m \sum_{\vec{z}^{(i)}} q_i(\vec{z}^{(i)}) \log \frac{p(\vec{x}^{(i)}, \vec{z}^{(i)}; \vec{\theta})}{q_i(\vec{z}^{(i)})} \quad (2.304)$$

the lower bound holds true for any valid distr. q_i , how to choose q_i ?

- if we have some current guess $\vec{\theta}$ of the parameters, it seems natural to try to make the lower-bound tight at that value of $\vec{\theta}$
- ie., we'll make the inequality above hold with equality at our particular value of $\vec{\theta}$
- this enables us to prove that $l(\vec{\theta})$ increases monotonically with successive iterations of EM

if we want “=” holds true in Jensen's Ineq.,

$$\frac{p(\vec{x}^{(i)}, \vec{z}^{(i)}; \vec{\theta})}{q_i(\vec{z}^{(i)})} = \text{const} \quad (2.305)$$

$$\therefore q_i(\vec{z}^{(i)}) \propto p(\vec{x}^{(i)}, \vec{z}^{(i)}; \vec{\theta}) \quad (2.306)$$

$$\therefore \sum_{\vec{z}^{(i)}} q_i(\vec{z}^{(i)}) = 1 \quad (2.307)$$

$$\therefore q_i(\vec{z}^{(i)}) = \frac{p(\vec{x}^{(i)}, \vec{z}^{(i)}; \vec{\theta})}{\sum_{\vec{z}^{(i)}} p(\vec{x}^{(i)}, \vec{z}^{(i)}; \vec{\theta})} \quad (2.308)$$

coordinate ascent

$$= \frac{p(\vec{x}^{(i)}, \vec{z}^{(i)}; \vec{\theta})}{p(\vec{x}^{(i)}; \vec{\theta})} \quad (2.309)$$

Markov's Inequality

$$= p(\vec{z}^{(i)} | \vec{x}^{(i)}; \vec{\theta}) \quad (2.310)$$

Algorithm 1 EM Algorithm

idea: E-step gives a lower-bound on the log-likelihood l that we're trying to max. M-step max. that lower-bound to get a new $\vec{\theta}$.

loop until convergence

```

for  $i = 1 : m$ 
     $q_i(\vec{z}^{(i)}) \leftarrow p(\vec{z}^{(i)} | \vec{x}^{(i)}; \vec{\theta})$  // E-step
     $\vec{\theta} \leftarrow \arg \max_{\vec{\theta}} \sum_{i=1}^m \sum_{\vec{z}^{(i)}} q_i(\vec{z}^{(i)}) \log \frac{p(\vec{x}^{(i)}, \vec{z}^{(i)}; \vec{\theta})}{q_i(\vec{z}^{(i)})}$  // M-step

```

if we define

$$J(q_i, \vec{\theta}) = \sum_{i=1}^m \sum_{\vec{z}^{(i)}} q_i(\vec{z}^{(i)}) \log \frac{p(\vec{x}^{(i)}, \vec{z}^{(i)}; \vec{\theta})}{q_i(\vec{z}^{(i)})} \quad (2.311)$$

$$l(\vec{\theta}) \geq J(q_i, \vec{\theta}) \quad (2.312)$$

The EM can also be viewed a **coordinate ascent** on J

- E-step maximizes it wrt. q_i
- M-step maximizes it wrt. $\vec{\theta}$

Theorem 2.26. *EM causes the likelihood to converge monotonically.*

Proof. suppose $\vec{\theta}_t$ and $\vec{\theta}_{t+1}$ are the parameters from two successive iterations of EM. We will now prove that $l(\vec{\theta}_{t+1}) \geq l(\vec{\theta}_t)$, which shows EM always monotonically improves the log-likelihood.

$$l(\vec{\theta}_{t+1}) \geq \sum_{i=1}^m \sum_{\vec{z}^{(i)}} q_i(\vec{z}^{(i)}) \log \frac{p(\vec{x}^{(i)}, \vec{z}^{(i)}; \vec{\theta}_{t+1})}{q_i(\vec{z}^{(i)})} \quad // \text{by Jensen's Ineq} \quad (2.313)$$

$$\geq \sum_{i=1}^m \sum_{\vec{z}^{(i)}} q_i(\vec{z}^{(i)}) \log \frac{p(\vec{x}^{(i)}, \vec{z}^{(i)}; \vec{\theta}_t)}{q_i(\vec{z}^{(i)})} \quad // \vec{\theta}_{t+1} = \arg \max_{\vec{\theta}} (\bullet) \quad (2.314)$$

$$= l(\vec{\theta}_t) \quad // \text{at } \vec{\theta}_t \text{ “=} \text{ holds true} \quad (2.315)$$

□

2.7 Probability Inequalities

Theorem 2.27 (Markov's Inequality). *Let X be a non-negative rv. and suppose $\mu = \mathbb{E}[X]$ exists,*

$$\Pr(X > a) \leq \frac{\mu}{a}, \forall a > 0 \quad (2.316)$$

Proof.

Chebyshev's Inequality
Logistic Sigmoid
Function

(2.317) Softplus Function

$$\mathbb{E}[X] = \int_0^\infty xp(x)dx // by def. and X \geq 0$$

$$= \int_0^a xp(x)dx + \int_a^\infty xp(x)dx, \forall a > 0 \quad (2.318)$$

$$\geq \int_a^\infty xp(x)dx \quad (2.319)$$

$$\geq a \int_a^\infty p(x)dx \quad (2.320)$$

$$= a \Pr(X > a) \quad (2.321)$$

□

Theorem 2.28 (Chebyshev's Inequality). Let $\mu = \mathbb{E}[X], \sigma^2 = \text{Var}(X)$

$$\Pr(|X - \mu| \geq a) \leq \frac{\sigma^2}{a^2}, \forall a > 0 \quad (2.322)$$

or def. $Z = \frac{X - \mu}{\sigma}$

$$\Pr(|Z| \geq a) \leq \frac{1}{a^2}, \forall a > 0 \quad (2.323)$$

Proof.

$$\Pr(|X - \mu| \geq a) = \Pr((X - \mu)^2 \geq a^2) \quad (2.324)$$

$$\leq \frac{\mathbb{E}[(X - \mu)^2]}{a^2} // by Markov's Ineq. \quad (2.325)$$

$$= \frac{\sigma^2}{a^2} \quad (2.326)$$

□

2.8 Useful Properties of Common Functions

Define 2.69 (Logistic Sigmoid Function).

$$\sigma(s) = \frac{1}{1 + \exp(-s)} = \frac{\exp(s)}{1 + \exp(s)} \in [0, 1] \quad (2.327)$$

$$\sigma'(s) = \sigma(s)(1 - \sigma(s)) = \sigma(s)\sigma(-s) \quad (2.328)$$

$$\sigma^{-1}(s) = \log \frac{s}{1 - s}, s \in [0, 1] \quad (2.329)$$

Define 2.70 (Softplus Function).

$$\zeta(s) = \log(1 + \exp(s)) \in [0, \infty] \quad (2.330)$$

Random Functions

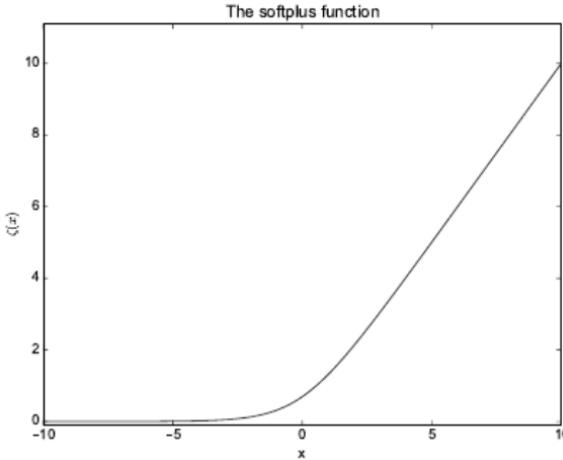


Figure 2.6: Softplus Function

it is the soft (smooth) version of

$$\max(0, s) \quad (2.331)$$

see Fig. 2.6

$$\log(\sigma(s)) = -\log(1 + \exp(-s)) = -\zeta(\sigma(-s)) \quad (2.332)$$

$$\zeta'(s) = \frac{\exp(s)}{1 + \exp(s)} = \sigma(s) \quad (2.333)$$

$$\int_{-\infty}^s \sigma(x) dx = \zeta(s) \quad (2.334)$$

$$\zeta^{-1}(s) = \log(\exp(s) - 1), x > 0 \quad (2.335)$$

$$\zeta(s) - \zeta(-s) = s \quad (2.336)$$

2.9 Gaussian Process

Gaussian distr.: model finite collections of real-valued variables

Gaussian process: model over random functions

Define 2.71 (Random Functions). *Let F be a set of functions deterministically mapping $\mathcal{X} \mapsto \mathcal{Y}$. A random function $f \in F$ is randomly drawn from F according to some prob. distr.*

2.9.1 Probability Distributions Over Functions with Finite Domains

let $\mathcal{X} = \{x_1, x_2, \dots, x_T\}$ be finite set of elements

F is all possible functions mapping $\mathcal{X} \mapsto \mathbb{R}$, eg. $f_1 \in F$, $f_1(x_1) = 11, f_1(x_2) = 26, \dots$

since \mathcal{X} has only T elements, we can repr. f completely as a T -dim. vec.

$$\vec{f} = \begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_T) \end{bmatrix} \quad (2.337)$$

one way to sefcify prob. distr. over functions is

$$\vec{f} \sim N(\vec{\mu}, \sigma^2 I) \quad (2.338)$$

$$p(f) = \prod_{t=1}^T \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(f(x_t) - \mu_t)^2\right) \quad (2.339)$$

2.9.2 Probability Distributions Over Functions with Infinite Domains

Define 2.72 (Stochasitic Process). *A collection of rv. $\{f(x) : x \in \mathcal{X}\}$, indexed by elements from some index set \mathcal{X} . Often $\mathcal{X} = \mathbb{R}$, the indices $x \in \mathcal{X}$ repr. times, the variables $f(x)$ repr. the temporal evolution of some quantity over time.*

Define 2.73 (Gauss Process). *A stochastic process st. any finite subcollection of rv. has a multivariate Gaussian distr. In particular, a collection of variables $\{f(x) : x \in \mathcal{X}\}$ is said to be drawn from a Gaussian process with **mean function** m and **covariance function** k if for any finite set of elements $x_1, x_2, \dots, x_T \in \mathcal{X}$, the associated finite set of rv. have distr.*

$$\begin{bmatrix} h(x_1) \\ \vdots \\ f(x_T) \end{bmatrix} \sim N\left(\begin{bmatrix} m(x_1) \\ \vdots \\ m(x_T) \end{bmatrix}, \begin{bmatrix} k(x_1, k_1) & \cdots & k(x_1, x_T) \\ \vdots & \ddots & \vdots \\ k(x_T, k_1) & \cdots & k(x_T, x_T) \end{bmatrix}\right) \quad (2.340)$$

We denote this as

$$f \sim GP(m, k) \quad (2.341)$$

For any $x, x' \in \mathcal{X}$

$$m(x) = \mathbb{E}[x] \quad (2.342)$$

$$k(x, x') = \mathbb{E}[(x - m(x))(x' - m(x'))] \quad (2.343)$$

squared exponential kernel (Gaussian kernel)

- can think of f drawn from a GP as an extremely high-dim. vec. drawn from an extremely high dim. multivariate Gaussian
- each dim. of the Gaussian corr. to an element $x \in \mathcal{X}$
how to choose m and k ?
- any real-val. function m is acceptable
- for k , the mat. $K \in \mathbb{R}^{T \times T}$ with $K_{ij} = k(x_i, x_j)$ is psd.
- Gaussian processes, therefore, are kernel-based probability distributions in the sense that any valid kernel function can be used as a covariance function

2.9.3 The Squared Exponential Kernel

consider a zero-mean Gaussian process

$$f \sim GP(0, k) \quad (2.344)$$

defined for $f : \mathbb{R} \mapsto \mathbb{R}$, choose k as **squared exponential kernel (Gaussian kernel)**

$$k(\vec{x}, \vec{x}') = \exp\left(-\frac{\|\vec{x} - \vec{x}'\|^2}{2\tau^2}\right) \quad (2.345)$$

- since GP is zero mean, we would expect that for the function vals. from the GP will tend to be distr. around 0
- if \vec{x} and \vec{x}' are nearby $\rightarrow f(\vec{x})$ and $f(\vec{x}')$ will tend to have high cov. $k(\vec{x}, \vec{x}') \approx 1$
- if \vec{x} and \vec{x}' are far apart $\rightarrow f(\vec{x})$ and $f(\vec{x}')$ will tend to have low cov. $k(\vec{x}, \vec{x}') \approx 0$

Why Use Gaussian Process

Gaussian process models allow one to quantify uncertainty in predictions resulting not just from intrinsic noise in the problem but also the errors in the parameter estimation procedure, many methods for model selection and hyperparameter selection in Bayesian methods are immediately applicable to Gaussian processes

like locally-weighted linear regression, Gaussian process regression is non-parametric and hence can model essentially arbitrary functions of the input points

Gaussian process regression models provide a natural way to introduce kernels into a regression modeling framework, by careful choice of kernels, Gaussian process regression models can sometimes take advantage of structure in the data

Gaussian process regression models, though perhaps somewhat tricky to understand conceptually, nonetheless lead to simple and straightforward linear algebra implementations

2.10 Markov Process

when data is repr. as a seq. of observations over time

- seq. of words that someone spoke
- part-of-speech tagging with a seq. of words

Markov model: reasoning about states over time

hidden Markov model: recover a series of states from a series of observations

Limited Horizon Assumption
Stationary Process Assumption
state transition matrix
 P

2.10.1 Markov Models

Markov Model

given a set of states

$$S = \{s_1, s_2, \dots, s_n\} \quad (2.346)$$

a series obeservation over time

$$\vec{x} \in S^T, x_t \in S, 1 \leq t \leq T \quad (2.347)$$

the observed states repr. the output of a random process over time

Two Markov Assumptions

Theorem 2.29 (Limited Horizon Assumption). *The probability of being in a state at time t depends only on the state at time $t - 1$. The state at time $t - 1$ repr. enough summary of the past to reasonably predict the future.*

$$p(x_t | x_1, \dots, x_{t-1}) = p(x_t | x_{t-1}) \quad (2.348)$$

Theorem 2.30 (Stationary Process Assumption). *The conditional distr. over next state given current state does not change over time.*

$$p(x_t | x_{t-1}) = p(x_t | x_1), \forall t \in \{2, 3, \dots, T\} \quad (2.349)$$

assume there is an init. state and init. obervation $x_0 = s_0$

- the prob. of seeing the first real state x_1 as $p(x_1 | x_0)$
- $p(x_t | x_1, \dots, x_{t-1}) = p(x_t | x_0, x_1, \dots, x_{t-1})$ because we define $x_0 = s_0$ for any state seq.

state transition matrix P

- $P \in \mathbb{R}^{n \times (n+1)}$
- P_{ij} : prob. of transitioning from state j to state i
eg., $S = \{\text{sun, cloud, rain}\}$

$$P = \begin{matrix} & \begin{matrix} s_0 & s_{\text{sun}} & s_{\text{cloud}} & s_{\text{rain}} \end{matrix} \\ \begin{matrix} s_{\text{sun}} \\ s_{\text{cloud}} \\ s_{\text{rain}} \end{matrix} & \begin{matrix} 0.33 & 0.8 & 0.2 & 0.1 \\ 0.33 & 0.1 & 0.6 & 0.2 \\ 0.33 & 0.1 & 0.2 & 0.7 \end{matrix} \end{matrix} \quad (2.350)$$

- a strong diag. in P means the states are self-correlated
- init. s_0 shows uniform probability of transitioning to each of the three states

Probability of a State Sequence

$$p(\vec{x}) = p(x_1, x_2, \dots, x_n) \quad (2.351)$$

$$= p(x_0, x_1, x_2, \dots, x_n) \quad (2.352)$$

$$= p(x_1|x_0)p(x_2|x_0, x_1) \cdots p(x_T|x_0, x_1, \dots, x_{T-1}) \quad (2.353)$$

$$\approx \prod_{t=1}^T p(x_t|x_{t-1}) \quad (2.354)$$

$$= P_{x_t, x_{t-1}} \quad (2.355)$$

Parameter Learning: MLE

find P to MLE the seq. of ob. \vec{x}

$$l(P) = \log p(\vec{x}) \quad (2.356)$$

$$= \log \prod_{t=1}^T P_{x_t, x_{t-1}} \quad (2.357)$$

$$= \sum_{t=1}^T \log P_{x_t, x_{t-1}} \quad (2.358)$$

$$= \sum_{i=1}^n \sum_{j=1}^n \sum_{t=1}^T \mathbb{1}\{x_t = s_i \wedge x_{t-1} = s_j\} \log P_{ij} \quad (2.359)$$

the opt. problem is

$$\min_P - \sum_{i=1}^n \sum_{j=1}^n \sum_{t=1}^T \mathbb{1}\{x_t = s_i \wedge x_{t-1} = s_j\} \log P_{ij} \quad (2.360)$$

$$\text{st. } \sum_{i=1}^n P_{ij} = 1, \forall j \quad (2.361)$$

$$P_{ij} \geq 0, \forall i, j \quad (2.362)$$

ignore the ≥ 0 constraints, the opt. sol. will produce $P_{ij} \geq 0$

$$L(P, \vec{\beta}) = - \sum_{i=1}^n \sum_{j=1}^n \sum_{t=1}^T \mathbb{1}\{x_t = s_i \wedge x_{t-1} = s_j\} \log P_{ij}$$

$$+ \sum_{j=1}^n \beta_j (\sum_{i=1}^n P_{ij} - 1) \quad (2.363)$$

$$\frac{\partial L}{\partial P_{ij}} = -\frac{\partial}{\partial P_{ij}} (\sum_{t=1}^T 1\{x_t = s_i \wedge x_{t-1} = s_j\} \log P_{ij}) + \beta_j \quad (2.364)$$

$$= -\sum_{t=1}^T 1\{x_t = s_i \wedge x_{t-1} = s_j\} \frac{1}{P_{ij}} + \beta_j \quad (2.365)$$

$$\stackrel{\text{set}}{=} 0 \quad (2.366)$$

$$P_{ij} = \frac{1}{\beta_j} \sum_{t=1}^T 1\{x_t = s_i \wedge x_{t-1} = s_j\} \quad (2.367)$$

substituting back

$$\frac{\partial L}{\partial \beta_j} = \frac{\partial}{\partial \beta_j} \sum_{j=1}^n \beta_j (\sum_{i=1}^n P_{ij} - 1) \quad (2.368)$$

$$= \sum_{i=1}^n P_{ij} - 1 \quad (2.369)$$

$$= \sum_{i=1}^n \frac{1}{\beta_j} \sum_{t=1}^T 1\{x_t = s_i \wedge x_{t-1} = s_j\} - 1 \quad (2.370)$$

$$\stackrel{\text{set}}{=} 0 \quad (2.371)$$

$$\beta_j = \sum_{i=1}^n \sum_{t=1}^T 1\{x_t = s_i \wedge x_{t-1} = s_j\} \quad (2.372)$$

$$= \sum_{t=1}^T 1\{x_{t-1} = s_j\} \quad (2.373)$$

substituting back

$$P_{ij} = \frac{\sum_{t=1}^T 1\{x_t = s_i \wedge x_{t-1} = s_j\}}{\sum_{t=1}^T 1\{x_{t-1} = s_j\}} = \frac{\#(s_j \rightarrow s_i)}{\#(s_j)} \quad (2.374)$$

2.10.2 Hidden Markov Model

Hidden Markov Model

consider again two examples

- seq. of words that someone spoke: you got sound seq. not the words generated it
- part-of-speech tagging with a seq. of words: words are observed but not pos. tag

Output Independence Assumption

in HMM, we don't get to observe the actual seq. of states,

$$\vec{x} \in S^T, x_t \in V, 1 \leq t \leq T \quad (2.375)$$

but we can only observe some outcome generated by each state

$$V = v_1, v_2, \dots, v_K \quad (2.376)$$

a series of observed outputs

$$\vec{y} \in V^T, y_t \in V, 1 \leq t \leq T \quad (2.377)$$

Assumption

Theorem 2.31 (Output Independence Assumption).

$$p(y_t = v_k | x_1, \dots, x_T, y_1, \dots, y_T) = p(y_t = v_k | x_t = s_i) \stackrel{\text{def}}{=} B_{ki} \quad (2.378)$$

$$B \in \mathbb{R}^{K \times n} \quad (2.379)$$

assume \vec{x} is generated by a Markov model parameterized by a state transition matrix P

Probability of an Observed Sequence: Forward Procedure

Probability of a Sequence of observations

$$p(\vec{y}) = \sum_{\vec{x}} p(\vec{y}, \vec{x}) \quad (2.380)$$

$$= \sum_{\vec{x}} p(\vec{y} | \vec{x}) p(\vec{x}) \quad (2.381)$$

$$\approx \sum_{\vec{x}} \prod_{t=1}^T p(y_t | x_t) \prod_{t=1}^T p(x_t | x_{t-1}) \quad (2.382)$$

$$= \sum_{\vec{x}} \prod_{t=1}^T B_{y_t, x_t} \prod_{t=1}^T A_{x_t, x_{t-1}} \quad (2.383)$$

- sum. over all possible \vec{x}
- x_t can take one of n possible vals.
- this sum directly require $O(n^T)$ op.

Dynamic Programming Technique def.

$$\alpha_j(t) = p(y_1, \dots, y_t, x_t = s_j) \quad (2.384)$$

it repr. the total prob. of all the obeservations up through time t and we are in state s_j at time t

$$p(\vec{y}) = p(y_1, y_2, \dots, y_T) \quad (2.385)$$

$$= \sum_{j=1}^n p(y_1, y_2, \dots, y_T, x_T = s_j) \quad (2.386)$$

$$= \sum_{j=1}^n \alpha_j(T) \quad (2.387)$$

use Forward Procedure to compute $\alpha_j(t)$

Algorithm 2 Forward Procedure for $\alpha_j(t)$

```

for  $i = 1 : n$  // base case
   $\alpha_i(1) = P_{i0}$ 
for  $t = 2 : T$  // recursion
  for  $i = 1 : n$ 
     $\alpha_i(t) = \sum_{j=1}^n \alpha_j(t-1) A_{ij} B_{ki}$ 

```

- compute complexity $O(nT)$

Backward Procedure : a similar alg. with

$$\beta_i(t) = p(y_{t+1}, y_{t+2}, \dots, y_T, x_t = s_i) \quad (2.388)$$

MLE: the Viterbi Algorithm

goal: given \vec{y} , what is the most probably \vec{x}

$$\vec{x} = \arg \max_{\vec{x}} p(\vec{x} | \vec{y}) \quad (2.389)$$

$$= \arg \max_{\vec{x}} p(\vec{y} | \vec{x}) p(\vec{x}) \quad (2.390)$$

$$= \arg \max_{\vec{x}} \prod_{t=1}^T B_{y_t, x_t} \prod_{t=1}^T A_{x_t, x_{t-1}} \quad (2.391)$$

naive approach

- try every \vec{x} and take the one with highest prob.

- $O(n^T)$ possibles of \vec{x}

dynamic programming sol. like Forward Algorithm

- if replace $\arg \max$ with $\sum_{\vec{x}}$, these two are the same

- **Viterbi Algorithm** is just like the Forward Procedure

Forward Procedure tracking the total prob. of generating the observations seen so far

Viterbi only track the max. prob. and record its corresponding state seq.

Backward Procedure
Viterbi Algorithm

Parameter Learning: EM for HMM

Algorithm 3 Naive EM for HMM

repeat until convergence

for every possible $\vec{x} \in S^T$

$$q(\vec{x}) = p(\vec{x}|\vec{y}) \quad // E \text{ step}$$

$$P, B = \arg \max_{P, B} \sum_{\vec{x}} q(\vec{x}) \log \frac{p(\vec{x}, \vec{y})}{p(\vec{x})}$$

$$\text{st. } \sum_{i=1}^n P_{ij} = 1, \forall j$$

$$P_{ij} \geq 0, \forall i, j$$

$$\sum_{k=1}^m B_{ki} = 1, \forall i$$

$$B_{ki} \geq 0, \forall k, i$$

- there are n^T possibles of \vec{x}
- can make use of Forward and Backward Procedure to compute tractably

$$\begin{aligned} P, B &= \arg \max_{P, B} \sum_{\vec{x}} q(\vec{x}) \log \frac{p(\vec{x}, \vec{y})}{p(\vec{x})} \\ &= \arg \max_{P, B} \sum_{\vec{x}} q(\vec{x}) \log p(\vec{x}, \vec{y}) \quad // \text{denominator does not dep. on } P, B \\ &= \arg \max_{P, B} \sum_{\vec{x}} q(\vec{x}) \log \prod_{t=1}^T p(y_t|x_t) \prod_{t=1}^T p(x_t|x_{t-1}) \\ &= \arg \max_{P, B} \sum_{\vec{x}} q(\vec{x}) \sum_{t=1}^T (\log B_{y_t, x_t} + \log P_{x_t, x_{t-1}}) \\ &= \arg \max_{P, B} \sum_{\vec{x}} q(\vec{x}) \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^K \sum_{t=1}^T (1\{y_t = v_k \wedge x_t = s_i\} \log B_{y_t, x_t} + 1\{x_t = s_i \wedge y_t = s_j\} \log P_{x_t, x_{t-1}}) \end{aligned}$$

like MLE for MM, ignore the ≥ 0 constraints, use Lagrangian to get

$$P_{ij} = \frac{\sum_{\vec{x}} q(\vec{x}) \sum_{t=1}^T 1\{x_t = s_i \wedge x_{t-1} = s_j\}}{\sum_{\vec{x}} q(\vec{x}) \sum_{t=1}^T 1\{x_{t-1} = s_j\}} \quad (2.397)$$

$$B_{ki} = \frac{\sum_{\vec{x}} q(\vec{x}) \sum_{t=1}^T 1\{y_t = v_k \wedge x_t = s_i\}}{\sum_{\vec{x}} q(\vec{x}) \sum_{t=1}^T 1\{x_t = s_i\}} \quad (2.398)$$

- sum is over all possible \vec{x}
- make use of Forward and Backward Procedure

$$\sum_{\vec{x}} q(\vec{x}) \sum_{t=1}^T 1\{x_t = s_i \wedge x_{t-1} = s_j\} \quad (2.399)$$

$$= \sum_{t=1}^T \sum_{\vec{x}} 1\{x_t = s_i \wedge x_{t-1} = s_j\} q(\vec{x}) \quad (2.400)$$

$$= \sum_{t=1}^T \sum_{\vec{x}} 1\{x_t = s_i \wedge x_{t-1} = s_j\} q(\vec{x}) \quad (2.401)$$

$$= \frac{1}{p(\vec{y})} \sum_{t=1}^T \sum_{\vec{x}} 1\{x_t = s_i \wedge x_{t-1} = s_j\} p(\vec{x}, \vec{y}) \quad (2.402)$$

$$= \frac{1}{p(\vec{y})} \sum_{t=1}^{T-1} p(\vec{y}, x_t = s_j, x_{t+1} = s_i) P_{ij} B_{y_t, i} \quad (2.403)$$

$$= \frac{1}{p(\vec{y})} \sum_{t=1}^{T-1} p(y_1, \dots, y_t, x_t = s_j) P_{ij} B_{y_t, i} p(y_{t+1}, \dots, y_T, x_{t+1} = s_i) \quad (2.404)$$

$$= \frac{1}{p(\vec{y})} \sum_{t=1}^{T-1} \alpha_j(t) P_{ij} B_{y_t, i} \beta_i(t+1) \quad (2.405)$$

$$\sum_{\vec{x}} q(\vec{x}) \sum_{t=1}^T 1\{x_{t-1} = s_j\} \quad (2.406)$$

$$= \sum_{i=1}^n \sum_{\vec{x}} q(\vec{x}) \sum_{t=1}^T 1\{x_{t-1} = s_j \wedge x_t = s_i\} \quad (2.407)$$

$$= \frac{1}{p(\vec{y})} \sum_{i=1}^n \sum_{t=1}^{T-1} \alpha_j(t) P_{ij} B_{y_t, i} \beta_i(t+1) \quad (2.408)$$

$$\sum_{\vec{x}} q(\vec{x}) \sum_{t=1}^T 1\{y_t = v_k \wedge x_t = s_i\} \quad (2.409)$$

$$= \frac{1}{p(\vec{y})} \sum_{t=1}^T \sum_{\vec{x}} 1\{y_t = v_k \wedge x_t = s_i\} p(\vec{x}, \vec{y}) \quad (2.410)$$

$$= \frac{1}{p(\vec{y})} \sum_{j=1}^n \sum_{t=1}^T \sum_{\vec{x}} 1\{y_t = v_k \wedge x_t = s_i \wedge x_{t-1} = s_j\} p(\vec{x}, \vec{y}) \quad (2.411)$$

$$= \frac{1}{p(\vec{y})} \sum_{j=1}^n \sum_{t=1}^{T-1} 1\{y_t = v_k\} \alpha_j(t) P_{ij} B_{y_t, i} \beta_i(t+1) \quad (2.412)$$

$$\sum_{\vec{x}} q(\vec{x}) \sum_{t=1}^T 1\{x_t = s_i\} \quad (2.413)$$

$$= \frac{1}{p(\vec{y})} \sum_{j=1}^n \sum_{t=1}^T \sum_{\vec{x}} 1\{x_t = s_i \wedge x_{t-1} = s_j\} p(\vec{x}, \vec{y}) \quad (2.414)$$

$$= \frac{1}{p(\vec{y})} \sum_{j=1}^n \sum_{t=1}^{T-1} \alpha_j(t) P_{ij} B_{y_t, i} \beta_i(t+1) \quad (2.415)$$

define

$$\gamma_t(i, j) = \alpha_j(t) P_{ij} B_{y_t, i} \beta_i(t+1) \quad (2.416)$$

comb. these expressions, we have MLE

Algorithm 4 Baum-Welch Algorithm

init. P, B as random valid probability matrices

repeat until convergence

// E-step

run Forward and Backward Algorithms to compute α, β

$$\gamma_t(i, j) = \alpha_j(t) P_{ij} B_{y_t, i} \beta_i(t+1)$$

// M-step

$$P_{ij} = \frac{\sum_{t=1}^{T-1} \gamma_t(i, j)}{\sum_{i=1}^n \sum_{t=1}^{T-1} \gamma_t(i, j)} = \frac{\#(s_j \rightarrow s_i)}{\#(s_j)}$$

$$B_{ki} = \frac{\sum_{j=1}^n \sum_{t=1}^{T-1} \mathbb{1}\{y_t=v_k\} \gamma_t(i, j)}{\sum_{j=1}^n \sum_{t=1}^{T-1} \gamma_t(i, j)} = \frac{\#(s_i \rightarrow v_k)}{\#(s_i)}$$

- in E-step we compute $\gamma_t(i, j)$ as a sufficient statistics of $q(\vec{x})$, which is prop. to the prob. of transitioning betw. state s_j and s_i at time t given all ob. \vec{y}
- para. learning for HMM is non-convex, multiple runs might be better
- usually smooth P, B st. no tranition or emission is assigned 0 prob.

*Rounding Error
Underflow
Overflow*

Chapter 3

Numerical Computation

Note: some of the material here (including some of the figures) is based on [?]

3.1 Underflow and Overflow

Define 3.1 (Rounding Error). *Approximation error when we represent the real number in the computer with a finite number of bit*

- it is problematic, especially when it compounds across many operations
 - it can cause algorithms that work in theory to fail in practice if they are not designed to minimize the accumulation of rounding error
- there are two forms rounding error

3.1.1 Underflow

Define 3.2 (Underflow). *Situation when numbers near zero are rounded to zero*

many functions behave qualitatively differently when their argument is zero rather than a small positive number

- division by zero
- take the logarithm of zero

3.1.2 Overflow

Define 3.3 (Overflow). *Situation when numbers with large magnitude are approx. as ∞ or $-\infty$*

Conditioning

Ex. Design implementations to deal with underflow and overflow when using softmax function

$$h(\vec{x})_k = \frac{\exp(\vec{\theta}_k^T \vec{x})}{\sum_{j=1}^K \exp(\vec{\theta}_j^T \vec{x})} \quad (3.1)$$

Sol def.

$$h(\vec{x})_k = \frac{\exp(\vec{\theta}_k^T \vec{x})}{\sum_{j=1}^K \exp(\vec{\theta}_j^T \vec{x})} = \frac{\exp(s_k)}{\sum_{j=1}^K \exp(s_j)} \quad (3.2)$$

consider

$$s_j = c, \forall j \quad (3.3)$$

$$\therefore h(\vec{x})_k = \frac{1}{K} \quad (3.4)$$

Numerically, this may not occur

- when c is very negative
 - $\exp(c)$ underflow
 - $\exp(c) = 0$
 - $h(\vec{x})_k = \frac{0}{0}$
- when c is very positive
 - $\exp(c)$ overflow
 - $\exp(c) = \infty$
 - $h(\vec{x})_k = \frac{\infty}{\infty}$

both of these difficulties can be resolved by instead evaluating

$$h(\vec{x})_k = \frac{\exp(s_k)}{\sum_{j=1}^K \exp(s_j)} \quad (3.5)$$

$$= \frac{\exp(s_k) \cdot \max_i \exp(-s_i)}{\sum_{j=1}^K \exp(s_j) \cdot \max_i \exp(-s_i)} \quad (3.6)$$

$$= \frac{\exp(s_k - \max_i s_i)}{\sum_{j=1}^K \exp(s_j - \max_i s_i)} \quad (3.7)$$

- subtracting $\max_i s_i$ results in the largest argument to \exp being 0, which rules out the possibility of overflow
- likewise, at least one term in the denominator has a value of 1, which rules out the possibility of underflow in the denominator leading to a division by zero

3.2 Poor Conditioning

Define 3.4 (Conditioning). Consider $\vec{f}(\vec{\theta})$, when $\vec{\theta}$ has small changes $\Delta\vec{\theta}$, how is $\vec{f}(\vec{\theta} + \Delta\vec{\theta}) - \vec{f}(\vec{\theta})$

when $\Delta\vec{\theta}$ is very small, but $\vec{f}(\vec{\theta} + \Delta\vec{\theta}) - \vec{f}(\vec{\theta})$ is large, it will be problematic for scientific computing because rounding error in the inputs $\vec{\theta}$ can result in large changes in the output

Condition Number
ill-conditioning
gradient descent
Critical Point / Stationary Point

Define 3.5 (Condition Number). *When $A \in \mathbb{R}^{n \times n}$ has an eigendecomposition, its condition number is*

$$\max_{i,j} \frac{|\lambda_i|}{|\lambda_j|} = \frac{\max_i |\lambda_i|}{\min_j |\lambda_j|} \quad (3.8)$$

when condition number is large, matrix inversion $A^{-1}\vec{x}$ is particularly sensitive to error in the input

- this is an intrinsic property of the matrix itself, not the result of rounding error during matrix inversion
- poorly conditioned matrices amplify pre-existing errors
- **ill-conditioning** (in that case of the linear system matrix) yields very slow convergence of the iterative algorithm solving a linear system

3.3 Gradient-Based Optimization

goal:

$$\vec{\theta}^* = \arg \min_{\vec{\theta}} f(\vec{\theta}) \quad (3.9)$$

3.3.1 Gradient-Based Method Intuition

consider one dim. case:

$$\theta^* = \arg \min_{\theta} f(\theta) \quad (3.10)$$

see Fig. 3.1, for small enough $\Delta\theta$

- $f(\theta - \Delta\theta) < f(\theta)$ if $\frac{df(\theta)}{d\theta} > 0$
- $f(\theta + \Delta\theta) < f(\theta)$ if $\frac{df(\theta)}{d\theta} < 0$
- thus, we can reduce $f(\theta)$ by moving θ in small steps with opposite sign of the derivative, this is the **gradient descent** technique, ie.,

$$f\left(\theta - \Delta\theta \operatorname{sign}\left(\frac{df(\theta)}{d\theta}\right)\right) < f(\theta) \quad (3.11)$$

Define 3.6 (Critical Point / Stationary Point). *Point where $\nabla_{\vec{\theta}} f(\theta) = \vec{0}$, the derivative provides no information about which direction to move. Such point can a local minimum, a local maximum, or a saddle point (neither maxima nor minima).*

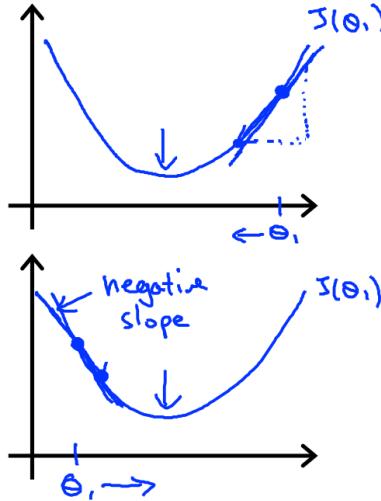


Figure 3.1: Gradient-Based Method Intuition

in the context of deep learning, $f(\vec{\theta})$ may have many local minima that are global minimum, and many saddle points surrounded by very flat regions, we therefore usually settle for finding a value of f that is very low, but not necessarily a global minimum

In some cases, we can directly get the critical point by

$$\nabla_{\vec{\theta}} f(\vec{\theta}) \stackrel{\text{set}}{=} \vec{0} \quad (3.12)$$

3.3.2 Gradient Descent

when we cannot get $\vec{\theta}^*$ by $\nabla_{\vec{\theta}} f(\vec{\theta}) \stackrel{\text{set}}{=} \vec{0}$ directly, we iteratively approach to the critical point

Iterative Optimization

to minimize $f(\vec{\theta})$, we would like to find the direction \vec{v} in which f decreases the fastest

- direction \vec{v} : (assumed) of unit length: $\|\vec{v}\|_2 = 1$, this is the direction of f to get the ball roll downhill
 - step size α : (assumed) pos.: $\alpha > 0$
- a greedy approach

$$\min_{\|\vec{v}\|=1} f(\vec{\theta} + \alpha \vec{v}) \quad (3.13)$$

this is non-lin. opt. with constraints

Linear Approximation

local approx. by lin. formula makes problem easier if α really small (Taylor expansion)

$$\vec{v}^* = \arg \min_{\|\vec{\theta}\|_2=1} f(\vec{\theta} + \alpha \vec{v}) \quad (3.14)$$

$$\approx \arg \min_{\|\vec{\theta}\|_2=1} f(\vec{\theta}) + \alpha \vec{v}^T \nabla f(\vec{\theta}) \quad (3.15)$$

$$= \arg \min_{\|\vec{\theta}\|_2=1} f(\vec{\theta}) + \alpha \|\vec{\theta}\| \|\nabla f(\vec{\theta})\| \cos(\vec{\theta}, \nabla f(\vec{\theta})) \quad (3.16)$$

$$= \arg \min_{\|\vec{\theta}\|_2=1} f(\vec{\theta}) + \alpha \|\nabla f(\vec{\theta})\| \cos(\vec{\theta}, \nabla f(\vec{\theta})) \quad (3.17)$$

$$= \arg \min_{\|\vec{\theta}\|_2=1} \cos(\vec{\theta}, \nabla f(\vec{\theta})) \quad (3.18)$$

$$= -\frac{\nabla f(\vec{\theta})}{\|\nabla f(\vec{\theta})\|} \quad (3.19)$$

Gradient Descent

opt. \vec{v}

- opposite direction of $\nabla J_{in}(\vec{\theta})$
- in other words, the gradient points directly uphill, and the negative gradient points directly downhill

$$\vec{\theta} \leftarrow \vec{\theta} - \alpha \frac{\nabla J_{in}(\vec{\theta})}{\|\nabla J_{in}(\vec{\theta})\|} \quad (3.20)$$

Choice of α

α better be monotinic of $\|\nabla J_{in}(\vec{\theta})\|$, see Fig 3.2

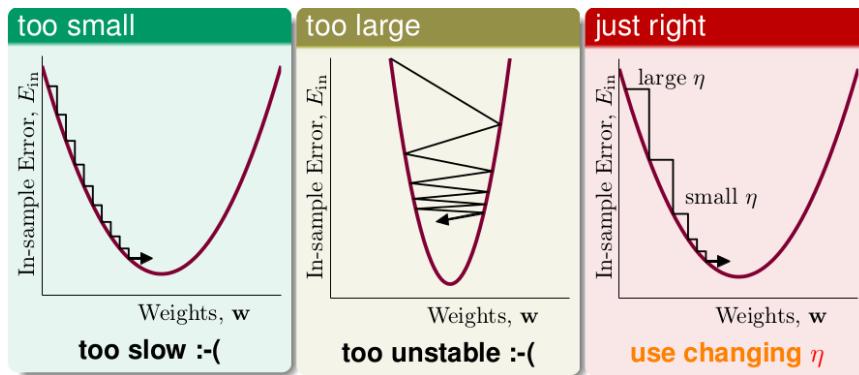


Figure 3.2: Choice of α

learning rate
method of steepest descent
line search
Jacobian matrix

if $\alpha \propto \|\nabla f(\vec{\theta})\| \cdot \alpha'$

$$\vec{\theta} \leftarrow \vec{\theta} - \alpha \frac{\nabla f(\vec{\theta})}{\|\nabla f(\vec{\theta})\|} \quad (3.21)$$

$$= \vec{\theta} - \alpha' \|\nabla f(\vec{\theta})\| \frac{\nabla f(\vec{\theta})}{\|\nabla f(\vec{\theta})\|} \quad (3.22)$$

$$= \vec{\theta} - \alpha' \nabla f(\vec{\theta}) \quad (3.23)$$

it can conv. to a local min., call α' the **learning rate**
usually write it directly as

$$\vec{\theta} \leftarrow \vec{\theta} - \alpha \nabla f(\vec{\theta}) \quad (3.24)$$

this is the **method of steepest descent**

- it conv. when $\nabla_{\vec{\theta}} f(\vec{\theta}) = \vec{0}$ or $\approx \vec{0}$
- often α is a preset small const.
- another way to choose α is **line search**

$$\alpha^* = \arg \min_{\alpha} f(\vec{\theta} - \alpha \nabla f(\vec{\theta})) \quad (3.25)$$

Gradient of a Vector-Valued Function (Jacobian)

this derivative is called **Jacobian matrix**

$$J = \frac{\partial \vec{f}(\vec{\theta})}{\partial \vec{\theta}} = \nabla_{\vec{\theta}^T} \vec{f}(\vec{\theta}) = (\nabla_{\vec{\theta}} \vec{f}(\vec{\theta})^T)^T \quad (3.26)$$

$$J_{ij} = \frac{\partial f(\vec{\theta})_i}{\partial \theta_j} \quad (3.27)$$

Implementation Notes on Gradient Descent

Simultaneously Update

$$temp_0 \leftarrow \theta_0 - \alpha \frac{\partial}{\partial \theta_0} f(\theta_0, \theta_1) \quad (3.28)$$

$$temp_1 \leftarrow \theta_1 - \alpha \frac{\partial}{\partial \theta_1} f(\theta_0, \theta_1) \quad (3.29)$$

$$\theta_0 \leftarrow temp_0 \quad (3.30)$$

$$\theta_1 \leftarrow temp_1 \quad (3.31)$$

Feature Scaling make sure features are on a similar scale (approx. a $[-1, +1]$ range)

$$x_j \leftarrow \frac{x_j - \mu_j}{\sigma_j}, \forall j \quad (3.32)$$

- $x_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)}$ is the avg. val. of x_j
- σ_j is the std. dev. or range (max - min) of x_j

Avoid Correlated Features when the contours of $f(\vec{\theta})$ is very elongated

- eg., if two dims. are highly correlated
- the direction of steepest descent is almost perpendicular to the direction towards the minimum, see Fig. 3.3

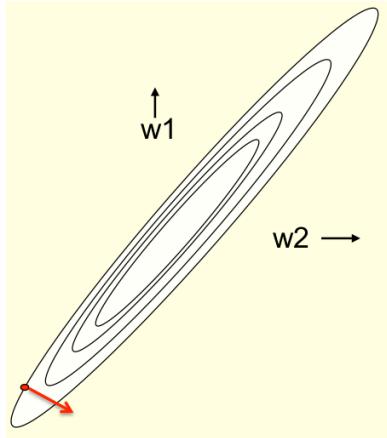


Figure 3.3: the case that the ellipse is very elongated

Practical Learning Rate for suff. small α , $f(\vec{\theta})$ should dec. on every iteration, see Fig. 3.4

- but if α too small, grad. desc. can be slow to conv.
- to choose α , try: 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1...

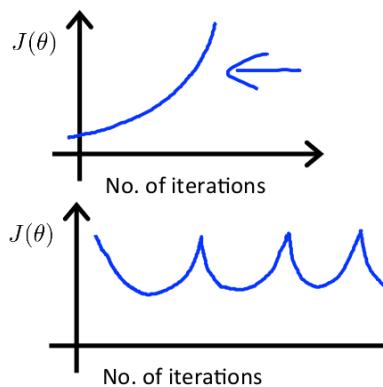


Figure 3.4: Grad. desc. is not working, try use smaller α .

3.3.3 Stochastic Gradient Descent

want

- update direction: $\vec{v} \approx -\nabla J_{in}(\vec{\theta})$
- computing \vec{v} by one single $(\vec{x}^{(i)}, y^{(i)})$
tech. on removing $\frac{1}{m} \sum_{i=1}^m$
- view as expectation E over uniform choice of i
- stochastic grad.: $\nabla_{\vec{\theta}} \text{err}(\vec{\theta}, \vec{x}^{(i)}, y^{(i)})$ with random i
stochastic grad. is an unbiased estimator of true grad.

$$\underset{\text{random } i}{\mathbb{E}} [\nabla_{\vec{\theta}} \text{err}(\vec{\theta}, \vec{x}^{(i)}, y^{(i)})] = \nabla_{\vec{\theta}} J_{in}(\vec{\theta}) \quad (3.33)$$

Stochastic Gradient Descent (SGD)

$$\vec{\theta} \leftarrow \vec{\theta} - \alpha \nabla_{\vec{\theta}} \text{err}(\vec{\theta}, \vec{x}^{(i)}, y^{(i)}) \quad (3.34)$$

stochastic grad. = true grad. + zero-mean noise directions
after enough steps, avg. true grad. \approx avg. stochastic grad.
pros:

- simple and cheaper computation
- gets $\vec{\theta}$ close to the min. much faster than batch grad. desc.
- useful for big data or online learning

cons

- less stable in nature
- it may never conv. to the min.
- $\vec{\theta}$ will keep oscillating around the min. of $J(\vec{\theta})$

Convergence Checking

batch grad. desc.

- plot $J_{train}(\vec{\theta})$ as a fcn. of # iterations of grad. desc.
- stochastic grad. desc.
 - during learning, compute $\text{err}(\vec{\theta}, \vec{x}^{(i)}, y^{(i)})$ before updating $\vec{\theta}$ using $(\vec{x}^{(i)}, y^{(i)})$
 - for every 1000 iterations, say, plot $\text{err}(\vec{\theta}, \vec{x}^{(i)}, y^{(i)})$ avg. over the last 1000 examples processed by alg., see Fig. 3.5

learning rate α is typically held const. Can slowly decrease α over time if we want $\vec{\theta}$ to conv. $\vec{\theta} = \frac{\text{const}_1}{\text{iterNum} + \text{const}_2}$ to ensure that the para. will conv. to the global min.

Batch Gradient Descent vs. Stochastic Gradient Descent

batch GD

- does steepest descent on the error surface
- this travels perpendicular to constraint from the contour lines, see Fig. 3.6
- after some # of iterations, batch GD will conv. to lower val. of obj. function

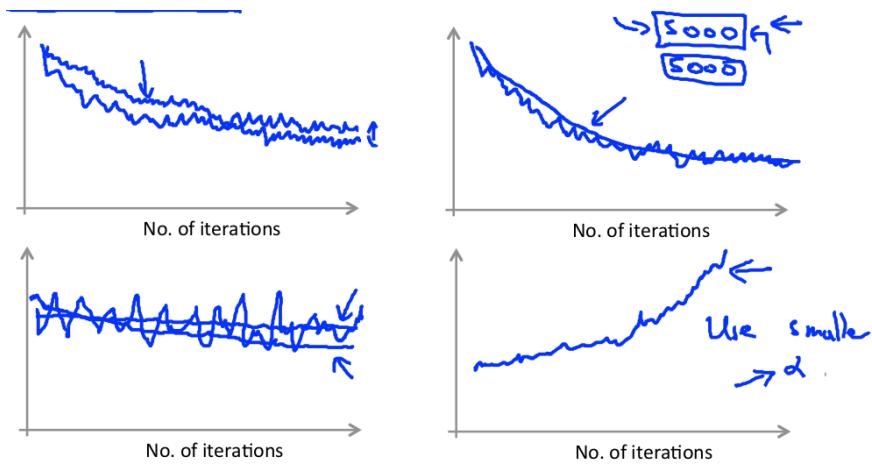


Figure 3.5: Convergence Checking

SGD

- zig-zags around the direction of steepest descent, see Fig. 3.7
- conv. init. faster than batch GD

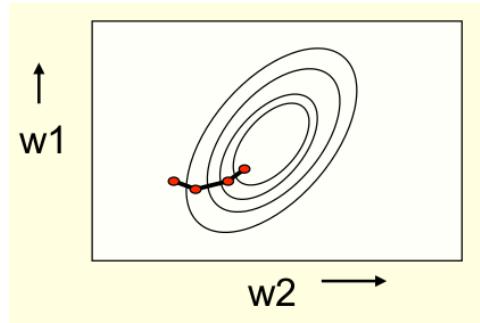


Figure 3.6: Batch Gradient Descent

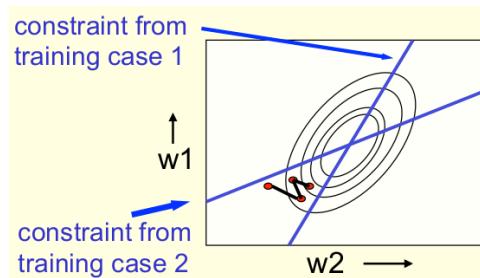


Figure 3.7: Stochastic Gradient Descent

how about SGD first, followed by batch GD?

- training err. can be greatly improved
- but it is at the cost of generation error

3.3.4 Mini-batch Gradient Descent

Three Gradient Descent Methods

batch grad. desc.: use all m examples in each iteration

stochastic grad. desc.: use 1 example in each iteration

mini-batch grad. desc.: use b examples in each iteration

how to choose b ?

- large b provides a more accurate estimate of the gradient, but with less than linear return (recall std. err. of the mean estimated from m samples is given by $\frac{\hat{\sigma}}{\sqrt{m}}$)
- extremely small $b \rightarrow$ multicore architectures are usually underutilized
- if all examples in the batch are to be processed in parallel, then the amount of memory scales with the batch size, this is the limiting factor in b
- when using GPU, it is common for power of 2 batch sizes to offer better runtime, typical 16, 32, ..., 256
- small b can offer a regularizing effect, generalization error is often best for a batch size of 1, though this might take a very long time to train and require a small α
- gradient based method can handle $b = 100$
- Hessian based method require $b = 10000$

Random Shuffling

computing the expected gradient from samples requires that those samples be indep.

- shuffle the examples before selecting minibatches
- in practice, shuffle the order of the dataset once and then minibatch through this fixed order

Algorithm 5 Mini-batch Logistic Regression Algorithm

```

init.  $\vec{\theta}$ 
randomly shaffle dataset
while there is not enough iterations
  for  $i = 1 : b : m$ 
     $\vec{\theta} \leftarrow \vec{\theta} + \alpha \frac{1}{b} \sum_{k=i}^{i+b-1} g(-y^{(i)} \vec{\theta}^T \vec{x}^{(i)}) (y^{(i)} \vec{x}^{(i)})$ 
return  $h(\vec{x}) = g(\vec{\theta}^T \vec{x})$ 

```

3.3.5 Map-reduce and Data Parallelism

batch grad. desc.

$$\vec{\theta} \leftarrow \vec{\theta} + \alpha \frac{1}{m} \sum_{i=1}^m g(-y^{(i)} \vec{\theta}^T \vec{x}^{(i)}) (y^{(i)} \vec{x}^{(i)}) \quad (3.35)$$

Algorithm 6 Map-reduce

Machine 1: use $(\vec{x}^{(1)}, y^{(1)}), \dots, (\vec{x}^{(100)}, y^{(100)})$
 $temp_1 = \sum_{i=1}^{100} g(-y^{(i)} \vec{\theta}^T \vec{x}^{(i)}) (y^{(i)} \vec{x}^{(i)})$

Machine 2: use $(\vec{x}^{(101)}, y^{(101)}), \dots, (\vec{x}^{(200)}, y^{(200)})$
 $temp_2 = \sum_{i=101}^{200} g(-y^{(i)} \vec{\theta}^T \vec{x}^{(i)}) (y^{(i)} \vec{x}^{(i)})$

Machine 3: use $(\vec{x}^{(201)}, y^{(201)}), \dots, (\vec{x}^{(300)}, y^{(300)})$
 $temp_3 = \sum_{i=201}^{300} g(-y^{(i)} \vec{\theta}^T \vec{x}^{(i)}) (y^{(i)} \vec{x}^{(i)})$

Machine 4: use $(\vec{x}^{(301)}, y^{(301)}), \dots, (\vec{x}^{(400)}, y^{(400)})$
 $temp_4 = \sum_{i=301}^{400} g(-y^{(i)} \vec{\theta}^T \vec{x}^{(i)}) (y^{(i)} \vec{x}^{(i)})$

comb.:
 $\vec{\theta} \leftarrow \vec{\theta} + \alpha \frac{1}{400} (temp_1 + temp_2 + temp_3 + temp_4)$

map-reduce and summation over the training set: many learning algs. can be expressed as computing sums of fcns. over the training set

3.3.6 Online Gradient Descent

$$J_{out}(\vec{\theta}) = \mathbb{E}[\text{err}(\vec{\theta}, \vec{x}, y)] = \int \text{err}(\vec{\theta}, \vec{x}, y) dp(\vec{x}, y) \quad (3.36)$$

$$\nabla J_{out}(\vec{\theta}) = \int \nabla \text{err}(\vec{\theta}, \vec{x}, y) dp(\vec{x}, y) \quad (3.37)$$

we can obtain an unbiased estimator of the exact gradient of generalization error by sampling one example (\vec{x}, y) (or equivalently a minibatch) from the data generating process p , and computing the gradient

- this is only possible if the examples are not repeated
- after we have pass through the training data once and begin to repeat minibatches, the two criteria diverge

3.3.7 Advanced Optimization

opt. alg.: given $J(\vec{\theta})$, want $\min_{\vec{\theta}} J(\vec{\theta})$

given $\vec{\theta}$, we have code that can compute $J(\vec{\theta}), \nabla J(\vec{\theta})$

grad. desc.: $\vec{\theta} \leftarrow \vec{\theta} - \alpha \nabla J(\vec{\theta})$

opt. algs.:

- Grad. Desc.
- Conjugate Grad.

- BFGS
 - L-BFGS
- advantages:
- no need to manually pick α
 - often faster than grad. desc.
- disadvantages: more complex

```

1 function [J, grad] = costFcn(theta)
2 % compute J
3 % compute grad
4
5 options = optimset('GradObj', 'on', 'MaxIter', '100');
6 theta = zeros(n+1, 1);
7 [theta, fcnVal, exitFlag] = fminunc(@costFcn, theta, options);

```

3.4 Hessian and Newton's Method

3.4.1 Second Derivative Test

second derivative can be used to determine whether a critical point is a local maximum, a local minimum, or saddle point

One Dimensional Case

- at critical point $f'(\theta) = 0$
 if $f''(\theta) > 0$
- $f'(\theta)$ increases as we move to the right \rightarrow the slope begins to point uphill to the right
 - $f'(\theta)$ decreases as we move to the left \rightarrow the slope begins to point uphill to the left
 - $\rightarrow \theta$ is a local minimum
- similarly, if $f''(\theta) < 0$, θ is a local maximum
 if $f''(\theta) = 0$, θ may be a saddle point, or a part of a flat region

Hessian Matrix

$$H = \nabla_{\vec{\theta}} (\nabla_{\vec{\theta}} f(\vec{\theta}))^T \in \mathbb{S}^n \quad (3.38)$$

$$H_{ij} = \frac{\partial^2 f(\vec{\theta})}{\partial \theta_i \partial \theta_j} \quad (3.39)$$

- at critical point $\nabla_{\vec{\theta}} f(\vec{\theta}) = \vec{0}$
 if $H \in \mathbb{S}_{++}^n$
- the directional second derivative in any direction must be positive
 - $\rightarrow \vec{\theta}$ is a local minimum

similarly, if $H \in \mathbb{S}_{--}^n$, $\vec{\theta}$ is a local maximum
if some of the eig.-vals. are pos., and some are neg.,
• $\vec{\theta}$ is a local maximum on some cross section of f
• but local minimum on another cross section
• $\vec{\theta}$ is a saddle point
if H has 0 eig.-val., the test is inconclusive

3.4.2 Understand the Performance of GD

when H has a poor condition number, GD perf. poorly

- geometrically, contours of f is like a long canyon with steep canyon walls
- because in one direction, the derivative increases rapidly, while in another direction, it increases slowly
- the large positive eigenvalue of the Hessian corresponding to the eigenvector pointed in this direction indicates that this directional derivative is rapidly increasing
- GD descending along canyon walls, because they are the steepest direction
- when step size is somewhat too large, it has a tendency to overshoot the bottom of the f and thus needs to descend the opposite canyon wall on the next iteration
- thus, the steepest direction is not actually a promising search direction in this context

3.4.3 Newton's Method

One Dimensional Case

consider $f : \mathbb{R} \mapsto R$, we want to find $\theta \in \mathbb{R}$, st. $f(\theta) = 0$

$$\theta \leftarrow \theta - \frac{f(\theta)}{f'(\theta)} \quad (3.40)$$

interpretation, see Fig. 3.8

- approximating the fcn. f via a lin. fcn. that is tangent to f at the curr. guess θ
- sol. for where that lin. fcn. equals to 0
- letting the next guess for θ be where that lin. fcn. is 0

Proof.

$$f'(\theta_t) = \frac{f(\theta_t)}{\theta_t - \theta_{t+1}} \quad (3.41)$$

$$\therefore \theta_t - \theta_{t+1} = \frac{f(\theta_t)}{f'(\theta_t)} \quad (3.42)$$

$$\theta_{t+1} = \theta_t - \frac{f(\theta_t)}{f'(\theta_t)} \quad (3.43)$$

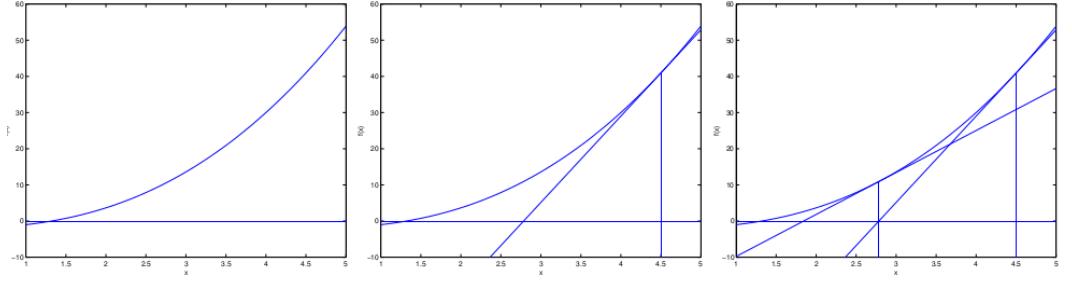


Figure 3.8: Newton's Method

□

Use Newton's Method for Optimization

$$\vec{\theta} \leftarrow \vec{\theta} - H^{-1} \nabla f(\vec{\theta}) \quad (3.44)$$

Proof. by second-order Taylor series expansion

$$\vec{\theta}_{t+1}^* = \arg \min_{\vec{\theta}_{t+1}} f(\vec{\theta}_{t+1}) \quad (3.45)$$

$$\begin{aligned} &\approx \arg \min_{\vec{\theta}_{t+1}} f(\vec{\theta}_t) + (\vec{\theta}_{t+1} - \vec{\theta}_t)^T \nabla_{\vec{\theta}} f(\vec{\theta}_t) \\ &\quad + \frac{1}{2} (\vec{\theta}_{t+1} - \vec{\theta}_t)^T H \Big|_{\vec{\theta}=\vec{\theta}_t} (\vec{\theta}_{t+1} - \vec{\theta}_t) \end{aligned} \quad (3.46)$$

$$\nabla_{\vec{\theta}_{t+1}} f(\vec{\theta}_{t+1}) = \nabla_{\vec{\theta}} f(\vec{\theta}_t) + H(\vec{\theta}_{t+1} - \vec{\theta}_t) \stackrel{\text{set}}{=} \vec{0} \quad (3.47)$$

$$\vec{\theta}_{t+1} = \vec{\theta}_t - H^{-1} \nabla_{\vec{\theta}} f(\vec{\theta}_t) \quad (3.48)$$

□

- locally approximated as quadratic, iteratively updating the approximation and jumping to the minimum of the approximation
- if the obj. function is convex but not quad., this update can be iterated

Algorithm 7 Newton's Method

repeat until convergence

$$\begin{aligned} H &= \nabla^2 J(\vec{\theta}) \\ \vec{\theta} &\leftarrow \vec{\theta} - H^{-1} \nabla J(\vec{\theta}) \end{aligned}$$

Understanding the Performance of Newton's Method

H : transform from Euclidean space where the problem is defined to a space where gradient opt. is simplified

because $H \in \mathbb{S}_+^n$, its eigendecomposition is given by $H = U\Lambda U^T$

consider transform $\vec{\theta}$ to $\vec{\phi} = \Lambda^{\frac{1}{2}}U^T\vec{\theta}$

$$\nabla_{\vec{\phi}} J(\vec{\theta}_t) = \left(\frac{\partial J(\vec{\theta}_t)}{\partial \vec{\theta}} \frac{\partial \vec{\theta}}{\partial \vec{\phi}} \right)^T \quad (3.49)$$

$$= \left(\frac{\partial \vec{\theta}}{\partial \vec{\phi}} \right)^T \nabla_{\vec{\theta}} J(\vec{\theta}_t) \quad (3.50)$$

$$= U\Lambda^{\frac{1}{2}} \nabla_{\vec{\theta}} J(\vec{\theta}_t) \quad (3.51)$$

$$\nabla_{\vec{\theta}} J(\vec{\theta}_t) = \Lambda^{frac{1}{2}} U^T \nabla_{\vec{\phi}} J(\vec{\theta}_t) \quad (3.52)$$

then, the Taylor expansion is

$$\begin{aligned} J(\vec{\theta}_{t+1}) &= J(\vec{\theta}_t) + (\vec{\theta}_{t+1} - \vec{\theta}_t)^T \nabla_{\vec{\theta}} J(\vec{\theta}_t) \\ &\quad + \frac{1}{2} (\vec{\theta}_{t+1} - \vec{\theta}_t)^T H \Big|_{\vec{\theta}=\vec{\theta}_t} (\vec{\theta}_{t+1} - \vec{\theta}_t) \end{aligned} \quad (3.53)$$

$$\begin{aligned} &= J(\vec{\theta}_t) + (\vec{\theta}_{t+1} - \vec{\theta}_t)^T U \Lambda^{\frac{1}{2}} \nabla_{\vec{\phi}} J(\vec{\theta}_t) \\ &\quad + \frac{1}{2} (\vec{\theta}_{t+1} - \vec{\theta}_t)^T U \Lambda U^T (\vec{\theta}_{t+1} - \vec{\theta}_t) \end{aligned} \quad (3.54)$$

$$\begin{aligned} &= J(\vec{\theta}_t) + (\Lambda^{\frac{1}{2}} U^T \vec{\theta}_{t+1} - \Lambda^{\frac{1}{2}} U^T \vec{\theta}_t)^T \nabla_{\vec{\phi}} J(\vec{\theta}_t) \\ &\quad + \frac{1}{2} (\Lambda^{\frac{1}{2}} U^T \vec{\theta}_{t+1} - \Lambda^{\frac{1}{2}} U^T \vec{\theta}_t)^T (\Lambda^{\frac{1}{2}} U^T \vec{\theta}_{t+1} - \Lambda^{\frac{1}{2}} U^T \vec{\theta}_t) \end{aligned} \quad (3.55)$$

$$= J(\vec{\theta}_t) + (\vec{\phi}_{t+1} - \vec{\phi}_t)^T \nabla_{\vec{\phi}} J(\vec{\theta}_t) + \frac{1}{2} (\vec{\phi}_{t+1} - \vec{\phi}_t)^T (\vec{\phi}_{t+1} - \vec{\phi}_t) \quad (3.56)$$

$$\nabla_{\vec{\phi}} J(\vec{\theta}_{t+1}) \stackrel{\text{set}}{=} \vec{0} \quad (3.57)$$

$$\vec{\phi}_{t+1} \leftarrow \vec{\phi}_t - \nabla_{\vec{\phi}} J(\vec{\phi}_t) \quad (3.58)$$

Newton's method maps an arbitrary and possibly ill-conditioned quadratic objective function from $\vec{\theta}$ space into an alternative $\vec{\phi}$ space where the quadratic objective function is isometric

in ϕ -space, the Newton's update is transformed into a std. GD with $\alpha = 1$, see Fig. 3.9

Newton's Method vs. Gradient Descent

Newton's method vs. GD

- faster conv. than (batch) GD
- require many fewer iterations to get very close to the min.

Fisher scoring

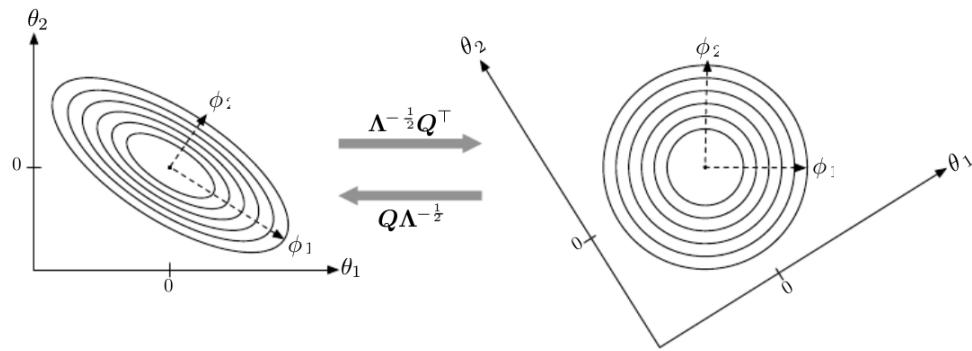


Figure 3.9: The effect of Newton's Method

- but one iteration of Newton's can be more expensive than one iteration of GD, since it needs to inv. a $n \times n$ mat.
- so long as n is not too large, it is usually much faster overall
- but Newton's Method is only appropriate when the nearby critical point is a minimum ($H \in \mathbb{S}_{++}^n$),
- whereas GD can in principle escape a saddle point, although it may take a lot of time if the negative eigenvalues are very small in magnitude, producing a kind of plateau around the saddle point

Fisher scoring : use Newton's method to max. the LogReg log likelihood fcn. $l(\vec{\theta})$

3.5 Convex Optimization

3.5.1 Introduction

Optimization with Guarantee

strictly convex functions are well-behaved because they lack saddle points and all of their local minima are necessarily global minima

Constrained Optimization

one way to constrained optimization is simply to modify GD

- if we use const. α , we make GD steps, then project the result back into constraints
- if we use a line search, we can search only over step sizes α that yield new $\vec{\theta}$ that are feasible, or we can project each point on the line back into the constraint region
- this method can be made more efficient by projecting the gradient into the tangent space of the feasible region before taking the step or beginning the line search the constraint region

a more sophisticated approach is to design a different, unconstrained optimization problem whose solution can be converted into a solution to the original, constrained optimization problem

Convex Set
convex combination

Ex. sol.

$$\min_{\vec{\theta} \in \mathbb{R}^2} f(\vec{\theta}) \quad (3.59)$$

$$\text{st. } \|\vec{\theta}\|_2 = 1 \quad (3.60)$$

Sol. we can instead minimize

$$\min_{\alpha \in \mathbb{R}} f\left(\begin{bmatrix} \cos \alpha \\ \sin \alpha \end{bmatrix}\right) \quad (3.61)$$

then return

$$\theta_1 = \cos \alpha, \theta_2 = \sin \alpha \quad (3.62)$$

3.5.2 Convex Sets

Define 3.7 (Convex Set). *A set S is convex if*

$$\alpha x + (1 - \alpha)y \in S, \forall x, y \in S, \forall \alpha \in \mathbb{R}, 0 \leq \alpha \leq 1 \quad (3.63)$$

- $\alpha x + (1 - \alpha)y$ is called a **convex combination** of x, y
- it means, if we take any two points in S , and draw a line segment between these two points
- then every point on that line segment also belongs to S , see Fig. 3.10

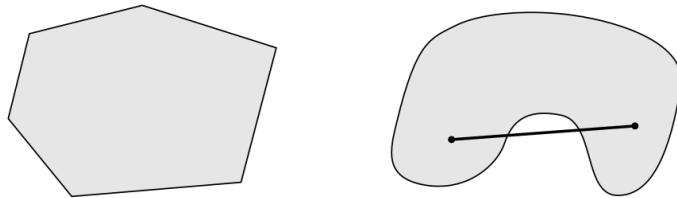


Figure 3.10: Examples of a convex set and a non-convex set

Examples

$$\mathbb{R}^n$$

$$\alpha \vec{x} + (1 - \alpha) \vec{y} \in \mathbb{R}^n, \forall \vec{x}, \vec{y} \in \mathbb{R}^n, \forall \alpha \in \mathbb{R}, 0 \leq \alpha \leq 1 \quad (3.64)$$

The Non-Negative Orthant \mathbb{R}_+^n

$$\mathbb{R}_+^n = \{\vec{x} \in \mathbb{R}^n : x_j \geq 0, \forall j \in \{1, 2, \dots, n\}\} \quad (3.65)$$

$$\alpha\vec{x} + (1 - \alpha)\vec{y} \geq 0, \forall \vec{x}, \vec{y} \in \mathbb{R}_+^n, \forall \alpha \in \mathbb{R}, 0 \leq \alpha \leq 1 \quad (3.66)$$

Norm Balls

$$S = \{\vec{x} \in \mathbb{R}^n : \|\vec{x}\| \leq 1\} \quad (3.67)$$

$$\|\alpha\vec{x} + (1 - \alpha)\vec{y}\| \leq \|\alpha\vec{x}\| + \|(1 - \alpha)\vec{y}\| \quad // triangle inequality of norm \quad (3.68)$$

$$= \alpha\|\vec{x}\| + (1 - \alpha)\|\vec{y}\| \quad // homogeneity of norm \quad (3.69)$$

$$\leq \alpha + (1 - \alpha) \quad (3.70)$$

$$= 1, \forall \vec{x}, \vec{y} \in S, \forall \alpha \in \mathbb{R}, 0 \leq \alpha \leq 1 \quad (3.71)$$

Affine Subspaces

$$S = \{\vec{x} \in \mathbb{R}^n : A\vec{x} = \vec{b}, A \in \mathbb{R}^{m \times n}, \vec{b} \in \mathbb{R}^m\} \quad (3.72)$$

note $S = \emptyset$ if $\vec{b} \notin C(A)$

$$A(\alpha\vec{x} + (1 - \alpha)\vec{y}) = \alpha A\vec{x} + (1 - \alpha)A\vec{y} \quad (3.73)$$

$$= \alpha\vec{b} + (1 - \alpha)\vec{b} \quad (3.74)$$

$$= \vec{b}, \forall \vec{x}, \vec{y} \in S, \forall \alpha \in \mathbb{R}, 0 \leq \alpha \leq 1 \quad (3.75)$$

Polyhedra this is componentwise inequality

$$S = \{\vec{x} \in \mathbb{R}^n : A\vec{x} \leq \vec{b}, A \in \mathbb{R}^{m \times n}, \vec{b} \in \mathbb{R}^m\} \quad (3.76)$$

$$A(\alpha\vec{x} + (1 - \alpha)\vec{y}) = \alpha A\vec{x} + (1 - \alpha)A\vec{y} \quad (3.77)$$

$$\leq \alpha\vec{b} + (1 - \alpha)\vec{b} \quad (3.78)$$

$$= \vec{b}, \forall \vec{x}, \vec{y} \in S, \forall \alpha \in \mathbb{R}, 0 \leq \alpha \leq 1 \quad (3.79)$$

PSD. Matrices \mathbb{S}_+^n

$$\vec{a}^T(\alpha A + (1 - \alpha)B)\vec{a} = \alpha\vec{a}^T A\vec{a} + \alpha\vec{a}^T B\vec{a} \geq 0 \quad (3.80)$$

$$\forall A, B \in \mathbb{S}_+^n, \forall \alpha \in \mathbb{R}, 0 \leq \alpha \leq 1, \forall \vec{x} \in \mathbb{R}^n \quad (3.81)$$

likewise, $\mathbb{S}_{++}^n, \mathbb{S}_-^n, \mathbb{S}_{--}^n$ are also convex sets

Intersections of Convex Sets

$$S = \bigcap_{k=1}^K S_k \quad \begin{array}{ll} \text{Convex Functions} \\ \text{strictly convex} \\ \text{concave} \\ \text{strictly concave} \end{array} \quad (3.82)$$

$$\forall x, y \in S \quad (3.83)$$

$$\therefore x, y \in S_k, \forall k \quad (3.84)$$

$$\therefore \alpha x + (1 - \alpha)y \in S_k, \forall k, \forall \alpha \in \mathbb{R}, 0 \leq \alpha \leq 1 \quad (3.85)$$

// by the def. of convex set

$$\therefore \alpha x + (1 - \alpha)y \in \bigcap_{k=1}^K S_k = S \quad (3.87)$$

however, that the union of convex sets in general will not be convex.

3.5.3 Convex Functions

Define 3.8 (Convex Functions). A function f is convex if its domain $\text{dom } f$ is a convex set¹ and if

$$f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y), \forall x, y \in \text{dom } f, \forall \alpha \in \mathbb{R}, 0 \leq \alpha \leq 1 \quad (3.88)$$

- if we pick any two points on the graph of a convex function and draw a straight line segment between them
- then the portion of the function between these two points will lie below this straight line, see Fig. 3.11

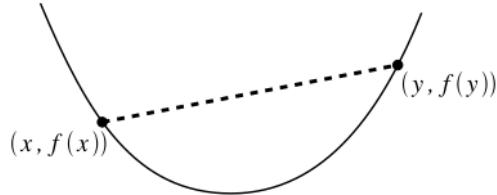


Figure 3.11: Graph of a convex function

f is **strictly convex** if

$$f(\alpha x + (1 - \alpha)y) < \alpha f(x) + (1 - \alpha)f(y), \forall x, y \in \text{dom } f, x \neq y, \forall \alpha \in \mathbb{R}, 0 < \alpha < 1 \quad (3.89)$$

f is **concave** if $-f$ is convex

f is **strictly concave** if $-f$ is strictly convex

¹this requirement is to ensure $\alpha x + (1 - \alpha)y$ is still in $\text{dom } f$ so that $f(\alpha x + (1 - \alpha)y)$ is defined

first-order
approximation

First Order Condition for Convexity

suppose $f : \mathbb{R}^n \mapsto \mathbb{R}$ is differentiable, then f is convex iff. $\text{dom } f$ is a convex set and if

$$f(\vec{y}) \geq f(\vec{x}) + (\nabla_{\vec{x}} f(\vec{x}))^T (\vec{y} - \vec{x}), \forall \vec{x}, \vec{y} \in \text{dom } f \quad (3.90)$$

- $f(\vec{x}) + (\nabla_{\vec{x}} f(\vec{x}))^T (\vec{y} - \vec{x})$ is the **first-order approximation** to $f(\vec{y})$
- this can be thought of as approximating f with its tangent line at the point \vec{x} , see Fig. 3.12
- f is convex iff. every point on the tangent line will lie below the corresponding point on f

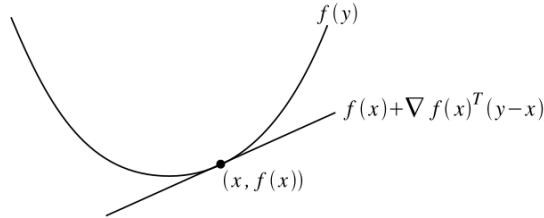


Figure 3.12: Illustration of the first-order condition for convexity

f is strictly convex if $>$ holds

f is concave if \leq holds

f is strictly concave if $<$ holds

Second Order Condition for Convexity

suppose $f : \mathbb{R}^n \mapsto \mathbb{R}$ is twice differentiable, then f is convex iff. $\text{dom } f$ is a convex set and if

$$H = \nabla_{\vec{x}}^2 f(\vec{x}) \in \mathbb{S}_+^n, \forall \vec{x} \in \text{dom } f \quad (3.91)$$

in one dim., it is equivalent to

$$f''(x) \geq 0 \quad (3.92)$$

f is strictly convex if $H \in \mathbb{S}_{++}^n$

f is concave if $H \in \mathbb{S}_-^n$

f is strictly concave if $H \in \mathbb{S}_{--}^n$

Jensen's Inequality

suppose f is convex

$$f(\mathbb{E}[\vec{x}]) \leq \mathbb{E}[f(\vec{x})], \forall \vec{x} \in \text{dom } f \quad (3.93)$$

Proof.

θ -sublevel set

$$\because f \text{ is convex} \quad (3.94)$$

$$\therefore f(\alpha x_1 + (1 - \alpha)x_2) \leq \alpha f(x_1) + (1 - \alpha)f(x_2), \quad (3.95)$$

$$\forall x_1, x_2 \in \text{dom } f, \forall \alpha \in \mathbb{R}, 0 \leq \alpha \leq 1 \quad (3.96)$$

$$\therefore f(\alpha'(\alpha x_1 + (1 - \alpha)x_2) + (1 - \alpha')x_3) \quad (3.97)$$

$$\leq \alpha'f(\alpha x_1 + (1 - \alpha)x_2) + (1 - \alpha')f(x_3) \quad (3.98)$$

$$\leq \alpha' \alpha f(x_1) + \alpha'(1 - \alpha)f(x_2) + (1 - \alpha')f(x_3) \quad (3.99)$$

$$\forall x_1, x_2, x_3 \in \text{dom } f, \forall \alpha, \alpha' \in \mathbb{R}, 0 \leq \alpha \leq 1, 0 \leq \alpha' \leq 1, \quad (3.100)$$

$$\stackrel{\text{def}}{=} \alpha_1 f(x_1) + \alpha_2 f(x_2) + \alpha_3 f(x_3), \quad (3.101)$$

$$\alpha_1 + \alpha_2 + \alpha_3 = \alpha' \alpha + \alpha'(1 - \alpha) + \alpha'(1 - \alpha) = 1 \quad (3.102)$$

$$\therefore f\left(\sum_{k=1}^K \alpha_k x_k\right) \leq \sum_{k=1}^K \alpha_k f(x_k), \quad (3.103)$$

$$\forall x_k \in \text{dom } f, \forall \alpha_k \in \mathbb{R}_+, \sum_{k=1}^K \alpha_k = 1 // \text{ by induction} \quad (3.104)$$

$$\therefore f\left(\int p(x) x dx\right) \leq \int p(x) f(x) dx, \quad (3.105)$$

$$\int p(x) dx = 1, p(x) \geq 0, \forall x // \text{ for inf. sum} \quad (3.106)$$

□

Sublevel Sets

suppose $f : \mathbb{R}^n \mapsto \mathbb{R}$ is a convex function, the **θ -sublevel set** is the set of all points \vec{x} such that $f(\vec{x}) \leq \theta$

$$S = \{\vec{x} \in \text{dom } f : f(\vec{x}) \leq \theta\} \quad (3.107)$$

S is a convex set

$$f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y) \quad (3.108)$$

$$\leq \alpha\theta + (1 - \alpha)\theta \quad (3.109)$$

$$= \theta, \forall x, y \in S, \forall \alpha \in \mathbb{R}, 0 \leq \alpha \leq 1 \quad (3.110)$$

Examples

Exponential

$$f(x) = \exp(ax), a \in \mathbb{R}, \text{dom } f = \mathbb{R} \quad (3.111)$$

$$f''(x) = a^2 \exp(ax) \geq 0, \forall x \in \text{dom } f \quad (3.112)$$

Negative Logarithm

$$f(x) = -\log x, \text{dom } f = \mathbb{R}_+ \quad (3.113)$$

$$f''(x) = \frac{1}{x^2} \geq 0, \forall x \in \text{dom } f \quad (3.114)$$

Affine Function

$$f(\vec{x}) = \vec{b}^T \vec{x} + c, \vec{b} \in \mathbb{R}^n, c \in \mathbb{R}, \text{dom } f = \mathbb{R}^n \quad (3.115)$$

$$H = 0 \in \mathbb{S}_{++}^n, \forall x \in \text{dom } f \quad (3.116)$$

H is both \mathbb{S}_+^n and \mathbb{S}_-^n , so f is (the only function) both convex and concave

Quadratic Functions

$$f(\vec{x}) = \vec{x}^T A \vec{x} + \vec{b}^T \vec{x} + c, A \in \mathbb{S}^n, \vec{b} \in \mathbb{R}^n, c \in \mathbb{R}, \text{dom } f = \mathbb{R}^n \quad (3.117)$$

$$H = 2A, \forall x \in \text{dom } f \quad (3.118)$$

thus, the convexity of f is determined entirely by A

- if $A \in \mathbb{S}_+^n \rightarrow f$ is convex
 - if A is indefinite then f is neither convex nor concave
- a special case: Euclidean norm is a strictly convex function

$$f(\vec{x}) = \|\vec{x}\|_2^2 = \vec{x}^T \vec{x}, \text{dom } f = \mathbb{R}^n \quad (3.119)$$

$$H = 2I \in \mathbb{S}_{++}^n \quad (3.120)$$

Norms

$$f(\vec{x}) = \|\vec{x}\|, \text{dom } f = \mathbb{R}^n \quad (3.121)$$

$$\|\alpha \vec{x} + (1 - \alpha) \vec{y}\| \leq \|\alpha \vec{x}\| + \|(1 - \alpha) \vec{y}\| = \alpha \|\vec{x}\| + (1 - \alpha) \|\vec{y}\|, \quad (3.122)$$

$$\forall \alpha \in \mathbb{R}, 0 \leq \alpha \leq 1 \quad (3.123)$$

norms are not generally not differentiable everywhere

Nonnegative Weighted Sums of Convex Functions

$$f(x) = \sum_{k=1}^K w_k f_k(x), \forall w_k \in \mathbb{R}_+ \quad (3.124)$$

$$f(\alpha x + (1 - \alpha) y) = \sum_{k=1}^K w_k f_k(\alpha x + (1 - \alpha) y) \quad (3.125)$$

$$\leq \sum_{k=1}^K w_k (\alpha f(x) + (1 - \alpha)f(y)), \forall \alpha \in \mathbb{R}, 0 \leq \alpha \quad \begin{array}{l} \text{Locally Optimal} \\ \text{Globally Optimal} \end{array} \quad (3.126)$$

$$= \alpha \sum_{k=1}^K w_k f(x) + (1 - \alpha) \sum_{k=1}^K w_k f(y) \quad (3.127)$$

$$= \alpha f(x) + (1 - \alpha)f(y) \quad (3.128)$$

3.5.4 Convex Optimization Problems

$$\min_x f(x) \quad (3.129)$$

$$\text{st. } x \in S \quad (3.130)$$

- f is a convex function
 - S is a convex set
- often write it as

$$\min_x f(x) \quad (3.131)$$

$$\text{st. } g_i(x) \leq 0, \forall i \quad (3.132)$$

$$h_j(x) = 0, \forall j \quad (3.133)$$

- f is a convex function
- g_i 's are convex functions
0-sublevel set of g_i is a convex set
intersection of many convex sets, is also convex
- h_j 's are affine functions
an equability constraint is equivalent to 2 inequalities:

$$h_j(x) \leq 0, h_j(x) \geq 0 \quad (3.134)$$

these will both be valid constraints iff. h_j is convex, and $-h_j$ is convex (ie., h_j is concave), so h_j must be affine

Global Optimality in Convex Problems

Define 3.9 (Locally Optimal). *a point x is locally optimal if it is feasible and if*

$$\exists \delta > 0, f(x) \leq f(y), \forall y \text{ st. } \|x - y\|_2 \leq \delta \text{ and } y \text{ is feasible} \quad (3.135)$$

Define 3.10 (Globally Optimal). *a point x is globally optimal if it is feasible and if*

$$f(x) \leq f(y), \forall y \text{ st. } y \text{ is feasible} \quad (3.136)$$

for a convex optimization problem all locally optimal points are globally optimal

Proof. by contradiction

suppose x is a locally optimal point which is not globally opt. ie., there exists a feasible point x^* st. $f(x^*) < f(x)$, by def. of local opt.

$$\exists \delta > 0, f(x) \leq f(y), \forall y \text{ st. } \|x - y\|_2 \leq \delta \text{ and } y \text{ is feasible} \quad (3.137)$$

choose

$$y = \alpha x^* + (1 - \alpha)x, \alpha = \frac{\delta}{\|x - x^*\|_2} \quad (3.138)$$

since the feasible set is a convex set, and x, x^* are both feasible, y is in that convex set, and y is feasible

$$\|x - y\|_2 = \|x - (\alpha x^* + (1 - \alpha)x)\|_2 \quad (3.139)$$

$$= \left\| x - \left(\frac{\delta}{\|x - x^*\|_2} x^* + \left(1 - \frac{\delta}{\|x - x^*\|_2}\right)x \right) \right\|_2 \quad (3.140)$$

$$= \left\| x - x - \frac{\delta}{\|x - x^*\|_2} (x^* - x) \right\|_2 \quad (3.141)$$

$$= \left\| \frac{\delta}{\|x - x^*\|_2} (x - x^*) \right\|_2 \quad (3.142)$$

$$= \frac{\delta}{2} \left\| \frac{(x - x^*)}{\|x - x^*\|_2} \right\| \quad (3.143)$$

$$= \frac{\delta}{2} \quad (3.144)$$

$$< \delta \quad (3.145)$$

$$f(y) = f(\alpha x^* + (1 - \alpha)x) \quad (3.146)$$

$$\leq \alpha f(x^*) + (1 - \alpha)f(x) \quad (3.147)$$

$$< \alpha f(x) + (1 - \alpha)f(x) \quad (3.148)$$

$$= f(x) \quad (3.149)$$

it is a contradiction, x cannot be locally opt. \square

Special Cases of Convex Problems

Linear Programming (LP) f and g_i are affine functions

$$\min_{\vec{x}} \vec{p}^T \vec{x} + r \quad (3.150)$$

$$\text{st. } B\vec{x} \leq \vec{c} \quad (3.151)$$

$$W\vec{x} = \vec{v} \quad (3.152)$$

Quadratic Programming (QP) f is a convex quadratic function and g' s are affine

$$\min_{\vec{x}} \vec{x}^T Q \vec{x} + \vec{p}^T \vec{x} + r, Q \in \mathbb{S}_+^n \quad (3.153)$$

$$\text{st. } B \vec{x} \leq \vec{c} \quad (3.154)$$

$$W \vec{x} = \vec{u} \quad (3.155)$$

Quadratically Constrained Quadratic Programming (QCQP) f and g_i 's are convex quadratic functions

$$\min_{\vec{x}} \vec{x}^T Q \vec{x} + \vec{p}^T \vec{x} + r, Q \in \mathbb{S}_+^n \quad (3.156)$$

$$\text{st. } \vec{x}^T A_i \vec{x} + \vec{b}_i^T \vec{x} + c_i \leq 0, A \in \mathbb{S}_+^n, \forall i \quad (3.157)$$

$$W \vec{x} = \vec{u} \quad (3.158)$$

Semidefinite Programming (SDP)

$$\min_X \text{tr } P X, P \in \mathbb{S}^n \quad (3.159)$$

$$\text{st. } \text{tr } W_i X = u_i, W_i \in \mathbb{S}^n, \forall i \quad (3.160)$$

$$X \in \mathbb{S}_+^n \quad (3.161)$$

$$\text{tr } P X = \text{tr} [\vec{p}_1 \vec{p}_2 \cdots \vec{p}_m] \begin{bmatrix} \vec{x}_1^T \\ \vec{x}_2^T \\ \vdots \\ \vec{x}_m^T \end{bmatrix} \quad (3.162)$$

$$= \text{tr} (\vec{p}_1 \vec{x}_1^T + \vec{p}_2 \vec{x}_2^T + \cdots + \vec{p}_m \vec{x}_m^T) \quad (3.163)$$

$$= \text{tr } \vec{p}_1 \vec{x}_1^T + \text{tr } \vec{p}_2 \vec{x}_2^T + \cdots + \text{tr } \vec{p}_m \vec{x}_m^T \quad (3.164)$$

$$= \vec{p}_1^T \vec{x}_1 + \vec{p}_2^T \vec{x}_2 + \cdots + \vec{p}_m^T \vec{x}_m \quad (3.165)$$

SDP are more general form of QCQP

Examples

Constrained Least Squares

$$\min_{\vec{\theta}} \|X \vec{\theta} - \vec{y}\|_2^2 \quad (3.166)$$

$$\text{st. } \vec{l} \leq \vec{\theta} \leq \vec{u} \quad (3.167)$$

this is a QP problem, with

$$Q = X^T X, \vec{p} = -X^T \vec{y}, r = \vec{y}^T \vec{y} \quad (3.168)$$

$$B_u = I, \vec{c}_u = \vec{u} \quad (3.169)$$

$$B_l = -I, \vec{c}_l = -\vec{l} \quad (3.170)$$

$$W = 0, \vec{u} = \vec{0} \quad (3.171)$$

Maximum Likelihood for Logistic Regression

$$l(\vec{\theta}) = -\frac{1}{m} \sum_{i=1}^m \log(1 + \exp(-y^{(i)} \vec{\theta}^T \vec{x}^{(i)})) \quad (3.172)$$

$$\begin{aligned} \nabla_{\vec{\theta}} l(\vec{\theta}) &= -\frac{1}{m} \sum_{i=1}^m \frac{1}{1 + \exp(-y^{(i)} \vec{\theta}^T \vec{x}^{(i)})} (1 + \exp(-y^{(i)} \vec{\theta}^T \vec{x}^{(i)})) (-y^{(i)} \vec{x}^{(i)}) \\ &= \frac{1}{m} \sum_{i=1}^m g(-y^{(i)} \vec{\theta}^T \vec{x}^{(i)}) y^{(i)} \vec{x}^{(i)} \end{aligned} \quad (3.174)$$

$$H = \nabla_{\vec{\theta}} (\nabla_{\vec{\theta}} l(\vec{\theta}))^T \quad (3.175)$$

$$= \nabla_{\vec{\theta}} \frac{1}{m} \sum_{i=1}^m g(-y^{(i)} \vec{\theta}^T \vec{x}^{(i)}) y^{(i)} \vec{x}^{(i)T} \quad (3.176)$$

$$= -\frac{1}{m} \sum_{i=1}^m g(-y^{(i)} \vec{\theta}^T \vec{x}^{(i)}) g(y^{(i)} \vec{\theta}^T \vec{x}^{(i)}) y^{(i)} y^{(i)} \vec{x}^{(i)T} \quad (3.177)$$

$$\vec{a}^T H \vec{a} = -\frac{1}{m} \sum_{i=1}^m g(-y^{(i)} \vec{\theta}^T \vec{x}^{(i)}) g(y^{(i)} \vec{\theta}^T \vec{x}^{(i)}) \vec{a}^T \vec{x}^{(i)} \vec{x}^{(i)T} \vec{a} \quad (3.178)$$

$$= -\frac{1}{m} \sum_{i=1}^m g(-y^{(i)} \vec{\theta}^T \vec{x}^{(i)}) g(y^{(i)} \vec{\theta}^T \vec{x}^{(i)}) (\vec{x}^{(i)T} \vec{a})^T \quad (3.179)$$

$$= -\frac{1}{m} \sum_{i=1}^m (\geq 0) \quad (3.180)$$

$$\leq 0 \quad (3.181)$$

$$\therefore H \in \mathbb{S}_-^n \quad (3.182)$$

MLE is equivalent to

$$\min_{\vec{\theta}} -l(\vec{\theta}) \quad (3.183)$$

with no constraints

however, it is not so easy to put this problem into a “standard” form optimization problem, but you can very efficiently find the global solution using an algorithm such as Newtons method

Implementation: Linear SVM using CVX

CVX² is a off-the-shelf MATLAB based software package to sol. convex opt. problem

- once you identify the convex optimization problem, you can solve it without worrying about how to implement

²<http://stanford.edu/~boyd/cvx/>

- but it can be much slower compared to the best possible implementation the algorithm yourself.

eg., use CVX to sol. linear primal SVM

```

1 cvx_begin
2     variables w(n) b xi(m)
3     minimize 1 / 2 * sum(w .* w) + C * sum(xi)
4     y .* (X * w + b) >= 1 - xi;
5     xi >= 0
6 cvx_end

```

3.5.5 Lagrange duality

Recall: Convex Optimization Problem

when sol.

$$\min_{\vec{x}} f(\vec{x}) \quad (3.184)$$

a necessary and sufficient condition for \vec{x}^* is

$$\nabla_{\vec{x}} f(\vec{x}^*) = \vec{0} \quad (3.185)$$

in the more general setting of convex optimization problem with constraints

$$\min_x f(x) \quad (3.186)$$

$$\text{st. } g_i(x) \leq 0, \forall i \quad (3.187)$$

$$h_j(x) = 0, \forall j \quad (3.188)$$

this simple optimality condition does not work

the theory of Lagrange duality is the study of optimal solutions to convex optimization problems

The Lagrangian

$$L(\vec{x}, \vec{\alpha}, \vec{\beta}) = f(\vec{x}) + \sum_i \alpha_i g_i(\vec{x}) + \sum_j \beta_j h_j(\vec{x}) \quad (3.189)$$

Primal Problem

the primal problem is a modified version of the original convex objective problem

$$\min_{\vec{x}} \max_{\vec{\alpha} \geq 0, \vec{\beta}} L(\vec{x}, \vec{\alpha}, \vec{\beta}) \quad (3.190)$$

weak duality

$$= \min_{\vec{x}} \max_{\vec{\alpha} \geq 0, \vec{\beta}} (f(\vec{x}) + \sum_i \alpha_i g_i(\vec{x}) + \sum_j \beta_j h_j(\vec{x})) \quad (3.191)$$

$$= \min_{\vec{x}} \left(f(\vec{x}) + \max_{\vec{\alpha} \geq 0, \vec{\beta}} \left(\sum_i \alpha_i g_i(\vec{x}) + \sum_j \beta_j h_j(\vec{x}) \right) \right) \quad (3.192)$$

$$= \min_{\vec{x}} \left(f(\vec{x}) + \max_{\vec{\alpha} \geq 0, \vec{\beta}} \begin{cases} \sum_i \alpha_i (\leq 0) + \sum_j \beta_j (= 0) & \text{if } \vec{x} \text{ is primal feasible} \\ \sum_i \alpha_i (> 0) + \sum_j \beta_j (\neq 0) & \text{if } \vec{x} \text{ is } g_i, h_j \text{ infeasible} \\ \sum_i \alpha_i (\leq 0) + \sum_j \beta_j (\neq 0) & \text{if } \vec{x} \text{ is } h_j \text{ infeasible} \\ \sum_i \alpha_i (> 0) + \sum_j \beta_j (= 0) & \text{if } \vec{x} \text{ is } g_i \text{ infeasible} \end{cases} \right) \quad (3.193)$$

$$= \min_{\vec{x}} \left(f(\vec{x}) + \begin{cases} 0 & \text{if } \vec{x} \text{ is primal feasible} \\ \infty & \text{if } \vec{x} \text{ is primal infeasible} \end{cases} \right) \quad (3.194)$$

$$= \min_{\vec{x}} f(\vec{x}) \text{ with } \vec{x} \text{ is primal feasible} \quad (3.195)$$

$$\max_{\vec{\alpha} \geq 0, \vec{\beta}} (f(\vec{x}) + \sum_i \alpha_i g_i(\vec{x}) + \sum_j \beta_j h_j(\vec{x})) \quad (3.196)$$

is a convex function of \vec{x}

- $g_i(\vec{x})$'s are convex function of \vec{x}
- $\sum_i \alpha_i g_i(\vec{x})$ is the non-negative weighted sums of convex functions, which is also convex
- $h_j(\vec{x})$'s are affine function, which are convex of \vec{x}
- $\sum_j \beta_j h_j(\vec{x})$ is convex regardless of the sign of β_j 's
- (.) is convex since the sum of convex functions is convex
- max. of convex functions is also convex

Dual Problem

for any fixed $\vec{\alpha}', \vec{\beta}'$ with $\alpha'_i \geq 0, \forall i$

$$\min_{\vec{x}} \max_{\vec{\alpha} \geq 0, \vec{\beta}} L(\vec{x}, \vec{\alpha}, \vec{\beta}) \geq \min_{\vec{x}} L(\vec{x}, \vec{\alpha}', \vec{\beta}') \quad (3.197)$$

because $\max \geq$ any

for best $\vec{\alpha} \geq 0, \vec{\beta}$

$$\min_{\vec{x}} \max_{\vec{\alpha} \geq 0, \vec{\beta}} L(\vec{x}, \vec{\alpha}, \vec{\beta}) \geq \max_{\vec{\alpha}' \geq 0, \vec{\beta}'} \min_{\vec{x}} L(\vec{x}, \vec{\alpha}', \vec{\beta}') \quad (3.198)$$

because best is one of any

this is the **weak duality**

$$\min_{\vec{x}} (f(\vec{x}) + \sum_i \alpha_i g_i(\vec{x}) + \sum_j \beta_j h_j(\vec{x})) \quad (3.199)$$

is a concave function of $\vec{\alpha}, \vec{\beta}$

- for fixed \vec{x} , it is a affine function of $\vec{\alpha}, \vec{\beta}$, hence concave

- min. of concave functions is concave
- “=” holds when satisfy certain **constraint qualifications**, this is **strong duality**

a number of different “constraint qualification” exist, of which the most commonly invoked constraint qualification is known as **Slater’s condition**: there exists some feasible primal sol. \vec{x}^* for which all inequality constraints are strictly satisfied

$$g_i(\vec{x}^*) < 0, \forall i \quad (3.200)$$

in practice, nearly all convex problems satisfy some type of constraint qualification, and hence the primal and dual problems have the same optimal value

Complementary slackness

one particularly interesting consequence of strong duality for convex optimization problems is a property known as **complementary slackness**

$$\alpha_i^* g_i(\vec{x}^*) = 0, \forall i \quad (3.201)$$

equivalent writing ways of complementary slackness

$$\alpha_i^* > 0 \rightarrow g_i(\vec{x}^*) = 0 \quad (3.202)$$

$$g_i(\vec{x}^*) < 0 \rightarrow \alpha_i^* = 0 \quad (3.203)$$

the KKT Condition

suppose that $\vec{x}^*, \vec{\alpha}^*, \vec{\beta}^*$ satisfy the following conditions:

- primal feasibility

$$g_i(\vec{x}^*) \leq 0, \forall i \quad (3.204)$$

$$h_j(\vec{x}^*) = 0, \forall j \quad (3.205)$$

- dual feasibility

$$\alpha_i^* \geq 0, \forall i \quad (3.206)$$

- complementary slackness

$$\alpha_i^* g_i(\vec{x}^*) = 0, \forall i \quad (3.207)$$

- Lagrangian stationarity

$$\nabla_{\vec{x}} L(\vec{x}^*, \vec{\alpha}, \vec{\beta}) = \vec{0} \quad (3.208)$$

then \vec{x}^* is primal opt. and $\vec{\alpha}^*, \vec{\beta}^*$ are dual opt.

futhermore, if strong duality holds, then any primal optimal \vec{x}^* and dual optimal $\vec{\alpha}^*, \vec{\beta}^*$ must satisfy these conditions

constraint qualifications
strong duality
Slater’s condition
complementary slackness

Ex.

$$\min_{\vec{x} \in \mathbb{R}^2} x_1^2 + x_2 \quad (3.209)$$

$$\text{st. } 2x_1 + x_2 \geq 4 \quad (3.210)$$

$$x_2 \geq 1 \quad (3.211)$$

Sol. note all these three are convex functions

$$L(\vec{x}, \vec{\alpha}) = x_1^2 + x_2 + \alpha_1(4 - 2x_1 - x_2) + \alpha_2(1 - x_2) \quad (3.212)$$

dual form

$$\max_{\vec{\alpha} \geq 0} \min_{\vec{x}} L(\vec{x}, \vec{\alpha}) = \max_{\vec{\alpha} \geq 0} \min_{\vec{x}} x_1^2 + x_2 + \alpha_1(4 - 2x_1 - x_2) + \alpha_2(1 - x_2) \quad (3.213)$$

$$\frac{\partial}{\partial x_1} L(\vec{x}, \vec{\alpha}) = 2x_1 - 2\alpha_2 \stackrel{\text{set}}{=} 0 \quad (3.214)$$

$$\therefore x_1 = \alpha_1 \quad (3.215)$$

$$\frac{\partial}{\partial x_2} L(\vec{x}, \vec{\alpha}) = 1 - \alpha_1 - \alpha_2 \stackrel{\text{set}}{=} 0 \quad (3.216)$$

$$\therefore \alpha_1 + \alpha_2 = 1 \quad (3.217)$$

$$\max_{\vec{\alpha} \geq 0} \min_{\vec{x}} x_1^2 + x_2 + \alpha_1(4 - 2x_1 - x_2) + \alpha_2(1 - x_2) = \max_{\vec{\alpha} \geq 0} -\alpha_1^2 + 4\alpha_1 \quad (3.218)$$

final form

$$\max_{\vec{\alpha}} -\alpha_1^2 + 4\alpha_1 + \alpha_2 \quad (3.219)$$

$$\text{st. } \alpha_1 \geq 0 \quad (3.220)$$

$$\alpha_2 \geq 0 \quad (3.221)$$

$$\alpha_1 + \alpha_2 = 1 \quad (3.222)$$

Part II

Machine Learning Foundation

Chapter 4

When Can Machines Learn

4.1 The Learning Problem

4.1.1 What is Machine Learning

From Learning to Machine Learning

Learning: observations → learning (with experience) → acquire skill.

Machine Learning: data → ML (with experience computed from data) → improved perf. measure.

Eg., learn to recognize tree.

Arthur Samuel's def.: Field of study that gives computers the ability to learn w/o being explicitly programmed

Tom M. Mitchell's def.: A computer program is said to learn from experience E wrt. some class of tasks T and perf. measure P, if its perf. at tasks in T, as measured by P, improves with experience E.

Why Use Machine Learning

When human cannot program the system manually: navigating on Mars.

When human cannot def. the sol. easily: speech/ visual recognition.

When needing rapid decs. that humans cannot do: high-freq. trading.

When needing to be user-oriented in a massive scale: consumer-targetted marketing.

Key Essence to Dec. Whether to Use Machine Learning

pattern: exists some underlying pattern, so perf. measure can be improved.

def.: no programmable easy def., so ML is needed.

data: there is data about the pattern, so ML has some inputs to learn from.

4.1.2 Components of Machine Learning

unknown pattern to be learned \Leftrightarrow target function: $f : \mathcal{X} \mapsto \mathcal{Y}$
hypothesis h^* \Leftrightarrow skill with hopefully good perf.: learned formula to be used.
ML: use data to compute hypothesis h^* to approx. target f .

4.1.3 Machine Learning and Other Fields

ML and DM

DM: use huge data to find property that is interesting.

- if interesting property same as compute hypothesis to approx. target f , then ML = DM.
- if interesting property related to compute hypothesis to approx. target f , then DM helps ML and vice versa.
- hard to distinguish ML and DM in reality.
- traditional DM also focuses on efficient computation in large database.

ML and AI

AI: compute sth. that shows intelligent behavior.

- ML is one possible route to realize AI: $h^* \approx f$ is sth. that shows intelligent behavior.

ML and Stat.

Stat.: use data to make inference about an unknown process.

- Stat. can be used to achieve ML: h^* is an inference outcome, f is sth. unknown.
- traditional statistics also focus on provable results with math assumptions, and care less about computation.

4.2 Learning to Answer Yes/ No

4.2.1 Perceptron Hypothesis Set

linear binary classifiers/ hyperplanes:

$$h(\vec{x}) = \text{sign}(\vec{\theta}^T \vec{x}) \in \{1, -1\} \quad (4.1)$$

hypothesis set: set of all classifiers considered by the learning alg.

$$\mathcal{H} = \{h : h(\vec{x}) = \text{sign}(\vec{\theta}^T \vec{\theta}), \vec{\theta} \in \mathbb{R}^{n+1}\} \quad (4.2)$$

Algorithm 8 PLA

idea: start with some h , repr. by its init. vec. $\vec{\theta}$, and correct its mistakes iteratively on D .

```

init.  $\vec{\theta}$ , say  $\vec{0}$ .
while there is mistake in  $D$  made by curr.  $\vec{\theta}$ 
  for  $\{\vec{x}^{(i)}, y^{(i)}\} \in D$  in naive cycle (1, ..., m) or precomputed rand. cycle
    if  $\text{sign}(\vec{\theta}^T \vec{x}^{(i)}) \neq y^{(i)}$  // a mistake has been found
       $\vec{\theta} \leftarrow \vec{\theta} + y^{(i)} \vec{x}^{(i)}$  // try to correct the mistake
return  $h(\vec{x}) = \text{sign}(\vec{\theta}^T \vec{x})$ 

```

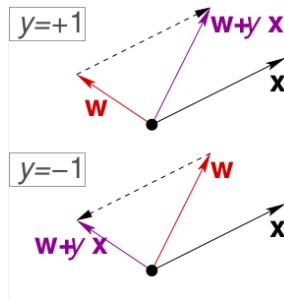


Figure 4.1: The update rule tries to correct the mistake.

4.2.2 Perceptron Learning Alg. (PLA)

After running the PLA and achieving zero errors, we will be able to express the final weight vector as the initial weight vector plus an integer combination of the inputs.

After one update: $\vec{\theta}_{t+1} = \vec{\theta}_t + y^{(i)} \vec{x}^{(i)}$. See Fig. 4.1.

- $y^{(i)} \vec{\theta}_{t+1}^T \vec{x}^{(i)} \geq y^{(i)} \vec{\theta}_t^T \vec{x}^{(i)}$
- ideally we want $y^{(i)} \vec{\theta}_{t+1}^T \vec{x}^{(i)} > 0$, but this may not always be true.

PLA halts (ie., no more mistakes) $\rightarrow D$ allows some $\vec{\theta}$ to make no mistake.

- call such D lin. separable.

4.2.3 Guarantee of PLA

Assume lin. sep. $D \xrightarrow{?} \text{PLA}$ always halt.

PLA fact: $\vec{\theta}_T$ Gets More Aligned with $\vec{\theta}^*$

lin. sep. $D \iff$ exists perfect $\vec{\theta}^*$ st. $\text{sign}(\vec{\theta}^{*T} \vec{x}^{(i)}) = y^{(i)}$
perfect $\vec{\theta}^*$ hence every $\vec{x}^{(i)}$ correctly away from line:

$$y^{(i)} \vec{\theta}^{*T} \vec{x}^{(i)} \geq \min_i y^{(i)} \vec{\theta}^{*T} \vec{x}^{(i)} > 0 \quad (4.3)$$

$\vec{\theta}^{\star T} \vec{\theta}_T$ inc. by updating with any $\{\vec{x}^{(i)}, y^{(i)}\}$, seems $\vec{\theta}_T$ appears more aligned with $\vec{\theta}^{\star}$

$$\vec{\theta}^{\star T} \vec{\theta}_T = \vec{\theta}^{\star T} (\vec{\theta}_{T-1} + y^{(i)} \vec{x}^{(i)}) \quad (4.4)$$

$$\geq \vec{\theta}^{\star T} \vec{\theta}_{T-1} + \min_i y^{(i)} \vec{\theta}^{\star T} \vec{x}^{(i)} // use \ lemma \quad (4.5)$$

$$// which is \dot{z} \vec{\theta}^{\star T} \vec{\theta}_{T-1} \quad (4.6)$$

$$> \vec{\theta}^{\star T} \vec{\theta}_{T-2} + 2 \cdot \min_i y^{(i)} \vec{\theta}^{\star T} \vec{x}^{(i)} \quad (4.7)$$

$$> \vec{\theta}^{\star T} \vec{\theta}_0 + T \cdot \min_i y^{(i)} \vec{\theta}^{\star T} \vec{x}^{(i)} // applying \ T \ times \quad (4.8)$$

$$(4.9)$$

PLA fact: $\vec{\theta}_T$ Does Not Grow Too Fast

$\vec{\theta}_t$ changed only when mistakes $\iff \text{sign}(\vec{\theta}_t^T \vec{x}^{(i)}) \neq y^{(i)} \iff y^{(i)} \vec{\theta}_t^T \vec{x}^{(i)} \leq 0$
 mistake limits $\|\vec{\theta}_T\|^2$ grows, even when updating with longest $\vec{x}^{(i)}$

$$\|\vec{\theta}_T\|^2 = \|\vec{\theta}_{T-1} + y^{(i)} \vec{x}^{(i)}\|^2 \quad (4.10)$$

$$= \|\vec{\theta}_{T-1}\|^2 + 2y^{(i)} \vec{\theta}_{T-1}^T \vec{x}^{(i)} + \|y^{(i)} \vec{x}^{(i)}\|^2 \quad (4.11)$$

$$// expand the sq. term \quad (4.12)$$

$$\leq \|\vec{\theta}_{T-1}\|^2 + 0 + \|y^{(i)} \vec{x}^{(i)}\|^2 \quad (4.13)$$

$$// update rule only be used when $y^{(i)} \vec{\theta}_{T-1}^T \vec{x}^{(i)} \leq 0 \quad (4.14)$$$

$$\leq \|\vec{\theta}_{T-1}\|^2 + \max_i \|\vec{x}^{(i)}\|^2 \quad (4.15)$$

$$// use the longest $\vec{x}^{(i)} \quad (4.16)$$$

$$// $y^{(i)}$ doesn't influence the norm \quad (4.17)$$

$$\leq \|\vec{\theta}_0\|^2 + T \cdot \max_i \|\vec{x}^{(i)}\|^2 // applying \ T \ times \quad (4.18)$$

$$(4.19)$$

Upper-Bound on T

start from $\vec{\theta}_0 = \vec{0}$, after T mistake corrections,

$$1 \geq \frac{\vec{\theta}^{\star T}}{\|\vec{\theta}^{\star}\|} \frac{\vec{\theta}_T}{\|\vec{\theta}_T\|} \quad (4.20)$$

$$\geq \frac{T \cdot \min_i y^{(i)} \vec{\theta}^{\star T} \vec{x}^{(i)}}{\|\vec{\theta}^{\star}\| \cdot \sqrt{T} \max_i \|\vec{x}^{(i)}\|} \quad (4.21)$$

$$// substitute two facts above \quad (4.22)$$

$$\geq \sqrt{T} \frac{\min_i y^{(i)} \vec{\theta}^{\star T} \vec{x}^{(i)}}{\|\vec{\theta}^{\star}\| \max_i \|\vec{x}^{(i)}\|} \quad (4.23)$$

$$= \sqrt{T} \cdot \frac{\gamma}{R} \quad (4.24)$$

$$\text{// def. } \gamma = \min_i \frac{y^{(i)} \vec{\theta}^{\star T} \vec{x}^{(i)}}{||\vec{\theta}^{\star}||} \quad (4.25)$$

$$\text{// def. } R = \max_i ||\vec{x}^{(i)}|| \quad (4.26)$$

(4.27)

Thus, T is an upper-bound, # mistakes that PLA tries to corrects

$$T \leq \frac{R^2}{\gamma^2} \quad (4.28)$$

lin. sep. on D

- inner prod. of $\vec{\theta}_T$ and $\vec{\theta}^{\star}$ grows fast
- length of $\vec{\theta}_T$ grows slowly
- PLA lines are more and more aligned with $\vec{\theta}^{\star} \rightarrow$ halts
- no mistake eventually

4.2.4 Non-Sep. Data

More about PLA

Pros:

- simple to implement, fast, works in any dim. n

Cons:

- assume lin. sep. on D : property unknown in advance (no need for PLA if we know $\vec{\theta}^{\star}$)
- not fully sure how long halting takes: γ depends on $\vec{\theta}^{\star}$

What if D is not lin. sep.

line with noise tol.: NP-hard to sol., unfortunately

$$\vec{\theta}^{\star} = \arg \min_{\vec{\theta}} \sum_{i=1}^m 1\{y^{(i)} \neq \text{sign}(\vec{\theta}^T \vec{x}^{(i)})\} \quad (4.29)$$

Pocket Alg.

If D is actually lin. sep.:

- pocket on D is slower than PLA: pocket needs to check whether $\vec{\theta}$ is better than $\vec{\theta}^{\star}$ in each iteration
- $\vec{\theta}_{POCKET}$ is the same as $\vec{\theta}_{PLA}$, both making no mistakes.

Note: more info. can be found in wiki¹

¹<https://en.wikipedia.org/wiki/Perceptron#Definitions>

Algorithm 9 Pocket Algorithm

idea: mod. PLA by keeping the (somewhat) best weights $\vec{\theta}^*$ in pocket (greedy approach)

```

init.  $\vec{\theta}^*$ , say  $\vec{0}$ . // init. pocket weights
while true
  for  $\{\vec{x}^{(i)}, y^{(i)}\} \in D$  randomly
    if  $\text{sign}(\vec{\theta}^T \vec{x}^{(i)}) \neq y^{(i)}$  // a mistake has been found
       $\vec{\theta} \leftarrow \vec{\theta} + y^{(i)} \vec{x}^{(i)}$  // try to correct the mistake
      if  $\vec{\theta}$  makes fewer mistakes than  $\vec{\theta}^*$ 
         $\vec{\theta}^* \leftarrow \vec{\theta}$ 
    if there is enough iterations on  $\vec{\theta}$ 
      return  $h(\vec{x}) = \text{sign}(\vec{\theta}^* T \vec{x})$ 

```

4.3 Types of Learning

4.3.1 Learning with Diff. Ouput Space \mathcal{Y}

Binary Classification

- $\mathcal{Y} = \{-1, +1\}$
- core and important problem with many tools as building block of other tools

Multiclass Classification

- $\mathcal{Y} = \{1, 2, \dots, K\}$
- many applications in practice, especially for recognition

Regression

- $\mathcal{Y} = \mathbb{R}$, or $\mathcal{Y} = [Lower, Upper] \subset \mathbb{R}$
- deeply studied in stat.
- core and important problem with many stat. tools as building block of other tools

Structured Learning

- Seq. Tagging Problem
- multiclass classification: word \rightarrow word class, eg., I \rightarrow pronoun, love \rightarrow verb
- structuered learning: sentence \rightarrow structure (class of each word). eg., I love ML \rightarrow (PVN)
- $\mathcal{Y} = PVN, PVP, NVN, PV, \dots$, not including VVVV

- huge multiclass classification problem (structure = hyperclass) w/o explicit class def.
- other structured learning problem: protein data \rightarrow protein folding, speech data \rightarrow speech parse tree

4.3.2 Learning with Diff. Data Label $y^{(i)}$

Supervised Learning

- every $\vec{x}^{(i)}$ comes with corresponding $y^{(i)}$

Unsupervised Learning

- learning w/o $y^{(i)}$
- diverse, w/ possibly very diff. perf. goals
- clustering: \approx unsupervised multiclass classification
- density estimation: \approx unsupervised bounded regression
- outlier detection: \approx unsupervised binary classification

Semi-supervised Learing

- labeled data is expensive and hard to acquire, unlabeled data is often copiously available
- learning with some $y^{(i)}$
- unlabeled data can be used to estimate the low-dim. manifold structure of the data
- unlabeled data can be used to estimate the joint prob. distr. of features

Self-Training procedure

- initial labeled set D_L and unlabeled set D_U
 - use algorithm A on the current labeled set D_L to identify the k instances in the unlabeled data D_U for which the classifier A is the most confident
 - assign labels to the most k confidently predicted instances and add them to D_L remove these instances from D_U
- overfitting: addition of predicted labels may propagate errors

Co-Training procedure

- two disjoint feature groups: F_1 and F_2
- labeled sets L_1 and L_2
- train classifier A_1 using labeled set L_1 and feature set F_1 , and add k most confidently predicted instances from unlabeled set $U - L_2$ to training data set L_2 for classifier A_2

- train classifier A_2 using labeled set L_2 and feature set F_2 , and add k most confidently predicted instances from unlabeled set $U - L_1$ to training data set L_1 for classifier A_1

Reinforcement Learning

- learning w/ partial/ implicit info. (often sequentially)
- cannot easily show $y^{(i)}$
- learning use $\{\vec{x}, y, goodness\}$

4.3.3 Learning with Diff. Protocol $f \mapsto \{\vec{x}^{(i)}, y^{(i)}\}$

Batch Learning

- learn from all known data, predict with fixed h
- a very common protocol

Online Learning

- hypothesis improves through receiving data instances sequentially
- observe $\vec{x}^{(i)}$, predict $h(\vec{x}^{(i)})$, receive desired label $y^{(i)}$, and then update h using $\{\vec{x}^{(i)}, y^{(i)}\}$
- PLA can be easily adapted to online protocol
- reinforcement learning is often done online
- can adapt to changing user preference
- be advantageous when there is a lot of redundancy in the data

Active Learning

- learning by asking when confused
- improve hypothesis with fewer labels (hopefully) by labeling the most informative data
- eg., label those instances for which the value of the label is the least certain, see Fig. 4.2, 4.3

4.3.4 Learning with Diff. Input Space \mathcal{X}

Concrete Features

- each dim. of $\vec{x} \in \mathbb{R}^n$ repr. sophisticated physical meaning
- often including human intelligence on the learning task
- the easy ones for ML

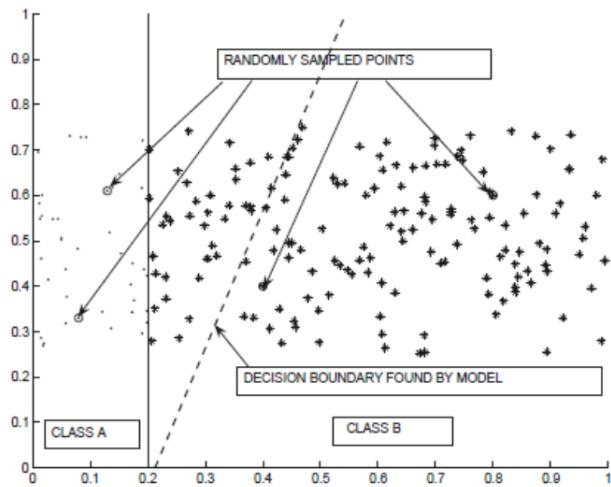


Figure 4.2: Random Sampling

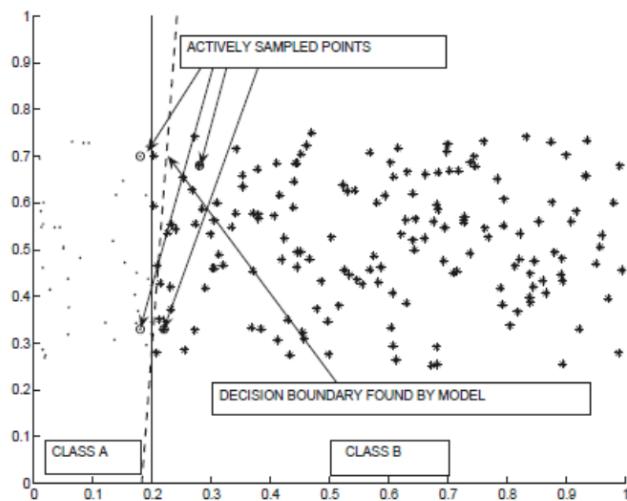


Figure 4.3: Active Sampling

Raw Features

- eg. digit recognition problem
- simple physical meaning: more difficult for ML
- often need human or machines to convert to concrete ones

Abstract Features

- eg. movie rating prediction problem

- no physical meaning: even more difficult for ML
- again need feature conversion/ extraction/ construction

4.4 Feasibility of Learning

4.4.1 Learning is Impossible?

No free lunch: learning from D (to infer sth. outside D) is doomed if any unknown f can happen
need additional assumption

4.4.2 Prob. to the Rescue

consider a bin of many many orange and green marbles, we don't know the orange portion (prob.), can you infer the orange prob.?

Inferring Orange Prob.

bin:

- assume $P\{\text{orange}\} = \mu$, $P\{\text{green}\} = 1 - \mu$
- μ unknown

sample:

- m marbles sampled indep.
- orange fraction = $\bar{\mu}$, green fraction = $1 - \bar{\mu}$
- $\bar{\mu}$ known

Does in-sample $\bar{\mu}$ say anything about out-of-sample μ

possibly not: sample can be mostly green while bin is mostly orange
probably yes: in-sample $\bar{\mu}$ likely close to unknown μ

Hoeffding's Ineq.

let Z_1, Z_2, \dots, Z_m be m rv.

$$Z_i = \begin{cases} 1 & \text{if get orange} \\ 0 & \text{if get green} \end{cases} \quad (4.30)$$

$$Z_i \stackrel{\text{iid}}{\sim} Ber(\mu) \quad (4.31)$$

$$\Pr(Z_i = 1) = \mu \quad (4.32)$$

$$\Pr(Z_i = 0) = 1 - \mu \quad (4.33)$$

let

$$\bar{\mu} = \frac{1}{m} \sum_{i=1}^m Z_i \quad (4.34)$$

be the mean of rv.

in big sample (m large), $\bar{\mu}$ is probably close to μ (within any fixed $\varepsilon > 0$)

$$P(|\mu - \bar{\mu}| > \varepsilon) \leq 2 \exp(-2\varepsilon^2 m) \quad (4.35)$$

the statement $\bar{\mu} = \mu$ is probably approximately correct (PAC)

- valid for all m and ε
- does not depend on μ : no need to known μ
- large sample size m or looser gap $\varepsilon \rightarrow$ higher prob. for $\mu \approx \bar{\mu}$

this is also called Chernoff bound

4.4.3 Connection to Learning

Table 4.1: learning

bin	learning
unknown orange prob. μ	fixed hypothesis $h(\vec{x}) \stackrel{?}{=} \text{unknown target } f$
marble in bin	$\vec{x} \in \mathcal{X}$
orange	h is wrong $\iff h(\vec{x}) \neq f(\vec{x})$
green	h is right $\iff h(\vec{x}) = f(\vec{x})$
size- m sample from bin of iid. marbles	check h on D with iid. $\vec{x}^{(i)}$

see Tab. 6.2

if large m and iid. $\vec{x}^{(i)}$, can probably infer unknown out of sample err.
 $J_{out}(h)$ by known in sample err. $J_{in}(h)$.

$$J_{out}(h) = E_{\vec{x} \sim P}[1\{h(\vec{x}) \neq f(\vec{x})\}] = \Pr(h(\vec{x}) \neq f(\vec{x})) \quad (4.36)$$

$$J_{in}(h) = \frac{1}{m} \sum_{i=1}^m 1\{h(\vec{x}^{(i)}) \neq f(\vec{x}^{(i)})\} \quad (4.37)$$

$J_{in}(h)$ is called training error, empirical risk or empirical error
 $J_{out}(h)$ is called generalization err.

The Formal Guarantee

let Z_1, Z_2, \dots, Z_m be m rv.

$$Z_i \stackrel{\text{iid}}{\sim} Ber(J_{out}(h)) \quad (4.38)$$

$$Z_i = 1\{h(\vec{x}^{(i)}) \neq y^{(i)}\} \quad (4.39)$$

$$\Pr(Z_i = 1) = J_{out}(h) \quad (4.40)$$

$$\Pr(Z_i = 0) = 1 - J_{out}(h) \quad (4.41)$$

let

$$J_{in}(h) = \frac{1}{m} \sum_{i=1}^m Z_i = \frac{1}{m} \sum_{i=1}^m 1\{h(\vec{x}^{(i)}) \neq y^{(i)}\} \quad (4.42)$$

be the mean of rv.

for any fixed h , in bid data (m is large), $J_{in}(h)$ is probably close to $J_{out}(h)$ (within ε)

$$\Pr(|J_{in}(h) - J_{out}(h)| > \varepsilon) \leq 2 \exp(-2\varepsilon^2 m) \quad (4.43)$$

- valid for all m and ε
- does not depend on $J_{out}(h)$: no need to known $J_{out}(h)$, f and P and stay unknown
- the statement $J_{in}(h) = J_{out}(h)$ is probably approximately correct (PAC)

if $J_{in}(h) \approx J_{out}(h)$ and $J_{in}(h)$ is small
pick

$$h^* = \arg \min_{h \in \mathcal{H}} J_{in}(h) \quad (4.44)$$

we call this process empirical risk min. (ERM)

Can we claim good learning ($h \approx f$) by $J_{in}(h) \approx J_{out}(h)$ PAC for fixed h ?

- Yes: if $J_{in}(h)$ is small
 $\rightarrow J_{out}(h)$ is small and A pick the h as h^*
 $\rightarrow h \approx f$ wrt. P PAC
- No: if A forced to pick THE h as h^*
 $\rightarrow J_{in}(h)$ almost always not small
 $\rightarrow h \neq f$ PAC!

	\mathcal{D}_1	\mathcal{D}_2	...	\mathcal{D}_{1126}	...	\mathcal{D}_{5678}	Hoeffding
h_1	BAD					BAD	$\mathbb{P}_{\mathcal{D}}[\text{BAD } \mathcal{D} \text{ for } h_1] \leq \dots$
h_2		BAD					$\mathbb{P}_{\mathcal{D}}[\text{BAD } \mathcal{D} \text{ for } h_2] \leq \dots$
h_3	BAD	BAD				BAD	$\mathbb{P}_{\mathcal{D}}[\text{BAD } \mathcal{D} \text{ for } h_3] \leq \dots$
...							
h_M	BAD					BAD	$\mathbb{P}_{\mathcal{D}}[\text{BAD } \mathcal{D} \text{ for } h_M] \leq \dots$
all	BAD	BAD				BAD	?

Figure 4.4: BAD D for many h

4.4.4 Connection to Real Learning

real learning: A shall make choices $\in \mathcal{H}$ (like PLA) rather than being forced to pick one h

Coin Game

150 students, everyone flips a fair coin 5 times, $P\{\text{one of the students gets 5H of for his coin}\} = 1 - (\frac{31}{32})^{150} > 99\%$

BAD sample: J_{out} and J_{in} far away \rightarrow get worse when involving multiple choice

BAD Sample and BAD Data

BAD Sample: eg., $J_{out} = \frac{1}{2}$, but getting all H's ($J_{in} = 0$)

BAD D for one h :

- $J_{out}(h)$ and $J_{in}(h)$ far away
- eg., J_{out} big (far from f), but J_{in} small (correct on most examples)
- Hoeffding ineq. says, $P\{\text{for fixed } h, \text{BAD } D \text{ was selected}\}$ is small

BAD D for many h , see Fig 4.4:

- for that D , there exists at least one h st. $J_{out}(h)$ and $J_{in}(h)$ far away
- eg., h_1 is good, but he meets BAD D , thus $J_{in}(h_1)$ is high; but h_2 is bad, he meets BAD D , $J_{in}(h_2)$ is small, A wrongly choose h_2 as h^*
- for M hypotheses, bound of $P_D\{\text{BAD } D\}$?

the Union Bound

let A_1, A_2, \dots, A_M be M def. events (that may not be indep.)

$$\Pr(A_1 \cup A_2 \cup \dots \cup A_M) \leq \Pr(A_1) + \Pr(A_2) + \dots + \Pr(A_M) \quad (4.45)$$

in prob. theory, the union bound is usually stated as an axiom

Bound of BAD Data

if $|\mathcal{H}| = M$ finite

$$\Pr_D\{\text{BAD } D\} \quad (4.46)$$

$$= \Pr(\exists h \in \mathcal{H} : |J_{in}(h) - J_{out}(h)| > \varepsilon) \quad (4.47)$$

$$= \Pr_D\{\text{BAD } D \text{ for } h_1 \vee \text{BAD } D \text{ for } h_2 \vee \dots \vee \text{BAD } D \text{ for } h_M\} \quad (4.48)$$

$$\leq \Pr_D\{\text{BAD } D \text{ for } h_1\} + \Pr_D\{\text{BAD } D \text{ for } h_2\} + \dots \quad (4.49)$$

$$\Pr_D\{\text{BAD } D \text{ for } h_M\} \quad // \text{ union bound} \quad (4.50)$$

$$\leq 2 \exp(-2\varepsilon^2 m) + 2 \exp(-2\varepsilon^2 m) + \dots + 2 \exp(-2\varepsilon^2 m) \quad (4.51)$$

$$= 2M \exp(-2\varepsilon^2 m) \quad (4.52)$$

finite-bin ver. of Hoeffding, valid for all M, m, ε

does not depend on any $J_{out}(h_j)$

if M is not too big, and m large enough, $\Pr_D\{\text{BAD } D\}$ is small, learning possible

for whatever h^* picked by A , $J_{out}(h^*) = J_{in}(h^*)$ is PAC, regardless of A

most reasonable A (like PLA/ pocket): pick the h_j with lowest $J_{in}(h_j)$

as h^*

Chapter 5

Why Can Machines Learn

5.1 Training vs. Testing

5.1.1 Recap and Preview

2 central questions

for batch & supervised learning, $h^* \approx f \iff J_{out}(h^*) \approx 0$, achieved through $J_{in}(h^*) \approx J_{out}(h^*)$ and $J_{in}(h^*) \approx 0$

learning split to 2 central questions

- 1. can we make sure $J_{in}(h^*) \approx J_{out}(h^*)$
- 2. can we make sure $J_{in}(h^*) \approx 0$

What role does M play for the 2 questions

small M

- 1. Yes! $\Pr\{\text{BAD}\} \leq 2M \exp(\dots)$
- 2. No! too few choice

large M

- 1. No! $\Pr\{\text{BAD}\} \leq 2M \exp(\dots)$
- 2. Yes! many choices

Preview

Know

$$\Pr(|J_{in}(h^*) - J_{out}(h^*)| > \varepsilon) \leq 2M \exp(-2\varepsilon^2 m) \quad (5.1)$$

- let $\Pr(|J_{in}(h^*) - J_{out}(h^*)| > \varepsilon) \leq \delta$ be the tol. BAD prob.

• data size needed is $m = \frac{1}{2\varepsilon^2} \log \frac{2M}{\delta}$

Todo

- establish a finite quantity that replaces M

$$\Pr(|J_{in}(h) - J_{out}(h)| > \varepsilon) \leq 2 \cdot \underset{\mathcal{H}}{\text{grow}}(m) \cdot \exp(-2\varepsilon^2 m) \quad (5.2)$$

- justify the feasibility of learning for inf. M
- study $\text{grow}_{\mathcal{H}}(m)$ to understand its trade-off for right \mathcal{H} , just like M

5.1.2 Effective # of Lines

Where Did Uniform Bound Fail

Uniform Bound

$$\begin{aligned}\Pr\{\text{BAD}\} &= \Pr\{\text{BAD for } h_1 \vee \text{BAD for } h_2 \vee \dots \vee \text{BAD for } h_M\} \\ &= \Pr\{\text{BAD for } h_1\} + \Pr\{\text{BAD for } h_2\} + \dots + \Pr\{\text{BAD for } h_M\}\end{aligned}\quad (5.3)$$

BAD events overlapping for similar hypotheses $h_1 \approx h_2$, see Fig. 5.1, why?

- $J_{out}(h_1) \approx J_{out}(h_2)$
- for most D , $J_{in}(h_1) \approx J_{in}(h_2)$
- union bound over-estimating
- to account for overlap, can we group similar hypotheses by kind?

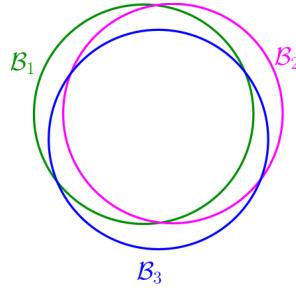


Figure 5.1: BAD events overlapping for similar hypotheses

How Many Lines Are there

$$\mathcal{H} = \{\text{all lines in } \mathbb{R}^2\}$$

how many lines? inf

how many kinds of lines if viewed from one input vec. $\vec{x}^{(1)}$? 2 kinds, see Fig. 5.2, one judge $\vec{x}^{(1)}$ as +1, one for -1

how many kinds of lines if viewed from two inputvecs. $\vec{x}^{(1)}, \vec{x}^{(2)}$? 4 kinds, see Fig. 5.3

how many kinds of lines if viewed from three inputvecs. $\vec{x}^{(1)}, \vec{x}^{(2)}, \vec{x}^{(3)}$? 8 kinds, see Fig. 5.4, but not always, see Fig. 5.5

how many kinds of lines if viewed from four inputvecs. $\vec{x}^{(1)}, \vec{x}^{(2)}, \vec{x}^{(3)}, \vec{x}^{(4)}$? ≤ 14 kinds, see Fig. 5.6

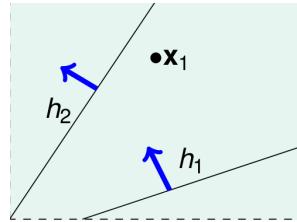


Figure 5.2: how many kinds of lines if viewed from one input vec.

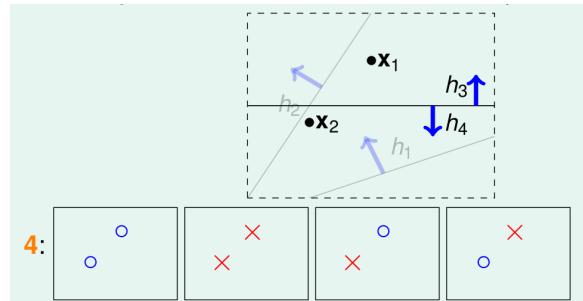


Figure 5.3: how many kinds of lines if viewed from two input vecs.

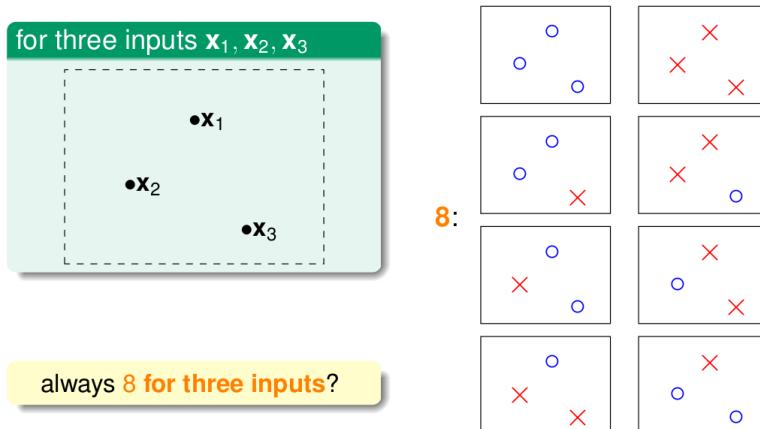


Figure 5.4: how many kinds of lines if viewed from three input vecs.

Effective # of Lines

effective # of lines: max. kinds of lines wrt. m inputs $\{\vec{x}^{(1)}, \vec{x}^{(2)}, \dots, \vec{x}^{(m)}\}$, see Tab. 5.1

- must be $\leq 2^m$
- fin. grouping of inf. many lines $\in \mathcal{H}$
- wish $\Pr(|J_{in}(h^*) - J_{out}(h^*)| > \varepsilon) \leq 2 \cdot \text{eff}(m) \cdot \exp(-2\varepsilon^2 m)$, if
 - $\text{eff}(m)$ can replace M
 - $\text{eff}(m) \ll 2^m$

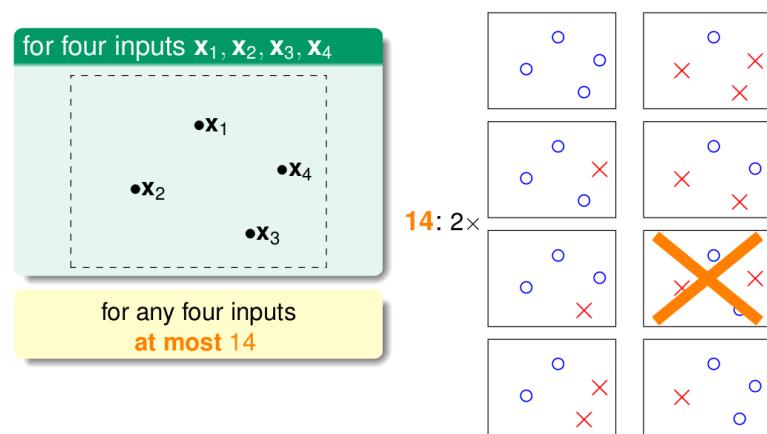
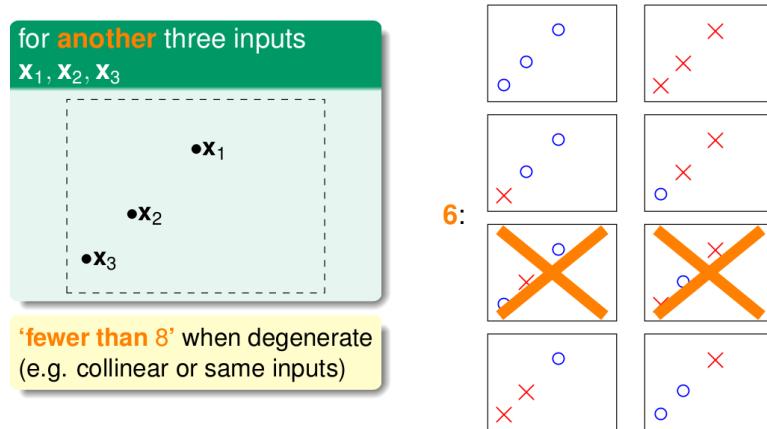


Figure 5.5: for another three input vecs.

learning possible with inf. lines

Table 5.1: lines in \mathbb{R}^2

m	$\text{eff}(m)$
1	2
2	4
3	8
4	$14 < 2^m$

5.1.3 Effective # of Hypotheses

Dichotomies: Mini-Hypotheses

$$\mathcal{H} = \{\text{hypotheses } h : \mathcal{X} \mapsto \{-1, +1\}\} \quad (5.5)$$

call $h(X) = \{h(\vec{x}^{(1)}), h(\vec{x}^{(2)}), \dots, h(\vec{x}^{(m)})\} \in \{-1, +1\}^m$ a dichotomy:
hypothesis limited to the eyes of $\{\vec{x}^{(1)}, \vec{x}^{(2)}, \dots, \vec{x}^{(m)}\}$, see Tab. 5.2

$\mathcal{H}(X)$: all dichotomies implemented by \mathcal{H} on $\{\vec{x}^{(1)}, \vec{x}^{(2)}, \dots, \vec{x}^{(m)}\}$

$|\mathcal{H}(X)|$: candidate for replacing M

Table 5.2: dichotomies: mini-hypotheses

	hypotheses \mathcal{H}	dichotomies $\mathcal{H}(X)$
eg.	all lines in \mathbb{R}^2	$\{\{-1, -1, \dots\}, \dots, \{-1, +1, \dots\}, \dots, \{+1, +1, \dots\}\}$
size	possibly inf.	upper bounded by 2^m

Growth Fcn.

$|\mathcal{H}(X)|$ depends on inputs $\{\vec{x}^{(1)}, \vec{x}^{(2)}, \dots, \vec{x}^{(m)}\}$
growth fcn.

- remove dep. by taking max. of all possible X , it dep. on m and the hypotheses type, see Tab. 5.3
- max. # of dichotomies to the eyes of X
- effective # of hypothesis $\leq \text{grow}_{\mathcal{H}}(m)$ through the eye of m inputs

$$\text{grow}(m) = \max_{\mathcal{H}} |\mathcal{H}(X)| \quad (5.6)$$

Table 5.3: growth fcn.

m	$\text{grow}_{\mathcal{H}}(m)$
1	2
2	4
3	$\max(\dots, 6, 8) = 8$
4	$14 < 2^m$

$\text{grow}_{\mathcal{H}}(m)$ is fin. and upper-bounded by 2^m

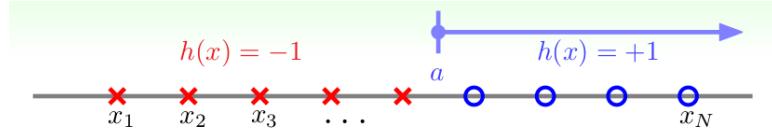


Figure 5.7: pos. rays.

x_1	x_2	x_3	x_4
o	o	o	o
x	o	o	o
x	x	o	o
x	x	x	o
x	x	x	x

Figure 5.8: growth fcn. for pos. rays.

Growth Fcn. for Pos. Rays

$\mathcal{X} = \mathbb{R}$: one dim.

\mathcal{H} contains h , where $h(x) = \text{sign}(x - \theta)$ for threshold θ
pos. half of 1-dim. perceptrons, see Fig 5.7, 5.8

$$\underset{\mathcal{H}}{\text{grow}}(m) = m + 1 \ll 2^m \text{ when } m \text{ is large} \quad (5.7)$$

Growth. Fcn. for pos. and neg. rays (dec. stumps)

$\mathcal{X} = \mathbb{R}$, equivalent to the perceptron hypothesis set in 1-dim.

\mathcal{H} contains h , where $h(x) = s \cdot \text{sign}(x - \theta)$ for threshold θ and sign s

$$\underset{\mathcal{H}}{\text{grow}}(m) = 2(m - 1) + 2 = 2m \quad (5.8)$$

2 dics. in each of the $m - 1$ spots plus all +1 and all -1

Growth Fcn. for pos. intervals

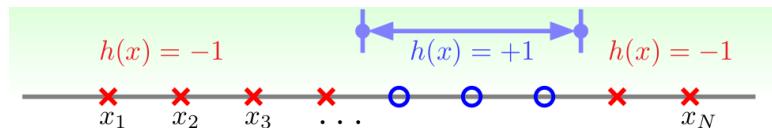


Figure 5.9: pos. ints.

x_1	x_2	x_3	x_4
o	x	x	x
o	o	x	x
o	o	o	x
o	o	o	o
x	o	x	x
x	o	o	x
x	o	o	o
x	x	o	x
x	x	o	o
x	x	x	o
x	x	x	x

Figure 5.10: growth fcn. for pos. ints.

$\mathcal{X} = \mathbb{R}$: one dim.

\mathcal{H} contains h , where $h(x) = +1$ iff. $x \in [l, r)$, -1 o.w.

$$\underset{\mathcal{H}}{\text{grow}}(m) = \binom{m+1}{2} + 1 = \frac{1}{2}m^2 + \frac{1}{2}m + 1 \ll 2^m \text{ when } m \text{ is large} \quad (5.9)$$

Growth Fcn. for Convex Sets

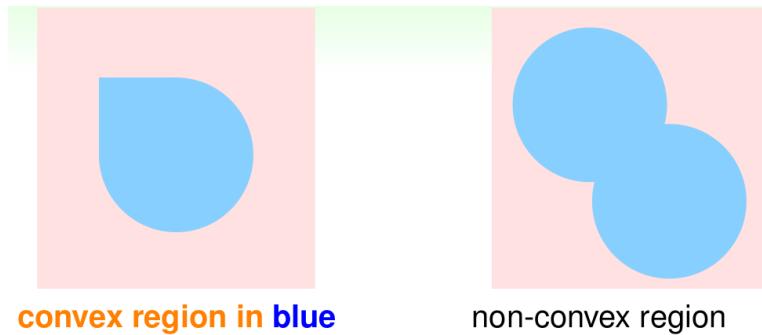


Figure 5.11: cvx. sets.

$\mathcal{X} = \mathbb{R}^2$: two dim.

\mathcal{H} contains h , where $h(\vec{x}) = +1$ iff. \vec{x} in a cvx. region, -1 o.w.

one possible set of m inputs: $\{\vec{x}^{(i)}\}_{i=1}^m$ on a big circle, see Fig 5.12

every dichotomy can be implemented by \mathcal{H} using a cvx region slightly extended from contour of pos. inputs

$$\underset{\mathcal{H}}{\text{grow}}(m) = 2^m \quad (5.10)$$

call those m inputs shattered by \mathcal{H} (\mathcal{H} can realize any labeling on D)

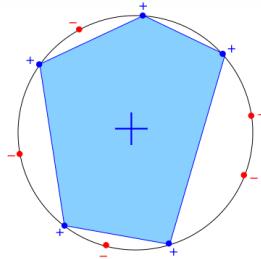


Figure 5.12: growth fcn. for cvx. sets.

5.1.4 Break Point

The Growth Fcn.

what if $\text{grow}_{\mathcal{H}}(m)$ replaces m :

$$\Pr(|J_{in}(h^*) - J_{out}(h^*)| > \varepsilon) \stackrel{?}{\leq} 2 \cdot \underset{\mathcal{H}}{\text{grow}}(m) \cdot \exp(-2\varepsilon^2 m) \quad (5.11)$$

- poly.: good
- exp.: bad

Break Point of \mathcal{H}

$\text{grow}_{\mathcal{H}}(k) = 2^k \iff$ exists k inputs that can be shattered

what do we know about 2-dim. perceptrons now?

- 3 inputs: exists shatter
- 4 inputs: for all, no shatter
- if no k inputs can be shattered by \mathcal{H} , call k a break point for \mathcal{H}
- $\text{grow}_{\mathcal{H}}(k) < 2^k$
- $k+1, k+2, k+3, \dots$ also break points!
- will study min. break point k
- 2-dim. perceptrons: break point 4
- effective price of choice in training: (wishfully) growth fcn. $\text{grow}_{\mathcal{H}}(m)$ with a break point

The Five Break Points

conjecture from Tab. 5.4

- no break point: $\text{grow}_{\mathcal{H}}(m) = 2^m$ (sure!)
- break point k : $\text{grow}_{\mathcal{H}}(m) = O(m^{k-1})$

Table 5.4: The Five Growth Fcns.

	$\text{grow}_{\mathcal{H}}(m)$	break point
pos. rays	$\text{grow}_{\mathcal{H}}(m) = m + 1$	break point at 2
dec. stumps	$\text{grow}_{\mathcal{H}}(m) = 2m$	break point at 3
pos. ints.	$\text{grow}_{\mathcal{H}}(m) = \frac{1}{2}m^2 + \frac{1}{2}m + 1$	break point at 3
cvx. sets	$\text{grow}_{\mathcal{H}}(m) = 2^m$	no break point
2-dim. perceptrons	$\text{grow}_{\mathcal{H}}(m) < 2^m$ in some cases	break points at 4

5.2 Theory of Generalization

5.2.1 Restriction of Break Point

what must be true when min. break point $k = 2$

- $m = 1$: $\text{grow}_{\mathcal{H}}(m) = 2$ by def.
 - $m = 2$: $\text{grow}_{\mathcal{H}}(m) < 4$ by def. (so max. possible = 3)
 - $m = 3$: see Fig 5.13, 5.14, so max. possible = 4 $\ll 2^m$
- break point k restricts max. possible $\text{grow}_{\mathcal{H}}(m)$ a lot for $m > k$
idea

$$\underset{\mathcal{H}}{\text{grow}(m)} \leq \text{max. possible } \underset{\mathcal{H}}{\text{grow}(m)} \text{ given } k \leq \text{poly}(m) \quad (5.12)$$

5.2.2 Bounding Fcn.: Basic Cases

Bounding Fcn.

bounding fcn. $B(m, k)$: max. possible $\text{grow}_{\mathcal{H}}(m)$ when break point k

- combinatorial quantity: max. num. of length- m vecs. with $\{-1, +1\}$ while no shatter any length- k subvecs.
- irrelevant of the details of \mathcal{H}
 - eg., $B(m, 3)$ bounds with pos. ints. ($k = 3$) and 1-dim. perceptrons ($k = 3$)
- can be loose in bounding $\text{grow}_{\mathcal{H}}(m)$
- new goal: $B(m, k) \leq \text{poly}(m)$

Table of Bounding Fcn.

known:

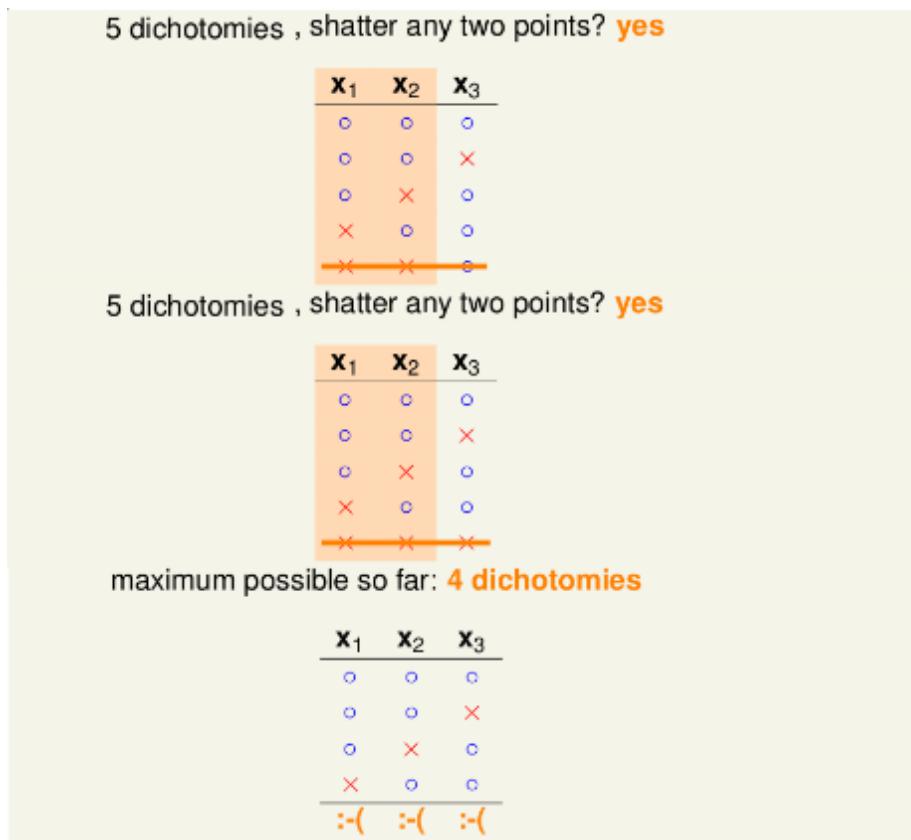
- $B(2, 2) = 3$ (max. < 4)
- $B(3, 2) = 4$ (pic. proof previously)
- $B(m, 1) = 1$ (leave it as ex.)
- $B(m, k) = 2^m$ for $m < k$ (including all dichotomies not violating break condition)

<p>1 dichotomy , shatter any two points? no</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>x_1</th> <th>x_2</th> <th>x_3</th> </tr> </thead> <tbody> <tr> <td>o</td> <td>o</td> <td>o</td> </tr> </tbody> </table>	x_1	x_2	x_3	o	o	o												
x_1	x_2	x_3																
o	o	o																
<p>2 dichotomies , shatter any two points? no</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>x_1</th> <th>x_2</th> <th>x_3</th> </tr> </thead> <tbody> <tr> <td>o</td> <td>o</td> <td>o</td> </tr> <tr> <td>o</td> <td>o</td> <td>x</td> </tr> </tbody> </table>	x_1	x_2	x_3	o	o	o	o	o	x									
x_1	x_2	x_3																
o	o	o																
o	o	x																
<p>3 dichotomies , shatter any two points? no</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>x_1</th> <th>x_2</th> <th>x_3</th> </tr> </thead> <tbody> <tr> <td>o</td> <td>o</td> <td>o</td> </tr> <tr> <td>o</td> <td>o</td> <td>x</td> </tr> <tr> <td>o</td> <td>x</td> <td>o</td> </tr> </tbody> </table>	x_1	x_2	x_3	o	o	o	o	o	x	o	x	o						
x_1	x_2	x_3																
o	o	o																
o	o	x																
o	x	o																
<p>4 dichotomies , shatter any two points? yes</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>x_1</th> <th>x_2</th> <th>x_3</th> </tr> </thead> <tbody> <tr> <td>o</td> <td>o</td> <td>o</td> </tr> <tr> <td>o</td> <td>o</td> <td>x</td> </tr> <tr> <td>o</td> <td>x</td> <td>o</td> </tr> <tr> <td>x</td> <td>o</td> <td>o</td> </tr> </tbody> </table>	x_1	x_2	x_3	o	o	o	o	o	x	o	x	o	x	o	o			
x_1	x_2	x_3																
o	o	o																
o	o	x																
o	x	o																
x	o	o																
<p>4 dichotomies , shatter any two points? no</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>x_1</th> <th>x_2</th> <th>x_3</th> </tr> </thead> <tbody> <tr> <td>o</td> <td>o</td> <td>o</td> </tr> <tr> <td>o</td> <td>o</td> <td>x</td> </tr> <tr> <td>o</td> <td>x</td> <td>o</td> </tr> <tr> <td>x</td> <td>o</td> <td>o</td> </tr> </tbody> </table>	x_1	x_2	x_3	o	o	o	o	o	x	o	x	o	x	o	o			
x_1	x_2	x_3																
o	o	o																
o	o	x																
o	x	o																
x	o	o																
<p>5 dichotomies , shatter any two points? yes</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>x_1</th> <th>x_2</th> <th>x_3</th> </tr> </thead> <tbody> <tr> <td>o</td> <td>o</td> <td>o</td> </tr> <tr> <td>o</td> <td>o</td> <td>x</td> </tr> <tr> <td>o</td> <td>x</td> <td>o</td> </tr> <tr> <td>x</td> <td>o</td> <td>o</td> </tr> <tr> <td>x</td> <td>o</td> <td>x</td> </tr> </tbody> </table>	x_1	x_2	x_3	o	o	o	o	o	x	o	x	o	x	o	o	x	o	x
x_1	x_2	x_3																
o	o	o																
o	o	x																
o	x	o																
x	o	o																
x	o	x																

Figure 5.13: max. possible $\text{grow}_{\mathcal{H}}(m)$ when $m = 3, k = 2$

- $B(k, k) = 2^k - 1$ removing a single dichotomy satisfies breaking condition

more than half done, see Fig 5.15

Figure 5.14: max. possible $\text{grow}_{\mathcal{H}}(m)$ when $m = 3, k = 2$

		k						
		1	2	3	4	5	6	...
B(N, k)		1	2	2	2	2	2	...
N	1	1	3	4	4	4	4	...
	2	1	3	4	4	4	4	...
	3	1	4	7	8	8	8	...
	4	1		15	16	16	16	...
	5	1			31	32	32	...
	6	1				63	63	...
	:	:					...	

Figure 5.15: table of bounding fcn.: base case

5.2.3 Bounding Fcn.: Inductive Cases

motivation:

- $B(4, 3)$ shall be related to $B(3, ?)$
- next: reduce $B(4, 3)$ to $B(3, ?)$

Achieving Dichotomies of $B(4, 3)$

after checking all 2^{2^4} sets of dichotomies, the winner is Fig 5.16

- $2^m = 2^4$ kinds of inputs length- m vecs.
- for each input vec., it has 2 choices: \mathcal{H} can shatter and \mathcal{H} cannot shatter
- check all 2^{2^4} sets of dichotomies to see whether they violate any 3 cannot be shattered

	x_1	x_2	x_3	x_4
01	o	o	o	o
02	x	o	o	o
03	o	x	o	o
04	o	o	x	o
05	o	o	o	x
06	x	x	o	x
07	x	o	x	o
08	x	o	o	x
09	o	x	x	o
10	o	x	o	x
11	o	o	x	x

$B(N, k)$	1	2	3	4	5	6
1	1	2	2	2	2	2
2	1	3	4	4	4	4
3	1	4	7	8	8	8
N	4	1	11	15	16	16
	5	1		31	32	
	6	1			63	

Figure 5.16: achieving dichotomies of $B(4, 3)$

Reorganized Dichotomies of $B(4, 3)$

in the view of $\vec{x}^{(4)}$, orange: pair, purple: single, see Fig 5.17.

	x_1	x_2	x_3	x_4
01	o	o	o	o
05	o	o	o	x
02	x	o	o	o
08	x	o	o	x
03	o	x	o	o
10	o	x	o	x
04	o	o	x	o
11	o	o	x	x
06	x	x	o	x
07	x	o	x	o
09	o	x	x	o

Figure 5.17: reorganized dichotomies of $B(4, 3)$

	\mathbf{x}_1	\mathbf{x}_2	\mathbf{x}_3
α	o	o	o
	x	o	o
	o	x	o
	o	o	x
β	x	x	o
	x	o	x
	o	x	x

Figure 5.18: $B(4, 3) = 11 = 2\alpha + \beta$

$$B(4, 3) = 11 = 2\alpha + \beta \quad (5.13)$$

$\alpha + \beta$: only look at $\{\vec{x}^{(1)}, \vec{x}^{(2)}, \vec{x}^{(3)}\}$

- dichotomies on $\{\vec{x}^{(1)}, \vec{x}^{(2)}, \vec{x}^{(3)}\}$
- $B(4, 3)$ no shatter any 3 inputs $\rightarrow \alpha + \beta$ no shatter any 3
- $\alpha + \beta \leq B(3, 3)$

α : only look at the 2α part

- dichotomies on $\{\vec{x}^{(1)}, \vec{x}^{(2)}, \vec{x}^{(3)}\}$ with $\vec{x}^{(4)}$ paired
- $B(4, 3)$ no shatter any 3 inputs $\rightarrow \alpha$ no shatter any 2
- $\alpha \leq B(3, 2)$

put it together

$$B(4, 3) = 2\alpha + \beta \quad (5.14)$$

$$\alpha + \beta \leq B(3, 3) \quad (5.15)$$

$$\alpha \leq B(3, 2) \quad (5.16)$$

$$B(4, 3) \leq B(3, 3) + B(3, 2) \quad (5.17)$$

$$B(m, k) = 2\alpha + \beta \quad (5.18)$$

$$\alpha + \beta \leq B(m - 1, k) \quad (5.19)$$

$$\alpha \leq B(m - 1, k - 1) \quad (5.20)$$

$$B(4, 3) \leq B(m - 1, k) + B(m - 1, k - 1) \quad (5.21)$$

now have upper bound of bounding fcn., see Fig 5.19

Bounding Fcn.: the Thm

$$\text{grow}_{\mathcal{H}}(m) \leq B(m, k) \leq \sum_{i=0}^{k-1} \binom{m}{i} = O(m^{k-1}) = \text{poly}(m) \text{ if break point exists} \quad (5.22)$$

		k					
		1	2	3	4	5	6
N	1	1	2	2	2	2	2
	2	1	3	4	4	4	4
	3	1	4	7	8	8	8
	4	1	≤ 5	11	15	16	16
	5	1	≤ 6	≤ 16	≤ 26	31	32
	6	1	≤ 7	≤ 22	≤ 42	≤ 57	63

Figure 5.19: table of bounding fcn.: base & inductive case

Proof by induction

- when $k = 1$

$$B(m, 1) = 1 = \sum_{i=0}^0 \binom{m}{i} = \binom{m}{0} = 1 \quad (5.23)$$

- when $k \geq 2$

– Base case: $m = 1$

$$B(1, k) = 2^1 = \sum_{i=0}^{k-1} \binom{1}{i} = 1 + 1 + 0 \dots = 2 \quad (5.24)$$

$$\text{// Note } \binom{m}{0} = 1 \forall m \in \mathbb{N}, \binom{0}{i} = 0 \forall i \in \mathbb{N}_+ \quad (5.25)$$

$$\text{// } \binom{m}{i} = \binom{m-1}{i-1} + \binom{m-1}{i} \forall m, i \in \mathbb{N}_+ \quad (5.26)$$

$$\text{// Thus } \binom{m}{i} = 0 \text{ if } i > m \quad (5.27)$$

– inductive step: supp. this hold true for $m \leq m_0$, when $m = m_0 + 1$

$$B(m_0 + 1, k) \quad (5.28)$$

$$\leq B(m_0, k) + B(m_0, k - 1) \quad (5.29)$$

$$\leq \sum_{i=0}^{k-1} \binom{m_0}{i} + \sum_{i=0}^{k-2} \binom{m_0}{i} \quad (5.30)$$

$$= 1 + \sum_{i=1}^{k-1} \binom{m_0}{i} + \sum_{i=1}^{k-1} \binom{m_0}{i-1} \quad (5.31)$$

$$= 1 + \sum_{i=1}^{k-1} (\binom{m_0}{i} + \binom{m_0}{i-1}) \quad (5.32)$$

$$= 1 + \sum_{i=1}^{k-1} \binom{m_0 + 1}{i} \quad (5.33)$$

$$= \sum_{i=0}^{k-1} \binom{m_0 + 1}{i} \quad (5.34)$$

■

The Five Break Points

see Tab. 5.5

Table 5.5: the five break points

	$\text{grow}_{\mathcal{H}}(m)$
pos. rays	$\text{grow}_{\mathcal{H}}(m) = m + 1 \leq B(m, 2) = O(m)$
dec. stumps	$\text{grow}_{\mathcal{H}}(m) = 2m \leq B(m, 3) = O(m^2)$
pos. ints.	$\text{grow}_{\mathcal{H}}(m) = \frac{1}{2}m^2 + \frac{1}{2}m + 1 \leq B(m, 3) = O(m^2)$
cvx. sets	$\text{grow}_{\mathcal{H}}(m) = 2^m$ no break point
2-dim. perceptrons	$\text{grow}_{\mathcal{H}}(m) \leq B(m, 4) = O(m^3) < 2^m$

5.2.4 A Pictorial Proof

want:

$$\Pr(\exists h \in \mathcal{H}, \text{st. } |J_{in}(h) - J_{out}(h)| > \varepsilon) \leq 2 \cdot \text{grow}_{\mathcal{H}}(m) \cdot \exp(-2\varepsilon^2 m) \quad (5.35)$$

actually, when m large enough, VC bound

$$\Pr(\exists h \in \mathcal{H}, \text{st. } |J_{in}(h) - J_{out}(h)| > \varepsilon) \leq 4 \cdot \text{grow}_{\mathcal{H}}(2m) \cdot \exp(-\frac{1}{8}\varepsilon^2 m) \quad (5.36)$$

$J_{in} \approx J_{out}$ possible if $\text{grow}_{\mathcal{H}}(m)$ breaks somewhere and m large enough

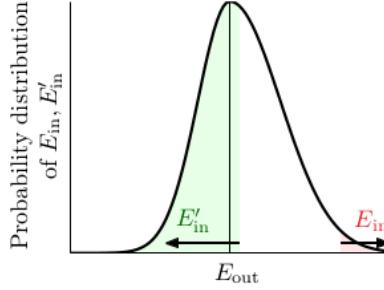
Proof

- replace J_{out} by J'_{in}
 - J_{out} is inf. many, how to replace it?
 - sample verification set D' of size m to cal. J'_{in}
 - BAD h of $J_{in} - J_{out}$ $\xrightarrow{\text{prob.}}$ BAD h of $J_{in} - J'_{in}$, see Fig 5.20

$$\frac{1}{2} \Pr(\exists h \in \mathcal{H}, \text{st. } |J_{in}(h) - J_{out}(h)| > \varepsilon) \quad (5.37)$$

$$\leq \Pr(\exists h \in \mathcal{H}, \text{st. } |J_{in}(h) - J'_{in}(h)| > \frac{\varepsilon}{2}) \quad (5.38)$$

- decompose \mathcal{H} by kind

Figure 5.20: prob. dist. of J_{in}, J'_{in}

- now $\text{grow}_{\mathcal{H}}(m)$ comes to play, how?
- inf. \mathcal{H} becomes $|\mathcal{H}(\mathcal{X}, \mathcal{X}')|$ kinds
- union bound on $\text{grow}_{\mathcal{H}}(2m)$ kinds

$$\text{BAD} \leq 2 \Pr(\exists h \in \mathcal{H}, \text{st. } |J_{in}(h) - J'_{in}(h)| > \frac{\varepsilon}{2}) \quad (5.39)$$

$$\leq 2 \underset{\mathcal{H}}{\text{grow}}(2m) \Pr(\text{fixed } h, \text{st. } |J_{in}(h) - J'_{in}(h)| > \frac{\varepsilon}{2}) \quad (5.40)$$

- use Hoeffding w/o replacement
 - consider bin of $2m$ examples, choose m to cal. J_{in} , leave others to cal. J'_{in}

$$|J_{in} - J'_{in}| > \frac{\varepsilon}{2} \iff |J_{in} - \frac{J_{in} + J'_{in}}{2}| > \frac{\varepsilon}{4} \quad (5.41)$$

$$\text{BAD} \leq 2 \underset{\mathcal{H}}{\text{grow}}(2m) \Pr(\text{fixed } h, \text{st. } |J_{in}(h) - J'_{in}(h)| > \frac{\varepsilon}{2}) \quad (5.42)$$

$$\leq 2 \underset{\mathcal{H}}{\text{grow}}(2m) \cdot 2 \exp(-2(\frac{\varepsilon}{4})^2 m) \quad (5.43)$$

■

5.3 The VC Dim.

5.3.1 Def. of VC Dim.

More on VC Bound

VC bound says that, if $\text{grow}_{\mathcal{H}}(m)$ breaks at k , m large enough \rightarrow probably generalized $J_{in} \approx J_{out}$

if A picks a h^* with small J_{in}
 \rightarrow prob. learned!

VC Dim.

$VC(\mathcal{H})$ = largest m for which $\text{grow}_{\mathcal{H}}(m) = 2^m$

- the most inputs \mathcal{H} that can shatter
- $VC(\mathcal{H}) \approx \min. k - 1$
- if $m \geq 2$, $VC(\mathcal{H}) \geq 2$, $\text{grow}_{\mathcal{H}}(m) \leq m^{VC(\mathcal{H})}$
- good: fin. $VC(\mathcal{H}) \rightarrow h^*$ will generalize ($J_{in} \approx J_{out}$)
- regardless of learning alg. A
- regardless of input distr. P
- regardless of target fcn. f
- worst case guarantee on generalization

VC Dimension of Union of Hypotheses

for hypothesis set \mathcal{H}_1 and hypotheses h_1, h_2, \dots, h_K , what is the upper bound of the $VC(\mathcal{H}_1 \cup \{h_1, h_2, \dots, h_K\})$?

consider $K = 1$ case: suppose $VC(\mathcal{H}) = VC(\mathcal{H}_1 \cup \{h_1\}) = n$, and let D be set of n points that can shattered by \mathcal{H}

pick an arbitrary $\vec{x} \in D$, since \mathcal{H} shatters D , there must be some $\bar{h} \in \mathcal{H}$ such that h and \bar{h} agree on labelings for all points in D except \vec{x}

this means that $\mathcal{H}' = \mathcal{H} - \{h\}$ achieves all possible labelings on $D' = D - \{\vec{x}\}$ (ie., \mathcal{H}' shatters D')

so,

$$VC(\mathcal{H}') \geq |D'| = n - 1 \quad (5.44)$$

but $\mathcal{H}' \subset \mathcal{H}_1$

$$n - 1 \leq VC(\mathcal{H}') \leq VC(\mathcal{H}_1) \quad (5.45)$$

$$VC(\mathcal{H}) \leq VC(\mathcal{H}_1) + 1 \quad (5.46)$$

the result stated in the problem set follows immediately by applying the same logic inductively, one hypothesis at a time.

VC Dim. of union of the Hypothesis Sets

for hypothesis sets $\mathcal{H}_1, \mathcal{H}_2, \dots, \mathcal{H}_K$ with fin., pos., VC dim. $VC(\mathcal{H}_k)$, what is the bound of the $VC(\bigcup_{k=1}^K \mathcal{H}_k)$?

lower bound is $\max_{k \in \{1, 2, \dots, K\}} VC(\mathcal{H}_k)$, ie., all \mathcal{H}_k is subset of the hypothesis set whose VC dim. is the largest

upper bound: when all \mathcal{H}_k are disjoint. consider $K = 2$ case: $VC(\mathcal{H}_1) = n_1, VC(\mathcal{H}_2) = n_2$

$$\underset{\mathcal{H}_1 \cup \mathcal{H}_2}{\text{grow}}(m) \quad (5.47)$$

$$\leq \underset{\mathcal{H}_1}{\text{grow}}(m) + \underset{\mathcal{H}_2}{\text{grow}}(m) \quad (5.48)$$

$$\leq \sum_{i=0}^{n_1} \binom{m}{i} + \sum_{i=0}^{n_2} \binom{m}{i} \quad (5.49)$$

$$= \sum_{i=0}^{n_1} \binom{m}{i} + \sum_{i=0}^{n_2} \binom{m}{m-i} \quad (5.50)$$

$$= \sum_{i=0}^{n_1} \binom{m}{i} + \sum_{i=m-n_2}^m \binom{m}{i} \quad (5.51)$$

we are going to find m st. $\text{grow}_{\mathcal{H}_1 \cup \mathcal{H}_2}(m) < 2^m$, that m is the break point
is $m = n_1$ big enough? No!

$$\sum_{i=0}^{n_1} \binom{m}{i} + \sum_{i=m-n_2}^m \binom{m}{i} \quad (5.52)$$

$$= 2^m + \sum_{i=m-n_2}^m \binom{m}{i} \quad (5.53)$$

$$> 2^m \quad (5.54)$$

is $m = n_1 + n_2 + 1$ big enough? No!

$$\sum_{i=0}^{n_1} \binom{m}{i} + \sum_{i=m-n_2}^m \binom{m}{i} \quad (5.55)$$

$$= \sum_{i=0}^{n_1} \binom{m}{i} + \sum_{i=n_1+1}^m \binom{m}{i} \quad (5.56)$$

$$= 2^m \quad (5.57)$$

is $m = n_1 + n_2 + 2$ big enough? Yes!

$$\sum_{i=0}^{n_1} \binom{m}{i} + \sum_{i=m-n_2}^m \binom{m}{i} \quad (5.58)$$

$$= \sum_{i=0}^{n_1} \binom{m}{i} + \sum_{i=n_1+2}^m \binom{m}{i} \quad (5.59)$$

$$= 2^m - \binom{m}{n_1+1} \quad (5.60)$$

$$< 2^m \quad (5.61)$$

thus, $VC(\mathcal{H}_1 \mathcal{H}_2) = n_1 + n_2 + 1 = VC(\mathcal{H})_1 + VC(\mathcal{H})_2 + 1$
we can use this to infer K case

5.3.2 VC Dim. of Perceptrons

1-dim. perceptron: $VC(\mathcal{H}) = 2$

2-dims. perceptrons: $VC(\mathcal{H}) = 3$

n -dims. perceptrons: $VC(\mathcal{H}) = n + 1$? two steps

- $VC(\mathcal{H}) \geq n + 1$: there are some $n + 1$ inputs we can shatter
- $VC(\mathcal{H}) \leq n + 1$: we cannot shatter any set of $n + 2$ inputs

Proof

- $VC(\mathcal{H}) \geq n + 1$
 - some trivial inputs

$$X = \begin{bmatrix} \vec{x}^{(1)T} \\ \vec{x}^{(2)T} \\ \vdots \\ \vec{x}^{(n+1)T} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & \cdots & 0 \\ 1 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & 0 & \cdots & 1 \end{bmatrix} \in \mathbb{R}^{(n+1) \times (n+1)} \quad (5.62)$$

- visually in 2-dims., see Fig 5.21



Figure 5.21: visually in 2-dims.

- X invertible
- to shatter, for any

$$\vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n+1)} \end{bmatrix} \in \mathbb{R}^{n+1} \quad (5.63)$$

find $\vec{\theta}$ st.

$$\text{sign}(X\vec{\theta}) = \vec{y} \leftarrow X\vec{\theta} = \vec{y} \xrightarrow{X \text{ invertible}} \vec{\theta} = X^{-1}\vec{y} \quad (5.64)$$

- special X can be shattered $\rightarrow VC(\mathcal{H}) \geq n + 1$
- $VC(\mathcal{H}) \leq n + 1$
 - a 2-dim. special case

$$X = \begin{bmatrix} \vec{x}^{(1)T} \\ \vec{x}^{(2)T} \\ \vec{x}^{(3)T} \\ \vec{x}^{(4)T} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \in \mathbb{R}^{(2+1) \times (2+1)} \quad (5.65)$$

- * Thus, $\vec{x}^{(4)} = \vec{x}^{(2)} + \vec{\theta}^T \vec{x}^{(3)} - \vec{x}^{(1)}$
- * ? cannot be \times , see Fig 5.22

$$\vec{\theta}^T \vec{x}^{(4)} = \underbrace{\vec{\theta}^T \vec{x}^{(2)}}_{>0} + \underbrace{\vec{\theta}^T \vec{x}^{(3)}}_{>0} - \underbrace{\vec{\theta}^T \vec{x}^{(1)}}_{<0} \stackrel{\text{must}}{>} 0 \quad (5.66)$$



Figure 5.22: a 2-dim. special case

- * lin. dep. restricts dichotomy
- n -dim. general case

$$X = \begin{bmatrix} \vec{x}^{(1)T} \\ \vec{x}^{(2)T} \\ \vdots \\ \vec{x}^{(n+1)T} \\ \vec{x}^{(n+2)T} \end{bmatrix} \in \mathbb{R}^{(n+2) \times (n+1)} \quad (5.67)$$

- * more rows than cols.
- * lin. dep. (some α_i non-zero)

$$\vec{x}^{(n+2)} = \alpha_1 \vec{x}^{(1)} + \alpha_2 \vec{x}^{(2)} + \dots + \alpha_{n+1} \vec{x}^{(n+1)} \quad (5.68)$$

- * can you gen.

$$\vec{y} = \begin{bmatrix} \text{sign}(\alpha_1) \\ \text{sign}(\alpha_2) \\ \vdots \\ \text{sign}(\alpha_{n+1}) \\ -1 \end{bmatrix} \in \mathbb{R}^{n+2} \quad (5.69)$$

- * a contradiction!

$$\vec{\theta}^T \vec{x}^{(n+2)} = \underbrace{\alpha_1 \vec{\theta}^T \vec{x}^{(1)}}_{>0} + \underbrace{\alpha_2 \vec{\theta}^T \vec{x}^{(2)}}_{>0} + \underbrace{\alpha_{n+1} \vec{\theta}^T \vec{x}^{(n+1)}}_{>0} \stackrel{\text{must}}{>} 0 \quad (5.70)$$

- * general X no shatter

■

5.3.3 Physical Intuition of VC Dim.

Deg. of Freedom

hypothesis para. $\vec{\theta} = [\theta_0 \ \theta_1 \ \dots \ \theta_n]^T$: create deg. of freedom

hypothesis quantity $M = |\mathcal{H}|$: analog deg. of freedom, see Fig 5.23, ie., how many knobs can we turn

hypothesis power $VC(\mathcal{H}) = n + 1$: effective binary deg. of freedom (for binary classification), ie., how many effective knobs can we turn

$VC(\mathcal{H})$: powerfulness of \mathcal{H}

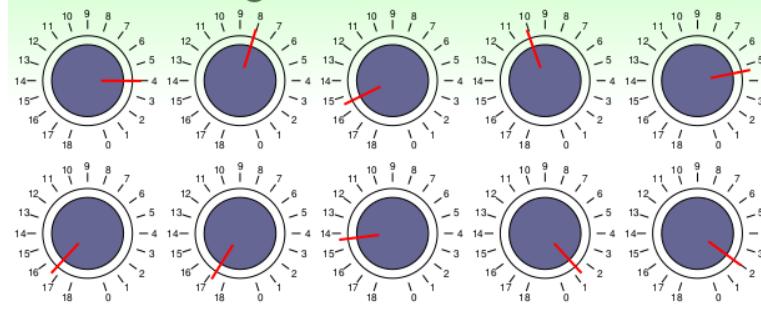


Figure 5.23: deg. of freedom

- pos. rays: $VC(\mathcal{H}) = 1$, free para. θ
 - pos. ints.: $VC(\mathcal{H}) = 2$, free paras. l, r
- practical rule of thumb: $VC(\mathcal{H}) \approx \# \text{ free paras.}$ (but not always)

M and $VC(\mathcal{H})$

2 central questions

- 1. can we make sure $J_{in}(h^*) \approx J_{out}(h^*)$
- 2. can we make sure $J_{in}(h^*) \approx 0$

small M

- 1. Yes! $\Pr\{\text{BAD}\} \leq 2M \exp(\dots)$
- 2. No! too few choice

large M

- 1. No! $\Pr\{\text{BAD}\} \leq 2M \exp(\dots)$
- 2. Yes! many choices

small $VC(\mathcal{H})$

- 1. Yes! $\Pr\{\text{BAD}\} \leq 2(2m)^{VC(\mathcal{H})} \exp(\dots)$
- 2. No! too limited power

large $VC(\mathcal{H})$

- 1. No! $\Pr\{\text{BAD}\} \leq 2(2m)^{VC(\mathcal{H})} \exp(\dots)$
- 2. Yes! lots of power

using the right $VC(\mathcal{H})$ (or \mathcal{H}) is important!

5.3.4 Interpreting VC Dim.

VC Bound Rephrase: Penalty for Model Complexity

for any $h^* = A(D)$, ie. $h^* = \arg \min_{h \in \mathcal{H}} J_{in}(h)$ and stat. large D , $VC(\mathcal{H}) \geq 2$

$$\Pr(|J_{in}(h^*) - J_{out}(h^*)| > \varepsilon) \leq 4 \cdot (2m)^{VC(\mathcal{H})} \cdot \exp(-\frac{1}{8}\varepsilon^2 m) = \delta \quad (5.71)$$

$$\varepsilon = \sqrt{\frac{8}{m} \log \frac{4(2m)^{VC(\mathcal{H})}}{\delta}} \quad (5.72)$$

wp. $\geq 1 - \delta$, GOOD (uniform conv., because this is a bound that holds simultaneously for all as opposed to just one h^*):

$$|J_{in}(h^*) - J_{out}(h^*)| \leq \varepsilon = \sqrt{\frac{8}{m} \log \frac{4(2m)^{VC(\mathcal{H})}}{\delta}} \quad (5.73)$$

$$J_{out}(h^*) \leq J_{in}(h^*) + \varepsilon(m, \mathcal{H}, \delta) \quad (5.74)$$

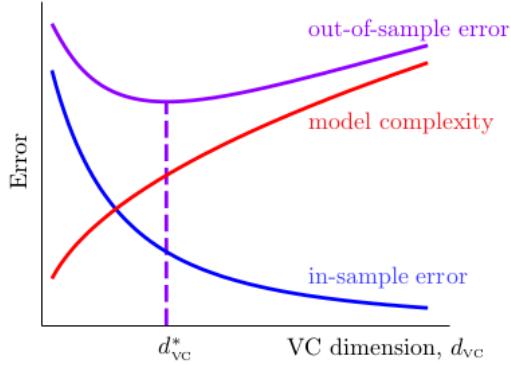


Figure 5.24: penalty for model complexity

ε : penalty for model complexity, see Fig 5.24

def. $h_{out}^* = \arg \min_{h \in \mathcal{H}} J_{out}(h)$ to be the best that we could possibly do given that we are using \mathcal{H}

$$J_{out}(h^*) \leq J_{in}(h^*) + \varepsilon // by\ uniform\ conv. \quad (5.75)$$

$$\leq J_{in}(h_{out}^*) + \varepsilon // h^* \text{ is the best one to min. } J_{in}(h) \quad (5.76)$$

$$\leq J_{out}(h_{out}^*) + 2\varepsilon // uniform\ conv.\ again \quad (5.77)$$

$$= bias + variance \quad (5.78)$$

if uniform conv. occurs, then the generalization err. of h^* is at most 2ε worse than the best possible hypo. in \mathcal{H}

- $VC(\mathcal{H}) \uparrow$:
 $J_{out}(h_{out}^*) \downarrow$ (since we are taking a min. over a larger set of \mathcal{H}), bias \downarrow
 $\varepsilon \uparrow$, variance \uparrow
- $VC(\mathcal{H}) \downarrow$: $\varepsilon \downarrow$ but $J_{out}(h_{out}^*) \uparrow$
- best $VC(\mathcal{H})^*$ in the middle
- powerful \mathcal{H} not always good

VC Bound Rephrase: Sample Complexity

given spec. $\varepsilon = 0.1, \delta = 0.1$, the bound tells us how many training examples we need in order to make a good guarantee

sample complexity: theory: $m \approx 10,000VC(\mathcal{H})$, practice: $m \approx 10VC(\mathcal{H}) = O(VC(\mathcal{H}))$ (roughly lin. in # of paras. of \mathcal{H})

why?

- Hoeffding for unknown J_{out} : any distr., any target
 - $\text{grow}_{\mathcal{H}}(m)$ instead of $|\mathcal{H}(X)|$: any data
 - $m^{VC(\mathcal{H})}$ instead of $\text{grow}_{\mathcal{H}}(m)$: any \mathcal{H} of same $VC(\mathcal{H})$
 - union bound on w.-c.: any choice made by A
- but hardly better, and similarly loose for all models

5.4 Noise and Error

5.4.1 Noise and Probabilistic Target

Does VC Bound work under noise?

deterministic marbles

- marble $\vec{x} \sim \Pr(\vec{x})$
- deterministic color $1\{f(\vec{x}) \neq h(\vec{x})\}$

probabilistic (noisy) marbles

- marble $\vec{x} \sim \Pr(\vec{x})$
- probabilistic color $1\{y \neq h(\vec{x})\}$ with $y \sim \Pr(y|\vec{x})$

same nature: can estimate $\Pr\{\text{orange}\}$ if iid. ...

VC holds for $\underbrace{\vec{x} \stackrel{iid.}{\sim} \Pr(\vec{x}), y \stackrel{iid.}{\sim} \Pr(y|\vec{x})}_{(\vec{x},y) \stackrel{iid.}{\sim} \Pr(\vec{x},y)}$

Target Distr. $\Pr(y|\vec{x})$

char. behavior of mini-target on one \vec{x}

can be viewed as ideal mini-target + noise, eg.

- $\Pr(+1|\vec{x}) = 0.7, \Pr(-1|\vec{x}) = 0.3$
- ideal mini-target $f(\vec{x}) = +1$
- flipping noise level $= 0.3 = \tau$

This models a setting in which an unreliable human (or other source) is labeling your training data for you, and on each example he/she has a probability τ of mislabeling it.

deterministic target f : special case of target distr.

- $\Pr(y|\vec{x}) = 1$ for $y = f(\vec{x})$
- $\Pr(y|\vec{x}) = 0$ for $y \neq f(\vec{x})$

goal of learning: predict ideal mini-target (wrt. $\Pr(y|\vec{x})$) on often-seen inputs (wrt. $\Pr(\vec{x})$)

VC Bound of Noisy Target

Specifically, we will consider what happens when the training labels are noisy, but the test labels are not.

let

$$J_{out}(h) = \mathbb{E}[h(\vec{x}) \neq f(\vec{x})] \quad (5.79)$$

be the clean, uncorrupted err.

$$\tilde{J}_{out}(h) = \mathbb{E}[h(\vec{x}) \neq y] \quad (5.80)$$

be the corrupted err.

then,

$$\tilde{J}_{out}(h) = \Pr(h(\vec{x}) \neq y) \quad (5.81)$$

$$= \Pr(h(\vec{x}) \neq y | y = f(\vec{x})) \Pr(y = f(\vec{x})) + \Pr(h(\vec{x}) \neq y | y \neq f(\vec{x})) \Pr(y \neq f(\vec{x})) \quad (5.82)$$

$$= J_{out}(h)(1 - \tau) + (1 - J_{out}(h))\tau \quad (5.83)$$

$$= (1 - 2\tau)J_{out}(h) + \tau \quad (5.84)$$

solving for $J_{out}(h)$

$$J_{out}(h) = \frac{\tilde{J}_{out}(h) - \tau}{1 - 2\tau} \quad (5.85)$$

apply Hoeffding: $\forall h \in \mathcal{H}, |\mathcal{H}| = M$ is fin.,

$$\Pr(|\tilde{J}_{out}(h) - \tilde{J}_{in}(h)| > \varepsilon) \leq 2M \exp(-2\varepsilon^2 m) = \delta \quad (5.86)$$

let

$$h^* = \arg \min \tilde{J}_{in}(h) \quad (5.87)$$

$$h_{out}^* = \arg \min J_{out}(h) \quad (5.88)$$

$$(5.89)$$

$$J_{out}(h^*) = \frac{\tilde{J}_{out}(h^*) - \tau}{1 - 2\tau} \quad (5.90)$$

$$\leq \frac{\tilde{J}_{in}(h^*) + \varepsilon - \tau}{1 - 2\tau} \quad wp.1 - \delta \quad (5.91)$$

$$\leq \frac{\tilde{J}_{in}(h_{out}^*) + \varepsilon - \tau}{1 - 2\tau} \quad wp.1 - \delta \quad (5.92)$$

$$\leq \frac{\tilde{J}_{out}(h_{out}^*) + 2\varepsilon - \tau}{1 - 2\tau} \quad wp.1 - \delta \quad (5.93)$$

$$\leq \frac{(1 - 2\tau)J_{out}(h_{out}^*) + \tau + 2\varepsilon - \tau}{1 - 2\tau} \quad wp.1 - \delta \quad (5.94)$$

$$= J_{out}(h_{out}^*) + \frac{2\varepsilon}{(1 - 2\tau)} \quad wp.1 - \delta \quad (5.95)$$

$$= J_{out}(h_{out}^*) + 2\varepsilon' \cdot wp.1 - \delta \quad (5.96)$$

sample complexity:

$$m = \frac{1}{2\varepsilon'^2 m} \log \frac{2M}{\delta} \quad (5.97)$$

$$= \frac{1}{2(1-2\tau)^2 \varepsilon'^2 m} \log \frac{2M}{\delta} \quad (5.98)$$

$$(5.99)$$

This result suggests that, roughly, m examples that have been corrupted at noise level τ are worth about as much as $(1-2\tau)^2 m$ uncorrupted training examples. This is a useful rule-of-thumb to know if you ever need to decide whether/how much to pay for a more reliable source of training data.

The closer τ is to 0.5, the more samples are needed to get the same generalization error bound. For τ approaching 0.5, the training data becomes more and more random; having no information at all about the underlying distribution for $\tau = 0.5$

5.4.2 Err. Measure

Pointwise Err. Measure

$$J_{out}(h^*) = \mathbb{E}_{\vec{x} \sim P} [1\{h^*(\vec{x}) \neq f(\vec{x})\}] = \mathbb{E}_{\vec{x} \sim P} [\text{err}(h^*(\vec{x}), f(\vec{x}))] \quad (5.100)$$

- out-of-sample: avgd over unknown \vec{x}
- pointwise: evaluated on one \vec{x}
- classification: $1\{\text{prediction} \neq \text{target}\}$
- classification err: 0/1 err.

$$J_{in}(h^*) = \frac{1}{m} \sum_{i=1}^m \text{err}(h^*(\vec{x}^{(i)}), f(\vec{x}^{(i)})) \quad (5.101)$$

Two Important Pointwise Error Measures

$$\text{err}(\hat{y} = h^*(\vec{x}), y = f(\vec{x})) \quad (5.102)$$

0/1 err.: $\text{err}(\hat{y}, y) = 1\{\hat{y} \neq y\}$

- correct or incorrect?
- often for classification

sq. err.: $\text{err}(\hat{y}, y) = (\hat{y} - y)^2$

- how far is \hat{y} from y ?
- often for regression

Ideal Mini-Target

interplay betw. noise and err.: $\Pr(y|\vec{x})$ and $\text{err}(\hat{y}, y)$ def. ideal mini-target $f(\vec{x})$
 for 0/1 err.: min. flipping err. — NP-hard to opt.

$$f(\vec{x}) = \arg \max_{y \in \mathcal{Y}} \Pr(y|\vec{x}) \quad (5.103)$$

for eq. err.: min. Gaussian noise

$$f(\vec{x}) = \sum_{y \in \mathcal{Y}} y \cdot \Pr(y|\vec{x}) \quad (5.104)$$

5.4.3 Alg. Err. Measure

Choice of Err. Measure

- two types of err.: FP and FN
- 0/1 err. penalizes both types equally
- err. is application/ user-dep, friendly: easy to opt. for A
- closed-form sol.
- cvx. obj. fcn.

5.4.4 Weighted Classification

Skewed classes

cancer classification examples, see Tab. 5.6

Table 5.6: Precision/ Recall		
	$y = 1$	$y = -1$
$h(\vec{x}) = 1$	TP	FP
$h(\vec{x}) = -1$	FN	TN

of all patients where we predicted $y = 1$, what frac. actually has cancer?

$$\text{Precision} = \frac{h(\vec{x}) = 1 \wedge y = 1}{h(\vec{x}) = 1} = \frac{TP}{TP + FP} \quad (5.105)$$

of all patients that usually have cancer, what frac. did we correctly detect as having cancer?

$$\text{Recall} = \frac{h(\vec{x}) = 1 \wedge y = 1}{y = 1} = \frac{TP}{TP + FN} \quad (5.106)$$

e.g., if we predict $h(\vec{x}) = 1$ all the time

- $Precision \approx 0$

- $Recall = 1$

suppose we want to predict $h(\vec{x}) = 1$ only if very confident

- higher precision
- lower recall

suppose we want to avoid missing too many cases of cancer (avoid FN)

- lower precision
- higher recall

more generally, in log. reg., predict 1 if $h(\vec{x}) \geq threshold$, see Fig. 5.25

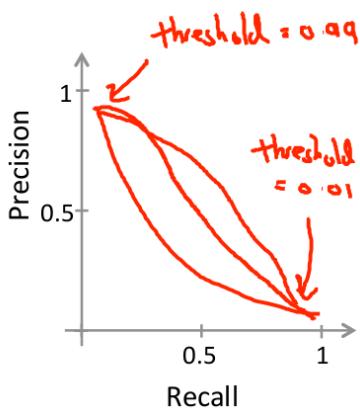


Figure 5.25: predict 1 if $h(\vec{x}) \geq threshold$

how to compare precision/ recall numbers?

$$F1 = 2 \frac{PR}{P+R} \quad (5.107)$$

- if $P = 0$ or $R = 0$, $F1 = 0$
- if $P = 1$ and $R = 1$, $F1 = 1$

Weighted Classification

CIA fingerprint for entrance problem
cost (error, loss) mat., see Fig 5.26

		$h(\mathbf{x})$	
		+1	-1
y	+1	0	1
	-1	1000	0

Figure 5.26: CIA cost mat.

$$J_{out}^w(h) = \underset{(\vec{x},y) \sim P}{\mathbb{E}} \left\{ \begin{array}{ll} 1 & \text{if } y = +1 \\ 1000 & \text{if } y = -1 \end{array} \right\} \cdot 1\{y \neq h(\vec{x})\} \quad (5.108)$$

$$J_{in}^w(h) = \frac{1}{m} \sum_{i=1}^m \left\{ \begin{array}{ll} 1 & \text{if } y = +1 \\ 1000 & \text{if } y = -1 \end{array} \right\} \cdot 1\{y^{(i)} \neq h(\vec{x}^{(i)})\} \quad (5.109)$$

weighted classification: diff. weight for diff. (\vec{x}, y)

Min. J_{in} for Weighted Classification

PLA: doesn't matter if lin. sep.

pocket: modify pocket-replacement rule

- if $\vec{\theta}$ reaches smaller J_{in}^w than $\vec{\theta}^*$, replace $\vec{\theta}^*$ by $\vec{\theta}$

pocket: some guarantee on $J_{in}^{0/1}$, modified pocket: similar guarantee on J_{in}^w ?

- after copying -1 examples 1000 times, J_{in}^w for lhs. $\equiv J_{in}^{0/1}$ for rhs., see Fig 5.27.

original problem		equivalent problem	
		$h(\mathbf{x})$	
y	$+1$	$+1$	-1
	-1	0	1
		1000	0
$\mathcal{D}:$			
	$(\mathbf{x}_1, +1)$		$(\mathbf{x}_1, +1)$
	$(\mathbf{x}_2, -1)$		$(\mathbf{x}_2, -1), (\mathbf{x}_2, -1), \dots, (\mathbf{x}_2, -1)$
	$(\mathbf{x}_3, -1)$		$(\mathbf{x}_3, -1), (\mathbf{x}_3, -1), \dots, (\mathbf{x}_3, -1)$
	\dots		\dots
	$(\mathbf{x}_{N-1}, +1)$		$(\mathbf{x}_{N-1}, +1)$
	$(\mathbf{x}_N, +1)$		$(\mathbf{x}_N, +1)$

Figure 5.27: systematic route: connect J_{in}^w and $J_{in}^{0/1}$

Weighted Pocket Alg.

using virtual copying, weighted pocket alg. include:

- weighted PLA: rand. check -1 example mistakes with 1000 times more probability
- weighted pocket replacement: if $\vec{\theta}$ reaches smaller J_{in}^w than $\vec{\theta}^*$, replace $\vec{\theta}^*$ by $\vec{\theta}$

Chapter 6

How Can Machine Learn

6.1 Linear Regression

6.1.1 Linear Regression Problem

lin. reg. hypo.

$$h(\vec{x}) = \vec{\theta}^T \vec{x} \quad (6.1)$$

the err. measure: sq. err.

$$err(\hat{y}, y) = (\hat{y} - y)^2 \quad (6.2)$$

$$J_{in}(\vec{\theta}) = \frac{1}{m} \sum_{i=1}^m (h(\vec{x}^{(i)}) - y^{(i)})^2 = \frac{1}{m} \sum_{i=1}^m (\vec{\theta}^T \vec{x}^{(i)} - y^{(i)})^2 \quad (6.3)$$

$$J_{out}(\vec{\theta}) = E(\vec{x}, y) \sim P[(\vec{\theta}^T \vec{x} - y)^2] \quad (6.4)$$

6.1.2 Linear Regression Algorithm

Matrix Form of $J_{in}(\vec{\theta})$

$$J_{in}(\vec{\theta}) = \frac{1}{m} \sum_{i=1}^m (\vec{\theta}^T \vec{x}^{(i)} - y^{(i)})^2 \quad (6.5)$$

$$= \frac{1}{m} \sum_{i=1}^m (\vec{x}^{(i)T} \vec{\theta} - y^{(i)})^2 \quad (6.6)$$

$$= \frac{1}{m} \left\| \begin{array}{c} \vec{x}^{(1)T} \vec{\theta} - y^{(1)} \\ \vec{x}^{(2)T} \vec{\theta} - y^{(2)} \\ \vdots \\ \vec{x}^{(m)T} \vec{\theta} - y^{(m)} \end{array} \right\|^2 \quad (6.7)$$

$$= \frac{1}{m} \|X\vec{\theta} - \vec{y}\|^2 \quad (6.8)$$

$J_{in}(\vec{\theta})$: continuouts, differentiable, convex

The Gradient of $\nabla J_{in}(\vec{\theta})$

$$J_{in}(\vec{\theta}) = \frac{1}{m} \|X\vec{\theta} - \vec{y}\|^2 \quad (6.9)$$

$$= \frac{1}{m} (\vec{\theta}^T X^T X \vec{\theta} - \vec{\theta}^T X^T \vec{y} - \vec{y}^T X \vec{\theta} + \vec{y}^T \vec{y}) \quad (6.10)$$

$$= \frac{1}{m} (\vec{\theta}^T X^T X \vec{\theta} - 2\vec{y}^T X \vec{\theta} + \vec{y}^T \vec{y}) // \text{tr } a = \text{tr } a^T \quad (6.11)$$

$$\nabla J_{in}(\vec{\theta}) = \frac{1}{m} \nabla (\vec{\theta}^T X^T X \vec{\theta} - 2\vec{y}^T X \vec{\theta} + \vec{y}^T \vec{y}) \quad (6.12)$$

$$= \frac{1}{m} (\nabla \vec{\theta}^T X^T X \vec{\theta} - 2\nabla \vec{y}^T X \vec{\theta} + \nabla_{\vec{\theta}} \vec{y}^T \vec{y}) \quad (6.13)$$

$$= \frac{2}{m} (X^T X \vec{\theta} - X^T \vec{y}) \quad (6.14)$$

$$// \nabla_{\vec{x}} \vec{b}^T \vec{x} = \vec{b}, \nabla_{\vec{x}} \vec{x}^T A \vec{x} = 2A \vec{x} \text{ if } A \in \S \quad (6.15)$$

$$\stackrel{\text{set}}{=} \vec{0} \quad (6.16)$$

Optimal Linear Regression Weights

if $X^T X$ inv.

- $\vec{\theta}^* = (X^T X)^{-1} X^T \vec{y}$

• often the case bacause $m >> n + 1$

if $X^T X$ singular

- many opt. sols.

- one of the sols.: $\vec{\theta}^* = X^+ \vec{y}$

what if $X^T X$ is singular?

- redundant features (lin. dep.), eg., $x_2 = 11.26x_1$

- too many features ($m \leq n$): delete some features, or use regularization

practical suggestion: use pinv routine instead of $(X^T X)^{-1} X^T \vec{y}$ for numerical stability when almost-singular

Algorithm 10 Linear Regression Algorithm

return $h(\vec{x}) = X^+ \vec{y}$ or $h(\vec{x}) = (X^T X)^{-1} X^T \vec{y}$

6.1.3 Probabilistic Interpretation

Why use lin. reg.? Why use sq. err.?

Probabilistic Assumption

assume

$$y^{(i)} = \vec{\theta}^T \vec{x}^{(i)} + \varepsilon^{(i)} \quad (6.17)$$

$\varepsilon^{(i)}$ is an err. term that captures

- unmodeled effects (eg., there are some features very pertinent to predicting housing price, but we'd left out of the regression)
- random noise

assume

$$\varepsilon^{(i)} \stackrel{\text{iid.}}{\sim} N(0, \sigma^2) \quad (6.18)$$

ie., density of $\varepsilon^{(i)}$ is given by

$$p(\varepsilon^{(i)}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\varepsilon^{(i)2}}{2\sigma^2}\right) \quad (6.19)$$

implies that

$$p(y^{(i)} | \vec{x}^{(i)}; \vec{\theta}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \vec{\theta}^T \vec{x}^{(i)})^2}{2\sigma^2}\right) \sim N(\vec{\theta}^T \vec{x}^{(i)}, \sigma^2) \quad (6.20)$$

$\vec{\theta}$ is a para. not a rv.

given $X, \vec{\theta}$, what is the distr. of the \vec{y} ? $p(\vec{y}|X; \vec{\theta})$

when we wish to explicitly view this as a fcn. of $\vec{\theta}$, we instead call it the likelihood fcn.

$$L(\vec{\theta}) = L(\vec{\theta}; X, \vec{y}) \quad (6.21)$$

$$= p(\vec{y}|X; \vec{\theta}) \quad (6.22)$$

$$= \prod_{i=1}^m p(y^{(i)} | \vec{x}^{(i)}; \vec{\theta}) // \text{indep. assumption of } \varepsilon^{(i)} \text{'s} \quad (6.23)$$

$$= \prod_{i=1}^m \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \vec{\theta}^T \vec{x}^{(i)})^2}{2\sigma^2}\right) \quad (6.24)$$

Maximum Likelihood

how to choose $\vec{\theta}$? principal of max. likelihood says that we should choose $\vec{\theta}$ so as to make the data high prob. as possible as possible.

$$\vec{\theta}^* = \arg \max_{\vec{\theta}} L(\vec{\theta}) \quad (6.25)$$

$$= \arg \max_{\vec{\theta}} \prod_{i=1}^m \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \vec{\theta}^T \vec{x}^{(i)})^2}{2\sigma^2}\right) \quad (6.26)$$

$$= \arg \max_{\vec{\theta}} \log \prod_{i=1}^m \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \vec{\theta}^T \vec{x}^{(i)})^2}{2\sigma^2}\right) \quad (6.27)$$

$$= \arg \max_{\vec{\theta}} \sum_{i=1}^m \log \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \vec{\theta}^T \vec{x}^{(i)})^2}{2\sigma^2}\right) \quad (6.28)$$

$$= \arg \max_{\vec{\theta}} \left(m \log \frac{1}{\sqrt{2\pi}\sigma}\right) - \frac{1}{2\sigma^2} \sum_{i=1}^m (y^{(i)} - \vec{\theta}^T \vec{x}^{(i)})^2 \quad (6.29)$$

$$= \arg \min_{\vec{\theta}} \sum_{i=1}^m (y^{(i)} - \vec{\theta}^T \vec{x}^{(i)})^2 \quad (6.30)$$

$$= \arg \min_{\vec{\theta}} J_{in}(\vec{\theta}) \quad (6.31)$$

under the prev. prob. assumptions on data, sq. err. corresponds to finding the max. likelihood estimate of $\vec{\theta}$.

6.1.4 Generalization Issue

Is Linear Regression a Learning Algorithm?

No!

- analytic (closed-form) sol., instantaneous
- not improving J_{in} nor J_{out} iteratively

Yes!

- good J_{in} ? yes, optimal!
- good J_{out} ? yes, finite $VC(\mathcal{H})$ like perceptrons (see below)
- improving iteratively? somewhat, with an iterative pinv routine

Benefit of Analytic Solution: Simpler-than-VC Guarantee

$$J_{in}(\vec{\theta}^*) = \frac{1}{m} \|\vec{y} - h(\vec{x})\|^2 = \frac{1}{m} \|\vec{y} - XX^+ \vec{y}\|^2 = \frac{1}{m} \|(I - XX^+) \vec{y}\|^2 \stackrel{def.}{=} \frac{1}{m} \|(I - P) \vec{y}\|^2 \quad (6.32)$$

Geometric View of P

$\hat{\vec{y}} = h(\vec{x}) = X\vec{\theta}^* \in \text{span } X$, see Fig 6.1

$\vec{y} - \hat{\vec{y}}$ smallest: $\vec{y} - \hat{\vec{y}} \perp \text{span } X$

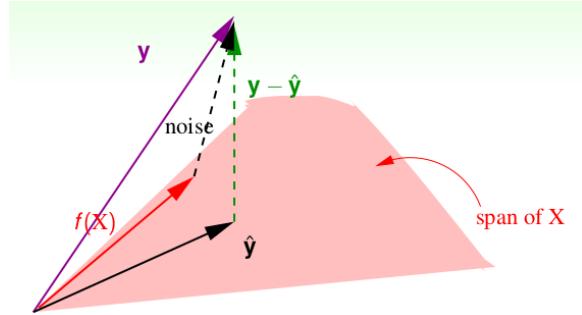
$\hat{\vec{y}} = P\vec{y}$: proj. \vec{y} into $\hat{\vec{y}} \in \text{span } X$

$\vec{y} - \hat{\vec{y}} = (I - P)\vec{y}$: proj. \vec{y} into $\vec{y} - \hat{\vec{y}} \perp \text{span } X$

$\text{tr}(I - P) = m - (n + 1)$

An Illustrative Proof

if \vec{y} comes from some idea $f(X) \in \text{span } X + \text{noise}$
 noise transformed by $I - P$ to be $\vec{y} - \hat{\vec{y}}$

Figure 6.1: Geometric View of P

$$J_{in}(\vec{\theta}^*) = \frac{1}{m} \|(\vec{y} - \hat{\vec{y}})\|^2 = \frac{1}{m} \|(I - P) \times noise\|^2 = \frac{1}{m} (m - (n + 1)) \|noise\|^2 \quad (6.33)$$

thus,

$$\bar{J}_{in} = E D \sim P[J_{in}(\vec{\theta}^* \text{ wrt. } D)] = noise \times \left(1 - \frac{n+1}{m}\right) \quad (6.34)$$

$$\bar{J}_{out} = noise \times \left(1 + \frac{n+1}{m}\right) \quad (6.35)$$

The Learning Curve

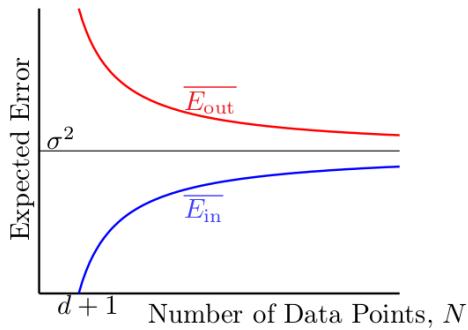


Figure 6.2: The Learning Curve

both conv. to σ^2 (noise level) for m , see Fig 6.2

expected generalization err.: $\frac{2(n+1)}{m}$: similar to w-c guarantee from VC

6.1.5 Locally Weighted Linear Regression

LWR Error

$$J(\vec{\theta}) = \frac{1}{m} \sum_{i=1}^m w^{(i)} (\vec{\theta}^T \vec{x}^{(i)} - y^{(i)})^2 \quad (6.36)$$

$w^{(i)} \geq 0$ are weights, a fairly std. choice is:

- 1-dim. case

$$w^{(i)} = \exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right) \quad (6.37)$$

- n -dim. case

$$w^{(i)} = \exp\left(-\frac{(\vec{x}^{(i)} - \vec{x})^T \Sigma^{-1} (\vec{x}^{(i)} - \vec{x})}{2}\right) \quad (6.38)$$

or,

$$w^{(i)} = \exp\left(-\frac{(\vec{x}^{(i)} - \vec{x})^T (\vec{x}^{(i)} - \vec{x})}{2\tau^2}\right) \quad (6.39)$$

x is the query point

if $|x^{(i)} - x|$ small

- $w^{(i)} \rightarrow 1$ large

• we will try hard to make $(y^{(i)} - \vec{\theta}^T \vec{x}^{(i)})^2$ small

if $|x^{(i)} - x|$ large

- $w^{(i)} \rightarrow 0$

• the $(y^{(i)} - \vec{\theta}^T \vec{x}^{(i)})^2$ error term will be pretty much ignored in the fit

τ is the bandwidth para. to control how quickly the weight of a training example falls off with dist. of its $x^{(i)}$ from the query point x

if τ small

- the fitting is dominated by the closest by training samples

• less training samples are actually taken into account when doing the regression

- regression results become very susceptible to noise

if τ large

- we have enough training samples to reliably fit straight lines

• unfortunately a straight line usually is not the right model for LWR

- we get a bad fit

Optimal LWR Weights

$$J(\vec{\theta}) = \frac{1}{m} \sum_{i=1}^m w^{(i)} (\vec{\theta}^T \vec{x}^{(i)} - y^{(i)})^2 \quad (6.40)$$

$$= \frac{1}{m} \sum_{i=1}^m w^{(i)} (\vec{\theta}^T \vec{x}^{(i)} - y^{(i)}) (\vec{\theta}^T \vec{x}^{(i)} - y^{(i)}) \quad (6.41)$$

$$= \frac{1}{m} [w^{(1)} (\vec{\theta}^T \vec{x}^{(1)} - y^{(1)}) \ w^{(2)} (\vec{\theta}^T \vec{x}^{(2)} - y^{(2)}) \dots w^{(m)} (\vec{\theta}^T \vec{x}^{(m)} - y^{(m)})] \begin{bmatrix} \vec{\theta}^T \vec{x}^{(1)} - y^{(1)} \\ \vec{\theta}^T \vec{x}^{(2)} - y^{(2)} \\ \vdots \\ \vec{\theta}^T \vec{x}^{(m)} - y^{(m)} \end{bmatrix} \quad (6.42)$$

$$= \frac{1}{m} [\vec{\theta}^T \vec{x}^{(1)} - y^{(1)} \ \vec{\theta}^T \vec{x}^{(2)} - y^{(2)} \ \dots \ \vec{\theta}^T \vec{x}^{(m)} - y^{(m)}] \begin{bmatrix} w^{(1)} & & & \\ & w^{(2)} & & \\ & & \ddots & \\ & & & w^{(m)} \end{bmatrix} \begin{bmatrix} \vec{\theta}^T \vec{x}^{(1)} - y^{(1)} \\ \vec{\theta}^T \vec{x}^{(2)} - y^{(2)} \\ \vdots \\ \vec{\theta}^T \vec{x}^{(m)} - y^{(m)} \end{bmatrix} \quad (6.43)$$

$$= \frac{1}{m} (X\vec{\theta} - \vec{y})^T W (X\vec{\theta} - \vec{y}) \quad (6.44)$$

$$\nabla J(\vec{\theta}) = \nabla \frac{1}{m} (X\vec{\theta} - \vec{y})^T W (X\vec{\theta} - \vec{y}) \quad (6.45)$$

$$= \frac{2}{m} (X^T W X \vec{\theta} - X^T W \vec{y}) \quad (6.46)$$

$$\stackrel{\text{set}}{=} \vec{0} \quad (6.47)$$

$$\vec{\theta} = (X^T W X)^{-1} X^T W \vec{y} \quad (6.48)$$

Non-parametric Algorithm

$\vec{\theta}$ is chosen giving a much higher weight to the (errors on) training examples close to the query point x

(original unweighted) lin. reg. is a para. learning alg.

- it has a fixed, finite # of paras. $\vec{\theta}$ to fit the data
- once we have fit the $\vec{\theta}$, we no longer need to keep the training data around to make future dec.

LWR is a non-para. alg.

- we need to keep the entire training set around to make future dec.
- the amount of stuff we need to keep in order to repr. the h is $O(m)$

6.1.6 Bayesian Learning Regression

$$y = \vec{\theta}^T \vec{x} + \varepsilon \quad (6.49)$$

$$\varepsilon \sim N(0, \sigma^2) \quad (6.50)$$

assume prior prob. of $\vec{\theta}$ is

$$\vec{\theta} \sim N(\vec{0}, \tau^2 I) \quad (6.51)$$

the posterior prob. is

$$p(\vec{\theta}|D) = \frac{\prod_{t=1}^m p(\vec{x}^{(i)}|\vec{\theta})p(\vec{\theta})}{\int_{\vec{\theta}} \prod_{t=1}^m p(\vec{x}^{(i)}|\vec{\theta})p(\vec{\theta})d\vec{\theta}} = N\left(\frac{1}{\sigma^2}A^{-1}X^T\vec{y}, A^{-1}\right) \quad (6.52)$$

where

$$A = \frac{1}{\sigma^2}X^T X + \frac{1}{\sigma^2}I \quad (6.53)$$

future prediction on \vec{x}

$$p(y|\vec{x}, D) = \int_{\vec{\theta}} p(y|\vec{x}, \vec{\theta})p(\vec{\theta}|D)d\vec{\theta} = N\left(\frac{1}{\sigma^2}\vec{x}^T A^{-1}X^T\vec{y}, \vec{x}^T A^{-1}\vec{x} + \sigma^2\right) \quad (6.54)$$

- reflects the uncertainty from $\vec{\theta}$ and ε
- MLE of $\vec{\theta}$ provides no estimate of how reliable these learned paras. may be

6.1.7 Gaussian Process Regression

The Gaussian Process Regression Model

let

$$y = f(x) + \varepsilon \quad (6.55)$$

$$\varepsilon \sim N(0, \sigma^2) \quad (6.56)$$

assume a prior distr. over f

$$f \sim GP(0, k) \quad (6.57)$$

let

$$D = \{\vec{x}^{(i)}, y^{(i)}\}_{i=1}^m \quad (6.58)$$

be training set of iid. examples from some unknown distr.

let

$$D_t = \{\vec{x}_t^{(i)}, y_t^{(i)}\}_{i=1}^{m_t} \quad (6.59)$$

be test set of iid. examples from same unknown distr., D and D_t are mutually indep.

Prediction

let

$$\vec{f} = \begin{bmatrix} f(\vec{x}^{(1)}) \\ f(\vec{x}^{(2)}) \\ \vdots \\ f(\vec{x}^{(m)}) \end{bmatrix}, \vec{f}_t = \begin{bmatrix} f(\vec{x}_t^{(1)}) \\ f(\vec{x}_t^{(2)}) \\ \vdots \\ f(\vec{x}_t^{(m)}) \end{bmatrix}, \quad (6.60)$$

if $f \sim GP(0, k)$, the marginal distr. over any set of input points belonging to \mathcal{X} must have a joint multivariate Gaussian distr., in particular, this must hold for the training and test points

$$\begin{bmatrix} \vec{f} \\ \vec{f}_t \end{bmatrix} \mid X, X_t \sim N(\vec{0}, \begin{bmatrix} K(X, X) & K(X, X_t) \\ K(X_t, X) & K(X_t, X_t) \end{bmatrix}) \quad (6.61)$$

$$\begin{bmatrix} \vec{\varepsilon} \\ \vec{\varepsilon}_t \end{bmatrix} \sim N(\vec{0}, \begin{bmatrix} \sigma^2 I & 0 \\ 0 & \sigma^2 I \end{bmatrix}) \quad (6.62)$$

$$\begin{bmatrix} \vec{y} \\ \vec{y}_t \end{bmatrix} \mid X, X_t = \begin{bmatrix} \vec{f} \\ \vec{f}_t \end{bmatrix} + \begin{bmatrix} \vec{\varepsilon} \\ \vec{\varepsilon}_t \end{bmatrix} \sim N(\vec{0}, \begin{bmatrix} K(X, X) + \sigma^2 I & K(X, X_t) \\ K(X_t, X) & K(X_t, X_t) + \sigma^2 I \end{bmatrix}) \quad (6.63)$$

$$\vec{y}_t | \vec{y}, X, X_t \sim N(K(X_t, X)(K(X, X) + \sigma^2 I)^{-1}\vec{y}, K(X_t, X_t) + \sigma^2 I - K(X_t, X)(K(X, X) + \sigma^2 I)^{-1}K(X, X_t)) \quad (6.64)$$

6.1.8 Linear Regression for Binary Classification

Linear Regression for Binary Classification

see Tab 6.1

Table 6.1: Linear Classification and Linear Regression

	\mathcal{Y}	$h(\vec{x})$	$err(\hat{y}, y)$
Linear Classification	$\{-1, +1\}$	$\text{sign } \vec{\theta}^T \vec{x}$	$1\{\hat{y} \neq y\}$
Linear Regression	\mathbb{R}	$\vec{\theta}^T \vec{x}$	$(\hat{y} - y)^2$

Algorithm 11 Linear Regression for Classification

input: binary classification data D

$\vec{\theta}^* = X^+ \vec{y}$
return $h(\vec{x}) = \text{sign } \vec{\theta}^{*T} \vec{x}$

Relation of Two Errs.

$$err_{01} = 1\{\text{sign } \vec{\theta}^T \vec{x} \neq y\} \quad (6.65)$$

$$err_{sq} = (\vec{\theta}^T \vec{x} - y)^2 \quad (6.66)$$

$err_{01} \leq err_{sq}$, see Fig 6.3

$$J_{out}^{01}(\vec{\theta}) \stackrel{VC}{\leq} J_{in}^{01} + \sqrt{\cdot} \leq J_{in}^{sq}(\vec{\theta}) + \sqrt{\cdot} \quad (6.67)$$

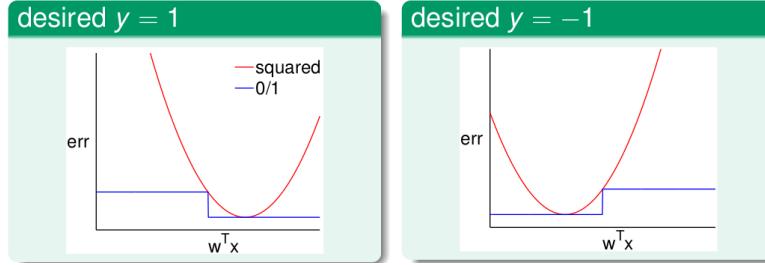


Figure 6.3: Relation of Two Errs.

- (loose) upper bound err_{sq} as $\hat{\text{err}}$ to approx. err_{01}
- trade bound tightness for efficiency
- $\vec{\theta}^*$: useful baseline classifier or as init. PLA/ pocket vec.

6.2 Logistic Regression

6.2.1 Logistic Regression Problem

binary classification: $f(\vec{x}) = \text{sign}(\Pr(y=1|\vec{x}) - \frac{1}{2}) \in \{-1, +1\}$

soft classification: $f(\vec{x}) = \Pr(y=1|\vec{x}) \in [0, 1]$

score: $s = \vec{\theta}^T \vec{x}$

logistic hypo.: $h(\vec{x}) = g(\vec{\theta}^T \vec{x}) = g(s)$

logistic fcn.

$$g(s) = \frac{1}{1 + e^{-s}} = \frac{e^s}{1 + e^s} \quad (6.68)$$

- convert the score to estimated prob.
- $g(-\infty) = 0, g(0) = \frac{1}{2}, g(\infty) = 1$
- smooth, monotonic, sigmoid fcn. of s

$$g'(s) = \frac{d}{ds} \frac{1}{1 + \exp(-s)} \quad (6.69)$$

$$= \frac{\exp(-s)}{(1 + \exp(-s))^2} \quad (6.70)$$

$$= \frac{1}{1 + \exp(-s)} \cdot \frac{\exp(-s)}{1 + \exp(-s)} \quad (6.71)$$

$$= \frac{1}{1 + \exp(-s)} \cdot \frac{1}{1 + \exp(s)} \quad (6.72)$$

$$= g(s)g(-s) \quad (6.73)$$

$$1 - g(s) = 1 - \frac{1}{1 + \exp(-s)} \quad (6.74)$$

$$= \frac{\exp(-s)}{1 + \exp(-s)} \quad (6.75)$$

$$= \frac{1}{1 + \exp(s)} \quad (6.76)$$

$$= g(-s) \quad (6.77)$$

logistic regression

$$h(\vec{x}) = \frac{1}{1 + \exp(-\vec{\theta}^T \vec{x})} \quad (6.78)$$

to approx. target fcn. $f(\vec{x}) = \Pr(y = 1 | \vec{x})$

6.2.2 Logistic Regression Error

How About Square Error?

use $y^{(i)} \in \{0, 1\}$ instead, the sq. err.

$$J(\vec{\theta}) = \frac{1}{m} \sum_{i=1}^m (h(\vec{x}^{(i)}) - y^{(i)})^2 \quad (6.79)$$

non-cvx.

Likelihood of Logistic Hypothesis

$$\Pr(y = 1 | \vec{x}) = h(\vec{x}) = g(\vec{\theta}^T \vec{x}) \quad (6.80)$$

$$\Pr(y = -1 | \vec{x}) = 1 - h(\vec{x}) = 1 - g(\vec{\theta}^T \vec{x}) = g(-\vec{\theta}^T \vec{x}) = h(-\vec{x}) \quad (6.81)$$

it can be written as

$$p(y | \vec{x}; \vec{\theta}) = h(y \vec{x}) = g(y \vec{\theta}^T \vec{x}) \quad (6.82)$$

assuming m training examples are iid.

$$\vec{\theta}^* = \arg \max_{\vec{\theta}} L(\vec{\theta}) \quad (6.83)$$

$$= \arg \max_{\vec{\theta}} \prod_{i=1}^m g(y^{(i)} \vec{\theta}^T \vec{x}^{(i)}) \quad (6.84)$$

$$= \arg \max_{\vec{\theta}} \log \prod_{i=1}^m g(y^{(i)} \vec{\theta}^T \vec{x}^{(i)}) \quad (6.85)$$

$$= \arg \max_{\vec{\theta}} \sum_{i=1}^m \log g(y^{(i)} \vec{\theta}^T \vec{x}^{(i)}) \quad (6.86)$$

Cross-Entropy Error

$$\min_{\vec{\theta}} \frac{1}{m} \sum_{i=1}^m -\log g(y^{(i)} \vec{\theta}^T \vec{x}^{(i)}) \quad (6.87)$$

$$= \min_{\vec{\theta}} \frac{1}{m} \sum_{i=1}^m -\log \frac{1}{1 + \exp(-y^{(i)} \vec{\theta}^T \vec{x}^{(i)})} \quad (6.88)$$

$$= \min_{\vec{\theta}} \frac{1}{m} \sum_{i=1}^m \log(1 + \exp(-y^{(i)} \vec{\theta}^T \vec{x}^{(i)})) \quad (6.89)$$

$$= \min_{\vec{\theta}} \frac{1}{m} \sum_{i=1}^m err(\vec{\theta}, \vec{x}^{(i)}, y^{(i)}) \quad (6.90)$$

$$= \min_{\vec{\theta}} J_{in}(\vec{\theta}) \quad (6.91)$$

if $y = 1$, $err = \log(1 + \exp(-\vec{\theta}^T \vec{x}^{(i)}))$, see Fig. 6.4

- if $h(\vec{x}) = 1$, $\vec{\theta}^T \vec{x} \gg 0$, $err = \log(1 + 0) = 0$
- if $h(\vec{w}) = 0$, $\vec{\theta}^T \vec{w} \ll 0$, $err = \log(1 + \infty) = \infty$, the alg. will penalize a lot

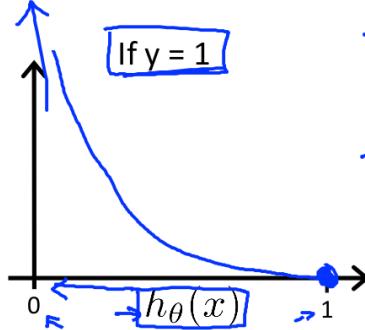


Figure 6.4: Cross-Entropy Error with $y = 1$

if $y = -1$, $err = \log(1 + \exp(\vec{\theta}^T \vec{x}^{(i)}))$, see Fig. 6.5

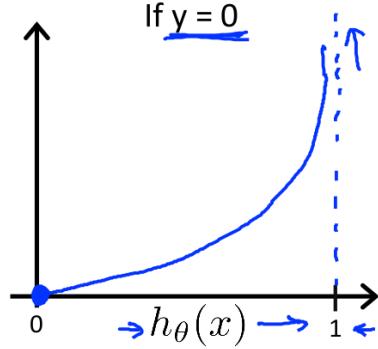
- if $h(\vec{w}) = 0$, $\vec{\theta}^T \vec{w} \ll 0$, $err = \log(1 + 0) = 0$
- if $h(\vec{x}) = 1$, $\vec{\theta}^T \vec{x} \gg 0$, $err = \log(1 + \infty) = \infty$

6.2.3 Gradient of Logistic Regression Error

Minimizing $J_{in}(\vec{\theta})$

$$\min_{\vec{\theta}} J_{in}(\vec{\theta}) = \frac{1}{m} \sum_{i=1}^m \log(1 + \exp(-y^{(i)} \vec{\theta}^T \vec{x}^{(i)})) \quad (6.92)$$

$J_{in}(\vec{\theta})$: continuous, differentiable, twice-differentiable, convex

Figure 6.5: Cross-Entropy Error with $y = -1$

The Gradient $\nabla J_{in}(\vec{\theta})$

$$\frac{\partial J_{in}(\vec{\theta})}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^m \frac{\partial \log(\cdot)}{\partial \cdot} \frac{\partial (1 + \exp(\cdot))}{\partial \cdot} \frac{\partial (-y^{(i)} \vec{\theta}^T \vec{x}^{(i)})}{\theta_j} \quad (6.93)$$

$$= \frac{1}{m} \sum_{i=1}^m \frac{1}{\exp(\cdot)} (-y^{(i)} x_j^{(i)}) \quad (6.94)$$

$$= \frac{1}{m} \sum_{i=1}^m \frac{\exp(\cdot)}{1 + \exp(\cdot)} (-y^{(i)} x_j^{(i)}) \quad (6.95)$$

$$= \frac{1}{m} \sum_{i=1}^m g(\cdot) (-y^{(i)} x_j^{(i)}) \quad (6.96)$$

$$= \frac{1}{m} \sum_{i=1}^m g(-y^{(i)} \vec{\theta}^T \vec{x}^{(i)}) (-y^{(i)} x_j^{(i)}) \quad (6.97)$$

$$\nabla J_{in}(\vec{\theta}) = \frac{1}{m} \sum_{i=1}^m g(-y^{(i)} \vec{\theta}^T \vec{x}^{(i)}) (-y^{(i)} \vec{x}^{(i)}) \stackrel{want}{=} \vec{0} \quad (6.98)$$

scaled g -weighted sum of $y^{(i)} \vec{x}^{(i)}$

- all $g(\cdot) = 0$: only if $y^{(i)} \vec{\theta}^T \vec{x}^{(i)} \gg 0 \rightarrow$ lin. sep. D
- weighted sum = $\vec{0}$: non-lin. eqn. of $\vec{\theta}$
- no closed-form sol.

PLA Revisited: Iterative Optimization

or, equivalently

Algorithm 12 PLA

idea : start with some h , repr. by its init. vec. $\vec{\theta}$, and correct its mistakes iteratively on D .

init. $\vec{\theta}$, say $\vec{0}$.

while there is mistake in D made by curr. $\vec{\theta}$

for $\{\vec{x}^{(i)}, y^{(i)}\} \in D$ **in** naive cycle (1, ..., m) or precomputed rand. cycle

if $\text{sign}(\vec{\theta}^T \vec{x}^{(i)}) \neq y^{(i)}$ // a mistake has been found

$\vec{\theta} \leftarrow \vec{\theta} + y^{(i)} \vec{x}^{(i)}$ // try to correct the mistake

return $h(\vec{x}) = \text{sign}(\vec{\theta}^T \vec{x})$

Algorithm 13 PLA

idea: start with some h , repr. by its init. vec. $\vec{\theta}$, and correct its mistakes iteratively on D .

init. $\vec{\theta}$, say $\vec{0}$.

while there is mistake in D made by curr. $\vec{\theta}$

for $\{\vec{x}^{(i)}, y^{(i)}\} \in D$ **in** naive cycle (1, ..., m) or precomputed rand. cycle

// iterative opt. approach

$\vec{\theta} \leftarrow \vec{\theta} + 1 \cdot 1\{\text{sign}(\vec{\theta}^T \vec{x}^{(i)}) \neq y^{(i)}\} y^{(i)} \vec{x}^{(i)}$

return $h(\vec{x}) = \text{sign}(\vec{\theta}^T \vec{x})$

6.2.4 Gradient Descent

Put Everything Together

batch grad. desc.: each step of grad. desc. uses all the training examples
similar time complexity to pocket per iteration

regardless of the initial $\vec{\theta}$, the final $\vec{\theta}^*$ should be similar if (0) T is large enough (1) α is properly set. Technically, (1) is more sophisticated than what's taught in the class.

Gradient Descent on Linear Regression

recall

$$J_{in}(\vec{\theta}) = \frac{1}{m} \|X\vec{\theta} - \vec{y}\|^2 = \frac{1}{m} (\vec{\theta}^T X^T X \vec{\theta} - 2\vec{\theta}^T X^T \vec{y} + \vec{y}^T \vec{y}) \quad (6.99)$$

it is cvx. quad.

$$\nabla J_{in}(\vec{\theta}) = \frac{1}{m} (2X^T X \vec{\theta} - 2X^T \vec{y}) = \frac{2}{m} (X^T X \vec{\theta} - X^T \vec{y}) \quad (6.100)$$

the magnitude of the update is prop. to the err. $X\vec{\theta} - \vec{y}$

Algorithm 14 Logistic Regression Algorithm

```

init  $\vec{\theta}$ 
while  $\nabla J_{in}(\vec{\theta}) > \varepsilon \wedge$  there is not enough iterations
     $\nabla J_{in}(\vec{\theta}) = \frac{1}{m} \sum_{i=1}^m g(-y^{(i)}\vec{\theta}^T \vec{x}^{(i)}) (-y^{(i)}\vec{x}^{(i)})$ 
     $\vec{\theta} \leftarrow \vec{\theta} - \alpha \nabla J_{in}(\vec{\theta})$ 
return  $h(\vec{x}) = g(\vec{\theta}^T \vec{x})$ 

```

Algorithm 15 Gradient Descent on Linear Regression

```

init  $\vec{\theta}$ 
while  $\nabla J_{in}(\vec{\theta}) > \varepsilon \wedge$  there is not enough iterations
     $\nabla J_{in}(\vec{\theta}) = \frac{2}{m} X^T (X\vec{\theta} - \vec{y})$ 
     $\vec{\theta} \leftarrow \vec{\theta} - \alpha \nabla J_{in}(\vec{\theta})$ 
return  $h(\vec{x}) = \vec{\theta}^T \vec{x}$ 

```

- if our prediction $X\vec{\theta}$ nearly matches the actual value of \vec{y} , then there is little need to change the paras.
- if our prediction $X\vec{\theta}$ is far from \vec{y} , a large change to the paras. will be made

6.3 Generalized Linear Models

regression: $y|x; \theta \sim N(\mu, \sigma^2)$

classification, consider $y \in \{0, 1\}$: $y|x; \theta \sim Ber(\phi)$

6.3.1 the Exponential Family

$$p(y; \vec{\eta}) = b(y) \exp(\vec{\eta}^T \vec{T}(y) - a(\vec{\eta})) \quad (6.101)$$

$\vec{\eta}$: natural para., canonical para.

$\vec{T}(y)$: sufficient statistic

$a(\vec{\eta})$: log partition fcn., usually use to be a normalization const. to let $p(y; \vec{\eta})$ sum to one

Bernoulli Distribution is in Exponential Family

$$y; \phi \sim Ber(\phi) \quad (6.102)$$

$$\Pr(y = 1; \phi) = \phi \quad (6.103)$$

$$\Pr(y = 0; \phi) = 1 - \phi \quad (6.104)$$

as we vary ϕ , we obtain Ber. with diff. means

$$p(y; \phi) = \phi^y (1 - \phi)^{1-y} \quad (6.105)$$

$$= \exp \log \phi^y (1 - \phi)^{1-y} \quad (6.106)$$

$$= \exp(\log \phi^y + \log(1 - \phi)^{1-y}) \quad (6.107)$$

$$= \exp(y \log \phi + (1 - y) \log(1 - \phi)) \quad (6.108)$$

$$= \exp\left(\log \frac{\phi}{1 - \phi} y + \log(1 - \phi)\right) \quad (6.109)$$

Thus,

$$T(y) = y \quad (6.110)$$

$$\eta = \log \frac{\phi}{1 - \phi} \quad (6.111)$$

$$\phi = \frac{\exp(\eta)}{1 + \exp(\eta)} = \frac{1}{1 + \exp(-\eta)} \quad (6.112)$$

$$a(\eta) = -\log(1 - \phi) \quad (6.113)$$

$$= -\log\left(1 - \frac{1}{1 + \exp(-\eta)}\right) \quad (6.114)$$

$$= -\log\left(\frac{\exp(-\eta)}{1 + \exp(-\eta)}\right) \quad (6.115)$$

$$= -\log\left(\frac{1}{1 + \exp(\eta)}\right) \quad (6.116)$$

$$= \log(1 + \exp(\eta)) \quad (6.117)$$

$$b(y) = 1 \quad (6.118)$$

Gaussian is in Exponential Family

$$y; \mu, \sigma \sim N(\mu, \sigma^2) \quad (6.119)$$

recall that when deriving lin. reg., σ had no effect on our final choice of $\vec{\theta}$ and $h(\vec{x})$, choose σ to be an arbitrary val. will not changing anything. So, let $\sigma = 1$ for convient.

$$p(y; \mu) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}(y - \mu)^2\right) \quad (6.120)$$

$$= \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}(y^2 - 2\mu y + \mu^2)\right) \quad (6.121)$$

$$= \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}y^2\right) \cdot \exp(\mu y - \frac{1}{2}\mu^2) \quad (6.122)$$

Thus,

$$T(y) = y \quad (6.123)$$

$$\eta = \mu \quad (6.124)$$

$$a(\eta) = \frac{1}{2}\mu^2 = \frac{1}{2}\eta^2 \quad (6.125)$$

$$b(y) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}y^2) \quad (6.126)$$

Other Distributions

multinomial

Poisson: model count-data

gamma

exponential: model continuous, non-neg. rv., such as time intervals

beta

Dirichlet: distr. over prob.

6.3.2 Constructing GLMs

consider a classification or regression problem where we would like to predict the val. of some rv. y as a fcn. of x

3 assumptions:

- $y|\vec{x}; \eta \sim \text{ExpFamily}(\vec{\eta})$
- $h(\vec{x}) = \text{E}[\vec{T}(y)|\vec{x}]$ a
- $\eta = \vec{\theta}^T \vec{x}$ or $\eta_k = \vec{\theta}_k^T \vec{x}$

Ordinary Least Squares

model

$$y|x \sim N(\mu, \sigma^2) \quad (6.127)$$

as we saw prev., Gaussian is in ExpFamily, and

$$\eta = \mu \quad (6.128)$$

$$h(\vec{x}) = \text{E}[\vec{T}(y)|\vec{x}] = \text{E}[y|\vec{x}] = \mu = \eta = \vec{\theta}^T \vec{x} \quad (6.129)$$

Logistic Regression

let $y \in \{0, 1\}$, model

$$y|x; \eta \sim \text{Ber}(\phi) \quad (6.130)$$

as we saw prev., Bernoulli is in ExpFamily, and

$$\phi = \frac{1}{1 + \exp(-\eta)} \quad (6.131)$$

$$h(\vec{x}) = \text{E}[\vec{T}(y)|\vec{x}] = \text{E}[y|\vec{x}] = \phi = \frac{1}{1 + \exp(-\eta)} = \frac{1}{1 + \exp(-\vec{\theta}^T \vec{x})} \quad (6.132)$$

$g(\eta) = \text{E}[\vec{T}(y); \eta]$: canonical response fcn., for ExpFamily, g is identify fcn.

g^{-1} : canonical link fcn., for Bernoulli, g^{-1} is the logistic fcn.

Softmax Regression

consider $y \in \{1, 2, \dots, K\}$, model it as multinomial distr.
it needs $K - 1$ paras.

$$\phi_k = p(y = k; \vec{\phi}), \text{ for } k \in \{1, 2, \dots, K - 1\} \quad (6.133)$$

ϕ_K is not a para., just for notational convenience

$$\phi_K = p(y = K; \vec{\phi}) = 1 - \sum_{k=1}^{K-1} \phi_k \quad (6.134)$$

proof multinomial is in ExpFamily def $\vec{T}(y) \in \mathbb{R}^{K-1}$ as

$$\vec{T}(1) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \vec{T}(2) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \vec{T}(3) = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \dots, \vec{T}(K-1) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}, \vec{T}(K) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad (6.135)$$

$$T(y)_k = 1\{y = k\}, \text{ for } k \in \{1, \dots, K - 1\} \quad (6.136)$$

$$\text{E}[T(y)_k] = \Pr(y = k) = \phi_k \quad (6.137)$$

$$p(y; \vec{\phi}) = \phi_1^{1\{y=1\}} \phi_2^{1\{y=2\}} \dots \phi_K^{1\{y=K\}} \quad (6.138)$$

$$= \exp \log \phi_1^{1\{y=1\}} \phi_2^{1\{y=2\}} \dots \phi_K^{1\{y=K\}} \quad (6.139)$$

$$= \exp(1\{y = 1\} \log \phi_1 + 1\{y = 2\} \log \phi_2 + \dots + 1\{y = K\} \log \phi_K) \quad (6.140)$$

$$= \exp \left(1\{y = 1\} \log \phi_1 + 1\{y = 2\} \log \phi_2 + \dots \right) \quad (6.141)$$

$$+ \left(1 - \sum_{k=1}^{K-1} 1\{y = k\} \right) \log \phi_K \quad (6.142)$$

$$= \exp \left(T(y)_1 \log \phi_1 + T(y)_2 \log \phi_2 + \dots \right) \quad (6.143)$$

$$+ \left(1 - \sum_{k=1}^{K-1} T(y)_k \right) \log \phi_K \quad (6.144)$$

$$= \exp \left(T(y)_1 \log \frac{\phi_1}{\phi_K} + T(y)_2 \log \frac{\phi_2}{\phi_K} + \dots \right) \quad (6.145) \quad \text{softmax regression}$$

$$+ T(y)_{K-1} \log \frac{\phi_{K-1}}{\phi_K} + \log \phi_K \right) \quad (6.146)$$

$$= b(y) \exp(\vec{\eta}^T \vec{T}(y) - a(\eta)) \quad (6.147)$$

$$\eta = \begin{bmatrix} \log \frac{\phi_1}{\phi_K} \\ \log \frac{\phi_2}{\phi_K} \\ \vdots \\ \log \frac{\phi_{K-1}}{\phi_K} \end{bmatrix} \in \mathbb{R}^{K-1} \quad (6.148)$$

$$b(y) = 1 \quad (6.149)$$

$$a(\eta) = -\log \phi_K \quad (6.150)$$

$$\eta_k = \log \frac{\phi_k}{\phi_K}, \text{ for } k \in \{1, 2, \dots, K-1\} \quad (6.151)$$

for convenient, also def.

$$\eta_K = \log \frac{\phi_K}{\phi_K} = 0 \quad (6.152)$$

so

$$\exp(\eta_k) = \frac{\phi_k}{\phi_K}, \text{ for } k \in \{1, 2, \dots, K\} \quad (6.153)$$

$$\sum_{k=1}^K \exp(\eta_k) = \sum_{k=1}^K \frac{\phi_k}{\phi_K} = \frac{\sum_{k=1}^K \phi_k}{\phi_K} = \frac{1}{\phi_K} \quad (6.154)$$

$$\phi_k = \exp(\eta_k) \phi_K = \frac{\exp(\eta_k)}{\sum_{j=1}^K \exp(\eta_j)} \quad (6.155)$$

softmax regression

$$\eta_k = \vec{\theta}_k^T \vec{x}, \text{ for } k = \{1, 2, \dots, K-1\} \quad (6.156)$$

because

$$\eta_K = \log \frac{\phi_K}{\phi_k} = 0 = \vec{\theta}_K^T \vec{x} \quad (6.157)$$

for notational convenience, def. $\vec{\theta}_K = \vec{0}$

$$h(\vec{x})_k = \mathbb{E}[T(y)_k | \vec{x}; \vec{\theta}] \quad (6.158)$$

$$= \mathbb{E}[1\{y = k\} | \vec{x}; \vec{\theta}] \quad (6.159)$$

$$= p(y = k | \vec{x}; \Theta) \quad (6.160)$$

$$= \phi_k \quad (6.161)$$

$$= \frac{\exp(\eta_k)}{\sum_{j=1}^K \exp(\eta_j)} \quad (6.162)$$

$$= \frac{\exp(\vec{\theta}_k^T \vec{x})}{\sum_{j=1}^K \exp(\vec{\theta}_j^T \vec{x})}, \text{ for } k = \{1, 2, \dots, K\} \quad (6.163)$$

$$h(\vec{x})_K = 1 - \sum_{k=1}^{K-1} h(\vec{x})_k \quad (6.164)$$

in other words, our hypo. will output the estimated prob. that $p(y = k | \vec{x}; \vec{\theta})$ for $k = \{1, 2, \dots, K\}$

para. fitting max. likelihood

$$\Theta^* = \arg \max_{\Theta} \prod_{i=1}^m p(y^{(i)} | \vec{x}^{(i)}; \Theta) \quad (6.165)$$

$$= \arg \max_{\Theta} \log \prod_{i=1}^m p(y^{(i)} | \vec{x}^{(i)}; \Theta) \quad (6.166)$$

$$= \arg \max_{\Theta} \log \prod_{i=1}^m \phi_1^{1\{y^{(i)}=1\}} \phi_2^{1\{y^{(i)}=2\}} \dots \phi_K^{1\{y^{(i)}=K\}} \quad (6.167)$$

$$= \arg \max_{\Theta} \sum_{i=1}^m \log \prod_{k=1}^K \phi_k^{1\{y^{(i)}=k\}} \quad (6.168)$$

$$= \arg \max_{\Theta} \sum_{i=1}^m \log \prod_{k=1}^K \left(\frac{\exp(\vec{\theta}_k^T \vec{x}^{(i)})}{\sum_{j=1}^K \exp(\vec{\theta}_j^T \vec{x}^{(i)})} \right)^{1\{y^{(i)}=k\}} \quad (6.169)$$

$$= \arg \max_{\Theta} \sum_{i=1}^m \sum_{k=1}^K \log \left(\frac{\exp(\vec{\theta}_k^T \vec{x}^{(i)})}{\sum_{j=1}^K \exp(\vec{\theta}_j^T \vec{x}^{(i)})} \right)^{1\{y^{(i)}=k\}} \quad (6.170)$$

$$= \arg \max_{\Theta} \sum_{i=1}^m \sum_{k=1}^K 1\{y^{(i)} = k\} \left(\vec{\theta}_k^T \vec{x}^{(i)} - \log \sum_{j=1}^K \exp(\vec{\theta}_j^T \vec{x}^{(i)}) \right) \quad (6.171)$$

$$= \arg \min_{\Theta} \sum_{i=1}^m \sum_{k=1}^K 1\{y^{(i)} = k\} \left(\log \sum_{j=1}^K \exp(\vec{\theta}_j^T \vec{x}^{(i)}) - \vec{\theta}_k^T \vec{x}^{(i)} \right) \quad (6.172)$$

$$= \arg \min_{\Theta} J_{in}(\Theta) \quad (6.173)$$

$$\nabla_{\theta_k} J_{in}(\Theta) = \sum_{i=1}^m \left(\frac{\exp(\vec{\theta}_k^T \vec{x}^{(i)}) \vec{x}^{(i)}}{\sum_{j=1}^K \exp(\vec{\theta}_j^T \vec{x}^{(i)})} - 1\{y^{(i)} = k\} \vec{x}^{(i)} \right) \quad (6.174)$$

$$= \sum_{i=1}^m \left(\frac{\exp(\vec{\theta}_k^T \vec{x}^{(i)})}{\sum_{j=1}^K \exp(\vec{\theta}_j^T \vec{x}^{(i)})} - 1\{y^{(i)} = k\} \right) \vec{x}^{(i)} \quad (6.175)$$

$$= \sum_{i=1}^m (h(\vec{x}^{(i)})_k - 1\{y^{(i)} = k\}) \vec{x}^{(i)} \quad (6.176)$$

can use a method such as grad. ascent or Newton's method

6.3.3 Stochastic Gradient Ascent on GLM

consider case $\vec{T}(y) = y$
max likelihood on GLM

$$\theta^* = \arg \max_{\vec{\theta}} L(\vec{\theta}) \quad (6.177)$$

$$= \arg \max_{\vec{\theta}} \log \prod_{i=1}^m p(y^{(i)} | \vec{x}^{(i)}; \vec{\theta}) \quad (6.178)$$

$$= \arg \max_{\vec{\theta}} \sum_{i=1}^m \log b(y^{(i)}) \exp(\eta y^{(i)} - a(\eta)) \quad (6.179)$$

$$= \arg \max_{\vec{\theta}} \sum_{i=1}^m \log b(y^{(i)}) + \sum_{i=1}^m (\eta y^{(i)} - a(\eta)) \quad (6.180)$$

$$= \arg \max_{\vec{\theta}} \sum_{i=1}^m (\eta y^{(i)} - a(\eta)) \quad (6.181)$$

$$= \arg \max_{\vec{\theta}} \sum_{i=1}^m (y^{(i)} \vec{\theta}^T \vec{x}^{(i)} - a(\vec{\theta}^T \vec{x}^{(i)})) \quad (6.182)$$

$$= \arg \max_{\vec{\theta}} \sum_{i=1}^m err(\vec{\theta}, \vec{x}^{(i)}, y^{(i)}) \quad (6.183)$$

$$\int_{-\infty}^{\infty} p(y | \vec{x}; \vec{\theta}) dy = \int_{-\infty}^{\infty} b(y) \exp(\eta y - a(\eta)) dy = 1 \quad (6.184)$$

$$\therefore \int_{-\infty}^{\infty} b(y) \exp(\eta y) dy = \exp(a(\eta)) \quad (6.185)$$

∇_{η} on both side

$$\int_{-\infty}^{\infty} b(y) \exp(\eta y) y dy = \exp(a(\eta)) \nabla a(\eta) \quad (6.186)$$

$$\therefore \nabla a(\eta) = \int_{-\infty}^{\infty} y b(y) \exp(\eta y - a(\eta)) dy \quad (6.187)$$

$$= \int_{-\infty}^{\infty} yp(y|\vec{x}; \vec{\theta}) dy \quad (6.188)$$

$$= E[y|\vec{x}; \vec{\theta}] \quad (6.189)$$

$$= h(\vec{x}) \quad (6.190)$$

$$\nabla err(\vec{\theta}, \vec{x}^{(i)}, y^{(i)}) = y^{(i)}\vec{x}^{(i)} - \nabla a(\vec{\theta}^T \vec{x}^{(i)})\vec{x}^{(i)} \quad (6.191)$$

$$= (y^{(i)} - \nabla a(\vec{\theta}^T \vec{x}^{(i)}))\vec{x}^{(i)} \quad (6.192)$$

$$= (y^{(i)} - h(\vec{x}^{(i)}))\vec{x}^{(i)} \quad (6.193)$$

$$(6.194)$$

stochastic grad. ascent rule

$$\vec{\theta} \leftarrow \vec{\theta} + \alpha \nabla err(\vec{\theta}, \vec{x}^{(i)}, y^{(i)}) \quad (6.195)$$

$$= \vec{\theta} - \alpha(h(\vec{x}^{(i)}) - y^{(i)})\vec{x}^{(i)} \quad (6.196)$$

$$(6.197)$$

6.4 Generative Learning Algorithms

6.4.1 Discriminant and Generative Learning Algorithms

consider a classification problem in which we want to distinguish betw. elephants ($y = 1$) and dogs ($y = 0$)

discriminative learning alg.

- tries to find a dec. boundary that sep. the elephants and dogs
- prediction: check on which side of the dec. boundary it falls
- model $p(y|\vec{x})$ directly (as log. reg.) or the mapping directly from $\mathcal{X} \rightarrow \{-1, +1\}$ (as perceptron)

generative learning alg.

- looking at elephants, build a model of what elephants look like
- looking at dogs, build a model of what dogs look like
- prediction: match the new animal against the elephant model and dog model, to see whether the new animal looks more like the elephants or more like the dogs
- model $p(x|y)$ and $p(y)$
- use Bayes rule to derive the posterior distr.

$$y = \arg \max_y p(y|\vec{x}) \quad (6.198)$$

$$= \arg \max_y \frac{p(\vec{x}|y)p(y)}{p(\vec{x})} \quad (6.199)$$

$$= \arg \max_y p(\vec{x}|y)p(y) \quad (6.200)$$

$$p(\vec{x}) = \sum_k p(\vec{x}|y=k)p(y=k) \quad (6.201)$$

6.4.2 Gaussian Discriminant Analysis (GDA)

assume $p(\vec{x}|y) \sim N$

the GDA Model

model when \vec{x} is continuous

$$p(y) = \phi^y(1 - \phi)^{1-y} \sim Ber(\phi) \quad (6.202)$$

$$p(\vec{x}|y=0) = \frac{1}{\sqrt{2\pi^n} \sqrt{|\Sigma|}} \exp\left(-\frac{1}{2}(\vec{x} - \vec{\mu}_0)^T \Sigma^{-1} (\vec{x} - \vec{\mu}_0)\right) \sim N(\vec{\mu}_0, \Sigma) \quad (6.203)$$

$$p(\vec{x}|y=1) = \frac{1}{\sqrt{2\pi^n} \sqrt{|\Sigma|}} \exp\left(-\frac{1}{2}(\vec{x} - \vec{\mu}_1)^T \Sigma^{-1} (\vec{x} - \vec{\mu}_1)\right) \sim N(\vec{\mu}_1, \Sigma) \quad (6.204)$$

by max. l wrt. the paras.

$$\arg \max l(\phi, \vec{\mu}_0, \vec{\mu}_1, \Sigma) \quad (6.205)$$

$$= \arg \max \log \prod_{i=1}^m p(\vec{x}^{(i)}, y^{(i)}; \phi, \vec{\mu}_0, \vec{\mu}_1, \Sigma) \quad (6.206)$$

$$= \arg \max \log \prod_{i=1}^m p(\vec{x}^{(i)}|y^{(i)}; \vec{\mu}_0, \vec{\mu}_1, \Sigma) p(y^{(i)}; \phi) \quad (6.207)$$

$$= \arg \max \sum_{i=1}^m \left(-\log |\Sigma| - (\vec{x}^{(i)} - \vec{\mu}_{y^{(i)}})^T \Sigma^{-1} (\vec{x}^{(i)} - \vec{\mu}_{y^{(i)}}) + y^{(i)} \log \phi + (1 - y^{(i)}) \log (1 - \phi) \right) \quad (6.208)$$

$$\phi = \frac{1}{m} \sum_{i=1}^m 1\{y^{(i)} = 1\} \quad (6.209)$$

$$\vec{\mu}_0 = \frac{\sum_{i=1}^m 1\{y^{(i)} = 0\} \vec{x}^{(i)}}{\sum_{i=1}^m 1\{y^{(i)} = 0\}} \quad (6.210)$$

$$\vec{\mu}_1 = \frac{\sum_{i=1}^m 1\{y^{(i)} = 1\} \vec{x}^{(i)}}{\sum_{i=1}^m 1\{y^{(i)} = 1\}} \quad (6.211)$$

$$\Sigma = \frac{1}{m} \sum_{i=1}^m (\vec{x}^{(i)} - \vec{\mu}_{y^{(i)}})(\vec{x}^{(i)} - \vec{\mu}_{y^{(i)}})^T \quad (6.212)$$

what the alg. is doing can be seen in Fig. 6.6

Discussion: GDA and Logistic Regression

if $p(x|y)$ is Gaussian, then $p(y|x)$ is log. fcn.

$$p(y=1|\vec{x}; \phi, \vec{\mu}_0, \vec{\mu}_1, \Sigma) \quad (6.213)$$

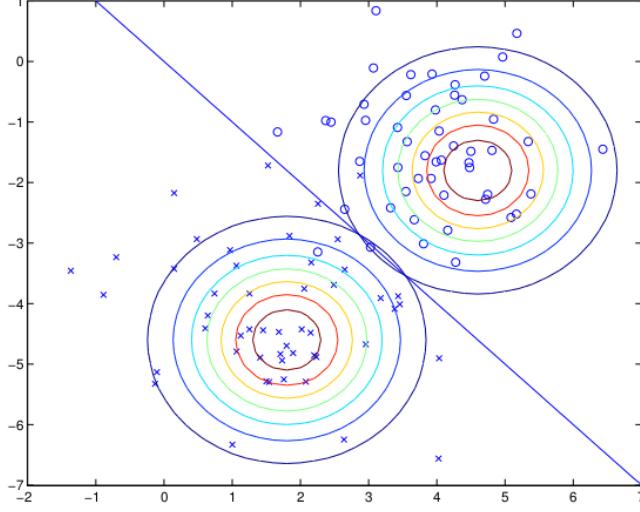


Figure 6.6: GDA

$$= \frac{p(\vec{x}|y=1)p(y=1)}{p(\vec{x}|y=1)p(y=1) + p(\vec{x}|y=0)p(y=0)} \quad (6.214)$$

$$= \frac{\frac{1}{\sqrt{2\pi^n} \sqrt{|\Sigma|}} \exp(-\frac{1}{2}(\vec{x} - \vec{\mu}_1)^T \Sigma^{-1} (\vec{x} - \vec{\mu}_1))\phi}{\frac{1}{\sqrt{2\pi^n} \sqrt{|\Sigma|}} \exp(-\frac{1}{2}(\vec{x} - \vec{\mu}_1)^T \Sigma^{-1} (\vec{x} - \vec{\mu}_1))\phi + \frac{1}{\sqrt{2\pi^n} \sqrt{|\Sigma|}} \exp(-\frac{1}{2}(\vec{x} - \vec{\mu}_0)^T \Sigma^{-1} (\vec{x} - \vec{\mu}_0))(1-\phi)} \quad (6.215)$$

$$= \frac{1}{1 + \exp\left((\mu_0 - \mu_1)^T \Sigma^{-1} \vec{x} + \frac{1}{2}(\mu_1^T \Sigma^{-1} \mu_1 - \mu_0^T \Sigma^{-1} \mu_0) + \log \frac{1-\phi}{\phi}\right)} \quad (6.216)$$

$$= \frac{1}{1 + \exp(-\vec{\theta}^T \vec{x})} \quad (6.217)$$

with

$$\vec{\theta} = \begin{bmatrix} \frac{1}{2}(\mu_0^T \Sigma^{-1} \mu_0 - \mu_1^T \Sigma^{-1} \mu_1) - \log \frac{1-\phi}{\phi} \\ \Sigma^{-1}(\mu_1 - \mu_0) \end{bmatrix} \in \mathbb{R}^{n+1} \quad (6.218)$$

and $\vec{x} \in \mathbb{R}^{n+1}$

the converse is not true: $p(y|x)$ is log. fcn. does not imply $p(x|y)$ is Gaussian

GDA makes stronger modelling assumptions about the data than log. reg.

when $p(x|y)$ is indeed gaussian, then GDA is asymptotically efficient

by making significantly weaker assumptions, log. reg. is more robust and less sensitive to incorrect modelling assumptions. When the data is indeed non-Gaussian, then log. reg. will almost always do better than GDA

for this reason, in practice log. reg. is used more often than GDA

6.4.3 Naive Bayes

Text Classification Problem

specify feature $\vec{x} \in \mathbb{R}^n$

$$x_j = 1\{\text{email contains the } j\text{-th word in the vocabulary}\} \quad (6.219)$$

vocabulary

- build: by looking through our training set and encode in our feature vec. only the words that occur at least once there
- it can reduce # words modeled and allow us to model/ include as a feature many words that may appear in your email but not in the dict.
- exclude very high freq. words (like "the", "of", "and") since they occur in so many docs. and do little to indicate whether an email is spam or not spam
- n is equal to the size of vocabulary

build a generative model: model $p(\vec{x}|y)$

$x_j \in \{0, 1\}$, if we were to model \vec{x} explicitly with a multinomial distr. over the 2^n possible outcomes, then we'd end up with a $2^n - 1$ dim. para. vec.

Naive Bayes

to model $p(\vec{x}|y)$, we therefore make a very strong assumption

Naive Bayes assumption: assume that the x_j 's are conditionally indep. given y , eg.

- if I tell you $y = 1$ (that email is spam)
- knowledge of x_{1126} (whether 'buy' appears in the msg.) will have no effect on your beliefs about the val. of x_{2087} (whether 'price' appears)
- $p(x_{1126}|y) = p(x_{2087}|y)$
- this does not mean x_{1126} and x_{2087} are indep. which would be written as $p(x_{1126}) = p(x_{1126}|x_{2087})$

now we have

$$p(\vec{x}|y) = p(x_1, x_2, \dots, x_n|y) \quad (6.220)$$

$$= p(x_1|y)p(x_2|y, x_1)p(x_3|y, x_1, x_2)\dots p(x_n|y, x_1, x_2, \dots, x_{n-1}) \quad (6.221)$$

$$= p(x_1|y)p(x_2|y)p(x_3|y)\dots p(x_n|y) \quad (6.222)$$

$$= \prod_{j=1}^n p(x_j|y) \quad (6.223)$$

model it as multi-variate Bernoulli event model

$$x_j|y = 1 \sim Ber(\phi_{j|y=1}), \forall j \quad (6.224)$$

$$x_j|y=0 \sim Ber(\phi_{j|y=0}), \forall j \quad (6.225)$$

$$\phi_{j|y=1} = p(x_j = 1|y = 1), \forall j \quad (6.226)$$

$$\phi_{j|y=0} = p(x_j = 1|y = 0), \forall j \quad (6.227)$$

$$\phi_y = p(y = 1) \quad (6.228)$$

$$p(x_j|y = 1) = (\phi_{j|y=1})^{x_j} (1 - \phi_{j|y=1})^{1-x_j}, x_j = \{0, 1\}, \forall j \quad (6.229)$$

$$p(x_j|y = 0) = (\phi_{j|y=0})^{x_j} (1 - \phi_{j|y=0})^{1-x_j}, x_j = \{0, 1\}, \forall j \quad (6.230)$$

an email is gen.

- 1. randomly determined according to the class priors $p(y)$ whether a spammer or non-spammer will send you the email
- 2. the person sending the email runs through the vocabulary, dec. whether to include each word j in that email indep. according to $p(x_j = 1|y)$
 - for each word x_j
 - flip a skew coin with prob. of heads $p(x_j = 1|y)$
 - if heads, x_j is included in the email
- thus, the prob. of an email is given is $p(\vec{x}, y) = p(y) \prod_{j=1}^n p(x_j|y)$ joint likelihood

$$L(\phi_y, \phi_{j|y=1}, \phi_{j|y=0}) = \prod_{i=1}^m p(\vec{x}^{(i)}, y^{(i)}) \quad (6.231)$$

max. wrt. $\phi_y, \phi_{j|y=1}, \phi_{j|y=0}$

$$\phi_{j|y=1} = \frac{\sum_{i=1}^m 1\{x_j^{(i)} = 1 \wedge y^{(i)} = 1\}}{\sum_{i=1}^m 1\{y^{(i)} = 1\}} \quad (6.232)$$

$$\phi_{j|y=0} = \frac{\sum_{i=1}^m 1\{x_j^{(i)} = 1 \wedge y^{(i)} = 0\}}{\sum_{i=1}^m 1\{y^{(i)} = 0\}} \quad (6.233)$$

$$\phi_y = \frac{\sum_{i=1}^m 1\{y^{(i)} = 1\}}{m} \quad (6.234)$$

prediction:

$$p(y = 1|\vec{x}) = \frac{p(\vec{x}|y = 1)p(y = 1)}{p(\vec{x})} \quad (6.235)$$

$$= \frac{p(\vec{x}|y = 1)p(y = 1)}{p(\vec{x}|y = 1)p(y = 1) + p(\vec{x}|y = 0)p(y = 0)} \quad (6.236)$$

$$= \frac{\prod_{j=1}^n p(x_j|y = 1)p(y = 1)}{\prod_{j=1}^n p(x_j|y = 1)p(y = 1) + \prod_{j=1}^n p(x_j|y = 0)p(y = 0)} \quad (6.237)$$

and pick whichever class has the higher posterior prob.

when x_j can take val. in $\{1, 2, \dots, K_j\}$ rather than binary val., we would simple model $p(x_j|y)$ as multinomial rather than Bernoulli

when x_j is continuous, it is quite common to discretize it and apply Naive Bayes, and model $p(x_j|y)$ as multinomial, see Fig. 6.7

when the orig. continuous-val. attrs. are not well modeled by a multi-variate Gaussian distr., discretizing and using Naive Bayes (instead of GDA) will often rlt. in a better classifier

Living area (sq. feet)	< 400	400-800	800-1200	1200-1600	>1600
x_i	1	2	3	4	5

Figure 6.7: discretize

Laplace Smoothing

if word 'nips' did not appear in your training set of spam/ non-spam emails, assume 'nips' is the 3500-th word in vocabulary, then

$$\phi_{3500|y=1} = \frac{\sum_{i=1}^m 1\{x_{3500}^{(i)} = 1 \wedge y^{(i)} = 1\}}{\sum_{i=1}^m 1\{y^{(i)} = 1\}} = 0 \quad (6.238)$$

$$\phi_{3500|y=0} = \frac{\sum_{i=1}^m 1\{x_{3500}^{(i)} = 1 \wedge y^{(i)} = 0\}}{\sum_{i=1}^m 1\{y^{(i)} = 0\}} = 0 \quad (6.239)$$

MLE thinks the prob. of seeing it in either type of email is zero

when trying to dec. if one of these msgs. containing 'nips' is spam

$$p(y = 1|\vec{x}) = \frac{p(\vec{x}|y = 1)p(y = 1)}{p(\vec{x})} \quad (6.240)$$

$$= \frac{\prod_{j=1}^n p(x_j|y = 1)p(y = 1)}{\prod_{j=1}^n p(x_j|y = 1)p(y = 1) + \prod_{j=1}^n p(x_j|y = 0)p(y = 0)} \quad (6.241)$$

$$\text{// because } \prod_{j=1}^n p(x_j|y) \text{ includes a term } p(x_{3500}|y) = (6.242)$$

$$= \frac{0}{0} \quad (6.243)$$

it is stat. a bad idea to estimate the prob. of some event to be zero just because you haven't seen it before in your training set.

consider case a multinomial rv. $y \in \{1, 2, \dots, K\}$, para.

$$\phi_k = p(y = k) \quad (6.244)$$

MLE:

$$\phi_k = \frac{\sum_{i=1}^m 1\{y^{(i)} = k\}}{m} \quad (6.245)$$

to avoid some of the ϕ_k to be zero, we use Laplace Smoothing

$$\phi_k = \frac{\sum_{i=1}^m 1\{y^{(i)} = k\} + 1}{m + K} \quad (6.246)$$

- $\sum_{k=1}^K \phi_k = 1$ still holds true
- $\phi_k \neq 0, \forall k$
- under certain conditions, it can be shown that the Laplace Smoothing actually gives the opt. estimator of the ϕ_j 's back to Naive Bayes classifier

$$\phi_{j|y=1} = \frac{\sum_{i=1}^m 1\{x_j^{(i)} = 1 \wedge y^{(i)} = 1\} + 1}{\sum_{i=1}^m 1\{y^{(i)} = 1\} + 2} \quad (6.247)$$

$$\phi_{j|y=0} = \frac{\sum_{i=1}^m 1\{x_j^{(i)} = 1 \wedge y^{(i)} = 0\} + 1}{\sum_{i=1}^m 1\{y^{(i)} = 0\} + 2} \quad (6.248)$$

it usually does not matter much whether we apply Laplace Smoothing to ϕ_y or not, since we typically have a fair frac. each of spam and non-spam emails, so ϕ_y will be quite far from 0

Naive Bayes often works surprisingly well, it is often also a very good "first thing" to try

Multinomial Event Model for Text Classification

does even better than Naive Bayes
 $\vec{x} \in \mathbb{R}^n$ where

$$x_j = \# \text{ times the email contains the } j\text{-th word in the vocabulary} \quad (6.249)$$

$$\vec{x}|y = 1 \sim \text{Multinomial}(K, \vec{\phi}_{y=1}) \quad (6.250)$$

$$\vec{x}|y = 0 \sim \text{Multinomial}(K, \vec{\phi}_{y=0}) \quad (6.251)$$

$$\vec{\phi}_{y=1} = \begin{bmatrix} \phi_{1|y=1} \\ \phi_{2|y=1} \\ \vdots \\ \phi_{n|y=1} \end{bmatrix} \in \mathbb{R}^n \quad (6.252)$$

$$\vec{\phi}_{y=0} = \begin{bmatrix} \phi_{1|y=0} \\ \phi_{2|y=0} \\ \vdots \\ \phi_{n|y=0} \end{bmatrix} \in \mathbb{R}^n \quad (6.253)$$

$$\phi_{j|y=1} = p(x_j|y = 1), \forall j \quad (6.254)$$

$$\phi_{j|y=0} = p(x_j|y = 0), \forall j \quad (6.255)$$

$$p(\vec{x}|y=1) = \binom{K}{x_1, x_2, \dots, x_n} \prod_{j=1}^n (\phi_{j|y=1})^{x_j} \quad (6.256)$$

$$= \frac{K!}{\prod_{j=1}^n x_j!} \prod_{j=1}^n (\phi_{j|y=1})^{x_j} \quad (6.257)$$

$$p(\vec{x}|y=0) = \binom{K}{x_1, x_2, \dots, x_n} \prod_{j=1}^n (\phi_{j|y=0})^{x_j} \quad (6.258)$$

$$= \frac{K!}{\prod_{j=1}^n x_j!} \prod_{j=1}^n (\phi_{j|y=0})^{x_j} \quad (6.259)$$

$$\phi_y = p(y=1) \quad (6.260)$$

$K = \sum_{j=1}^n x_j$ is # words in email \vec{x}

$p(x_j|y)$: prob. of j -th word in the vocabulary occurring in the email an email is gen.

- 1. spam/ non-spam is determined according to $p(y)$ as before
 - 2. sender use a n -sided skew dice, each side j corresponds to the j -th word in the vocabulary
 - 3. roll the side and insert the corresponding words
 - 4. repeat it K times
- thus, the prob. of an email is given is $p(\vec{x}, y) = p(y)p(\vec{x}|y)$, it looks the same, but meaning is diff.

likelihood

$$L(\vec{\phi}_{y=1}, \vec{\phi}_{y=0}, \phi_y) \quad (6.261)$$

$$= \prod_{i=1}^m p(\vec{x}^{(i)}, y^{(i)}) \quad (6.262)$$

$$= \prod_{i=1}^m p(\vec{x}^{(i)}|y^{(i)})p(y^{(i)}) \quad (6.263)$$

$$= \prod_{i=1}^m \binom{K^{(i)}}{x_1^{(i)}, x_2^{(i)}, \dots, x_n^{(i)}} \prod_{j=1}^n (\phi_{j|y^{(i)}})^{x_j^{(i)}} (\phi_y)^{y^{(i)}} (1 - \phi_y)^{1-y^{(i)}} \quad (6.264)$$

MLE and Laplace Smoothing

$$\phi_{j|y=1} = \frac{\sum_{i=1}^m 1\{y^{(i)} = 1\} x_j^{(i)} + 1}{\sum_{i=1}^m 1\{y^{(i)} = 1\} K_i + n} \quad (6.265)$$

$$\phi_{j|y=0} = \frac{\sum_{i=1}^m 1\{y^{(i)} = 0\} x_j^{(i)} + 1}{\sum_{i=1}^m 1\{y^{(i)} = 0\} K_i + n} \quad (6.266)$$

$$\phi_y = \frac{\sum_{i=1}^m 1\{y^{(i)} = 1\}}{m} \quad (6.267)$$

6.5 Linear Models for Classification

6.5.1 Linear Models for Binary Classification

Lin. Models Revisited

lin. score fcn. $s = \vec{\theta}^T \vec{x}$
 lin. models, see Tab 6.2

Table 6.2: Linear Models

	$h(\vec{x})$	err	$J_{in}(\vec{\theta})$
Linear Classification	$\text{sign}(s)$	$err_{01} = 1\{\text{sign}(s) \neq y\}$	discrete, NP-hard to solve
Linear Regression	s	$err_{sq} = (s - y)^2$	quad. cvx., closed-form sol.
Logistic Regression	$g(s)$	$err_{ce} = \log(1 + \exp(-ys))$	smooth cvx., grad. desc.

Visualizing Error Fcns.

see Fig. 6.8

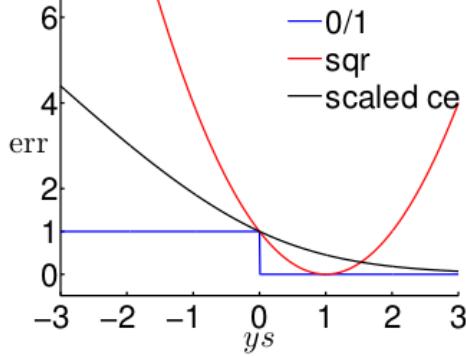


Figure 6.8: Visualizing Error Fcns.

$$err_{01} = 1\{\text{sign}(s) \neq y\} = 1\{\text{sign}(ys) \neq 1\} \quad (6.268)$$

- 1 iff. $ys \leq 0$

$$err_{sq} = (s - y)^2 = (s - y)^2 y^2 = (ys - 1)^2 \quad (6.269)$$

- large if $ys \ll 1$
- but overlarge $ys \gg 1$
- small $err_{sq} \rightarrow$ small err_{01}

$$err_{ce} = \log(1 + \exp(-ys)) \quad (6.270)$$

- monotonic of ys
- small $err_{ce} \rightarrow$ small err_{01}

$$err_{sce} = \lg(1 + \exp(-ys)) \quad (6.271)$$

- a proper upper bound of err_{01}
 - small $err_{sce} \rightarrow$ small err_{01}
- upper bound: useful for designing alg. err \hat{err}

Theoretical Implication of Upper Bound

$$err_{01}(s, y) \leq err_{sce}(s, y) = \frac{1}{\lg 2} err_{ce}(s, y) \quad (6.272)$$

VC on 01:

$$J_{out}^{01}(\vec{\theta}) \leq J_{in}^{01}(\vec{\theta}) + \sqrt{\cdot} \leq \frac{1}{\lg 2} J_{in}^{ce}(\vec{\theta}) + \sqrt{\cdot} \quad (6.273)$$

VC-Reg on CE:

$$J_{out}^{01}(\vec{\theta}) \leq \frac{1}{\lg 2} J_{out}^{ce}(\vec{\theta}) \leq \frac{1}{\lg 2} (J_{in}^{ce}(\vec{\theta}) + \sqrt{\cdot}) \quad (6.274)$$

small $J_{in}^{ce}(\vec{\theta}) \rightarrow$ small $J_{out}^{01}(\vec{\theta})$
 log./ lin. reg. for lin. classification

Reg. for Classification

Algorithm 16 Regression for Classification

input: D with $y^{(i)} \in \{-1, +1\}$

$\vec{\theta} \leftarrow \text{logReg}(D)$ or $\vec{\theta} \leftarrow \text{linReg}(D)$
return $h(\vec{x}) = \text{sign}(\vec{\theta}^T \vec{x})$

PLA

- pros: efficient and strong guarantee if lin. sep.
 - cons: works only if lin. sep., ow. needing pocket heuristic
- Lin. Reg.

- pros: easiest opt.

- cons: loose bound of err_{01} for large $|ys|$

Log. Reg.

- pros: easy opt.

- cons: loose bound of err_{01} for very neg. y_s
- lin. reg. sometimes used to set $\vec{\theta}_0$ for PLA/ pocket/ log. reg.
- log. reg. often preferred over pocket

6.5.2 Stochastic Gradient Descent

Two Iterative Optimization Schemes

PLA

- pick $(\vec{x}^{(i)}, y^{(i)})$ and decide $\vec{\theta}_{t+1}$ by the one example
- $O(1)$ time per iteration
- log. reg. (pocket)
- check D and decide $\vec{\theta}_{t+1}$ by all examples
- $O(m)$ time per iteration

Log. Reg. Revisited

Algorithm 17 Logistic Regression Algorithm

```

init  $\vec{\theta}$ 
while  $\nabla J_{in}(\vec{\theta}) \neq 0 \wedge$  there is not enough iterations
     $\vec{\theta} \leftarrow \vec{\theta} - \alpha \frac{1}{m} \sum_{i=1}^m g(-y^{(i)} \vec{\theta}^T \vec{x}^{(i)}) (-y^{(i)} \vec{x}^{(i)})$ 
return  $h(\vec{x}) = g(\vec{\theta}^T \vec{x})$ 

```

Algorithm 18 Stochastic Logistic Regression Algorithm

```

init  $\vec{\theta}$ 
randomly shaffle dataset
while there is not enough iterations
    for  $i = 1 : m$ 
         $\vec{\theta} \leftarrow \vec{\theta} + \alpha g(-y^{(i)} \vec{\theta}^T \vec{x}^{(i)}) (y^{(i)} \vec{x}^{(i)})$ 
return  $h(\vec{x}) = g(\vec{\theta}^T \vec{x})$ 

```

rule-of-thumb: $\alpha = 0.1$ when $\vec{x}^{(i)}$ in proper range

PLA Revisited

SGD log. reg. \approx soft PLA with $\alpha = 1$ when $\vec{\theta}^T \vec{x}^{(i)}$ large

6.5.3 Multiclass via Logistic Regression

$$\mathcal{Y} = \{y_1, y_2, y_3, y_4\}$$

Algorithm 19 PLA

idea: start with some h , repr. by its init. vec. $\vec{\theta}$, and correct its mistakes iteratively on D .

```

init.  $\vec{\theta}$ , say  $\vec{0}$ .
while there is mistake in  $D$  made by curr.  $\vec{\theta}$ 
  for  $\{\vec{x}^{(i)}, y^{(i)}\} \in D$  in naive cycle (1, ..., m) or precomputed rand. cycle
    // iterative opt. approach
     $\vec{\theta} \leftarrow \vec{\theta} + 1 \cdot 1\{\text{sign}(\vec{\theta}^T \vec{x}^{(i)}) \neq y^{(i)}\} y^{(i)} \vec{x}^{(i)}$ 
return  $h(\vec{x}) = \text{sign}(\vec{\theta}^T \vec{x})$ 

```

One Class at a Time

y_1 or not?

y_2 or not?

y_3 or not?

y_4 or not?

prediction: comb. binary classifiers, see Fig 6.9

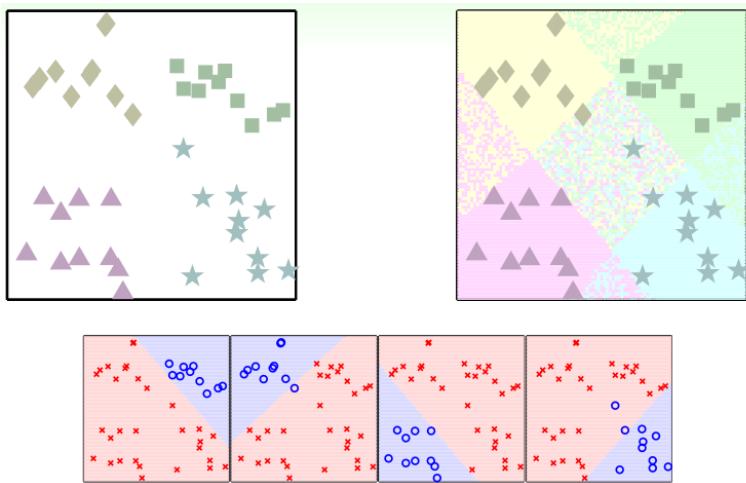


Figure 6.9: One Class at a Time

One Class at a Time Softly

$\Pr(y_1|\vec{x})? \rightarrow \vec{\theta}_1$

$\Pr(y_2|\vec{x})? \rightarrow \vec{\theta}_2$

$\Pr(y_3|\vec{x})? \rightarrow \vec{\theta}_3$

$\Pr(y_4|\vec{x})? \rightarrow \vec{\theta}_4$

prediction: comb. soft classifiers, see Fig 6.10

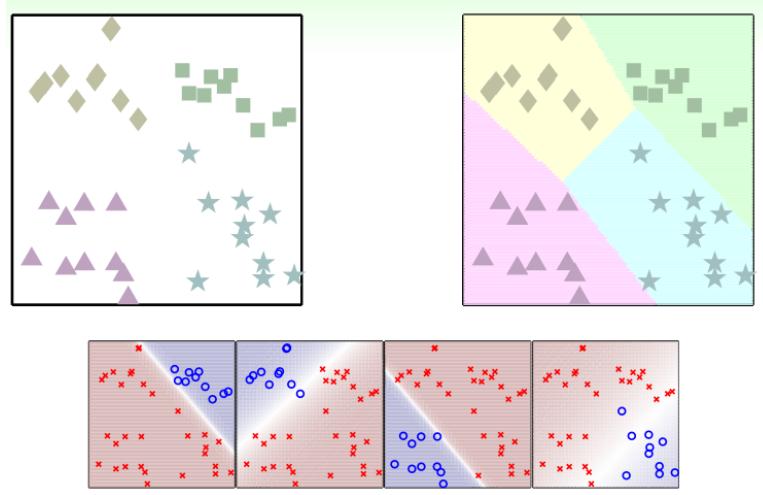


Figure 6.10: One Class at a Time Softly

$$h(\vec{x}) = \arg \max_k g(\vec{\theta}_k^T \vec{x}) \quad (6.275)$$

Algorithm 20 One-Versus-All (OVA) Decomposition

input: D with $\mathcal{Y} = \{y_1, y_2, \dots, y_K\}$

for $k = 1 : K$
 $D_k = \{(\vec{x}^{(i)}, \tilde{y}^{(i)} = 2 \cdot 1\{y^{(i)} = y_k\} - 1)\}_{i=1}^m$
 $\vec{\theta}_k \leftarrow \text{logReg}(D_k)$
return $h(\vec{x}) = \arg \max_k g(\vec{\theta}_k^T \vec{x})$

pros: simple, efficient, can be easily done in parallel, can be coupled with any log. reg.-like approaches

cons: often unbalanced D_k when K large

extension: multinomial (coupled) log. reg.

6.5.4 Multiclass via Binary Classification

Source of Unbalance: One vs. All

idea: make binary classification problems more balanced by one vs. one

One vs. One at a Time

$$\begin{aligned} y_1 \text{ or } y_2? &\rightarrow \vec{\theta}_{12} \\ y_1 \text{ or } y_3? &\rightarrow \vec{\theta}_{13} \\ y_1 \text{ or } y_4? &\rightarrow \vec{\theta}_{14} \end{aligned}$$

y_2 or $y_3?$ $\rightarrow \vec{\theta}_{23}$

y_2 or $y_4?$ $\rightarrow \vec{\theta}_{24}$

y_3 or $y_4?$ $\rightarrow \vec{\theta}_{34}$

prediction: comb. pairwise classifiers, see Fig. 6.11

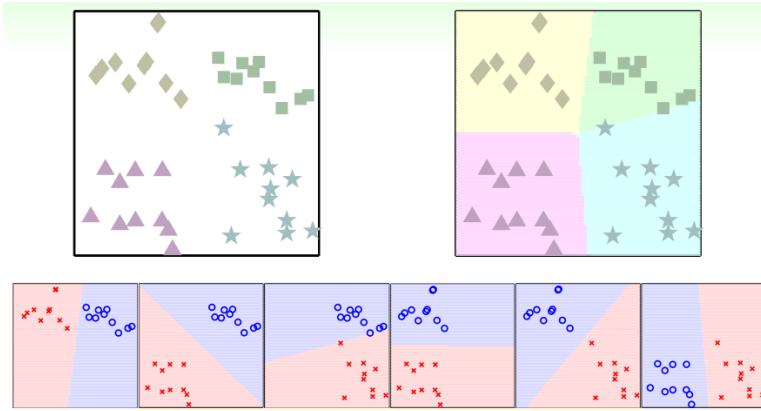


Figure 6.11: One vs. One at a Time

$$h(\vec{x}) = \text{voting}(g(\vec{\theta}_{kl}^T \vec{x})) \quad (6.276)$$

Algorithm 21 One-Versus-One (OVO) Decomposition

input: D with $\mathcal{Y} = \{y_1, y_2, \dots, y_K\}$

```

for  $k = 1 : (K - 1)$ 
  for  $l = (k + 1) : K$ 
     $D_{kl} = \emptyset$ 
    for  $i = 1 : m$ 
      if  $y^{(i)} = y_k \vee y^{(i)} = y_l$ 
         $D_{kl} = D_{kl} \cup \{(\vec{x}^{(i)}, \tilde{y}^{(i)}) = 2 \cdot 1\{y^{(i)} = y_k\} - 1\}$ 
     $\vec{\theta}_{kl} \leftarrow \text{logReg}(D_{kl})$ 
return  $h(\vec{x}) = \text{voting}(g(\vec{\theta}_{kl}^T \vec{x}))$ 

```

pros: efficient (smaller training problems), stable, can be coupled with any binary classification approaches

cons: use $O(K^2)$ $\vec{\theta}_{kl}$ — more space, slower prediction, more training

6.6 Nonlinear Transformation

6.6.1 Quadratic Hypotheses

Linear Hypothesis

up to now: lin. hypo.

- visually: line-like boundary
- mathematically: lin. scores $s = \vec{\theta}^T \vec{x}$
- theoretically: $VC(\mathcal{H})$ under control
- practically: on some D , large J_{in} for every line

Lin. Hypo. in \mathcal{Z} space

$$\Phi : \mathcal{X} \mapsto \mathcal{Z}$$

$$\vec{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}, \vec{\Phi}(\vec{x}) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_1 x_2 \\ x_2^2 \end{bmatrix} = \begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \end{bmatrix} = \vec{z} \quad (6.277)$$

$$h(\vec{x}) = \tilde{h}(\vec{z}) = \text{sign}(\tilde{\vec{\theta}}^T \vec{\Phi}(\vec{x})) = \text{sign}(\tilde{\theta}_0 + \tilde{\theta}_1 x_1 \tilde{\theta}_2 x_2 + \tilde{\theta}_3 x_1^2 + \tilde{\theta}_4 x_1 x_2 + \tilde{\theta}_5 x_2^2) \quad (6.278)$$

lin. in \mathcal{Z} space \iff quad. curves in \mathcal{X} space, including lines and const. as degenerate cases

6.6.2 Nonlinear Transform

Algorithm 22 The Nonlinear Transform Steps

transfer orig. data $D = \{(\vec{x}^{(i)}, y^{(i)})\}_{i=1}^m$ to $\tilde{D} = \{(\vec{z}^{(i)} = \vec{\Phi}(\vec{x}^{(i)}), y^{(i)})\}_{i=1}^m$

$\tilde{\vec{\theta}} \leftarrow A(\tilde{D})$ using your favorite lin. classification alg.

return $h(\vec{x}) = \text{sign}(\tilde{\vec{\theta}}^T \vec{x})$

two choices:

- feature transform Φ
- lin. model A

6.6.3 Price of Nonlinear Transform

\tilde{n} -th order poly. transform:

$$\vec{z}^{(i)} = \vec{\Phi}(\vec{x}^{(i)}) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_n \\ x_1^2 \\ x_1 x_2 \\ \vdots \\ x_n^2 \\ x_{\tilde{n}} \\ x_1^{\tilde{n}-1} x_2 \\ \vdots \\ x_n^{\tilde{n}} \end{bmatrix} \in \mathbb{R}^{\tilde{n}+1} \quad (6.279)$$

Computation/ Storage Price

$$\tilde{n} + 1 = \binom{n + \tilde{n}}{\tilde{n}} = \binom{n + \tilde{n}}{n} = O(\tilde{n}^n) \quad (6.280)$$

\tilde{n} large \rightarrow difficult to compute/ store

Model Complexity Price

free vars.: $\tilde{n} + 1 \approx VC(\mathcal{H})$

$VC(\mathcal{H}) \leq \tilde{n} + 1$, why?

- any $\tilde{n} + 2$ inputs not shattered in \mathcal{Z}
 - any $\tilde{n} + 2$ inputs not shattered in \mathcal{X}
- \tilde{n} large \rightarrow large $VC(\mathcal{H})$

Generalization Issue

Two centre issue

- 1. $J_{in} \approx J_{out}$?
- 2. $J_{in} \approx 0$

Generalization Issue, see Tab. 6.3

Table 6.3: Generalization Issue

\tilde{n}	1	1
higher	N	Y
smaller	Y	N

Danger of Visual Choices

1. can not visualize when $\mathcal{X} = \mathbb{R}^{10}$
2. careful about your brain's model complexity
for VC-safety, Φ shall be decided w/o peeking data

6.6.4 Structured Hypothesis Sets

safe routine: lin. model first, see Fig. 6.12

- if $J_{in}(h_1)$ good enough, live happy thereafter
- ow., move right of the curve with nothing lost except wasted computation

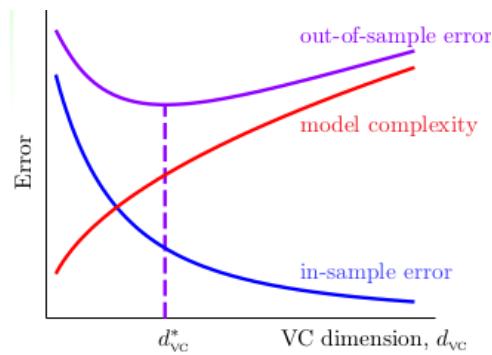


Figure 6.12: Structured Hypothesis Sets

Chapter 7

How Can Machine Learn Better

7.1 Hazard of Overfitting

7.1.1 What is Overfitting?

bad generalization: low J_{in} , high J_{out}

underfitting: model fail to accurately capture the structure in the data

overfitting:

- we are fitting patterns in the data that happened to be present in our small, finite training set, but that do not reflect the wider pattern of the relationship betw. \vec{x} and y
- lower J_{in} , higher J_{out}

cause of overfitting: a driving analogy, see Tab. 7.1

Table 7.1: cause of overfitting

overfit	commit a car accident
use excessive $VC(\mathcal{H})$	drive too fast
noise	bumpy road
limited data size m	limited obs. about road condition

7.1.2 The Role of Noise and Data Size

Case Study

See Fig. 7.1,g, overfitting from h_2 to h_{10} ? both yes!

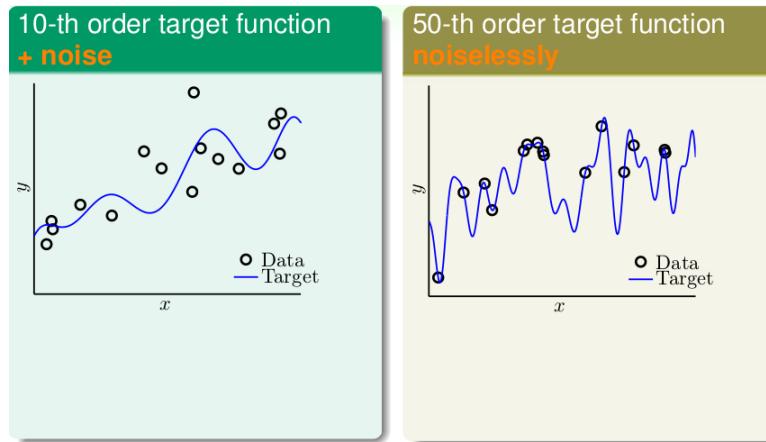


Figure 7.1: Case Study 1

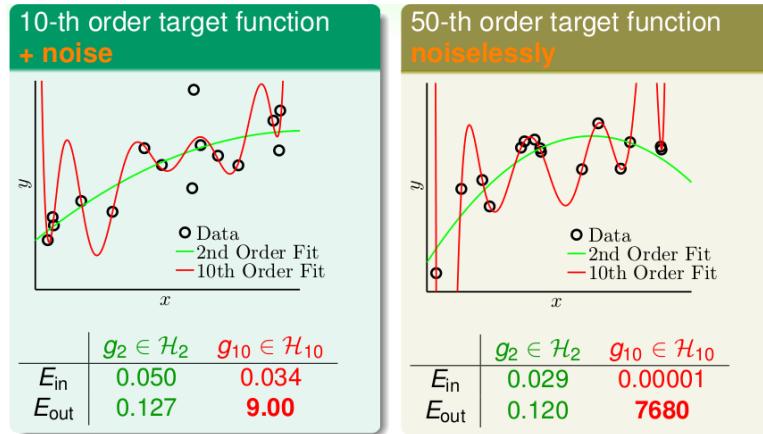


Figure 7.2: Case Study 2

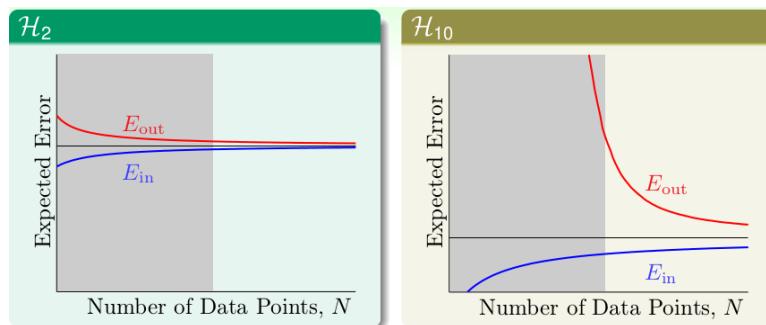


Figure 7.3: Learning Curves Revisited

Noisy Case: Learning Curves Revisited

See Fig. 7.3, \mathcal{H}_{10} : lower J_{out} when $m \rightarrow \infty$, but much larger generalization err. for small m

gray area: h_{10} overfits, h_2 always wins in J_{out} if m is small

No Noise Case

h_2 still wins

is there really no noise? target complexity acts like noise

7.1.3 Deterministic Noise

A Detailed Experiment

$$y = f(x) + \varepsilon \quad (7.1)$$

where $f(x) = Q_f$ -th ploy. of x , $\varepsilon \sim N(0, \sigma^2)$

data size m

$h_2 \in \mathcal{H}_2$, $h_{10} \in \mathcal{H}_{10}$

$J_{in}(h_{10}) \leq J_{in}(h_2)$ for sure

overfit measure $J_{in}(h_{10}) - J_{in}(h_2)$ as a fcn. of (m, σ^2) and (m, Q_f) , at $Q_f = 10$, see Fig. 7.4

- impact of (m, σ^2) : stochastic noise
- impact of (m, Q_f) : deterministic noise

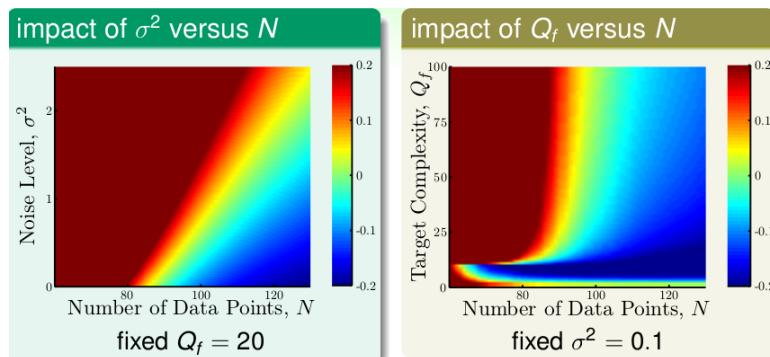


Figure 7.4: overfit measure $J_{in}(h_{10}) - J_{in}(h_2)$ as a fcn. of (m, σ^2) and (m, Q_f)

four reasons of serious overfitting:

- $m \downarrow$
- stochastic noise \uparrow
- deterministic noise \uparrow
- excessive power \uparrow

Deterministic Noise

if $f \notin \mathcal{H}$ sth. of f cannot be captured by \mathcal{H}

deterministic noise: difference betw. best $h^* \in \mathcal{H}$ and f
 acts like stochastic noise: cf., pseudo-random generator
 difference to stochastic noise

- depends on \mathcal{H}
- fixed for a given \vec{x}

when teaching a kid, perhaps better not to use examples from a complicated target function, see lower-left corner of the right of Fig. 7.4

7.1.4 Dealing with Overfitting

Driving Analogy Revisited

see Tab. 7.2

Table 7.2: Driving Analogy Revisited

learning	driving
start from simple model	drive slowly
data cleaning/ pruning	use more accurate road info.
data hinting	exploit more road info.
regularization	put the brakes
validation	monitor the dashboard

Validation

reduce # features

- manually select which features to keep
- model selection alg.

Regularization

keep all the features, but reduce magnitude / values of paras. θ_j
 works by trading increased bias for reduced variance
 works well when we have a lot of features, each of which contributes a bit to predicting y

Data Cleaning/ Pruning

if detect the outlier 5 (see Fig. 7.5) at the top by

- too close to other 1, or too far from other 5
- wrong by curr. classifier

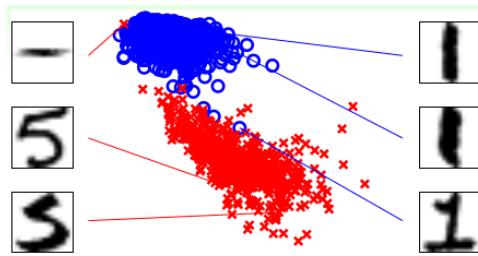


Figure 7.5: Data Cleaning/ Pruning

possible action 1: correct the label (data cleaning)

possible action 2: remove the example (data pruning)

possibly helps, but effect varies

Data Hinting

the best way to make a ML model generalize better is to train it on more data

how much work would it be to get 10x as much data as we currently have?

- artificial data synthesis
- collect/ label it yourself
- crowd source

slightly shifted / rotated digits carry the same meaning

create fake data: add virtual examples by shifting / rotating the given digits (data hinting), see Fig. 7.6

- often greatly improve generalization
- distortion introduced should be repr. of the type of noise / distortions in the test set, eg., audio: background noise, bad cellphone connection
- usually does not help to add purely random / meaningless noise to your data
- possibly help, but watch out: virtual examples not $\stackrel{iid}{\sim} p(\vec{x}, y)$

7	4	7	3	6	3	1	0	1
8	1	1	7	7	4	8	0	1
2	7	4	8	7	3	7	4	1
0	7	4	1	3	7	7	4	5
9	7	4	1	3	7	7	4	0
8	2	0	8	6	6	2	0	8

Figure 7.6: Data Hinting

7.2 Regularization

7.2.1 Regularized Hypothesis Set

Regularization: The Magic

idea: step back from H_{10} to \mathcal{H}_2 using constraints
name history: fcn. approx. for ill-posed problems

Regression with Constraint

regression with $\mathcal{H}_{10} = \{\vec{\theta} \in \mathbb{R}^{10+1}\}$

$$\min_{\vec{\theta} \in \mathbb{R}^{10+1}} J_{in}(\vec{\theta}) \quad (7.2)$$

regression with $\mathcal{H}_2 = \{\vec{\theta} \in \mathbb{R}^{10+1} \text{ while } \theta_3 = \theta_4 = \dots = \theta_{10} = 0\}$

$$\min_{\vec{\theta} \in \mathbb{R}^{10+1}} J_{in}(\vec{\theta}) \quad (7.3)$$

$$\text{st. } \theta_3 = \theta_4 = \dots = \theta_{10} = 0 \quad (7.4)$$

step back = constrained opt. of J_{in}

Regression with Looser Constraint

regression with $\mathcal{H}'_2 = \{\vec{\theta} \in \mathbb{R}^{10+1} \text{ while } \geq 8 \text{ of } w_j = 0\}$

$$\min_{\vec{\theta} \in \mathbb{R}^{10+1}} J_{in}(\vec{\theta}) \quad (7.5)$$

$$\text{st. } \sum_{j=1}^n 1\{w_j \neq 0\} \leq 3 \quad (7.6)$$

NP-hard to sol.

Regression with Softer Constraint

regression with $\mathcal{H}(C) = \{\vec{\theta} \in \mathbb{R}^{10+1} \text{ while } \|\vec{w}^T\|^2 \leq C\}$

$$\min_{\vec{\theta} \in \mathbb{R}^{10+1}} J_{in}(\vec{\theta}) \quad (7.7)$$

$$\text{st. } \sum_{j=1}^n w_j^2 \leq C \quad (7.8)$$

$\mathcal{H}(C)$: overlaps but not exactly the same as \mathcal{H}'_2
soft and smooth structure over $C \geq 0$

$$\mathcal{H}(0) \subset \dots \subset \mathcal{H}(1126) \subset \dots \subset \mathcal{H}(\infty) = \mathcal{H}_{10} \quad (7.9)$$

regularized hypo.: opt. sol. from regularized hypo. set $\mathcal{H}(C)$

7.2.2 Weight Decay Regularization

Matrix Form of Regularized Problem

$$\min_{\vec{\theta} \in \mathbb{R}^{10+1}} J_{in}(\vec{\theta}) = \frac{1}{m} \sum_{i=1}^m (\vec{\theta}^T \vec{z}^{(i)} - y^{(i)})^2 = \frac{1}{m} (\vec{Z} \vec{\theta} - \vec{y})^T (\vec{Z} \vec{\theta} - \vec{y}) \quad (7.10)$$

$$\text{st. } \sum_{j=0}^n \theta_j^2 = \vec{\theta}^T \vec{\theta} \leq C \quad (7.11)$$

feasible vt within a radius- \sqrt{C} hypersphere

the Lagrange Multiplier

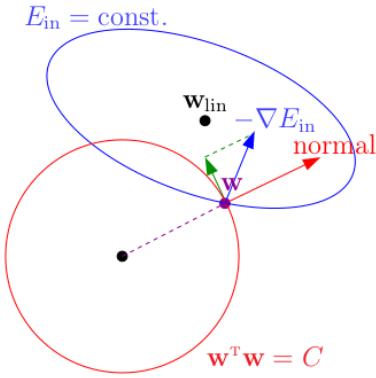


Figure 7.7: the Lagrange Multiplier

decreasing direction: $-\nabla J_{in}(\vec{\theta})$
 normal vec. of $\vec{\theta}^T \vec{\theta} = C$ is $\vec{\theta}$
 if $-\nabla J_{in}(\vec{\theta})$ and $\vec{\theta}$ not parallel: can decrease $J_{in}(\vec{\theta})$ w/o violating the constraint, see Fig. 7.7
 at opt. sol. $\vec{\theta}^*$: $-\nabla J_{in}(\vec{\theta}^*) \propto \vec{\theta}^*$
 want: find Lagrange multiplier $\lambda > 0$ and $\vec{\theta}^*$ st. $\nabla J_{in}(\vec{\theta}^*) + \frac{2\lambda}{m} \vec{\theta}^* = 0$

Augmented Err.

once find λ , then solving

$$\nabla J_{in}(\vec{\theta}^*) + \frac{2\lambda}{m} \vec{\theta}^* = \frac{2}{m} (\vec{Z}^T \vec{Z} \vec{\theta}^* - \vec{Z}^T \vec{y}) + \frac{2\lambda}{m} \vec{\theta}^* = 0 \quad (7.12)$$

is equivalent to

$$\min_{\vec{\theta}} J_{in}(\vec{\theta}) + \frac{\lambda}{m} \vec{\theta}^T \vec{\theta} \quad (7.13)$$

usually we leave offset term b unregularized

ridge regression

- b requires less data to fit accurately than \vec{w}
- \vec{w} specifies how two variables interact, fitting the weight well requires observing both variables in a variety of conditions
- offset b controls only a single variable, this means that we do not induce too much variance by leaving the offsets unregularized
- regularizing the offsets can introduce a significant amount of underfitting

$$\min_{\vec{\theta}} J_{in}(\vec{\theta}) + \frac{\lambda}{m} \vec{w}^T \vec{w} \stackrel{\text{def.}}{=} \min_{\vec{\theta}} J_{aug}(\vec{\theta}) \quad (7.14)$$

- call $\frac{\lambda}{m} \vec{\theta}^T \vec{\theta}$ the weighted-decay regularization
- the relationship betw. C and λ depends on the form of J
- larger $\lambda \iff$ shorter $\vec{\theta} \iff$ smaller C , see Fig. 7.8
- go with any transform + lin. model

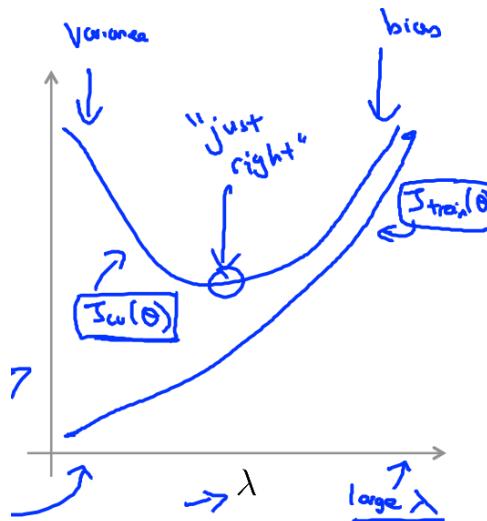


Figure 7.8: Overfit/ Underfit as a fcn. of the regularization para. λ
the opt. sol.

$$\vec{w}^* = \arg \min_{\vec{w}} J_{aug}(\vec{w}) = (Z^T Z + \lambda I)^{-1} Z^T \vec{y} \quad (7.15)$$

- $Z^T Z + \lambda I$ is inv.
- $Z^T Z = m\Sigma$ psd., λI pd.
- $Z^T Z + \lambda I$ pd.
- this makes Z to have higher variance, which makes it shrink the weights on features whose covariance with the output target is low compared to this added variance
- called **ridge regression** in Stat.

Some Detail: Legendre Polynomials

$$\min_{\vec{\theta}} \frac{1}{m} \sum_{i=1}^m (\vec{\theta}^T \vec{\Phi}(\vec{x}^{(i)}) - y^{(i)})^2 + \frac{\lambda}{m} \vec{\theta}^T \vec{\theta} \quad (7.16)$$

naive poly. transform

- $\vec{\Phi}(\vec{x}) = [1 \ x \ x^2 \ \dots \ x^{\tilde{n}}]^T$
- when $x \in [-1, +1]$, x^j really small, needing large θ_j
- normalized poly. transform:
- $\vec{\Phi}(\vec{x}) = [1 \ L_1(x) \ L_2(x) \ \dots \ L_{\tilde{n}}(x)]^T$
- orthonormal basis fcns., see Fig. 7.9

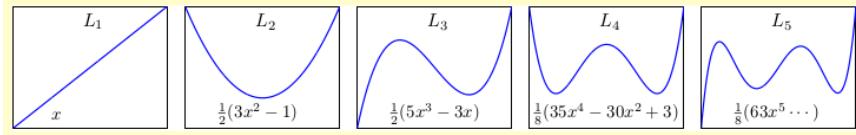


Figure 7.9: Legendre Polynomials

7.2.3 Gradient Descent with Regularization

recall GD

$$\vec{\theta} \leftarrow \vec{\theta} - \alpha \nabla J(\vec{\theta}) \quad (7.17)$$

now

$$\nabla J_{aug}(\vec{w}) = \nabla J_{in}(\vec{w}) + \nabla \frac{\lambda}{m} \vec{w}^T \vec{w} \quad (7.18)$$

$$= \nabla J_{in}(\vec{w}) + \frac{2\lambda}{m} \vec{w} \quad (7.19)$$

GD with regularization

$$\vec{w} \leftarrow \vec{w} - \alpha \nabla J_{aug}(\vec{w}) \quad (7.20)$$

$$= \vec{w} - \alpha \nabla J_{in}(\vec{w}) - \frac{2\alpha\lambda}{m} \vec{w} \quad (7.21)$$

$$= (1 - \frac{2\alpha\lambda}{m}) \vec{w} - \alpha \nabla J_{in}(\vec{w}) \quad (7.22)$$

$1 - \frac{2\alpha\lambda}{m} < 1$: shrink the weight vector by a const. factor on each step, towards 0, before performing the usual GD

7.2.4 Effect of Regularization

assume $b = 0$ then $\vec{\theta}$ is \vec{w}

suppose the \vec{w}^* is the opt. \vec{w} w/o regularization

Effective Number of Parameters

approx. the const function as

$$J(\vec{w}) \approx J(\vec{w}^*) + \nabla J(\vec{w}^*)(\vec{w} - \vec{w}^*) + \frac{1}{2}(\vec{w} - \vec{w}^*)^T H(\vec{w} - \vec{w}^*) \quad (7.23)$$

$$= J(\vec{w}^*) + \frac{1}{2}(\vec{w} - \vec{w}^*)^T H(\vec{w} - \vec{w}^*) // \nabla J(\vec{w}^*) = \vec{0} \quad (7.24)$$

the regularized opt. problem

$$\nabla J_{aug}(\vec{w}) = \nabla J(\vec{w}) + \frac{2\alpha\lambda}{m}\vec{w} \quad (7.25)$$

$$\approx \nabla J(\vec{w}^*) + \nabla \frac{1}{2}(\vec{w} - \vec{w}^*)^T H(\vec{w} - \vec{w}^*) + \frac{2\alpha\lambda}{m}\vec{w} \quad (7.26)$$

$$= H(\vec{w} - \vec{w}^*) + \frac{2\alpha\lambda}{m}\vec{w} \quad (7.27)$$

$$= (H - \frac{2\alpha\lambda}{m}I)\vec{w} - H\vec{w}^* \quad (7.28)$$

$$\stackrel{\text{set}}{=} \vec{0} \quad (7.29)$$

$$\vec{w} = (H - \frac{2\alpha\lambda}{m})^{-1}H\vec{w}^* \quad (7.30)$$

// this is the opt. \vec{w} when regularized

// as $\lambda \rightarrow 0$, $\vec{w} \rightarrow \vec{w}^*$

$$= (U\Lambda U^T - \frac{2\alpha\lambda}{m})^{-1}U\Lambda U^T\vec{w}^* \quad (7.33)$$

// H is psd., then $H = U\Lambda U^T$

$$= (U(\Lambda + \frac{2\alpha\lambda}{m}I)U^T)^{-1}U\Lambda U^T\vec{w}^* \quad (7.35)$$

$$= U(\Lambda + \frac{2\alpha\lambda}{m}I)^{-1}\Lambda U^T\vec{w}^* \quad (7.36)$$

$$U^T\vec{w} = (\Lambda + \frac{2\alpha\lambda}{m}I)^{-1}\Lambda U^T\vec{w}^* \quad (7.37)$$

- $U^T\vec{w}$: rotating the sol. \vec{w} into the basis defined by the eig.-vecs. U of H
- the effect of L_2 is rescale the coefficients of eig.-vecs.
- the j -th component is rescaled by a factor of $\frac{\lambda_j}{\lambda_j + \frac{2\alpha\lambda}{m}}$
- along the direction which λ_j is large, movement in this direction will significantly increase J , $\frac{\lambda_j}{\lambda_j + \frac{2\alpha\lambda}{m}} \rightarrow 1$, w_j , w_j is reduced a little
- along the direction which λ_j is small, movement in this direction will not significantly increase J , $\frac{\lambda_j}{\lambda_j + \frac{2\alpha\lambda}{m}} \rightarrow 0$, w_j is reduced a lot, see Fig. 7.10

Define 7.1 (Effective Number of Parameters). *It measures the effect of suppressing contributions to \vec{w} along eig. direction of H*

$$\gamma = \sum_j \frac{\lambda_j}{\lambda_j + \frac{2\alpha\lambda}{m}} \quad (7.38)$$

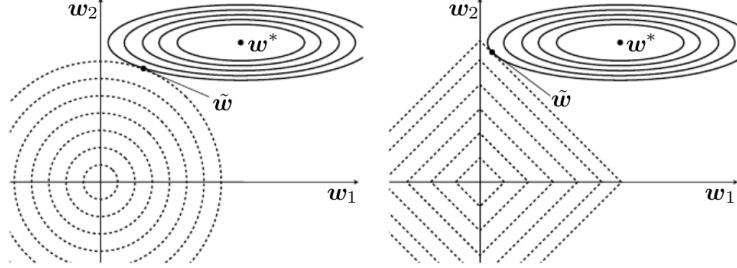


Figure 7.10: Left: first eig.-vec. \vec{u}_1 direction is w_1 's direction, λ_1 is small, w_1 is reduced alot. second eig.-vec. \vec{u}_2 direction is w_2 's direction, λ_2 is small, w_2 is reduced alot.

as $\lambda \uparrow, \gamma \downarrow$

7.2.5 Regularization and VC Theory

Regularization and VC Theory

$$\min_{\vec{\theta}} J_{aug}(\vec{\theta}) \stackrel{\text{equiv. to some } \lambda}{\iff} \min_{\vec{\theta}} J_{in}(\vec{\theta}) \text{ st. } \vec{\theta}^T \vec{\theta} \leq C \quad (7.39)$$

$$\stackrel{\text{VC}}{\Rightarrow} J_{out}(\vec{\theta}) \leq J_{in}(\vec{\theta}) + \sqrt{\cdot} \quad (7.40)$$

$\min J_{aug}$: indirectly getting VC guarantee w/o confining to $VC(\mathcal{H})$

Another View of Augmented Err.

aug. err.

$$J_{aug}(\vec{\theta}) = J_{in}(\vec{\theta}) + \frac{\lambda}{m} \vec{\theta}^T \vec{\theta} \quad (7.41)$$

VC Bound

$$J_{out}(\vec{\theta}) \leq J_{in}(\vec{\theta}) + \sqrt{\cdot} \quad (7.42)$$

regularizer $\vec{\theta}^T \vec{\theta}$: complexity of a single hypo.

generalization price $\sqrt{\cdot}$: complexity of a hypo. set

if $\frac{\lambda}{m} \vec{\theta}^T \vec{\theta}$ repr. $\sqrt{\cdot}$ well, J_{aug} is a better proxy of J_{out} than J_{in}

$\min J_{aug}$

- heuristically operating with the better proxy
- technically enjoying flexibility of whole \mathcal{H}

L₂ regularizer, weight decay

Effective VC Dimension

model complexity: $VC(\mathcal{H}) = \tilde{n} + 1$

$\vec{\theta}$ actually needed: $\mathcal{H}(C)$

$VC(\mathcal{H}(C))$: effective VC dim.

explanation of regularization

- $VC(\mathcal{H})$ large
- $VC(\mathcal{H}(C))$ small if regularized

7.2.6 General Regularizers

General Regularizers

target-dep.

- some properties of target, if known
- sym. regularizer: $\sum 1\{j \text{ is odd}\} \theta_j^2$
plausible
- direction towards smoother or simpler
- stochastic/ deterministic noise both non-smooth
- sparsity (L_1) regularizer: $\sum |\theta_j|$
friendly
- easy to opt.
- weight-decay (L_2) regularizer: $\sum \theta_j^2$
cf. err.-measure: user-dep., plausible, or friendly

L₂ Regularizer

Define 7.2 (L_2 regularizer, weight decay).

$$\Omega(\vec{w}) = \frac{1}{2} \sum_{j=1}^n \vec{w}_j^2 = \frac{1}{2} \|\vec{w}\|_2^2 \quad (7.43)$$

- cvx., differentiable everywhere
- easy to opt.
- L_2 regularization has the intuitive interpretation of heavily penalizing peaky weight vectors and preferring diffuse weight vectors
- it encourages the network to use all of its inputs a little rather than some of its inputs a lot
- during gradient descent update, using the L_2 regularization ultimately means that every weight is decayed linearly towards zero

L_2 regularizer can be interpreted as a Bayesian prior with

$$\vec{w} \sim N(\vec{0}, \lambda I) \quad (7.44)$$

$$p(\vec{w}) = \frac{1}{\sqrt{2\pi}^n} \exp\left(-\frac{\lambda}{2} \vec{w}^T \vec{w}\right) \quad (7.45)$$

L_1 Regularizer**Define 7.3** (L_1 regularizer).

L_1 regularizer
sparsity
LASSO

$$\Omega(\vec{w}) = \sum_{j=0}^n |w_j| = \|\vec{w}\|_1 \quad (7.46)$$

$$\begin{aligned} \nabla J_{aug}(\vec{w}) &= \nabla J(\vec{w}) + \frac{\lambda}{m} \text{sign}(\vec{w}) \\ &= \nabla(J(\vec{w}^*)) + \nabla J(\vec{w}^*)(\vec{w} - \vec{w}^*) + \frac{1}{2}(\vec{w} - \vec{w}^*)^T H(\vec{w} - \vec{w}^*) + \frac{\lambda}{m} \text{sign}(\vec{w}) \end{aligned} \quad (7.47)$$

$$= H(\vec{w} - \vec{w}^*) + \frac{\lambda}{m} \text{sign}(\vec{w}) \quad (7.49)$$

$$(\nabla J_{aug}(\vec{w}))_j = \gamma_j(w_j - w_j^*) + \frac{\lambda}{m} \text{sign}(w_j) // \text{assume } H = \text{diag}(\gamma_1, \gamma_2, \dots, \gamma_n) \quad (7.50)$$

$$= \gamma_j w_j + \frac{\lambda}{m} \text{sign}(w_j) - \gamma_j w_j^* \quad (7.51)$$

$$\stackrel{\text{set}}{=} 0 \quad (7.52)$$

$$w_j = \text{sign}(w_j^*) \max(|w_j^*| - \frac{\lambda}{m\gamma_j}, 0) \quad (7.53)$$

when $w_j^* > 0, \forall j$

- if $w_j^* \leq \frac{\lambda}{m\gamma_j}$, $w_j = 0$
- if $w_j^* > \frac{\lambda}{m\gamma_j}$, $w_j = w_j^* - \frac{\lambda}{m\gamma_j}$, it shift w_j^* , see Fig. 7.10
- cvx., not differentiable everywhere
- the solution is more likely to end up close to or at an axis, where some of the parameters are set to 0, it is called **sparsity**
- L_1 regularization end up using only a sparse subset of their most important inputs and become nearly invariant to the “noisy” inputs
- the sparsity can be used for feature selection, **LASSO** integrates an L_1 penalty with a linear model and a least square cost function
- in practice, L_2 regularization can be expected to give superior performance over L_1

 L_1 regularizer can be interpreted as a Bayesian prior with

$$w_j \sim Lap(\frac{m}{\lambda}, 0), \forall j \quad (7.54)$$

$$p(\vec{w}) = \frac{\lambda}{2m} \exp(-\frac{\lambda}{m} \sum_{j=1}^n |w_j|) \quad (7.55)$$

the Optimal λ

more noise \iff more regularization needed
 cf. more bumpy road \iff putting brakes more

dead units

7.2.7 Regularization as Constrained Optimization

use SGD on $J(\vec{\theta})$ and project $\vec{\theta}$ back to the nearest point satisfies $\vec{w}^T \vec{w} \leq C$

- useful if we know what val. of C is appropriate and do not want to find the corresponding λ
- sometimes penalties can cause non-convex opt.

Define 7.4 (dead units). *Units in neural network that do not contribute much to the behavior of NNet because the weights going into or out of them are all very small.*

when training with a penalty on the norm of the weights, these config. can be local opt.

- it can impose some stability on the opt. procedure
when α is large \rightarrow weight update is large $\rightarrow \vec{\theta}$ may consistly increase until numerical overflow
explicit constraints with reprojection allow us to terminate when $\vec{\theta}$ have reached a certain magnitude
- using constraints combined with a high learning rate to allow rapid exploration of para. space while maintain some stability

7.2.8 Regularization and Under-Constrained Problems

in regression problem, regularization insure $X^T X$ is invertible

most form of regularization are able to guarantee the conv. of iterative methods applied to underdetermined problems, eg., in LogReg, when data is lin. sep., w/o regularization, SGD will continue increase \vec{w} and in theory, it never halt, which cause numerical overflow,

7.3 Validation

7.3.1 Model Selection Problem

Model Selection by Best J_{in}

Φ_{1126} always more preferred over Φ_1

$\lambda = 0$ always more preferred over $\lambda = 0.1$ – overfitting?

if A_1 min. J_{in} over \mathcal{H}_1 and A_2 min. J_{in} over \mathcal{H}_2

- h^* achieves min. J_{in} over $\mathcal{H}_1 \cup \mathcal{H}_2$
- learning + model selection pays $VC(\mathcal{H}_1 \cup \mathcal{H}_2)$
- bad generalization?

Model Selection by Best J_{out}

$$h^* = \arg \min_{1 \leq k \leq K} J_{test}(A_k(D)) \quad (7.56)$$

generalization guarantee (fin.-bin Hoeffding)

$$J_{out}(h^*) \leq J_{test}(h^*) + O\left(\sqrt{\frac{\log K}{m_{test}}}\right) \quad (7.57)$$

7.3.2 Validation

$$D = D_{train} + D_{val} \quad (7.58)$$

make sure D_{val} clean: feed only D_{train} to A for model selection

$$h^- = A(D_{train}), h = A(D)$$

$$J_{out}(h) \leq J_{out}(h^-) \leq J_{val}(h^-) + O\left(\sqrt{\frac{\log K}{m_{val}}}\right)$$

the dilemma about m_{test}

$$J_{out}(h) \xrightarrow{\text{small } m_{val}} J_{out}(h^-) \xrightarrow{\text{large } m_{val}} J_{val}(h^-) \quad (7.59)$$

practical rule of thumb: $m_{val} = \frac{m}{5}$

finally, retrain on D

it seems waste $\frac{m}{5}$ data, since we testing models that were trained on only $\frac{4}{5}m$ each time

7.3.3 Leave-One-Out Cross Validation

Extreme Case: $m_{val} = 1$

$$J_{val}^{(i)}(h^-) = err(h^-(\vec{x}^{(i)}), y^{(i)}) = e^{(i)} \quad (7.60)$$

make $e^{(i)}$ closer to $J_{out}(h)$: avg. over possible $J_{val}^{(i)}$

$$J_{loocv}(h_k) = \frac{1}{m} \sum_{i=1}^m e_k^{(i)} = \frac{1}{m} \sum_{i=1}^m err(h_k^-(\vec{x}^{(i)}), y^{(i)}) \quad (7.61)$$

hope $J_{loocv}(h_k) \approx J_{out}(h_k)$

$$h^* = \arg \min_{1 \leq k \leq K} J_{loocv}(h_k) \quad (7.62)$$

Theoretical Guarantee of Leave-One-Out Estimate

does $J_{loocv}(h)$ say sth. about $J_{out}(h)$?

yes! for avg. J_{out} on size $m - 1$ data

$$\mathbb{E} D[J_{loocv}(h)] = \mathbb{E} D\left[\frac{1}{m} \sum_{i=1}^m e^{(i)}\right] \quad (7.63)$$

$$= \frac{1}{m} \sum_{i=1}^m \mathbb{E} D[e^{(i)}] \quad (7.64)$$

$$= \frac{1}{m} \sum_{i=1}^m \mathbb{E} D_{train} \mathbb{E}(\vec{x}^{(i)}, y^{(i)})[e^{(i)}] \quad (7.65)$$

$$= \frac{1}{m} \sum_{i=1}^m \mathbb{E} D_{train} \mathbb{E}(\vec{x}^{(i)}, y^{(i)})[err(h^-(\vec{x}^{(i)}), y^{(i)})] \quad (7.66)$$

$$= \frac{1}{m} \sum_{i=1}^m \mathbb{E} D_{train}[J_{out}(h^-)] \quad (7.67)$$

$$= \frac{1}{m} \sum_{i=1}^m \bar{J}_{out}(m-1) \quad (7.68)$$

$$= \bar{J}_{out}(m-1) \quad (7.69)$$

expected $J_{loocv}(h)$ says sth. about expected $J_{out}(h)$: often called almost unbiased estimate of $J_{out}(h)$

7.3.4 K-Fold Cross Validation

Disadvantages of LOOCV

m additional training per model, not always feasible in practice
except special case like analytic sol. for lin. regression
stability: due to variance of single-point estimates

K-Fold Cross Validation

random partition of D to K equal parts, take $K - 1$ for training and 1 for val. orderly

$$J_{cv}(h) = \frac{1}{K} \sum_{k=1}^K J_{val}^{(k)}(h^-) \quad (7.70)$$

practical rule of thumb: $K = 5, 10$

Final Words on Validation

K -fold generally preferred over single validation if computation allows nature of val.

- all training models: select among hypo.
- all val. schemes: select among finalists
- all test methods: just evaluate

val. still more optimistic than testing, report test rlt., not best val. rlt.

you can use validation to evaluate a single model or alg. to estimate how well it perf. for your application

7.3.5 Use Validation to Diagnosing Underfitting and Overfitting

see Fig. 7.11

- underfit: $J_{train}(\vec{\theta})$ will be high, $J_{train}(\vec{\theta}) \approx J_{cv}(\vec{\theta})$
- overfit: $J_{train}(\vec{\theta})$ will be low, $J_{train}(\vec{\theta}) \ll J_{cv}(\vec{\theta})$

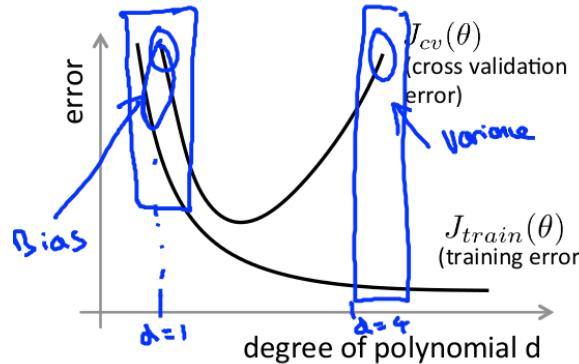


Figure 7.11: Use Validation to Diagnosing Underfitting and Overfitting

7.3.6 Feature Selection

this is one special and important case of model selection

it helps to determine which subset of features is relevant to the learning task

reduce n can help overcome overfitting

Naive Feature Selection

given n , there are 2^n possible feature subsets

thus, feature selection can be posed as a model selection problem over 2^n possible models

for large n , it's usually too expensive to explicitly enumerate over and compare all 2^n models

Wrapper Model Feature Selection

wraps around your learning alg., and repeatedly makes calls to the learning alg. to evaluate how well it does using diff. feature subsets

backward search starts off with $F = 1\{1, 2, \dots, n\}$ as the set of all features, and repeatedly deletes features one at a time until $F = \emptyset$

mutual information
KL divergence

Algorithm 23 Forward Search

```

init.  $F^{(0)} = \emptyset$ 
for  $t = 1 : n$ 
    // or  $t = 1 : th$  where  $th$  is some pre-set threshold corresponding
    to the max. # of features you want your alg. to use
    for  $j = 1 : n$ 
        if  $j \notin F^{(t-1)}$ 
             $F_j = F^{(t-1)} \cup \{j\}$ 
            train your alg. using  $F_j$  and use validation to get  $J_{cv}(F_j)$ 
             $F^{(t)} = \arg \min_{F_j} J_{cv}(F_j)$ 
    return best  $F^{(t)}$  that was evaluated during the entire search procedure

```

Filter Feature Selection

this is heuristic, but computationally much cheaper

idea: compute some simple score that measures how informative each feature x_j is about the class label y , then pick the \tilde{n} features with the largest scores

one possible choice of score would be the absolute val. of the correlation betw. x_j and y , as measured on training data. This would rlt. in our choosing the features that are the most strongly correlated with the class label.

more common (particularly for discrete-val. feature x_j) choice: **mutual information** , can also be expressed as **KL divergence**

$$MI(x_j, y) = KL(p(x_j, y) || p(x_j)p(y)) \quad (7.71)$$

$$= \sum_{x_j \in \{0,1\}} \sum_{y \in \{0,1\}} p(x_j, y) \log \frac{p(x_j, y)}{p(x_j)p(y)} \quad (7.72)$$

// assume x_j, y are binary-val. \quad (7.73)

- this gives a measure of how diff. the prob. distr. $p(x_j, y)$ and $p(x_j)p(y)$
- if x_j, y are indep. rv., then $p(x_j, y) = p(x_j)p(y)$, $KL(p(x_j, y) || p(x_j)p(y)) = 0$
- this means x_j clearly very non-informative about y , then the score is small
- $p(x_j, y), p(x_j), p(y)$ can be estimated on the D
how to dec. \tilde{n} ? use CV to select among the possible vals. of \tilde{n}

7.4 Noise Injection

for many classification and even some regression tasks, the task should still be possible to solve even if small random noise is added to the input

one way to improve the robustness of neural networks is simply to train them with random noise applied to their inputs

noise injection also works when the noise is applied to the hidden units, which can be seen as doing dataset augmentation at multiple levels of abstraction

7.4.1 Injecting Noise at the Input

consider a regression problem, add a random permutation to each \vec{x}

$$\tilde{\vec{x}} = \vec{x} + \vec{\varepsilon} \quad (7.74)$$

$$\vec{\varepsilon} \sim N(\vec{0}, \tau I) \quad (7.75)$$

$$h(\tilde{\vec{x}}) = h(\vec{x} + \vec{\varepsilon}) \quad (7.76)$$

$$\approx h(\vec{x}) + (\nabla_{\vec{x}} h(\vec{x}))^T \vec{\varepsilon} + \frac{1}{2} \vec{\varepsilon}^T H \vec{\varepsilon} \quad (7.77)$$

$$J_{out}(h(\tilde{\vec{x}})) = E[(h(\tilde{\vec{x}}) - y)^2] \quad (7.78)$$

$$= E[h(\tilde{\vec{x}})^2 - 2h(\tilde{\vec{x}})y + y^2] \quad (7.79)$$

$$\begin{aligned} &= E[(h(\vec{x}) + (\nabla h(\vec{x}))^T \vec{\varepsilon} + \frac{1}{2} \vec{\varepsilon}^T H \vec{\varepsilon})^2 \\ &\quad - 2(h(\vec{x}) + (\nabla h(\vec{x}))^T \vec{\varepsilon} + \frac{1}{2} \vec{\varepsilon}^T H \vec{\varepsilon})y + y^2] \end{aligned} \quad (7.80)$$

$$= E[(h(\vec{x}) - y)^2 + h(\vec{x})\vec{\varepsilon}^T H \vec{\varepsilon} + ((\nabla h(\vec{x}))^T \vec{\varepsilon})^2 - \vec{\varepsilon}^T H \vec{\varepsilon}] \quad (7.81)$$

// ignore high order term of $\vec{\varepsilon}$

$$= J_{out}(h(\vec{x})) + \tau E[(h(\vec{x}) - y)H] + \tau E[\|\nabla h(\vec{x})\|^2] \quad (7.82)$$

$$\frac{\partial h(\tilde{\vec{x}})}{\partial h(\vec{x})} \stackrel{\text{set}}{=} 0 \quad (7.83)$$

$$h(\vec{x}) = E[y|\vec{x}] \quad (7.84)$$

$$J_{out}(h(\tilde{\vec{x}})) = J_{out}(h(\vec{x})) + \tau E[\|\nabla h(\vec{x})\|^2] \quad (7.85)$$

- for small τ , $\tau E[\|\nabla h(\vec{x})\|^2]$ acts as a regularization term
- it penalize large gradients of the $h(\vec{x})$, it reduces the sensitivity of the output of the network wrt. small variations in its input \vec{x}
- we can interpret this as attempting to build in some local robustness into the model and thereby promote generalization

7.4.2 Injecting Noise at the Weights

the Bayesian treatment of learning would consider the model weights to be uncertain and representable via a probability distribution that reflects this uncertainty

adding the noise to the weights can be interpreted as a stochastic implementation of a Bayesian inference to reflect this uncertainty

consider a regression problem, add a random permutation to $\vec{\theta}$

$$\tilde{\vec{\theta}} = \vec{\theta} + \vec{\varepsilon} \quad (7.86)$$

$$\vec{\varepsilon} \sim N(\vec{0}, \tau I) \quad (7.87)$$

$$h_{\tilde{\vec{\theta}}}(\vec{x}) \approx h(\vec{x}) + (\nabla_{\vec{\theta}} h(\vec{x}))^T \vec{\varepsilon} + \frac{1}{2} \vec{\varepsilon}^T H \vec{\varepsilon} \quad (7.88)$$

$$J_{out}(h_{\tilde{\vec{\theta}}}(\vec{x})) = J_{out}(h(\vec{x})) + \tau E[(h(\vec{x}) - y)H] + \tau E[\|\nabla h(\vec{x})\|^2] \quad (7.89)$$

$$\frac{\partial h(\tilde{\vec{x}})}{\partial h(\vec{x})} \stackrel{\text{set}}{=} 0 \quad (7.90)$$

$$h(\vec{x}) = E[y|\vec{x}] \quad (7.91)$$

$$J_{out}(h_{\tilde{\vec{\theta}}}(\vec{x})) = J_{out}(h(\vec{x})) + \tau E[\|\nabla h(\vec{x})\|^2] \quad (7.92)$$

- for small τ , $\tau E[\|\nabla h(\vec{x})\|^2]$ acts as a regularization term
- it penalizes large gradients of the $h(\vec{x})$, it reduces the sensitivity of the output of the network wrt. small variations in $\vec{\theta}$
- we can interpret this as attempting to build in some local robustness into the model and thereby promote generalization

7.5 Decide What to Do Next

7.5.1 Do More Data Really works?

learning curves

if a learning alg. is suffering from high bias, getting more training data will not (by itself) help much, see Fig. 7.12

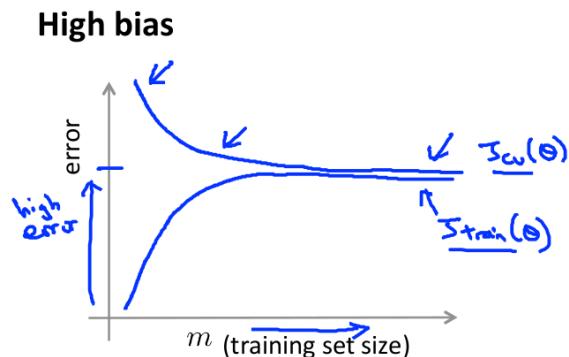


Figure 7.12: High Bias

if a learning alg. is suffering from high variance, getting more training data is likely to help, see Fig. 7.13

useful test: given the input \vec{x} , can a human expert confidently predict y large data rationale

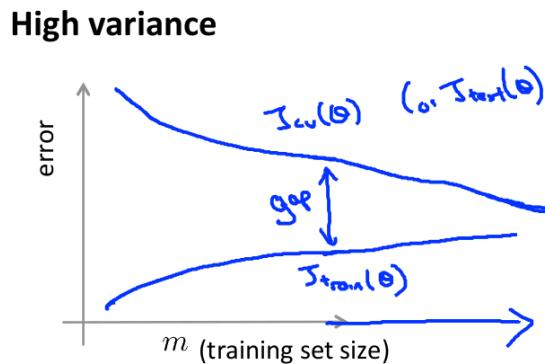


Figure 7.13: High Variance

- use a learning alg. with many paras.: low bias, $J_{\text{train}}(\vec{\theta})$ will be small
- use a very large training set: low variance, $J_{\text{train}} \approx J_{\text{test}}$

7.5.2 Optimization Algorithm Diagnostics

Bias vs. variance is one common diagnostic

for other problems, it's usually up to your own ingenuity to construct your own diagnostics to figure out what is wrong

Spam Classification Problem

eg.,

- Bayesian logistic regression gets 2% error on spam, and 2% error on non-spam.
- SVM using a lin. kernel gets 10% err. on spam, and 0.01% error on non-spam.
- But you want to use logistic regression, because of computational efficiency, etc.

what to do next?

Bayesian logistic regression

$$\min_{\vec{\theta}} J_{\text{in}}^{ce}(\vec{\theta}) = -\frac{1}{m} \sum_{i=1}^m \log p(y^{(i)} | \vec{x}^{(i)}, \vec{\theta}) + \frac{\lambda}{m} \|\vec{\theta}\|^2 \quad (7.93)$$

- correct value for λ ?
- Is the algorithm (gradient descent for logistic regression) converging? Its often very hard to tell if an algorithm has converged yet by looking at the objective curve.

Diagnostic

let $\vec{\theta}_{\text{SVM}}$ be the para. learned by an SVM

let $\vec{\theta}_{BLR}$ be the para. learned by Bayesian log. reg.
what you care about:

$$J_{in}^{01}(\vec{\theta}) = \frac{1}{m} \sum_{i=1}^m w^{(i)} 1\{h(\vec{x}) \neq y^{(i)}\} \quad (7.94)$$

weights $w^{(i)}$ higher for non-spam than for spam
 $\vec{\theta}_{SVM}$ outperforms $\vec{\theta}_{BLR}$, so

$$J_{in}^{01}(\vec{\theta}_{SVM}) < J_{in}^{01}(\vec{\theta}_{BLR}) \quad (7.95)$$

Two Cases

- if $J_{in}^{ce}(\vec{\theta}_{SVM}) < J_{in}^{ce}(\vec{\theta}_{BLR})$
 - but BLR was trying to min. $J_{in}^{ce}(\vec{\theta})$
 - this means that $\vec{\theta}_{BLR}$ fails to min. J^{ce} ,
 - the problem is with the convergence of the algorithm
 - try run gradient descent for more iterations or try Newtons method
- if $J_{in}^{ce}(\vec{\theta}_{SVM}) \geq J_{in}^{ce}(\vec{\theta}_{BLR})$
 - $\vec{\theta}_{BLR}$ succeeded at min. $J_{in}^{ce}(\vec{\theta})$
 - but the SVM, which does worse on $J_{in}^{ce}(\vec{\theta})$, actually does better on weighted accuracy $J_{in}^{01}(\vec{\theta})$
 - problem is with objective function of the min. problem
 - try use a different value for λ .

the Stanford Autonomous Helicopter Problem

- build a simulator of helicopter
- choose a cost function, say

$$J(\vec{\theta}) = \|\vec{x} - \vec{\mu}\|^2 \quad (7.96)$$

- \vec{x} is the helicopter position, $\vec{\mu}$ is the desired position
- run reinforcement learning (RL) algorithm to fly helicopter in simulation, so as to try to minimize cost function:

$$\vec{\theta}_{RL} = \arg \min_{\vec{\theta}} J(\vec{\theta}) \quad (7.97)$$

Suppose you do this, and the resulting controller parameters $\vec{\theta}_{RL}$ gives much worse performance than your human pilot. What to do next?

- Improve simulator?
- Modify cost function J?
- Modify RL algorithm?

Diagnostics

if $\vec{\theta}_{RL}$ flies well in simulation, but not in real life, then the problem is in the simulator.

let $\vec{\theta}_{human}$ be the human control policy. If $J(\vec{\theta}_{human}) < J(\vec{\theta}_{RL})$, then the problem is in the reinforcement learning algorithm. (Failing to minimize the cost function J .)

if $J(\vec{\theta}_{human}) \geq J(\vec{\theta}_{RL})$, then the problem is in the cost function. (min. it doesn't correspond to good autonomous flight.)

Final Note on Diagnostics

even if a learning algorithm is working well, you might also run diagnostics to make sure you understand what's going on. This is useful for:

- understanding your application problem: If you're working on one important ML application for months/years, it's very valuable for you personally to get an intuitive understanding of what works and what doesn't work in your problem.
- writing research papers: Diagnostics and error analysis help convey insight about the problem, and justify your research claims.
- i.e., Rather than saying Here's an algorithm that works, it's more interesting to say Here's an algorithm that works because of component X, and here's my justification.

7.5.3 Error Analysis

Many applications combine many different learning components into a pipeline.

Start with a simple alg. that you can implement quickly. Implement it and test it on your CV data.

plot learning curves to decide if more data, more features, etc. are likely to help

error analysis: manually examine the examples (in CV set) that your alg. make errors on. See if you spot any systematic trend in what type of examples it is making errors on

error analysis may not be helpful for deciding if this is likely to improve perf. Only sol. is to try it and see if it works

7.5.4 Ceiling Analysis

plug in ground-truth for each component, and see how accuracy changes. estimating the errors due to each component

what part of the pipeline should you spend the most time trying to improve? see Fig. 7.14

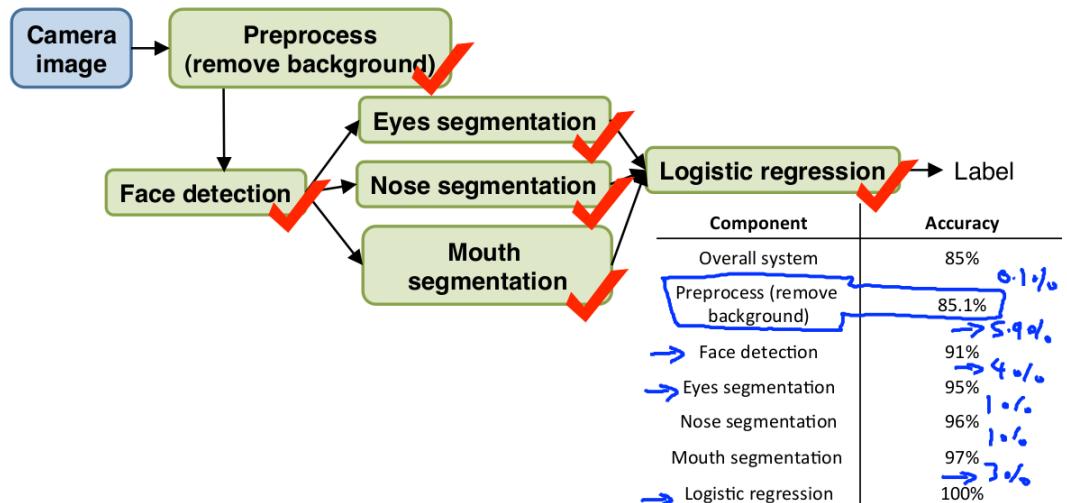


Figure 7.14: Ceiling Analysis

7.5.5 Ablative Analysis

ceiling analysis tries to explain the difference between current performance and perfect performance.

ablate analysis tries to explain the difference between some baseline (much poorer) performance and current performance.

eg., Suppose that youve build a good anti-spam classifier by adding lots of clever features to logistic regression, how much did each of these components really help?

simple logistic regression without any clever features get 94Just what accounts for your improvement from 94 to 99.9

ablate analysis: remove components from your system one at a time, to see how it breaks., see Fig. 7.15

Component	Accuracy
Overall system	99.9%
Spelling correction	99.0
Sender host features	98.9%
Email header features	98.9%
Email text parser features	95%
Javascript parser	94.5%
Features from images	94.0%

Figure 7.15: Ablative Analysis

7.5.6 Getting Started on a Problem

Two Approaches

approach 1: careful design

- spend a long term designing exactly the right features, collecting the right dataset, and designing the right algorithmic architecture.
- implement it and hope it works.
- benefit: Nicer, perhaps more scalable algorithms. May come up with new, elegant, learning algorithms; contribute to basic research in machine learning.

approach 2: build-and-fix

- implement something quick-and-dirty.
- run error analyses and diagnostics to see what's wrong with it, and fix its errors.
- benefit: Will often get your application problem working more quickly. Faster time to market.

Premature Statistical Optimization

very often, it's not clear what parts of a system are easy or difficult to build, and which parts you need to spend lots of time focusing on.

the only way to find out what needs work is to implement something quickly, and find out what parts break.

but this may be bad advice if your goal is to come up with new machine learning algorithms.

step 1 of designing a learning system: Plot the data.

7.6 Three Learning Principles

7.6.1 Occam's Razor

The simplest model that fits the data is also the most plausible.

Simple Model

simple hypo.

- small $\sqrt{.}$
- few paras.

simple model

- small $\sqrt{.}$
- contains small # of hypos.

simple model \rightarrow simple hypo.

Simple is Better

simple $\mathcal{H} \rightarrow$ smaller $\text{grow}_{\mathcal{H}}(m) \rightarrow$ less likely to fit data perfectly $\frac{\text{grow}_{\mathcal{H}}(m)}{2^m} \rightarrow$ more significant when fit happens

direction action: lin. first

always ask whether data over-modeled

7.6.2 Sampling Bias

If the data is sampled in a biased way, learning will produce a similarly biased outcome.

- tech. explanation: data from P_1 but test under P_2 : VC fails
- philosophical explanation: study Math hard but test English: no strong test guarantee
- minor VC assumption: data and testing both iid from P
- practical rule of thumb: match test scenario as much as possible. eg., if test: last user records after D
- training: emphasize later examples
- validation: use late user records

7.6.3 Data Snooping

If a data set has affected any step in the learning process, its ability to assess the outcome has been compromised.

truth very hard to avoid, unless being extremely honest

- extremely honest: lock your test data in safe
- less honest: reserve validation and use cautiously

be blind: avoid making modeling decision by data

be suspicious: interpret research results (including your own) by proper feeling of contamination

one secret to winning KDDCups: careful balance between data-driven modeling (snooping) and validation (no-snooping)

7.6.4 Power of Three

Three Related Fields

- DM
- AI
- Stat.

Three Theoretical Bounds

- Hoeffding
 - $\Pr(BAD) \leq 2 \exp(-2\varepsilon^2 m)$
 - one hypo.
 - useful for verifying/testing
- Multi-Bin Hoeffding

- $\Pr(BAD) \leq 2M \exp(-2\epsilon^2 m)$
- M hypos.
- useful for validation
- VC
 - $\Pr(BAD) \leq 4\text{grow}_{\mathcal{H}}(2m) \exp(\cdot)$
 - all \mathcal{H}
 - useful for training

Three Linear Models

- PLA/pocket: 0/1 err, minimize specially
- linear regression: sq. err, minimize analytically
- logistic regression: ce err, minimize iteratively

Three Key Tools

- Feature Transform
 - by using more complicated Φ
 - lower J_{in}
 - higher $VC(\mathcal{H})$
- Regularization
 - by augmenting regularizer
 - lower effective $VC(\mathcal{H})$
 - higher J_{in}
- Validation
 - by reserving examples as D_{val}
 - fewer choices
 - fewer examples

Three Learning Principles

- Occams Razer: simple is good
- Sampling Bias: class matches exam
- Data Snooping: honesty is best policy

Three Future Directions

- More Transform
- More Regularization
- Less Label

Chapter 8

Embedding Numerous Features: Kernel Models

8.1 Three Major Techniques Surrounding feature transform

embedding numerous features

- how to exploit and regularize numerous features
- SVM model

comb. predictive features

- how to construct and blend predictive features
- adaptive boosting (AdaBoost) model

distilling implicit features

- how to identify and learn implicit features
- deep learning model

8.2 Linear Support Vector Machine

8.2.1 Large-Margin Separating Hyperplane

Which Line is Better

see Fig. 8.1

PLA? depending on randomness

VC? whichever you like!

$$J_{out}(\vec{\theta}) \leq J_{in}(\vec{\theta}) + \Omega(\mathcal{H}) \quad (8.1)$$

- $J_{in}(vt) = 0$
- $VC(\mathcal{H}) = n + 1$

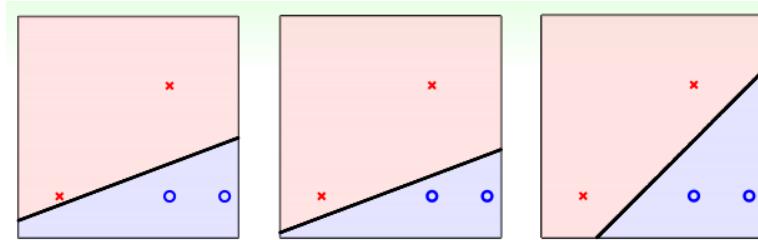


Figure 8.1: Which Line is Better

Why Rightmost Hyperplane

see Fig. 8.2

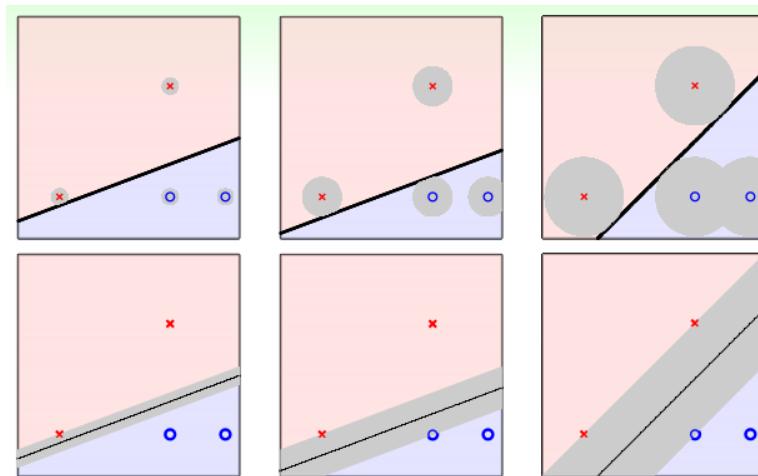


Figure 8.2: Rightmost Hyperplane is Better

if Gaussian-like noise on future $\vec{x} \approx \vec{x}^{(i)}$

- $\vec{x}^{(i)}$ further from hyperplane
 \iff tol. more noise
- \iff more robust to overfitting
 dist. from hyperplane to closest $\vec{x}^{(i)}$
- \iff amount of noise tol.
- \iff robustness of hyperplane
- \iff fatness of hyperplane
- \iff margin of hyperplane

Large-Margin Separating Hyperplane

$\vec{\theta}$ classifies every $(\vec{x}^{(i)}, y^{(i)})$ correctly: $y^{(i)}\vec{\theta}^T\vec{x}^{(i)} > 0, \forall i$

$$\max_{\vec{\theta}} \text{margin}(\vec{\theta}) \quad (8.2)$$

$$\text{st. } y^{(i)} \vec{\theta}^T \vec{x}^{(i)} > 0, \forall i \quad (8.3)$$

$$\text{where } \text{margin}(\vec{\theta}) = \min_i \text{dist}(\vec{x}^{(i)}, \vec{\theta}) \quad (8.4)$$

8.2.2 Standard Large-Margin Problem

Distance to Hyperplane

hyperplane

$$\vec{\theta}^T \vec{x} = \vec{w}^T \vec{x} + b = 0 \quad (8.5)$$

consider \vec{x}_1, \vec{x}_2 on hyperplane:

$$\vec{w}^T \vec{x}_1 = -b \quad (8.6)$$

$$\vec{w}^T \vec{x}_2 = -b \quad (8.7)$$

minus,

$$\vec{w}^T (\vec{x}_1 - \vec{x}_2) = (-b) - (-b) = 0 \quad (8.8)$$

$\vec{x}_1 - \vec{x}_2$ is parallel to the hyperplane, so $\vec{w} \perp$ hyperplane
 $\text{dist}(\vec{x}, \vec{\theta})$ is proj. $\vec{x} - \vec{x}_1$ to \perp hyperplane, see Fig. 8.3

$$\text{dist}(\vec{x}, \vec{w}, b) = \left| \frac{\vec{w}^T}{\|\vec{w}\|} (\vec{x} - \vec{x}_1) \right| = \frac{1}{\|\vec{w}\|} |\vec{w}^T \vec{x} + b| \quad (8.9)$$

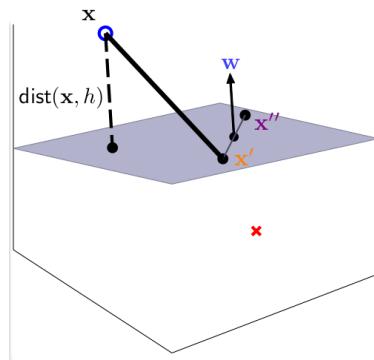


Figure 8.3: Distance to Hyperplane

Distance to Separating Hyperplane

separating hyperplane:

$$y^{(i)} \vec{\theta}^T \vec{x}^{(i)} = y^{(i)} (\vec{w}^T \vec{x}^{(i)} + b) > 0, \forall i \quad (8.10)$$

$y^{(i)} = \{+1, -1\}$, so dist. to sep. hyperplane

$$\text{dist}(\vec{x}, \vec{w}, b) = \frac{1}{\|\vec{w}\|} y^{(i)} (\vec{w}^T \vec{x}^{(i)} + b) \quad (8.11)$$

Margin of Special Separating Hyperplane

$$\max_{\vec{w}, b} \text{margin}(\vec{w}, b) \quad (8.12)$$

$$\text{st. } y^{(i)} (\vec{w}^T \vec{x}^{(i)} + b) > 0, \forall i \quad (8.13)$$

$$\text{where } \text{margin}(\vec{w}, b) = \min_i \frac{1}{\|\vec{w}\|} y^{(i)} (\vec{w}^T \vec{x}^{(i)} + b) \quad (8.14)$$

$\vec{w}^T \vec{x} + b = 0$ same as $3\vec{w}^T \vec{x} + 3b = 0$: scaling does not change $\text{margin}(\vec{w}, b)$
special scaling: only consider sep. (\vec{w}, b) st.

$$\min_i y^{(i)} (\vec{w}^T \vec{x}^{(i)} + b) = 1 \quad (8.15)$$

thus,

$$\text{margin}(\vec{\theta}) = \min_i \frac{1}{\|\vec{w}\|} y^{(i)} (\vec{w}^T \vec{x}^{(i)} + b) = \frac{1}{\|\vec{w}\|} \quad (8.16)$$

$$\max_{\vec{w}, b} \frac{1}{\|\vec{w}\|} \quad (8.17)$$

$$\text{st. } \min_i y^{(i)} (\vec{w}^T \vec{x}^{(i)} + b) = 1, \forall i \quad (8.18)$$

final ver.

Algorithm 24 SVM

$$\min_{\vec{w}, b} \frac{1}{2} \vec{w}^T \vec{w} \quad (8.19)$$

$$\text{st. } y^{(i)} (\vec{w}^T \vec{x}^{(i)} + b) \geq 1, \forall i \quad (8.20)$$

Proof. if opt. (\vec{w}, b) satisfy $\min_i y^{(i)} (\vec{w}^T \vec{x}^{(i)} + b) = 1.126$, can scale (\vec{w}, b) to
more opt. $(\frac{\vec{w}}{1.126}, \frac{b}{1.126})$, contradiction \square

8.2.3 SVM

SVM, see Fig. 8.4

- examples on boundary: locates fattest hyperplane
- call boudry examples: SV (candidate)
- other examples: not needed

SVM: learn fattest hyperplanes with help of SVs
solve?

- grad. desc.? not easy with constraints
- cvx. quad. obj. fcn. of (\vec{w}, b)
- lin. constraint of (\vec{w}, b)
- this is quad. programming (QP)

Algorithm 25 QP

$$\min_{\vec{u}} \frac{1}{2} \vec{u}^T Q \vec{u} + \vec{p}^T \vec{u} \quad (8.21)$$

$$\text{st. } \vec{a}_i^T \vec{u} \geq c_i, \forall i \in \{1, 2, \dots, m\} \quad (8.22)$$

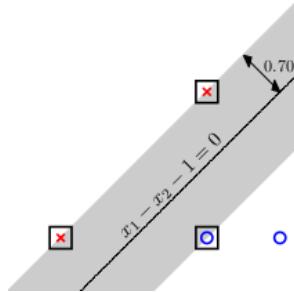


Figure 8.4: SVM

Algorithm 26 SVM with QP solver

$$Q = \begin{bmatrix} 0 & \vec{0} \\ \vec{0} & I_n \end{bmatrix} \in \mathbb{R}^{(1+n) \times (1+n)}$$

$$\vec{p} = \vec{0} \in \mathbb{R}^{1+n}$$

$$\vec{a}_i = y^{(i)} \begin{bmatrix} 1 \\ \vec{x}^{(i)} \end{bmatrix} \in \mathbb{R}^{1+n}, \forall i \in \{1, 2, \dots, m\}$$

$$c_i = 1, \forall i \in \{1, 2, \dots, m\}$$

$$\begin{bmatrix} b \\ \vec{w} \end{bmatrix} = \vec{u} = \text{QP}(Q, \vec{p}, A, \vec{c})$$

return $h(\vec{x}) = \text{sign}(\vec{w}^T \vec{x} + b)$

hard-margin: nothing violate fat boundry

lin. $\vec{x}^{(i)}$

8.2.4 Reasons behind Large-Margin Hyperplane

Why Large-Margin Hyperplane?

Table 8.1: Why Large-Margin Hyperplane?

	min.	constraint
regularization	J_{in}	$\vec{\theta}^T \vec{\theta} \leq C$
SVM	$\vec{w}^T \vec{w}$	$J_{in} = 0$ (and more)

see Tab. 8.1, SVM (large-margin hyperplane): weight-decay regularization within $J_{in} = 0$

Large Margin Restricts Dichotomies

SVM returns h with $\text{margin}(h) \geq \gamma$ (if exists), or 0 otherwise.

PLA: can shatter general $n + 1$ inputs

SVM: more strict than PLA, cannot shatter all $n + 1$ inputs
fewer dichotomies \rightarrow smaller $VC(\mathcal{H}) \rightarrow$ better generalization

Benefits of Large-Margin Hyperplanes

Table 8.2: Benefits of Large-Margin Hyperplanes

	large-margin hyperplanes	hyperplanes	hyperplanes + feature transform Φ
#	even fewer	not many	many
boundary	simple	simple	sophisticated

see Tab. 8.2

not many: good, for $VC(\mathcal{H})$ and generalization

sophisticated: good, for possible better J_{in}

a new possibility: non-linear SVM, see Tab. 8.3

Table 8.3: non-linear SVM

	large-margin hyperplanes + feature transform Φ
#	not many
boundary	sophisticated

8.3 Dual SVM

8.3.1 Motivation of Dual SVM

Non-linear SVM Revisited

Algorithm 27 SVM

$$\min_{\vec{w}, b} \frac{1}{2} \vec{w}^T \vec{w} \quad (8.23)$$

$$\text{st. } y^{(i)}(\vec{w}^T \vec{z}^{(i)} + b) \geq 1, \forall i \quad (8.24)$$

orig. SVM

- $\tilde{n} + 1$ vars.
- m constraints
want equiv.
- m vars.
- $m + 1$ constraints

Key Tool: Lagrange Multipliers

def.

$$L(\vec{w}, b, \vec{\alpha}) = \frac{1}{2} \vec{w}^T \vec{w} + \sum_{i=1}^m \alpha_i (1 - y^{(i)}(\vec{w}^T \vec{z}^{(i)} + b)) \quad (8.25)$$

$$\text{SVM} = \min_{\vec{w}, b} \max_{\alpha_i \geq 0} L(\vec{w}, b, \vec{\alpha}) \quad (8.26)$$

$$= \min_{\vec{w}, b} \max_{\alpha_i \geq 0} \frac{1}{2} \vec{w}^T \vec{w} + \sum_{i=1}^m \alpha_i (1 - y^{(i)}(\vec{w}^T \vec{z}^{(i)} + b)) \quad (8.27)$$

$$= \min_{\vec{w}, b} \max_{\alpha_i \geq 0} \begin{cases} \frac{1}{2} \vec{w}^T \vec{w} + \sum_{i=1}^m \alpha_i (\text{some } > 0) & \text{if violate} \\ \frac{1}{2} \vec{w}^T \vec{w} + \sum_{i=1}^m \alpha_i (\text{all } \leq 0) & \text{if feasible} \end{cases} \quad (8.28)$$

$$= \min_{\vec{w}, b} \begin{cases} \infty & \text{if violate} \\ \frac{1}{2} \vec{w}^T \vec{w} & \text{if feasible} \end{cases} \quad (8.29)$$

constraints now hidden in max

8.3.2 Lagrange Dual SVM

Lagrange Dual SVM

for any fixed $\vec{\alpha}'$ with $\alpha'_i \geq 0, \forall i$

$$\min_{\vec{w}, b} \max_{\alpha_i \geq 0} L(\vec{w}, b, \vec{\alpha}) \geq \min_{\vec{w}, b} L(\vec{w}, b, \vec{\alpha}') \quad (8.30)$$

because $\max \geq$ any
for best $\vec{\alpha}'$

$$\min_{\vec{w}, b} \max_{\alpha_i \geq 0} L(\vec{w}, b, \vec{\alpha}) \geq \max_{\alpha'_i \geq 0} \min_{\vec{w}, b} L(\vec{w}, b, \vec{\alpha}') \quad (8.31)$$

because best is one of any

Strong Duality of Quadratic Programming

$$\min_{\vec{w}, b} \max_{\alpha_i \geq 0} L(\vec{w}, b, \vec{\alpha}) = \max_{\alpha_i \geq 0} \min_{\vec{w}, b} L(\vec{w}, b, \vec{\alpha}) \quad (8.32)$$

true for QP if satisfy constraint qualification

- cvx. primal
 - feasible primal (true if Φ -sep.)
 - lin. constraints
- exists primal-dual opt. sol. $(\vec{w}, b, \vec{\alpha})$ for both sides

Solving Lagrange Dual: Simplification

$$\max_{\alpha_i \geq 0} \min_{\vec{w}, b} L(\vec{w}, b, \vec{\alpha}) = \max_{\alpha_i \geq 0} \min_{\vec{w}, b} \frac{1}{2} \vec{w}^T \vec{w} + \sum_{i=1}^m \alpha_i (1 - y^{(i)} (\vec{w}^T \vec{z}^{(i)} + b)) \quad (8.33)$$

inner problem unconstrained, at opt.

$$\frac{\partial L}{\partial b} = - \sum_{i=1}^m \alpha_i y^{(i)} \stackrel{\text{set}}{=} 0 \quad (8.34)$$

then

$$\max_{\alpha_i \geq 0} \min_{\vec{w}, b} L = \max_{\alpha_i \geq 0} \min_{\vec{w}, b} \frac{1}{2} \vec{w}^T \vec{w} + \sum_{i=1}^m \alpha_i (1 - y^{(i)} \vec{w}^T \vec{z}^{(i)}) - \sum_{i=1}^m \alpha_i y^{(i)} \quad (8.35)$$

$$= \max_{\alpha_i \geq 0} \min_{\vec{w}} \frac{1}{2} \vec{w}^T \vec{w} + \sum_{i=1}^m \alpha_i (1 - y^{(i)} \vec{w}^T \vec{z}^{(i)}) \quad (8.36)$$

inner problem unconstrained, at opt.

$$\frac{\partial L}{\partial \vec{w}} = \vec{w} - \sum_{i=1}^m \alpha_i y^{(i)} \vec{z}^{(i)} \stackrel{\text{set}}{=} \vec{0} \quad (8.37)$$

then

$$\max_{\alpha_i \geq 0} \min_{\vec{w}, b} L = \max_{\alpha_i \geq 0} \min_{\vec{w}} \frac{1}{2} \vec{w}^T \vec{w} + \sum_{i=1}^m \alpha_i - \vec{w}^T \left(\sum_{i=1}^m \alpha_i y^{(i)} \vec{z}^{(i)} \right) \quad (8.38)$$

$$= \max_{\alpha_i \geq 0} \min_{\vec{w}} \frac{1}{2} \vec{w}^T \vec{w} + \sum_{i=1}^m \alpha_i - \vec{w}^T \vec{w} \quad (8.39) \quad \begin{matrix} \text{complimentary} \\ \text{slackness} \end{matrix}$$

$$= \max_{\alpha_i \geq 0} \min_{\vec{w}} -\frac{1}{2} \vec{w}^T \vec{w} + \sum_{i=1}^m \alpha_i \quad (8.40)$$

$$= \max_{\alpha_i \geq 0} -\frac{1}{2} \left\| \sum_{i=1}^m \alpha_i y^{(i)} \vec{z}^{(i)} \right\|^2 + \sum_{i=1}^m \alpha_i \quad (8.41)$$

$$= \max_{\alpha_i \geq 0} -\frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y^{(i)} y^{(j)} \vec{z}^{(i)T} \vec{z}^{(j)} + \sum_{i=1}^m \alpha_i \quad (8.42)$$

KKT Optimality Conditions

- if primal-dual opt. $(\vec{w}, b, \vec{\alpha})$
- primal feasible: $y^{(i)}(\vec{w}^T \vec{z}^{(i)} + b) \geq 1, \forall i$
- dual feasible: $\alpha_i \geq 0, \forall i$
- primal-ineer opt. (at opt. all Lagrange terms disappear)

$$\alpha_i (1 - y^{(i)}(\vec{w}^T \vec{z}^{(i)} + b)) = 0, \forall i \quad (8.43)$$

- call it **complimentary slackness**, if $\alpha_i > 0 \rightarrow \vec{z}^{(i)}$ on fat boundary (SV!)
- dual-inner opt.: $\sum_{i=1}^m \alpha_i y^{(i)} = 0, \vec{w} = \sum_{i=1}^m \alpha_i y^{(i)} \vec{z}^{(i)}$

8.3.3 Solving Dual SVM

Dual Formulation of SVM

Algorithm 28 Dual SVM

$$\min_{\vec{\alpha} \in \mathbb{R}^m} \frac{1}{2} \vec{\alpha}^T Q \vec{\alpha} - \vec{1}^T \vec{\alpha} \quad (8.44)$$

$$\text{st. } \vec{y}^T \vec{\alpha} = 0 // \text{ from dual-inner opt.} \quad (8.45)$$

$$\alpha_i \geq 0, \forall i = \{1, 2, \dots, m\} // \text{ from dual feasible} \quad (8.46)$$

m vars.

$m + 1$ constraints

$$Q_{ij} = y^{(i)} y^{(j)} \vec{z}^{(i)T} \vec{z}^{(j)} \quad (8.47)$$

- $Q \in \mathbb{S}^m$
- if $m = 30,000$, dense Q takes $> 3G$ RAM
- need special solver for not storing whole Q and utilizing special constraints properly to scale up to large m

Dual SVM with QP Solver

Algorithm 29 QP

$$\min_{\vec{u}} \frac{1}{2} \vec{u}^T Q \vec{u} + \vec{p}^T \vec{u} \quad (8.48)$$

$$\text{st. } \vec{a}_i^T \vec{u} \geq c_i, \forall i \in \{1, 2, \dots, m\} \quad (8.49)$$

Algorithm 30 Dual SVM with QP solver

$$Q_{ij} = y^{(i)} y^{(j)} \vec{z}^{(i)T} \vec{z}^{(j)}$$

$$\vec{p} = -\vec{1} \in \mathbb{R}^m$$

$$\vec{a}_{\geq} = \vec{y}, c_{\geq} = 0$$

$$\vec{a}_{\leq} = \vec{y}, c_{\leq} = 0$$

$$\vec{a}_i = i\text{-th unit direction in } \mathbb{R}^m, \forall i \in \{1, 2, \dots, m\}$$

$$c_i = 0, \forall i \in \{1, 2, \dots, m\}$$

$$\vec{\alpha} = \vec{u} = \text{QP}(Q, \vec{p}, A, \vec{c})$$

$$\vec{w} = \sum_{i=1}^m \alpha_i y^{(i)} \vec{z}^{(i)} \quad // \text{from dual-inner opt.}$$

$$b = y^{(i)} - \vec{w}^T \vec{z}^{(i)}, \text{ for one } \alpha_i > 0 \quad // \text{from complimentary slackness}$$

return $h(\vec{x}) = \text{sign}(\vec{w}^T \vec{x} + b)$

note: many solvers treat equality $\vec{a}_{\geq}, \vec{a}_{\leq}$ & bound \vec{a}_i constraints specially for numerical stability

8.3.4 Messages behind Dual SVM

SV Revisited

on boundary: locates fattest hyperplane; others: not needed

examples with $\alpha_i > 0$: on boundary

call $\alpha_i > 0$ examples $(\vec{z}^{(i)}, y^{(i)})$: SVs

SV ($\alpha_i > 0$) \subseteq SV candidates (on boundary)

only SV needed to compute \vec{w}

$$\vec{w} = \sum_{i=1}^m \alpha_i y^{(i)} \vec{z}^{(i)} = \sum_{\text{SV}} \alpha_i y^{(i)} \vec{z}^{(i)} \quad (8.50)$$

only SV needed to compute b , for one SV $(\vec{z}^{(i)}, y^{(i)})$

$$b = y^{(i)} - \vec{w}^T \vec{z}^{(i)} \quad (8.51)$$

Representation of Fattest Hyperplane

SVM

$$\vec{w} = \sum_{i=1}^m \alpha_i y^{(i)} \vec{z}^{(i)} \quad (8.52)$$

α_i from dual sol.

PLA

$$\vec{\theta} = \sum_{i=1}^m \beta_i y^{(i)} \vec{z}^{(i)} \quad (8.53)$$

β_i from # mistakes corrections

\vec{w} = lin. comb. of $y^{(i)} \vec{z}^{(i)}$

- also true for GD/ SGD-based LogReg/ LinReg when $\vec{\theta}_0 = \vec{0}$
- call \vec{w} repr. by data
- SVM: repr. \vec{w} by SVs only

Summary: Two Forms of Hard-Margin SVM

Algorithm 31 Primal Hard-Margin SVM

$$\min_{\vec{w}, b} \frac{1}{2} \vec{w}^T \vec{w} \quad (8.54)$$

$$\text{st. } y^{(i)} (\vec{w}^T \vec{x}^{(i)} + b) \geq 1, \forall i \in \{1, 2, \dots, m\} \quad (8.55)$$

- $\tilde{n} + 1$ vars., m constraints
- suitable when $\tilde{n} + 1$ is small
- physical meaning: locate specially-scaled (\vec{w}, b)

Algorithm 32 Dual Hard-Margin SVM

$$\min_{\vec{\alpha} \in \mathbb{R}^m} \frac{1}{2} \vec{\alpha}^T Q \vec{\alpha} - \vec{1}^T \vec{\alpha} \quad (8.56)$$

$$\text{st. } \vec{y}^T \vec{\alpha} = 0 \quad (8.57)$$

$$\alpha_i \geq 0, \forall i = \{1, 2, \dots, m\} \quad (8.58)$$

- m vars., $m + 1$ constraints
- suitable when m is small
- $Q_{ij} = y^{(i)} y^{(j)} \vec{z}^{(i)T} \vec{z}^{(j)}$: inner prod. in $\mathbb{R}^{\tilde{n}}$
- physical meaning: locate SVs $(\vec{z}^{(i)}, y^{(i)})$ and their α_i

kernel function

8.4 Kernel Support Vector Machine

8.4.1 Kernel Trick

Fast Inner Product for $\vec{\phi}_2$

2nd order poly. transform

$$\vec{\phi}_2(\vec{x}) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_n \\ x_1^2 \\ x_1x_2 \\ \vdots \\ x_1x_n \\ x_2x_1 \\ \vdots \\ x_n^2 \end{bmatrix} \in \mathbb{R}^{\tilde{n}+1} \quad (8.59)$$

include both x_1x_2 and x_2x_1 for simplicity

$$\vec{\phi}_2(\vec{x})^T \vec{\phi}_2(\vec{x}') = 1 + \sum_{j=1}^n x_j x'_j + \sum_{j=1}^n \sum_{k=1}^n x_j x_k x'_j x'_k \quad (8.60)$$

$$= 1 + \sum_{j=1}^n x_j x'_j + \sum_{j=1}^n x_j x'_j \sum_{k=1}^n x_k x'_k \quad (8.61)$$

$$= 1 + \vec{x}^T \vec{x}' + (\vec{x}^T \vec{x}')(\vec{x}^T \vec{x}') \quad (8.62)$$

$$= K_{\vec{\phi}_2}(\vec{x}, \vec{x}') \quad (8.63)$$

for $\vec{\phi}_2$, transform and inner prod. can be carefully done with **kernel function** in $O(n)$ instead of $O(\tilde{n})$

Kernel: Transform + Inner Product

$$Q_{ij} = y^{(i)} y^{(j)} \vec{z}^{(i)T} \vec{z}^{(j)} = y^{(i)} y^{(j)} K(\vec{x}^{(i)}, \vec{x}^{(j)}) \quad (8.64)$$

opt. b ? from one SV $(\vec{x}^{(s)}, y^{(s)})$

$$b = y^{(s)} - \vec{w}^T \vec{z}^{(s)} \quad (8.65)$$

$$= y^{(s)} - \left(\sum_{SV} \alpha_i y^{(i)} \vec{z}^{(i)} \right)^T \vec{z}^{(s)} \quad (8.66)$$

$$= y^{(s)} - \sum_{SV} \alpha_i y^{(i)} K(\vec{x}^{(i)}, \vec{x}^{(s)}) \quad (8.67) \text{ kernel trick}$$

hypothesis $h(\vec{x})$ for test input \vec{x}

$$h(\vec{x}) = \text{sign}(\vec{w}^T \vec{x} + b) \quad (8.68)$$

$$= \text{sign}\left(\sum_{SV} \alpha_i y^{(i)} K(\vec{x}^{(i)}, \vec{x}) + b\right) \quad (8.69)$$

kernel trick : plug in efficient kernel function to avoid dep. on \tilde{n}

Kernel SVM with QP

Algorithm 33 Kernel Hard-Margin Dual SVM with QP solver

```

 $Q_{ij} = y^{(i)} y^{(j)} K(\vec{x}^{(i)}, \vec{x}^{(j)})$ 
// time complexity  $O(m^2)$  · (kernel evaluation)
 $\vec{p} = -\vec{1} \in \mathbb{R}^m$ 
 $\vec{a}_{\geq} = \vec{y}, c_{\geq} = 0$ 
 $\vec{a}_{\leq} = \vec{y}, c_{\leq} = 0$ 
 $\vec{a}_i = i\text{-th unit direction in } \mathbb{R}^m, \forall i \in \{1, 2, \dots, m\}$ 
 $c_i = 0, \forall i \in \{1, 2, \dots, m\}$ 
 $\vec{a} = \vec{u} = \text{QP}(Q, \vec{p}, A, \vec{c})$  // QP with  $m$  vars. and  $m+1$  constraints
 $b = y^{(s)} - \sum_{SV} \alpha_i y^{(i)} K(\vec{x}^{(i)}, \vec{x}^{(s)}),$  for one SV  $(\vec{x}^{(s)}, y^{(s)})$ 
// time complexity  $O(\#SVs)$  · (kernel evaluation)
return  $h(\vec{x}) = \text{sign}(\sum_{SV} \alpha_i y^{(i)} K(\vec{x}^{(i)}, \vec{x}) + b)$ 
// time complexity  $O(\#SVs)$  · (kernel evaluation)

```

8.4.2 Polynomial Kernel

General Poly-2 Kernel

$$\vec{\phi}_2(\vec{x}) = [1 \ x_1 \ \dots \ x_n \ x_1^2 \ \dots \ x_n^2]^T \quad (8.70)$$

$$\iff K_{\vec{\phi}_2}(\vec{x}, \vec{x}') = 1 + \vec{x}^T \vec{x}' + (\vec{x}^T \vec{x}')^2 \quad (8.71)$$

$$\vec{\phi}_2(\vec{x}) = [1 \ \sqrt{2}x_1 \ \dots \ \sqrt{2}x_n \ x_1^2 \ \dots \ x_n^2]^T \quad (8.72)$$

$$\iff K_{\vec{\phi}_2}(\vec{x}, \vec{x}') = 1 + 2\vec{x}^T \vec{x}' + (\vec{x}^T \vec{x}')^2 \quad (8.73)$$

$$\vec{\phi}_2(\vec{x}) = [1 \ \sqrt{2\gamma}x_1 \ \dots \ \sqrt{2\gamma}x_n \ \gamma x_1^2 \ \dots \ \gamma x_n^2]^T \quad (8.74)$$

$$\iff K_2(\vec{x}, \vec{x}') = 1 + 2\gamma \vec{x}^T \vec{x}' + \gamma^2 (\vec{x}^T \vec{x}')^2 = (1 + \gamma \vec{x}^T \vec{x}')^2 \quad (8.75)$$

with $\gamma > 0$

all these have equiv. power, but have diff. inner prod. → diff. geometry
 K_2 is somewhat easier to cal. than $K_{\vec{\phi}_2}$, it is more commonly used

Poly-2 Kernels in Action

$h(\vec{x})$ diff., SVs diff., see Fig. ref{poly}
 hard to say which is better before learning
 $\text{change of kernel} \iff \text{change of margin def.}$

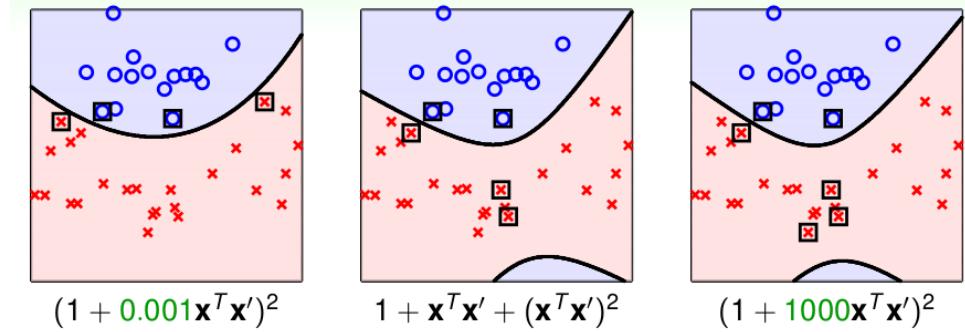


Figure 8.5: Poly-2 Kernels in Action

General Polynomial Kernel

$$K(\vec{x}, \vec{x}') = (\nu + \gamma \vec{x}^T \vec{x}')^d, \nu \geq 0, \gamma > 0 \quad (8.76)$$

Special Case: Linear Kernel

$$K_1(\vec{x}, \vec{x}') = (0 + 1 \cdot \vec{x}^T \vec{x}')^1 = \vec{x}^T \vec{x}' \quad (8.77)$$

just usual inner prod., can be sol. (often in primal form) efficiently

8.4.3 Gaussian Kernel

Kernel of Infinite Dimensional Transform

consider $x \in \mathbb{R}$

$$K(x, x') = \exp(-(x - x')^2) \quad (8.78)$$

$$= \exp(-x^2) \exp(-x'^2) \exp(2xx') \quad (8.79)$$

$$= \exp(-x^2) \exp(-x'^2) \sum_{k=0}^{\infty} \frac{(2xx')^k}{k!} \quad (8.80)$$

$$\text{// } \exp(a) = \sum_{k=0}^{\infty} \frac{a^k}{k!} \text{ Taylor expansion} \quad (8.81)$$

$$= \sum_{k=0}^{\infty} \exp(-x^2) \exp(-x'^2) \sqrt{\frac{2^k}{k!}} \sqrt{\frac{2^k}{k!}} x^k x'^k \quad (8.82)$$

$$= \vec{\phi}(\vec{x})^T \vec{\phi}(\vec{x}') \quad (8.83) \text{ Radial Basis Function (RBF) kernel}$$

where

$$\vec{\phi}(\vec{x}) = \exp(-x^2) \begin{bmatrix} 1 \\ \sqrt{\frac{2}{1!}} x \\ \sqrt{\frac{2^2}{2!}} x^2 \\ \vdots \end{bmatrix} \in \mathbb{R}^\infty \quad (8.84)$$

more generally, Gaussian Kernel

$$K(\vec{x}, \vec{x}') = \exp\left(-\frac{\|\vec{x} - \vec{x}'\|^2}{2\sigma^2}\right) \quad (8.85)$$

- close to 1 if \vec{x} and \vec{x}' are close
- near 0 if \vec{x} and \vec{x}' are far apart

Hypothesis of Gaussian SVM

$$h(\vec{x}) = \text{sign}\left(\sum_{SV} \alpha_i y^{(i)} \exp\left(-\frac{\|\vec{x} - \vec{x}^{(i)}\|^2}{2\sigma^2}\right)\right) + b \quad (8.86)$$

lin. comb. of Gaussians centered at SVs $\vec{x}^{(i)}$

also called **Radial Basis Function (RBF) kernel**

Support Vector Mechanism

large margin hyperplane: not many

high-order transforms $\vec{z} = \vec{\phi}(\vec{x})$ with kernel trick: sophisticated

store opt. \vec{w} : by a few SVs and α_i

new possibility by Gaussian SVM: infinite-dimensional linear classification, with generalization guarded by large-margin

Gaussian SVM in Action

small $\sigma \rightarrow$ sharp Gaussians \rightarrow overfit, see Fig. 8.6

8.4.4 Comparison of Kernels

Linear Kernel: Cons and Pros

Pros

- safe: lin. first
- fast: with special QP solver in primal
- very explainable: \vec{w} and SVs say sth.

Cons

- restricted: not always separable

Mercers condition

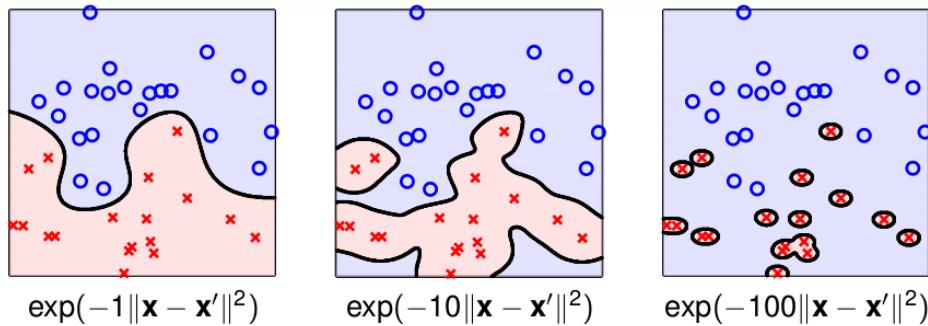


Figure 8.6: Gaussian SVM in Action

Polynomial Kernel: Cons and Pros

Pros

- less restricted than linear
- strong physical control: ‘knows’ deg. d

Cons

- numerical difficulty for large d
if $\|\xi + \gamma\vec{x}^T \vec{x}'\| < 1$, then $K \rightarrow 0$
if $\|\xi + \gamma\vec{x}^T \vec{x}'\| > 1$, then $K \rightarrow \text{big}$
- three parameters (γ, ν, d): more difficult to select
perhaps small- d only, sometimes efficiently done by linear on $\vec{\phi}_d(\vec{x})$

Gaussian Kernel: Cons and Pros

Pros

- more powerful than linear/poly.
- boundedless numerical difficulty than poly.
- one parameter only easier to select than poly.

Cons

- mysterious no \vec{w}
 - slower than linear
 - too powerful
- one of most popular but shall be used with care

Other Valid Kernels

More esoteric: String kernel, chi-square kernel, histogram intersection kernel

kernel represents special similarity: $K(\vec{x}, \vec{x}') = \vec{\phi}(\vec{x})^T \vec{\phi}(\vec{x}')$
for valid kernel let

$$K_{ij} = K(\vec{x}^{(i)}, \vec{x}^{(j)}) \quad (8.87)$$

Mercers condition, K must always be psd.

proof: for a valid kernel $K \in \mathbb{R}^{m \times m}$

$$K_{ij} = K(\vec{x}^{(i)}, \vec{x}^{(j)}) = \vec{\phi}(\vec{x}^{(i)})^T \vec{\phi}(\vec{x}^{(j)}) = \vec{\phi}(\vec{x}^{(j)})^T \vec{\phi}(\vec{x}^{(i)}) = K(\vec{x}^{(j)}, \vec{x}^{(i)}) = K_{ji} \quad (8.88)$$

thus, $K \in \mathbb{S}$

for $\forall \vec{a} \neq \vec{z}$

$$\vec{a}^T K \vec{a} = \sum_{i=1}^m \sum_{j=1}^m a_i K_{ij} a_j \quad (8.89)$$

$$= \sum_{i=1}^m \sum_{j=1}^m a_i \vec{\phi}(\vec{x}^{(i)})^T \vec{\phi}(\vec{x}^{(j)}) a_j \quad (8.90)$$

$$= \sum_{i=1}^m \sum_{j=1}^m a_i \left(\sum_{k=1}^{\tilde{n}} \phi(\vec{x}^{(i)})_k \phi(\vec{x}^{(j)})_k \right) a_j \quad (8.91)$$

$$= \sum_{i=1}^m \sum_{j=1}^m \sum_{k=1}^{\tilde{n}} a_i \phi(\vec{x}^{(i)})_k \phi(\vec{x}^{(j)})_k a_j \quad (8.92)$$

$$= \sum_{k=1}^{\tilde{n}} \sum_{i=1}^m a_i \phi(\vec{x}^{(i)})_k \sum_{j=1}^m a_j \phi(\vec{x}^{(j)})_k \quad (8.93)$$

$$= \sum_{k=1}^{\tilde{n}} \left(\sum_{i=1}^m a_i \phi(\vec{x}^{(i)})_k \right)^2 \quad (8.94)$$

$$\geq 0 \quad (8.95)$$

thus, K is psd. ■

8.5 Soft-Margin Support Vector Machine

8.5.1 Motivation and Primal Problem

Cons of Hard-Margin SVM

SVM can still overfit, reason

- ϕ : too powerful
- if always insisting on separable: have power to overfit to noise, see Fig. 8.7

Give Up on Some Examples

to give up some noisy examples

$$\min_{\vec{w}, b} \frac{1}{2} \vec{w}^T \vec{w} + C \sum_{i=1}^m \mathbb{1}\{y^{(i)} \neq \text{sign}(\vec{w}^T \vec{z}^{(i)} + b)\} \quad (8.96)$$

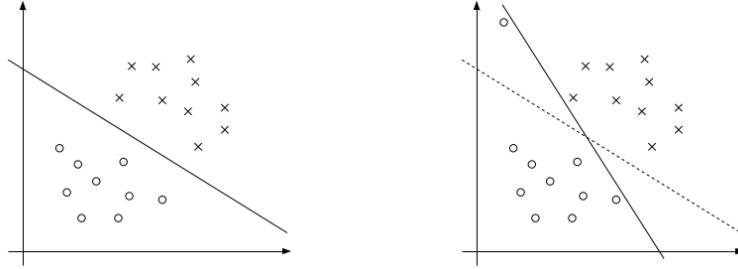


Figure 8.7: the left figure below shows an optimal margin classifier, and when a single outlier is added in the upper-left region (right figure), it causes the decision boundary to make a dramatic swing, and the resulting classifier has a much smaller margin.

$$\text{st. } y^{(i)}(\vec{w}^T \vec{z}^{(i)} + b) \geq 1 \text{ for correct } i \quad (8.97)$$

$$y^{(i)}(\vec{w}^T \vec{z}^{(i)} + b) \geq -\infty \text{ for incorrect } i \quad (8.98)$$

C : trade-off betw. large margin and noise tol.

Soft-Margin SVM

$$\min_{\vec{w}, b} \frac{1}{2} \vec{w}^T \vec{w} + C \sum_{i=1}^m \mathbb{1}\{y^{(i)} \neq \text{sign}(\vec{w}^T \vec{z}^{(i)} + b)\} \quad (8.99)$$

$$\text{st. } y^{(i)}(\vec{w}^T \vec{z}^{(i)} + b) \geq 1 - \infty \cdot \mathbb{1}\{y^{(i)} \neq \text{sign}(\vec{w}^T \vec{z}^{(i)} + b)\} \quad (8.100)$$

$\mathbb{1}\{\cdot\}$ non-lin., not QP anymore
 cannot distinguish small err. (slightly away from boundary), or large err.
 (far away from fat boundary)
 record margin violation ξ_i : lin. constraints
 penalize with margin violation instead of err. count: QP

Algorithm 34 Soft-Margin Primal SVM

$$\min_{\vec{w}, b, \xi} \frac{1}{2} \vec{w}^T \vec{w} + C \sum_{i=1}^m \xi_i \quad (8.101)$$

$$\text{st. } y^{(i)}(\vec{w}^T \vec{z}^{(i)} + b) \geq 1 - \xi_i, \forall i \in \{1, 2, \dots, m\} \quad (8.102)$$

$$\xi_i \geq 0, \forall i \in \{1, 2, \dots, m\} \quad (8.103)$$

C : trade-off betw. large margin and most examples have margin ≥ 1 ,
 see Fig. 8.8

- large C : want less margin violation
- small C : want large margin

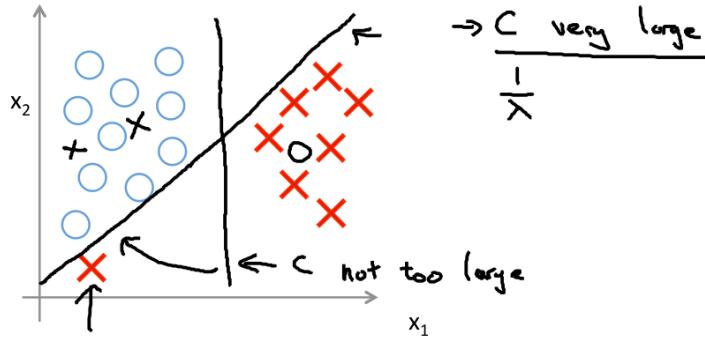


Figure 8.8: trade-off betw. large margin and less margin violation

QP with $\tilde{n} + 1 + m$ vars., $2m$ constraints

8.5.2 Dual Problem

Lagrange Dual

$$L(\vec{w}, b, \vec{\xi}, \vec{\alpha}, \vec{\beta}) = \frac{1}{2} \vec{w}^T \vec{w} + C \sum_{i=1}^m \xi_i + \sum_{i=1}^m \alpha_i (1 - \xi_i - y^{(i)} (\vec{w}^T \vec{z}^{(i)} + b)) + \sum_{i=1}^m \beta_i (-\xi_i) \quad (8.104)$$

dual problem

$$\max_{\alpha_i \geq 0, \beta_i \geq 0} \min_{\vec{w}, b, \vec{\xi}} \frac{1}{2} \vec{w}^T \vec{w} + C \sum_{i=1}^m \xi_i + \sum_{i=1}^m \alpha_i (1 - \xi_i - y^{(i)} (\vec{w}^T \vec{z}^{(i)} + b)) + \sum_{i=1}^m \beta_i (-\xi_i) \quad (8.105)$$

Simplify ξ_i and β_i

$$\frac{\partial L}{\partial \xi_i} = C - \alpha_i - \beta_i \stackrel{\text{set}}{=} 0 \quad (8.106)$$

$$\beta_i = C - \alpha_i \geq 0 \quad (8.107)$$

wlog., solving with constraints

$$0 \leq \alpha_i \leq C \quad (8.108)$$

β_i can be removed

SMO (sequential minimal optimization)

$$\max_{0 \leq \alpha_i \leq C} \min_{\vec{w}, b} \frac{1}{2} \vec{w}^T \vec{w} + \sum_{i=1}^m \alpha_i (1 - y^{(i)} (\vec{w}^T \vec{z}^{(i)} + b)) \quad (8.109)$$

inner problem same as hard-margin SVM

$$\vec{w} = \sum_{i=1}^m \alpha_i y^{(i)} \vec{z}^{(i)} \quad (8.110)$$

Standard Soft-Margin SVM Dual

Algorithm 35 Soft-Margin Dual SVM

$$\min_{\vec{\alpha}} \frac{1}{2} \vec{\alpha}^T Q \vec{\alpha} - \vec{1}^T \vec{\alpha} \quad (8.111)$$

$$\text{st. } \vec{y}^T \vec{\alpha} = 0 \quad (8.112)$$

$$0 \leq \alpha_i \leq C, \forall i \in \{1, 2, \dots, m\} \quad (8.113)$$

m vars., $2m + 1$ constraints
large $C \rightarrow$ less noise tol. \rightarrow overfit

8.5.3 SMO Algorithm

The **SMO (sequential minimal optimization)** algorithm, gives an efficient way of solving the dual problem arising from the derivation of the SVM

Coordinate Ascent

Consider trying to solve the unconstrained optimization problem

$$\max_{\vec{\alpha}} f(\vec{\alpha}) \quad (8.114)$$

Algorithm 36 Coordinate Ascent

loop until conv.

for $j = 1 : n$
 $\alpha_j \leftarrow \arg \max_{\alpha_j} f(\vec{\alpha})$

in the innermost loop of this algorithm, we will hold all the variables except for α_j fixed, and reoptimize f wrt. just the α_j , see Fig. 8.9

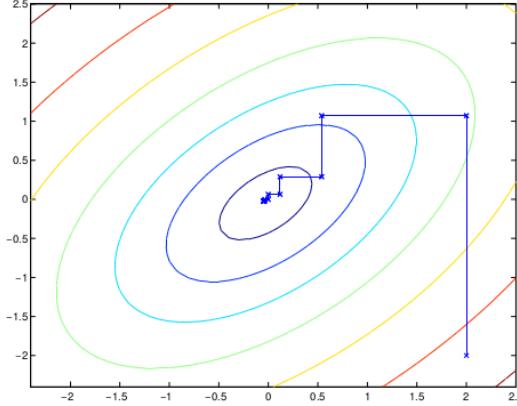


Figure 8.9: Coordinate ascend in action. Notice that on each step, coordinate ascent takes a step that's parallel to one of the axes, since only one variable is being optimized at a time.

SMO

recall one of the constraints in Soft-Margin Dual SVM

$$\vec{y}^T \vec{\alpha} = 0 \quad (8.115)$$

suppose we want to hold $\alpha_2, \dots, \alpha_m$ fixed, and take a coordinate ascent step and reoptimize the objective wrt. α_1 . Can we make any progress? No! because

$$\alpha_1 = -y^{(1)} \sum_{i=2}^m \alpha_i y^{(i)} \quad (8.116)$$

Hence, α_1 is exactly determined by other α_i 's, we cannot make any change w/o violating the constraint in the opt. problem

Thus, if we want to update some subject of the α_i 's, we must update at least two of them simultaneously in order to keep satisfying the constraints.

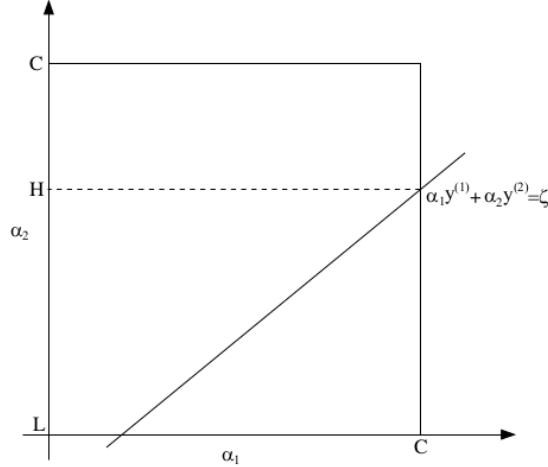
The key reason that SMO is an efficient algorithm is that the update to α_i and α_j can be computed very efficiently.

Lets say we currently have some setting of the $\vec{\alpha}$ that satisfy the Soft-Margin Dual SVM constraints, suppose we've decided to hold $\alpha_3, \alpha_4, \dots, \alpha_m$ fixed, and want to reopt. wrt. to α_1 and α_2 subject to the constraints

$$0 \leq \alpha_1 \leq C \quad (8.117)$$

$$0 \leq \alpha_2 \leq C \quad (8.118)$$

$$\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = - \sum_{i=3}^m \alpha_i y^{(i)} \stackrel{\text{def.}}{=} \zeta \quad (8.119)$$

Figure 8.10: Constraints on α_1 and α_2

see Fig. 8.10, α_1 and α_2 must lie within the box $[0, C] \times [0, C]$ and on the line $\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = \zeta$. Note also that, from these constraints, $L \leq \alpha_2 \leq H$ rewrite constraints

$$\alpha_1 = (\zeta - \alpha_2 y^{(2)}) y^{(1)} \quad (8.120)$$

then $f(\vec{\alpha})$ is some quad. form of α_2 , if ignore the constraints $L \leq \alpha_2 \leq H$, we easily maximize this quadratic function by setting its derivative to zero and solving to get $\hat{\alpha}_2$

when apply the constraints, we can find the opt. α_2^* by clipping $\hat{\alpha}_2$ to lie in the $[L, H]$ int.

To test for convergence of this algorithm, we can check whether the KKT conditions are satisfied to within some tol. Here, tol is the convergence tolerance parameter, and is typically set to around 0.01 to 0.001.

8.5.4 Messages behind Soft-Margin SVM

Solving for b

complimentary slackness:

$$\alpha_i (1 - \xi_i - y^{(i)} (\vec{w}^T \vec{z}^{(i)} + b)) = 0 \quad (8.126)$$

$$\beta_i \xi_i = (C - \alpha_i) \xi_i = 0 \quad (8.127)$$

$$\begin{aligned} \text{SV } (\alpha_s > 0) \\ b &= y^{(s)} - y^{(s)} \xi_s - \vec{w}^T \vec{z}^{(s)} \end{aligned} \quad (8.128)$$

$$\begin{aligned} \text{free } (\alpha_s < C, \beta_s > 0) \\ \xi_s &= 0 \end{aligned} \quad (8.129)$$

Algorithm 37 SMO

loop until conv.

select some α_j and α_k to update next (using a heuristic that tries to pick the two that will allow us to make the biggest progress towards the global maximum)

quad. opt. on $f(\alpha_1, \dots, (\zeta - \alpha_k y^{(k)})y^{(j)}, \dots, \alpha_k, \dots, \alpha_m)$ w/o constraints to get $\hat{\alpha}_k$

get constraints $L \leq \alpha_k \leq H$ from

$$0 \leq \alpha_j \leq C \quad (8.121)$$

$$0 \leq \alpha_k \leq C \quad (8.122)$$

$$\alpha_j y^{(j)} + \alpha_k y^{(k)} = - \sum_{i \text{ except } j,k} \alpha_i y^{(i)} \stackrel{\text{def.}}{=} \zeta \quad (8.123)$$

update rule of α_k is

$$\alpha_k \leftarrow \begin{cases} H & \text{if } \hat{\alpha}_k > H \\ \hat{\alpha}_j & \text{if } L \leq \hat{\alpha}_k \leq H \\ L & \text{if } \hat{\alpha}_k < L \end{cases} \quad (8.124)$$

update rule of α_j is

$$\alpha_j \leftarrow (\zeta - \alpha_k y^{(k)})y^{(j)} \quad (8.125)$$

sol. b with free SV $(\vec{x}^{(s)}, y^{(s)})$

$$b = y^{(s)} - \sum_{SV} \alpha_i y^{(i)} K(\vec{x}^{(i)}, \vec{x}^{(s)}) \quad (8.130)$$

Physical Meaning of α_i

see Fig. 8.11

non SV \circlearrowleft

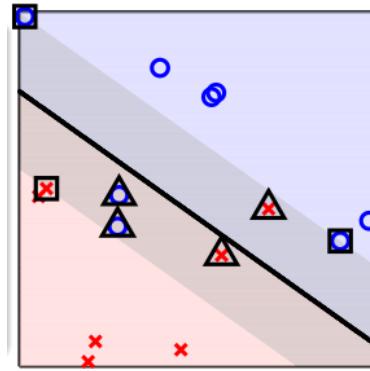
- $\alpha_i = 0 : \xi_i = 0$
- away from fat boundary or on boundary (SV candidate)

free SV \square

- $0 < \alpha_i < C : \xi_i = 0$
- locate b , on fat boundary

bounded SV \triangle

- $\alpha_i = C, \xi_i$: violation amount
- violate or on fat boundary

Figure 8.11: Physical Meaning of α_i

8.5.5 Model Selection

Model Selection by CV

eg., need select (C, γ) for Gaussians SVM

$J_{CV}(C, \gamma)$: non-smooth function of (C, γ) , difficult to opt.

proper models can be chosen by V-fold cross validation on a few grid values of (C, γ)

LOOCV Error for SVM

recall: $J_{LOOCV} = J_{CV}$ with m -fold
scaled # SV bounds LOOCV error

$$J_{LOOCV} \leq \frac{\#SV}{m} \quad (8.131)$$

proof: for $(\vec{x}^{(m)}, y^{(m)})$

- if opt. $\alpha_m = 0$ (non-SV)
- $(\alpha_1, \alpha_2, \dots, \alpha_{m-1})$ still opt. when leaving out $(\vec{x}^{(m)}, y^{(m)})$ (proof it by contradiction)
- $h^- = h$

$$e^{(\text{non SV})} = \text{err}(g^-, \vec{x}^{(\text{non SV})}, y^{(\text{non SV})}) = \text{err}(g, \vec{x}^{(\text{non SV})}, y^{(\text{non SV})}) = 0 \quad (8.132)$$

$$e^{(\text{SV})} = \text{err}(g^-, \vec{x}^{(\text{SV})}, y^{(\text{SV})}) \leq 1 \quad (8.133)$$

■

Selection by # SV

SV(C, γ): non-smooth function of (C, γ) , difficult to opt.

just an upper bound

dangerous model can be ruled out by # SV on a few grid of (C, γ)

#SV: often used as a safety check when computing J_{CV} is too time-consuming hinge error measure

8.6 Kernel Logistic Regression

8.6.1 Soft-Margin SVM as Regularized Model

Slack Variable ξ_i

soft-margin SVM

$$\min_{\vec{w}, b, \vec{\xi}} \frac{1}{2} \vec{w}^T \vec{w} + C \sum_{i=1}^m \xi_i \quad (8.134)$$

$$\text{st. } y^{(i)}(\vec{w}^T \vec{z}^{(i)} + b) \geq 1 - \xi_i, \forall i \in \{1, 2, \dots, m\} \quad (8.135)$$

$$\xi_i \geq 0, \forall i \in \{1, 2, \dots, m\} \quad (8.136)$$

margin violation

$$\xi_i = \max(0, 1 - y^{(i)}(\vec{w}^T \vec{z}^{(i)} + b)) = \begin{cases} 0 & \text{if } y^{(i)}(\vec{w}^T \vec{z}^{(i)} + b) \geq 1 \\ 1 - y^{(i)}(\vec{w}^T \vec{z}^{(i)} + b) & \text{if } y^{(i)}(\vec{w}^T \vec{z}^{(i)} + b) < 1 \end{cases} \quad (8.137)$$

Unconstrained Form

$$\min_{\vec{w}, b} \frac{1}{2} \vec{w}^T \vec{w} + C \sum_{i=1}^m \max(0, 1 - y^{(i)}(\vec{w}^T \vec{z}^{(i)} + b)) \quad (8.138)$$

$$= \frac{1}{2} \vec{w}^T \vec{w} + C \sum_{i=1}^m \text{err}(\vec{w}, b, \vec{x}^{(i)}, y^{(i)}) \quad (8.139)$$

- just L_2 regularization, with \vec{w} not $\vec{\theta}$, C not λ , **hinge error measure** $\text{err}(\vec{w}, b, \vec{x}^{(i)}, y^{(i)})$
- large margin \iff fewer hyperplanes \iff L_2 regularization of \vec{w}
- $C \uparrow \iff \lambda \downarrow \iff$ less regularization
- not QP, no (?) kernel trick
- not differentiable, harder to sol.

view SVM as regularized model: allows extending / connecting to other learning models

8.6.2 SVM vs. Logistic Regression

Error Measures

$$s = \vec{w}^T \vec{z} + b \quad (8.140)$$

$$\text{err}_{01}(s, y) = 1\{sy \leq 0\} \quad (8.141)$$

$$\text{err}_{svm}(s, y) = \max(0, 1 - ys) \geq \text{err}_{01}(s, y) \quad (8.142)$$

$$\text{err}_{sce}(s, y) = \lg(1 + \exp(-ys)) \geq \text{err}_{01}(s, y) \quad (8.143)$$

see Fig. 8.12

SVM $\approx L_2$ -regularized logistic regression, Tab. 8.4

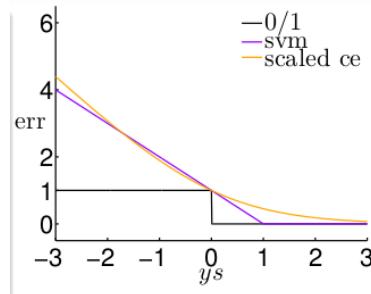


Figure 8.12: Three Different Error Measure

Table 8.4: Hinge Error and SCE Error

ys	$-\infty$	∞
$\text{err}_{svm}(s, y)$	$-ys$	0
$\text{err}_{sce}(s, y)$	$-ys$	0

Linear Models for Binary Classification

PLA: min. err_{01} specially

- pros: efficient if lin. sep.
 - cons: works only if lin. sep., ov. needing pocket
- regularized logistic regression for classification: min. err_{ce} by GD / SGD
- pros: easy opt. & regularization guard
 - cons: loose bound of err_{01} for very negative ys
- soft-margin SVM: min. regularized err_{svm} by QP
- pros: easy opt. & theoretical guarantee
 - cons: loose bound of err_{01} for very negative ys

8.6.3 SVM for Soft Binary Classification

SVM for Soft Binary Classification

naive idea 1

- 1. run SVM get \vec{w}_{svm}, b_{svm}

- 2. return $h(\vec{x}) = g(\vec{w}_{svm}^T \vec{x} + b_{svm})$
- directly use of similarity: works reasonably well
- no LogReg flavor
- naive idea 2
 - 1. run SVM get \vec{w}_{svm}, b_{svm}
 - 2. $\vec{\theta}_0 = [b_{svm} \vec{w}_{svm}]^T$
 - 3. run LogReg with init. $\vec{\theta}_0$ to get $h(\vec{x})$
 - not really easier than original LogReg
 - SVM flavor (kernel) lost

A Possible Model: Two-Level Learning

$$h(\vec{x}) = g(\theta_1(\vec{w}_{svm}^T \vec{\phi}(\vec{x}) + b_{svm}) + \theta_0) \quad (8.144)$$

- SVM flavor: fix hyperplane direction by \vec{w}_{svm} : kernel applies
- LogReg flavor: fine-tune hyperplane to match MLE by scaling (θ_1) and shifting (θ_0)
 - often $\theta_1 > 0$ when \vec{w}_{svm} reasonably good
 - often $\theta_0 \approx 0$ when b_{svm} reasonably good

Probabilistic SVM

Algorithm 38 Platt's Model of Probabilistic SVM for Soft Binary Classification

```
[ $\vec{w}_{svm}, b_{svm}$ ]  $\leftarrow$  SVM( $\{\vec{x}^{(i)}, y^{(i)}\}_{i=1}^m$ ) // or equiv.  $\vec{\alpha}$ 
for  $i = 1 : m$ 
   $z^{(i)} = \vec{w}_{svm}^T \vec{\phi}(\vec{x}^{(i)}) + b_{svm}$ 
   $[\theta_1, \theta_0] \leftarrow$  LogReg( $\{z^{(i)}, y^{(i)}\}_{i=1}^m$ )
return  $h(\vec{x}) = g(\theta_1(\vec{w}_{svm}^T \vec{\phi}(\vec{x}) + b_{svm}) + \theta_0)$ 
```

new LogReg problem

$$\min_{\theta_1, \theta_0} \frac{1}{m} \sum_{i=1}^m \log(1 + \exp(-y^{(i)}(\theta_1(\vec{w}_{svm}^T \vec{\phi}(\vec{x}^{(i)}) + b_{svm}) + \theta_0))) \quad (8.145)$$

how to sol. LogReg? GD / SGD or better: because only two vars.
kernel SVM \rightarrow approx. LogReg in \mathcal{Z} space

8.6.4 Kernel Logistic Regression

Key Behind Kernel Trick

$$\vec{\theta}^* = \sum_{i=1}^m \beta_i \vec{z}^{(i)} \quad (8.146)$$

Proof

$$\therefore \vec{\theta}^* \vec{z} = \sum_{i=1}^m \beta_i \vec{z}^{(i)T} \vec{z} = \sum_{i=1}^m \beta_i K(\vec{x}^{(i)}, \vec{x}) \quad (8.147)$$

SVM: α_i from dual sols.

$$\vec{w}^* = \sum_{i=1}^m \alpha_i y^{(i)} \vec{z}^{(i)} \quad (8.148)$$

PLA: $\alpha_i = \#$ mistakes corrections

$$\vec{\theta}^* = \sum_{i=1}^m \alpha_i y^{(i)} \vec{z}^{(i)} \quad (8.149)$$

LogReg by SGD:

$$\vec{\theta}^* = \sum_{i=1}^m \alpha g(-y^{(i)} \vec{\theta}^T \vec{z}^{(i)}) y^{(i)} \vec{z}^{(i)} \quad (8.150)$$

$$= \sum_{i=1}^m \beta_i \vec{z}^{(i)} \quad (8.151)$$

Representer Theorem

for any L_2 -regularized lin. model

$$\min_{\vec{\theta}} \frac{\lambda}{m} \vec{\theta}^T \vec{\theta} + \frac{1}{m} \sum_{i=1}^m \text{err}(y^{(i)}, \vec{\theta}^T \vec{z}^{(i)}) \quad (8.152)$$

gives

$$\vec{\theta}^* = \sum_{i=1}^m \beta_i \vec{z}^{(i)} \quad (8.153)$$

which means it can be kernelized

Proof : def.

$$\mathcal{Z} = \text{span}(\vec{z}^{(1)}, \vec{z}^{(2)}, \dots, \vec{z}^{(m)}) \quad (8.154)$$

by contradiction suppose

$$\vec{\theta}^* = \vec{\theta}_{//} + \vec{\theta}_{\perp}, \vec{\theta}_{//} \in \mathcal{Z}, \vec{\theta}_{\perp} \perp \mathcal{Z}, \vec{\theta}_{\perp} \neq \vec{0} \quad (8.155)$$

$$\vec{\theta}^* \vec{z} = \vec{\theta}_{//} \vec{z} + \vec{\theta}_{\perp} \vec{z} = \vec{\theta}_{//} \vec{z} \quad (8.156)$$

$$\text{err}(y^{(i)}, \vec{\theta}^{*T} \vec{z}^{(i)}) = \text{err}(y^{(i)}, \vec{\theta}_{//}^T \vec{z}^{(i)}) \quad (8.157)$$

$$\vec{\theta}^* \vec{\theta} = \vec{\theta}_{//}^T \vec{\theta}_{//} + 2\vec{\theta}_{//}^T \vec{\theta}_{\perp} + \vec{\theta}_{\perp}^T \vec{\theta}_{\perp} < \vec{\theta}_{//}^T \vec{\theta}_{//} \quad (8.158)$$

$\vec{\theta}_{//}$ is more opt. than $\vec{\theta}^*$, contradiction

□

Kernel Logistic Regression

solving

$$\min_{\vec{\theta}} \frac{\lambda}{m} \vec{\theta}^T \vec{\theta} + \frac{1}{m} \sum_{i=1}^m \log(1 + \exp(-y^{(i)} \vec{\theta}^T \vec{z}^{(i)})) \quad (8.159)$$

gives

wlog., can solve for $\vec{\beta}$ instead of $\vec{\theta}$

$$\begin{aligned} \min_{\vec{\beta}} \quad & \frac{\lambda}{m} \sum_{i=1}^m \sum_{j=1}^m \beta_i \beta_j K(\vec{x}^{(i)}, \vec{x}^{(j)}) + \frac{1}{m} \sum_{i=1}^m \log(1 + \exp(-y^{(i)} \sum_{j=1}^m \beta_j K(\vec{x}^{(i)}, \vec{x}^{(j)}))) \\ = \quad & \frac{\lambda}{m} \vec{\beta}^T K \vec{\beta} + \frac{1}{m} \sum_{i=1}^m \log(1 + \exp(-y^{(i)} K_{i,i} \vec{\beta})) \end{aligned} \quad (8.160)$$

- $\vec{\beta}^T K \vec{\beta}$: a special kernel regularizer
- a lin. model of $\vec{\beta}$ with kernel as transform & kernel regularizer
- a lin. model of $\vec{\theta}$ with embedded-in-kernel transform & L_2 regularizer
- sol. by using GD / SGD / ... for unconstrained opt.
- β_i 's often non-zero

8.7 Support Vector Regression

8.7.1 Kernel Ridge Regression

Kernelized Linear Regression

solving

$$\min_{\vec{\theta}} \frac{\lambda}{m} \vec{\theta}^T \vec{\theta} + \frac{1}{m} \sum_{i=1}^m (y^{(i)} - \vec{\theta}^T \vec{z}^{(i)})^2 \quad (8.162)$$

gives

$$\vec{\theta}^* = \sum_{i=1}^m \beta_i \vec{z}^{(i)} \quad (8.163)$$

wlog., can solve for $\vec{\beta}$ instead of $\vec{\theta}$

$$\begin{aligned} \min_{\vec{\beta}} \quad & \frac{\lambda}{m} \sum_{i=1}^m \sum_{j=1}^m \beta_i \beta_j K(\vec{x}^{(i)}, \vec{x}^{(j)}) + \frac{1}{m} \sum_{i=1}^m (y^{(i)} - \sum_{j=1}^m \beta_j K(\vec{x}^{(i)}, \vec{x}^{(j)}))^2 \\ = \quad & \frac{\lambda}{m} \vec{\beta}^T K \vec{\beta} + \frac{1}{m} \sum_{i=1}^m (y^{(i)} - K_{i,i} \vec{\beta})^2 \end{aligned} \quad (8.164)$$

$$= \frac{\lambda}{m} \vec{\beta}^T K \vec{\beta} + \frac{1}{m} \|y - K \vec{\beta}\|^2 \quad (8.165)$$

$$= \frac{\lambda}{m} \vec{\beta}^T K \vec{\beta} + \frac{1}{m} (\vec{\beta}^T K^T K \vec{\beta} - 2 \vec{\beta}^T K \vec{\beta} + \vec{y}^T \vec{y}) \quad (8.166)$$

$$= \frac{\lambda}{m} \vec{\beta}^T K \vec{\beta} + \frac{1}{m} (\vec{\beta}^T K^T K \vec{\beta} - 2 \vec{\beta}^T K \vec{\beta} + \vec{y}^T \vec{y}) \quad (8.167)$$

Least-Squares SVM
(LSSVM)

$$= J(\vec{\beta}) \quad (8.168)$$

$$\nabla_{\vec{\beta}} J(\vec{\beta}) = \frac{2\lambda}{m} K \vec{\beta} + \frac{1}{m} (2K^T K \vec{\beta} - 2K^T \vec{y}) \quad (8.169)$$

$$= \frac{2}{m} (\lambda K \vec{\beta} + K^T K \vec{\beta} - K^T \vec{y}) \quad (8.170)$$

$$= \frac{2}{m} (\lambda K^T I \vec{\beta} + K^T K \vec{\beta} - K^T \vec{y}) \quad (8.171)$$

$$= \frac{2K^T}{m} (\lambda I \vec{\beta} + K \vec{\beta} - \vec{y}) \quad (8.172)$$

$$= \frac{2K^T}{m} ((\lambda I + K) \vec{\beta} - \vec{y}) \quad (8.173)$$

$$\stackrel{\text{set}}{=} \vec{0} \quad (8.174)$$

$$\vec{\beta} = (\lambda I + K)^{-1} \vec{y} \quad (8.175)$$

- because K is psd., $\lambda > 0$, $(.)^{-1}$ always exists
- time complexity: $O(m^3)$ with simple dense mat. inversion
- can now do non-lin. regression easily

Linear vs. Kernel Ridge Regression

lin. ridge regression

$$\vec{\theta} = (\lambda I + X^T X)^{-1} X^T \vec{y} \quad (8.176)$$

- more restricted
- training: $O(n^3 + n^2 m)$
- prediction: $O(n)$
- efficient when $m \gg n$

kernel ridge regression

$$\vec{\beta} = (\lambda I + K)^{-1} \vec{y} \quad (8.177)$$

- more flexible with K
- training: $O(m^3)$
- prediction: $O(m)$
- hard for big data

trade-off betw. efficiency and flexibility

8.7.2 Support Vector Regression Primal

Soft-Margin SVM vs. Least-Squares SVM

Least-Squares SVM (LSSVM) = kernel ridge regression for classification

- similar boundary, many more SVs, see Fig. 8.13
- slower prediction, dense $\vec{\beta}$

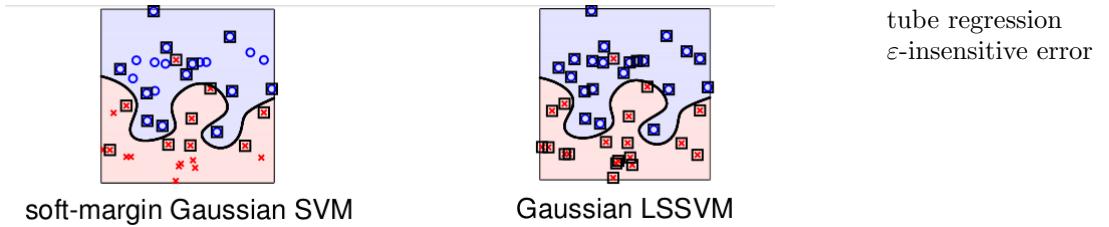


Figure 8.13: Soft-Margin SVM vs. Least-Squares SVM

Tube Regression

idea: want $\vec{\beta}$ like std. SVM

tube regression, see Fig. 8.14

- within a tube $|y - s| \leq \varepsilon$: no error
- outside a tube $|y - s| > \varepsilon$: error by distance to tube

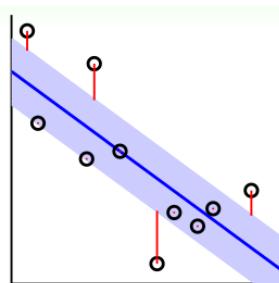


Figure 8.14: Tube Regression

ε -insensitive error with $\varepsilon > 0$

$$\text{err}(y, s) = \max(0, |y - s| - \varepsilon) \quad (8.178)$$

Tube vs. Squared Regression

$$\text{err}_{sq}(y, s) = (y - s)^2 \quad (8.179)$$

tube \approx squared when $|s - y|$ small & less affected by outliers, see Fig. 8.15

L_2 -Regularized Tube Regression

solving

$$\min_{\vec{\theta}} \frac{\lambda}{m} \vec{\theta}^T \vec{\theta} + \frac{1}{m} \sum_{i=1}^m \max(0, |y - \vec{\theta}^T \vec{z}^{(i)}| - \varepsilon) \quad (8.180)$$

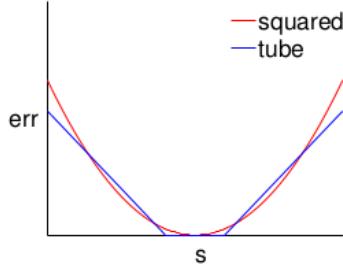


Figure 8.15: Tube vs. Squared Regression

gives

$$\vec{\theta}^* = \sum_{i=1}^m \beta_i \vec{z}^{(i)} \quad (8.181)$$

regularized tube regression

- unconstrained, but max. not differentiable
 - representer to kernelize, but no obvious sparsity
 - std. SVM
 - not differentiable, but QP
 - dual to kernelize, KKT condition \rightarrow sparsity
- will mimic std. SVM derivation

$$\min_{\vec{w}, b} \frac{1}{2} \vec{w}^T \vec{w} + C \sum_{i=1}^m \max(0, |y^{(i)} - \vec{w}^T \vec{z}^{(i)} - b| - \varepsilon) \quad (8.182)$$

- C : trade-off of regularization and tube violation
- ε : vertical tube width: one more para. to choose

Standard Support Vector Machine Primal

std. SVM

$$\min_{\vec{w}, b, \vec{\xi}} \frac{1}{2} \vec{w}^T \vec{w} + C \sum_{i=1}^m \xi_i \quad (8.183)$$

$$\text{st. } |\vec{w}^T \vec{z}^{(i)} + b - y^{(i)}| \leq \varepsilon + \xi_i, \forall i \quad (8.184)$$

$$\xi_i \geq 0, \forall i \quad (8.185)$$

making SVR lin. by consider upper tube violations ξ_i^u and lower violations ξ_i^l

this is QP with $\tilde{n} + 1 + 2m$ vars., $2m + 2m$ constraints

Algorithm 39 Primal SVR

$$\min_{\vec{w}, b, \xi} \frac{1}{2} \vec{w}^T \vec{w} + C \sum_{i=1}^m (\xi_i^u + \xi_i^l) \quad (8.186)$$

$$\text{st. } -\varepsilon - \xi_i^l \leq y^{(i)} - \vec{w}^T \vec{z}^{(i)} - b \leq \varepsilon + \xi_i^u, \forall i \quad (8.187)$$

$$\xi_i^u \geq 0, \forall i \quad (8.188)$$

$$\xi_i^l \geq 0, \forall i \quad (8.189)$$

8.7.3 Support Vector Regression Dual

Lagrange Multipliers α_i^u and α_i^l

$$\begin{aligned} \alpha_i^u &\text{ for } y^{(i)} - \vec{w}^T \vec{z}^{(i)} - b \leq \varepsilon + \xi_i^u \\ \alpha_i^l &\text{ for } -\varepsilon - \xi_i^l \leq y^{(i)} - \vec{w}^T \vec{z}^{(i)} - b \end{aligned}$$

$$\nabla_{\vec{w}} L = \vec{0} \quad (8.190)$$

$$\therefore \vec{w} = \sum_{i=1}^m (\alpha_i^u - \alpha_i^l) \vec{z}^{(i)} \quad (8.191)$$

$$= \sum_{i=1}^m \beta_i \vec{z}^{(i)} \quad (8.192)$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^m (\alpha_i^u - \alpha_i^l) = 0 \quad (8.193)$$

SVM Dual and SVR Dual

SVM primal

$$\min_{\vec{w}, b, \xi} \frac{1}{2} \vec{w}^T \vec{w} + C \sum_{i=1}^m \xi_i \quad (8.194)$$

$$\text{st. } y^{(i)}(\vec{w}^T \vec{z}^{(i)} + b) \geq 1 - \xi_i, \forall i \in \{1, 2, \dots, m\} \quad (8.195)$$

$$\xi_i \geq 0, \forall i \in \{1, 2, \dots, m\} \quad (8.196)$$

SVM dual

$$\min_{\vec{\alpha}} \frac{1}{2} \vec{\alpha}^T Q \vec{\alpha} - \vec{1}^T \vec{\alpha} \quad (8.197)$$

$$\text{st. } \vec{y}^T \vec{\alpha} = 0 \quad (8.198)$$

$$0 \leq \alpha_i \leq C, \forall i \in \{1, 2, \dots, m\} \quad (8.199)$$

SVR primal

$$\min_{\vec{w}, b, \xi} \frac{1}{2} \vec{w}^T \vec{w} + C \sum_{i=1}^m (\xi_i^u + \xi_i^l) \quad (8.200)$$

$$\text{st. } -\varepsilon - \xi_i^l \leq y^{(i)} - \vec{w}^T \vec{z}^{(i)} - b \leq \varepsilon + \xi_i^u, \forall i \quad (8.201)$$

$$\xi_i^u \geq 0, \forall i \quad (8.202)$$

$$\xi_i^l \geq 0, \forall i \quad (8.203)$$

Algorithm 40 Dual SVR

$$\min_{\vec{\alpha}^l, \vec{\alpha}^u} \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m (\alpha_i^u - \alpha_i^l)(\alpha_j^u - \alpha_j^l) K(\vec{x}^{(i)}, \vec{x}^{(j)}) + \quad (8.204)$$

$$\sum_{i=1}^m ((\varepsilon - y^{(i)})\alpha_i^l + \varepsilon + y^{(i)})\alpha_i^u \quad (8.205)$$

$$\text{st. } \sum_{i=1}^m (\alpha_i^u - \alpha_i^l) = 0, \forall i \quad (8.206)$$

$$0 \leq \alpha_i^l \leq C, \forall i \quad (8.207)$$

$$0 \leq \alpha_i^u \leq C, \forall i \quad (8.208)$$

complimentary slackness

$$\alpha_i^u(y^{(i)} - \vec{w}^T \vec{z}^{(i)} - b - \varepsilon - \xi_i^u) = 0 \quad (8.209)$$

$$\alpha_i^l(y^{(i)} - \vec{w}^T \vec{z}^{(i)} - b - \varepsilon - \xi_i^l) = 0 \quad (8.210)$$

strictly within the tube

$$|\vec{w}^T \vec{z}^{(i)} + b - y^{(i)}| < \varepsilon \quad (8.211)$$

$$\therefore \xi_i^u = 0 \wedge \xi_i^l = 0 \quad (8.212)$$

$$\therefore y^{(i)} - \vec{w}^T \vec{z}^{(i)} - b - \varepsilon - \xi_i^u \neq 0 \wedge y^{(i)} - \vec{w}^T \vec{z}^{(i)} - b - \varepsilon - \xi_i^l \neq 0 \quad (8.213)$$

$$\therefore \alpha_i^u = 0 \wedge \alpha_i^l = 0 \quad (8.214)$$

$$\therefore \beta_i = 0 \quad (8.215)$$

SVs $\beta_i \neq 0$: on or outside tube

SVR: allows sparse $\vec{\beta}$

8.7.4 Summary of Kernel Models

Map of Linear Models

first row: less used due to worse perf., see Fig. 8.16

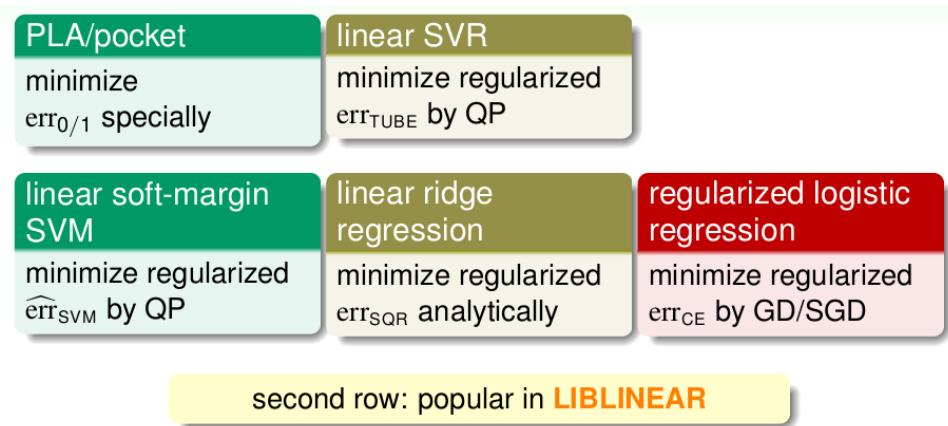


Figure 8.16: Maps of Linear Models

Map of Linear / Kernel Models

third row: less used due to dense $\vec{\beta}$, see Fig. 8.17
 kernel models are powerful extension of lin. models

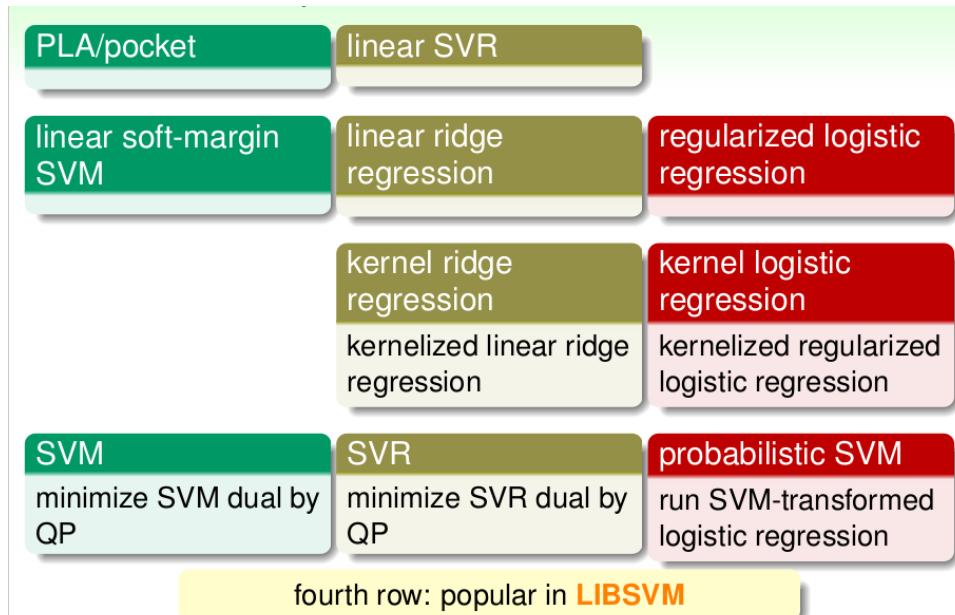


Figure 8.17: Map of Linear / Kernel Models

Decide Which One to Use

if $n \geq m$, eg., $n = 10^4, m = 10 \dots 10^3$

- use logistic regression
- or SVM with linear kernel
if n is small, m is intermediate, eg., $n = 1\dots 10^3, m = 10\dots 10^4$
- use SVM with Gaussian kernel
if n is small, m is large, eg., $n = 1\dots 10^3, m = 5 \times 10^4$
- create / add more features, then use logistic regression or SVM with linear kernel
neuron network likely to work well for most of these settings, but may be slower to train

Chapter 9

Combining Predictive Features: Aggregation Models

9.1 Blending and Bagging

9.1.1 Motivation of Aggregation

an Aggregation Story

your T friends h_1, \dots, h_T predicts whether stock will go up, you can

- select the most trust-worthy friend from their usual performance (validation!)

$$h(\vec{x}) = h_{t^*}(\vec{x}), t^* = \arg \min_t J_{val}(h_t^-) \quad (9.1)$$

- mix the predictions from all your friends uniformly (vote!)

$$h(\vec{x}) = \text{sign}\left(\sum_{t=1}^T h_t(\vec{x})\right) \quad (9.2)$$

- mix the predictions from all your friends non-uniformly (vote, but give some one more ballots!)

$$h(\vec{x}) = \text{sign}\left(\sum_{t=1}^T \alpha_t h_t(\vec{x})\right), \alpha_t \geq 0 \quad (9.3)$$

include select: $\alpha_t = 1\{J_{val}(h_t^-) \text{ is the smallest}\}$

include uniformly: $\alpha_t = 1$

- combine the predictions conditionally

$$h(\vec{x}) = \text{sign}\left(\sum_{t=1}^T c_t(\vec{x}) h_t(\vec{x})\right), c_t(\vec{x}) \geq 0 \quad (9.4)$$

include non-uniformly: $c_t(\vec{x}) = \alpha_t$

Recall: Selection by Validation

simple and popular

need one good h_t^- to guarantee small J_{val} (and small J_{out})

Why Might Aggregation Work?

idea: selection rely on one good hypothesis, can we do better with many (possibly weaker) hypotheses?

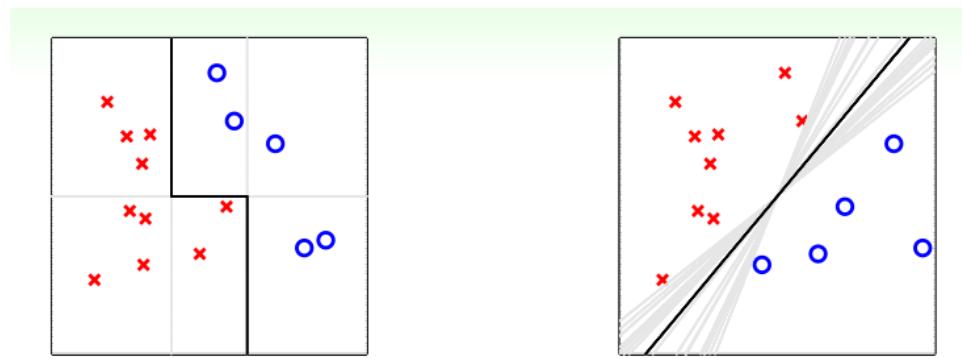


Figure 9.1: Why Might Aggregation Work?

cure underfitting: mix different weak hypotheses uniformly, see the left figure of Fig. 9.1

- $h(\vec{x})$ strong
- aggregation is like feature transform

cure overfitting: mix different random-PLA hypotheses uniformly, see the right figure of Fig. 9.1

- $h(\vec{x})$ moderate
- aggregation is like regularization

9.1.2 Uniform Blending

Uniform Blending (Voting) for Classification

$$h(\vec{x}) = \text{sign}\left(\sum_{t=1}^T h_t(\vec{x})\right) \quad (9.5)$$

- if h_t 's are same: as good as one single h_t
- if h_t 's are very different: majority can correct minority

similar results with uniform voting for multiclass for

$$h(\vec{x}) = \arg \max_{1 \leq k \leq K} \sum_{t=1}^T 1\{h_t(\vec{x}) = k\} \quad (9.6)$$

Uniform Blending for Regression

$$h(\vec{x}) = \frac{1}{T} \sum_{t=1}^T h_t(\vec{x}) \quad (9.7)$$

- if h_t 's are same: as good as one single h_t
- if h_t 's are very different: avg. could be more accurate than individual

Theoretical Analysis of Uniform Blending

$$\frac{1}{T} \sum_{t=1}^T (h_t(\vec{x}) - f(\vec{x}))^2 = \frac{1}{T} \sum_{t=1}^T (h_t(\vec{x})^2 - 2h_t(\vec{x})f(\vec{x}) + f(\vec{x})^2) \quad (9.8)$$

$$= \frac{1}{T} \sum_{t=1}^T h_t(\vec{x})^2 - 2f(\vec{x}) \frac{1}{T} \sum_{t=1}^T h_t(\vec{x}) + f(\vec{x})^2 \quad (9.9)$$

$$= \frac{1}{T} \sum_{t=1}^T h_t(\vec{x})^2 - 2f(\vec{x})h(\vec{x}) + f(\vec{x})^2 \quad (9.10)$$

$$= \frac{1}{T} \sum_{t=1}^T h_t(\vec{x})^2 - h(\vec{x})^2 + (h(\vec{x}) - f(\vec{x}))^2 \quad (9.11)$$

$$= \frac{1}{T} \sum_{t=1}^T h_t(\vec{x})^2 - 2h(\vec{x})h(\vec{x}) + h(\vec{x})^2 + (h(\vec{x}) - f(\vec{x}))^2 \quad (9.12)$$

$$= \frac{1}{T} \sum_{t=1}^T h_t(\vec{x})^2 - 2 \frac{1}{T} \sum_{t=1}^T h_t(\vec{x})h(\vec{x}) + \frac{1}{T} \sum_{t=1}^T h(\vec{x})^2 + (h(\vec{x}) - f(\vec{x}))^2 \quad (9.13)$$

$$= \frac{1}{T} \sum_{t=1}^T (h_t(\vec{x}) - h(\vec{x}))^2 + (h(\vec{x}) - f(\vec{x}))^2 \quad (9.14)$$

$$\frac{1}{T} \sum_{t=1}^T J_{out}(h_t(\vec{x})) = \frac{1}{T} \sum_{t=1}^T E[(h_t(\vec{x}) - h(\vec{x}))^2] + J_{out}(h(\vec{x})) \quad (9.15)$$

$$\geq J_{out}(h(\vec{x})) \quad (9.16)$$

Some Special h_t

consider a virtual iterative process
for $t = 1 : T$

get size m data D_t iid. from P
 get h_t by $A(D_t)$

$$h(\vec{x}) = \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=1}^T h_t(\vec{x}) = \mathbb{E} D[A(D)] \quad (9.17)$$

$$\frac{1}{T} \sum_{t=1}^T J_{out}(h_t(\vec{x})) = \frac{1}{T} \sum_{t=1}^T \mathbb{E} [(h_t(\vec{x}) - h(\vec{x}))^2] + J_{out}(h(\vec{x})) \quad (9.18)$$

expected performance of A
 = expected deviation to consensus + performance of consensus
 = variance + bias
 uniform blending: reduces variance for more stable performance

9.1.3 Linear and Any Blending

Linear Blending

$$h(\vec{x}) = \text{sign}(\sum_{t=1}^T \alpha_t h_t(\vec{x})), \alpha_t \geq 0 \quad (9.19)$$

need to compute α_t
 linear blending for regression

$$\min_{\vec{\alpha} > 0} \frac{1}{m} \sum_{i=1}^m (y^{(i)} - \sum_{t=1}^T \alpha_t h_t(\vec{x}))^2 \quad (9.20)$$

cf. LinReg + transformation

$$\min_{\vec{\theta}} \frac{1}{m} \sum_{i=1}^m (y^{(i)} - \sum_{j=1}^{\tilde{n}} \theta_j \phi(\vec{x})_j)^2 \quad (9.21)$$

linear blending = LinModel + hypotheses as transform + constraints

Constraint on α_t

if $\alpha_t < 0 \rightarrow \alpha_t h_t(\vec{x}) = |\alpha_t|(-h_t(\vec{x}))$
 negative α_t for $h_t \iff$ positive $|\alpha_t|$ for $-h_t(\vec{x})$
 in practice, often linear blending = LinModel + hypotheses as transform

Linear Blending vs. Selection

in practice, often $h_1 \in \mathcal{H}_1, h_2 \in \mathcal{H}_2, \dots, h_T \in \mathcal{H}_T$ by min. J_{in}
 selection by minimum J_{in} : best of best, paying $VC(\bigcup_{t=1}^T \mathcal{H}_t)$
 linear blending includes selection as special case: $\alpha_t = 1\{J_{val}(h_t^-) \text{ is the smallest}\}$
 complexity price of linear blending with J_{in} (aggregation of best): $\geq VC(\bigcup_{t=1}^T \mathcal{H}_t)$
 like selection, blending practically done with J_{val} instead of $J_{in} + h_t^-$
 from min. J_{train}

Any Blending

Algorithm 41 Linear Blending and Any Blending (Stacking)

input: $h_1^-, h_2^-, \dots, h_T^-$ from D_{train}

for $(\vec{x}^{(i)}, y^{(i)})$ in D_{val}

$$\vec{z}^{(i)} = \vec{\phi}^-(\vec{x}^{(i)}) = \begin{bmatrix} h_1^-(\vec{x}^{(i)}) \\ h_2^-(\vec{x}^{(i)}) \\ \vdots \\ h_T^-(\vec{x}^{(i)}) \end{bmatrix}$$

 // in linear blending
 $\vec{\alpha} \leftarrow \text{LinearModel}(\{\vec{z}^{(i)}, y^{(i)}\})$
return $h(\vec{x}) = \text{LinearHypothesis}_{\vec{\alpha}}(\vec{\phi}(\vec{x}))$

// any blending
 compute $\tilde{h} \leftarrow \text{AnyModel}(\{\vec{z}^{(i)}, y^{(i)}\})$
return $h(\vec{x}) = \tilde{h}(\vec{\phi}(\vec{x}))$

where

$$\vec{\phi}(\vec{x}^{(i)}) = \begin{bmatrix} h_1(\vec{x}) \\ h_2(\vec{x}) \\ \vdots \\ h_T(\vec{x}) \end{bmatrix} \quad (9.22)$$

any blending:

- powerful, achieves conditional blending
- but danger of overfitting, as always, and computational burden

9.1.4 Bagging (Bootstrap Aggregation)

What We Have Done

see Tab. 9.1

bagging

Table 9.1: What We Have Done

aggregation type	blending	learning
uniform	voting / averaging	?
non-uniform	linear	?
conditional	stacking	?

blending: aggregate after getting h_t learning: aggregate as well as getting h_t – diversity important

- diversity by different models $h_1 \in \mathcal{H}_1, h_2 \in \mathcal{H}_2, \dots, h_T \in \mathcal{H}_T$
- diversity by different parameters: GD with $\alpha = 0.001, 0.01, \dots, 10$
- diversity by algorithmic randomness: random PLA with different random seeds
- diversity by data randomness: within-cross-validation hypotheses h_t^-

Revisit of Bias-Variance

$$h(\vec{x}) = \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=1}^T h_t(\vec{x}) = \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=1}^T A(D_t \stackrel{\text{iid}}{\sim} P) = \mathbb{E} D[A(D)] \quad (9.23)$$

$$\frac{1}{T} \sum_{t=1}^T J_{out}(h_t(\vec{x})) = \frac{1}{T} \sum_{t=1}^T \mathbb{E} [(h_t(\vec{x}) - h(\vec{x}))^2] + J_{out}(h(\vec{x})) \quad (9.24)$$

expected performance of A

= expected deviation to consensus + performance of consensus

= variance + bias

consensus more stable than direct $A(D)$, but comes from many more D_t than the D on hand

want: approximate consensus by

- finite (large) T
- approximate $h_t = A(D_t \stackrel{\text{iid}}{\sim} P)$ using only D

Bootstrap Aggregation

bootstrap aggregation (**bagging**)

- a simple meta algorithm on top of base algorithm A
- idea: if the variance of a prediction is σ^2 , then the variance of the average of T iid. predictions is $\frac{\sigma^2}{T}$
- proper non-linear boundary after aggregating binary classifiers
- bagging works reasonably well if base algorithm sensitive to data randomness

Algorithm 42 Bootstrapp Aggregation

bootstrapping : a statistical tool that re-samples from D to ‘simulate’ D_t by re-sample m' (can be same with orig. m or can be arbitrary) examples from D uniformly with replacement

```

for  $t = 1 : T$ 
    get size  $m'$  data  $\hat{D}_t$  from  $D$  from bootstrapping
    get  $h_t$  by  $A(\hat{D}_t)$ 
return  $h(\vec{x}) = \frac{1}{T} \sum_{t=1}^T h_t(\vec{x})$ 

```

9.2 Adaptive Boosting

9.2.1 Motivation of Boosting

Apple Recognition Problem

Problem: describe an apple by looking at pictures of apples and non-apples

- A: apples are circular
class: apples are circular
- B: apples are red
class: apples are somewhat circular and somewhat red
- C: apples could also be green
class: apples are somewhat circular and

Motivation

- students: simple hypotheses h_t
- class: sophisticated hypothesis h

teacher: a tactic learning algorithm that directs the students to focus on key examples (aim to reduce bias)

9.2.2 Diversity by Re-weighting

Bootstrapping as Re-weighting Process

- suppose $D = \{(\vec{x}^{(1)}, y^{(1)}), (\vec{x}^{(2)}, y^{(2)}), (\vec{x}^{(3)}, y^{(3)}), (\vec{x}^{(4)}, y^{(4)})\}$
- bootstrap: $\hat{D} = \{(\vec{x}^{(1)}, y^{(1)}), (\vec{x}^{(1)}, y^{(1)}), (\vec{x}^{(3)}, y^{(3)}), (\vec{x}^{(4)}, y^{(4)})\}$
- each diverse h_t in bagging: by minimizing bootstrap-weighted error

$$J_{in}^u(h) = \frac{1}{m} \sum_{(\vec{x}^{(i)}, y^{(i)}) \in \hat{D}} 1\{h(\vec{x}^{(i)}) \neq y^{(i)}\} = \frac{1}{m} \sum_{i=1}^m u^{(i)} 1\{h(\vec{x}^{(i)}) \neq y^{(i)}\} \quad (9.25)$$

with

$$u^{(1)} = 2, u^{(2)} = 1, u^{(3)} = 0, u^{(4)} = 1 \quad (9.26)$$

$$\sum_{i=1}^m u^{(i)} = m \quad (9.27)$$

Weighted Base Algorithm

$$J_{in}^u(h) = \frac{1}{m} \sum_{i=1}^m u^{(i)} \text{err}(\vec{x}^{(i)}, y^{(i)}) \quad (9.28)$$

how to sol.? extension of class-weighted learning

SVM

- $J_{in}^u(h) \propto C \sum_{i=1}^m u^{(i)} \hat{\text{err}}_{svm}$ by dual QP
- adjusted upper bound $0 \leq \alpha_i \leq Cu^{(i)}$

LogReg

- $J_{in}^u(h) \propto \sum_{i=1}^m u^{(i)} \text{err}_{ce}$ by SGD
- sample $(\vec{x}^{(i)}, y^{(i)})$ with probability proportional to $u^{(i)}$

Re-weighting for More Diverse Hypothesis

‘improving’ bagging for binary classification: how to re-weight for more diverse hypotheses?

$$h_t = \arg \min_{h \in \mathcal{H}} \sum_{i=1}^m u_t^{(i)} \mathbb{1}\{h(\vec{x}^{(i)}) \neq y^{(i)}\} \quad (9.29)$$

$$h_{t+1} = \arg \min_{h \in \mathcal{H}} \sum_{i=1}^m u_{t+1}^{(i)} \mathbb{1}\{h(\vec{x}^{(i)}) \neq y^{(i)}\} \quad (9.30)$$

if h_t not good for u_{t+1}
 $\rightarrow h_t$ -like hypotheses not returned as h_{t+1}
 $\rightarrow h_{t+1}$ diverse from h_t

‘Optimal’ Re-weighting

idea construct u_{t+1} to make h_t random like

$$= \frac{\sum_{i=1}^m u_{t+1}^{(i)} \mathbb{1}\{h_t(\vec{x}^{(i)}) \neq y^{(i)}\}}{\sum_{i=1}^m u_{t+1}^{(i)} \mathbb{1}\{h_t(\vec{x}^{(i)}) \neq y^{(i)}\} + \sum_{i=1}^m u_{t+1}^{(i)} \mathbb{1}\{h_t(\vec{x}^{(i)}) = y^{(i)}\}} \quad (9.31)$$

$$= \frac{\sum_{i=1}^m u_{t+1}^{(i)} \mathbb{1}\{h_t(\vec{x}^{(i)}) \neq y^{(i)}\}}{\sum_{i=1}^m u_{t+1}^{(i)}} \quad (9.32)$$

$$= \frac{1}{2} \quad (9.33)$$

thus

$$\sum_{i=1}^m u_{t+1}^{(i)} \mathbf{1}\{h_t(\vec{x}^{(i)}) \neq y^{(i)}\} = \sum_{i=1}^m u_t^{(i)} \mathbf{1}\{h_t(\vec{x}^{(i)}) = y^{(i)}\} \quad (9.34)$$

want $\# \text{incorrect} = \# \text{correct}$: one possibility by re-scaling (multiplying) weights

- $\# \text{incorrect}$ with $u_t^{(i)} = 1126$, $\# \text{correct}$ with $u_t^{(i)} = 6211$
- incorrect rate $\varepsilon_t = \frac{1126}{7337}$, correct rate $= \frac{6211}{7337} = 1 - \varepsilon_t$
- incorrect $u_{t+1}^{(i)} = u_t^{(i)} \times (1 - \varepsilon_t)$, correct $u_{t+1}^{(i)} = u_t^{(i)} \times \varepsilon_t$

9.2.3 Adaptive Boosting Algorithm

Scaling Factor

'optimal' re-weighting

$$\varepsilon_t = \frac{\sum_{i=1}^m u_t^{(i)} \mathbf{1}\{h_t(\vec{x}^{(i)}) \neq y^{(i)}\}}{\sum_{i=1}^m u_t^{(i)}} \quad (9.35)$$

$$\# \text{incorrect} \times (1 - \varepsilon_t) = \# \text{correct} \times \varepsilon_t \quad (9.36)$$

$$\iff \# \text{incorrect} \times \frac{1 - \varepsilon_t}{\varepsilon_t} = \# \text{correct} \quad (9.37)$$

$$\iff \# \text{incorrect} \times \sqrt{\frac{1 - \varepsilon_t}{\varepsilon_t}} \sqrt{\frac{1 - \varepsilon_t}{\varepsilon_t}} = \# \text{correct} \quad (9.38)$$

$$\iff \# \text{incorrect} \times \sqrt{\frac{1 - \varepsilon_t}{\varepsilon_t}} = \frac{\# \text{correct}}{\sqrt{\frac{1 - \varepsilon_t}{\varepsilon_t}}} \quad (9.39)$$

$$(9.40)$$

def.

$$\rho_t = \sqrt{\frac{1 - \varepsilon_t}{\varepsilon_t}} \quad (9.41)$$

- $\rho_t \geq 1 \iff \varepsilon_t \leq \frac{1}{2}$
- physical meaning: scale up incorrect; scale down correct
- like what teacher does

Linear Aggregation on the Fly

$$h(\vec{x}) = \text{sign}(\sum_{t=1}^T \alpha_t h_t(\vec{x})) \quad (9.42)$$

uniform of $h_t(\vec{x})$'s? but h_2 very bad for J_{in}
want large α_t for good h_t

- α_t large $\iff \varepsilon_t$ small $\iff \rho_t$ large
- $\rightarrow \alpha_t = \text{monotonic}(\rho_t)$
- choose $\alpha_t = \log \rho_t$
 $\varepsilon_t = \frac{1}{2} \rightarrow \rho_t = 1 \rightarrow \alpha_t = 0$: bad h_t zero weight
 $\varepsilon_t = 0 \rightarrow \rho_t = \infty \rightarrow \alpha_t = \infty$: super h_t superior weight

Adaptive Boosting

= weak base learning algorithm A (student)

+ optimal re-weighting factor ρ_t (teacher)

+ ‘magic’ linear aggregation α_t (class)

Adaptive Boosting (AdaBoost) Algorithm

Algorithm 43 AdaBoost Algorithm

```

 $\vec{u}_1 = [\frac{1}{m} \frac{1}{m} \dots \frac{1}{m}]^T$  // want  $h_1$  best for  $J_{in}$ 
for  $t = 1 : T$ 
   $h_t = A(D, \vec{u}_t)$  //  $A$  tries to min.  $\vec{u}_t$ -weighted 0/1 err.
   $\varepsilon_t = \frac{\sum_{i=1}^m u_t^{(i)} \mathbb{1}\{h_t(\vec{x}^{(i)}) \neq y^{(i)}\}}{\sum_{i=1}^m u_t^{(i)}}$ 
   $\rho_t = \sqrt{\frac{1-\varepsilon_t}{\varepsilon_t}}$ 
   $\alpha_t = \log \rho_t$ 
  for  $i = 1 : m$ 
    if  $y^{(i)} \neq h_t(\vec{x}^{(i)})$  // incorrect sample
       $u_{t+1}^{(i)} = u_t^{(i)} \rho_t$ 
    else // correct sample
       $u_{t+1}^{(i)} = \frac{u_t^{(i)}}{\rho_t}$ 
  return  $h(\vec{x}) = \text{sign}(\sum_{t=1}^T \alpha_t h_t(\vec{x}))$ 

```

Theoretical Guarantee of AdaBoost

from VC bound

$$J_{out}(h) \leq J_{in}(h) + O\left(\sqrt{O(VC(\mathcal{H}) \cdot T \log T) \frac{\log m}{m}}\right) \quad (9.43)$$

- first term can be small
 $J_{in}(h) = 0$ after $T = O(\log m)$ iterations if $\varepsilon_t \leq \varepsilon < \frac{1}{2}, \forall t$
- second term can be small
 $O(VC(\mathcal{H}) \cdot T \log T)$ is the VC dim. of all possible h
it grows slowly with T

boosting view of AdaBoost: if A is weak but always slightly better than random $\varepsilon_t \leq \varepsilon < \frac{1}{2}, \forall t$, then (AdaBoost+ A) can be strong ($J_{in} = 0$ and J_{out} small)

9.2.4 Adaptive Boosting in Action

Decision Stump

want a weak base learning alg. A
a popular choice: decision stump

$$h(\vec{x}) = s \operatorname{sign}(x_j - \theta) \quad (9.44)$$

- three parameters: feature j , threshold θ , direction s
- physical meaning: vertical/horizontal lines when $n = 2$
- efficient to optimize: $O(nm \log m)$ time
- too weak to work by itself

AdaBoost-Stump in Application

AdaBoost-Stump: efficient feature selection and aggregation, see Fig. 9.2

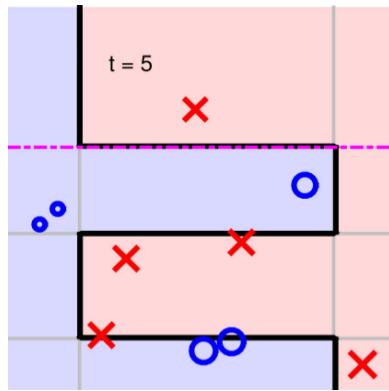


Figure 9.2: AdaBoost-Stump in Application

the world's first 'real-time' face detection program

9.3 Decision Tree

9.3.1 Decision Tree Hypothesis

What We Have Done

see Tab. 9.2

Decision Tree

$$h(\vec{x}) = \sum_{t=1}^T c_t(\vec{x}) h_t(\vec{x}) \quad (9.45)$$

Table 9.2: What We Have Done

aggregation type	blending	learning
uniform	voting / averaging	Bagging
non-uniform	linear	AdaBoost
conditional	stacking	Decision Tree

- base hypothesis $h_t(\vec{x})$: leaf at end of path t , a constant here
- condition $c_t(\vec{x}) = 1\{\text{is } \vec{x} \text{ on path } t\}$
- usually with simple internal nodes

Recursive View of Decision Tree

$$h(\vec{x}) = \sum_{q=1}^Q 1\{b(\vec{x}) = q\} h^{(q)}(\vec{x}) \quad (9.46)$$

- $h(\vec{x})$: full-tree hypothesis
- $b(\vec{x})$: branching criteria
- $h^{(q)}(\vec{x})$: sub-tree hypothesis at the q -th branch

Disclaimers about Decision Tree

usefulness

- human-explainable: widely used in business / medical data analysis
- simple to implement
- efficient in prediction and training

however

- heuristic: mostly little theoretical explanations
- heuristics: heuristics selection confusing to beginners
- arguably no single representative algorithm

9.3.2 Decision Tree Algorithm

A Basic Decision Tree Algorithm

four choices

- number of branches Q
- branching criteria $b(\vec{x})$
- termination criteria
- base hypothesis $h_t(\vec{x})$

Classification and Regression Tree (CART)

four choices

- number of branches $Q = 2$: binary tree
- base hypothesis $h_t(\vec{x}) = J_{in}$ -optimal constant
binary / multiclass classification (0/1 error): majority of $\{y^{(i)}\}$
regression (squared error): average of $\{y^{(i)}\}$
- termination criteria: ‘forced’ terminate when
all $y^{(i)}$ the same: $h_t(\vec{x}) = y^{(i)}$
all $\vec{x}^{(i)}$ the same: no decision stumps
- branching criteria $b(\vec{x})$: use $\{1, 2\}$ -output decision stump by purifying

$$D_q = \{(\vec{x}^{(i)}, y^{(i)}) : b(\vec{x}^{(i)}) = q, \forall i\}, q = \{1, 2\} \quad (9.47)$$

$$b(\vec{x}) = \arg \min_b \sum_{q=1}^2 |D_q| \cdot \text{impurity}(D_q) \quad (9.48)$$

CART: fully-grown tree with constant leaves that come from bi-branching by purifying

Impurity Functions

by J_{in} of optimal constant

- regression error

$$\bar{y} = \frac{1}{m'} \sum_{i=1}^{m'} y^{(i)} \quad (9.49)$$

$$\frac{1}{m'} \sum_{i=1}^{m'} (y^{(i)} - \bar{y})^2 \quad (9.50)$$

- classification error

$$\bar{y} = \text{majority}(\{y^{(i)}\}) \quad (9.51)$$

$$\frac{1}{m'} \sum_{i=1}^{m'} 1\{y^{(i)} \neq \bar{y}\} \quad (9.52)$$

for classification

- classification error

$$1 - \max_{1 \leq k \leq K} \frac{\sum_{i=1}^{m'} 1\{y^{(i)} = k\}}{m'} = \frac{1}{m'} \sum_{i=1}^{m'} 1\{y^{(i)} \neq \bar{y}\} \quad (9.53)$$

- Gini index

$$1 - \sum_{k=1}^K \left(\frac{\sum_{i=1}^{m'} 1\{y^{(i)} = k\}}{m'} \right)^2 \quad (9.54)$$

popular choices: Gini for classification, regression error for regression

pruned decision tree

9.3.3 Decision Tree Heuristics in CART

Basic CART Algorithm

Algorithm 44 Basic CART Algorithm

```

def DecisionTree( $D = \{(\vec{x}^{(i)}, y^{(i)})\}_{i=1}^m$ )
    if cannot branch anymore
        return  $h_t(\vec{x}) = J_{in}\text{-opt. const.}$ 
    else
        // learn branching criteria
         $b(\vec{x}) = \arg \min_b \sum_{q=1}^2 |D_q| \cdot \text{impurity}(D_q)$ 
        // split  $D$  into 2 parts
         $D = \sum_{q=1}^2 D_q = \sum_{q=1}^2 \{(\vec{x}^{(i)}, y^{(i)}) : b(\vec{x}^{(i)}) = q, \forall i\}$ 
         $h^{(1)}(\vec{x}) = \text{DecisionTree}(D_1)$ 
         $h^{(2)}(\vec{x}) = \text{DecisionTree}(D_2)$ 
        return  $h(\vec{x}) = \sum_{q=1}^2 1\{b(\vec{x}) = q\} h^{(q)}(\vec{x})$ 

```

Regularization by Pruning

fully-grown tree

- $J_{in}(h) = 0$ if all $\vec{x}^{(i)}$ different
- overfit (large $J_{out}(h)$) because low-level trees built with small D_q need a regularizer
- say, $\Omega(h) = \text{NumOfLeaves}(h)$
- regularized decision tree, called **pruned decision tree**

$$\arg \min_h J_{in}(h) + \lambda \Omega(h) \quad (9.55)$$

- cannot enumerate all possible h computationally, often consider only h : fully-grown tree

$$h^{(-i)} = \arg \min_{h^{(-i)}} J_{in}(h^{(-i)})$$
 st. $h^{(-i)}$ is one-leaf removed from $h^{(-(i-1))}$
- systematic choice of λ ? validation

Branching on Categorical Features

numerical features

- eg., blood pressure: 130, 98, 115, 147, 120
- branching for numerical

$$b(\vec{x}) = 1\{x_j \leq \theta\} + 1 \quad (9.56)$$

categorical features

- eg., major symptom: fever, pain, tired, sweaty

- branching for categorical

$$b(\vec{x}) = 1\{x_j \in S\} + 1 \quad (9.57)$$

$$S \subset \{1, 2, \dots, K\} \quad (9.58)$$

CART (& general decision trees): handles categorical features easily

Missing Features by Surrogate Branch

suppose

$$b(\vec{x}) = 1\{\text{weight} \leq 50\text{kg}\} + 1 \quad (9.59)$$

if weight missing during prediction

- what would human do?
go get weight
or, use threshold on height instead, because threshold on height \approx threshold on weight
- surrogate branch
maintain surrogate branch $b_1(\vec{x}), b_2(\vec{x}), \dots \approx$ best branch $b(\vec{x})$ during training
allow missing feature for $b(\vec{x})$ during prediction by using surrogate instead during training

CART: handles missing features easily

9.3.4 Decision Tree in Action

Decision Tree in Action

CART: ‘divide-and-conquer’, even more efficient than AdaBoost-Stump, see Fig. 9.3

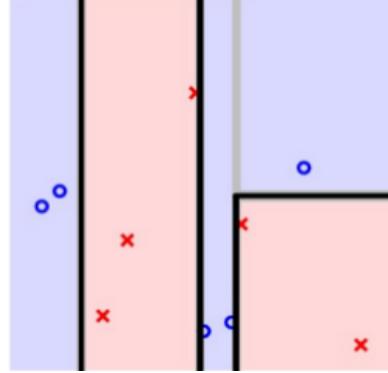


Figure 9.3: Decision Tree in Action

Practical Specialties of CART

almost no other learning model share all such specialties, except for other decision trees

- human-explainable
- multiclass easily
- categorical features easily
- missing features easily
- efficient non-linear training (and testing)

another popular decision tree algorithm: C4.5, with different choices of heuristics

9.4 Random Forest

9.4.1 Random Forest Algorithm

Random Forest (RF)

andom forest (RF) = bagging + fully-grown CART decision tree

Algorithm 45 Random Forest

input : D

for $t = 1 : T$
 $\hat{D}_t = \text{bootstrapping}(D)$ // get size m' data
 $h_t = \text{DecisionTree}(\hat{D}_t)$
return $h(\vec{x}) = \frac{1}{T} \sum_{t=1}^T h_t(\vec{x})$

def DecisionTree($D = \{(\vec{x}^{(i)}, y^{(i)})\}_{i=1}^m$)
if termination
return base h_t
else
 $b(\vec{x}) = \arg \min_b \sum_{q=1}^2 |D_q| \cdot \text{impurity}(D_q)$
 $D = \sum_{q=1}^2 D_q = \sum_{q=1}^2 \{(\vec{x}^{(i)}, y^{(i)}) : b(\vec{x}^{(i)}) = q, \forall i\}$
 $h^{(1)}(\vec{x}) = \text{DecisionTree}(D_1)$
 $h^{(2)}(\vec{x}) = \text{DecisionTree}(D_2)$
return $h(\vec{x}) = \sum_{q=1}^2 1\{b(\vec{x}) = q\} h^{(q)}(\vec{x})$

highly parallel/efficient to learn
inherit pros of CART
eliminate cons of fully-grown tree

Diversifying by Feature Projection

out-of-bag (OOB)

recall: data randomness for diversity in bagging: randomly sample m' examples from D

another possibility for diversity: randomly sample n' features from \vec{x}

- when sampling index $j_1, j_2, \dots, j_{n'}$, $\vec{\phi}(\vec{x}) = [x_{j_1} \ x_{j_2} \ \dots \ x_{j_{n'}}]^T$
- $\mathcal{Z} \subseteq \mathbb{R}^{n'}$: a random subspace $\mathcal{X} \subseteq \mathbb{R}^n$
- often $n' \ll n$, efficient for large n : can be generally applied on other models
- original RF re-sample new subspace for each $b(\vec{x})$ in CART

RF = bagging + random-subspace CART

Diversifying by Feature Expansion

randomly sample n' features from \vec{x} : $\vec{\phi}(\vec{x}) = P\vec{x}$ with row j of P sampled randomly from natural basis

more powerful features for diversity: row j other than natural basis

- projection (combination) with random row \vec{p}_j^T of P : $\phi(\vec{x})_j = \vec{p}_j^T \vec{x}$ (eg., $\phi(\vec{x})_j = x_1 + x_3 + x_6$)
- often consider low-dimensional projection: only n'' non-zero components in \vec{p}_j^T
- includes random subspace (see prev. subsubsect.) as special case: $n'' = 1$ and \vec{p}_j^T from natural basis
- original RF consider n' random low-dimensional projections for each $b(\vec{x})$ in CART

RF = bagging + random-combination CART – randomness everywhere!

9.4.2 Out-Of-Bag Estimate**Bagging Revisited**

see Fig. 9.4, \star : in t -th column: not used for obtaining h_t – called **out-of-bag (OOB)** examples of h_t

	g_1	g_2	g_3	\dots	g_T
(\mathbf{x}_1, y_1)	\tilde{D}_1	\star	\tilde{D}_3		\tilde{D}_T
(\mathbf{x}_2, y_2)	\star	\star	\tilde{D}_3		\tilde{D}_T
(\mathbf{x}_3, y_3)	\star	\tilde{D}_2	\star		\tilde{D}_T
\dots					
(\mathbf{x}_N, y_N)	\tilde{D}_1	\tilde{D}_2	\star		\star

Figure 9.4: Bagging Revisited

Number of OOB Examples

OOB: not sampled after one bootstrap
if $m' = m$, m large, probability for $(\bar{x}^{(i)}, y^{(i)})$ to be OOB for h_t

$$(1 - \frac{1}{m})^m \approx \frac{1}{e} = 0.37 \quad (9.60)$$

OOB size per h_t : $\frac{m}{e} = 0.37m$

OOB vs. Validation

see Fig. 9.5

OOB						Validation					
	g_1	g_2	g_3	\dots	g_T		g_1^-	g_2^-	\dots	g_M^-	
(\mathbf{x}_1, y_1)	$\tilde{\mathcal{D}}_1$	★	$\tilde{\mathcal{D}}_3$		$\tilde{\mathcal{D}}_T$		$\mathcal{D}_{\text{train}}$	$\mathcal{D}_{\text{train}}$		$\mathcal{D}_{\text{train}}$	
(\mathbf{x}_2, y_2)	★	★	$\tilde{\mathcal{D}}_3$		$\tilde{\mathcal{D}}_T$		\mathcal{D}_{val}	\mathcal{D}_{val}		\mathcal{D}_{val}	
(\mathbf{x}_3, y_3)	★	$\tilde{\mathcal{D}}_2$	★		$\tilde{\mathcal{D}}_T$		\mathcal{D}_{val}	\mathcal{D}_{val}		\mathcal{D}_{val}	
...											
(\mathbf{x}_N, y_N)	$\tilde{\mathcal{D}}_1$	★	★				$\mathcal{D}_{\text{train}}$	$\mathcal{D}_{\text{train}}$		$\mathcal{D}_{\text{train}}$	

Figure 9.5: OOB vs. Validation

- ★ like D_{val} : ‘enough’ random examples unused during training
- use ★ to validate h_t ? easy, but rarely needed
- use ★ to validate h ?

$$J_{\text{oob}}(h) = \frac{1}{m} \sum_{i=1}^m \text{err}(h^{(i)}(\bar{x}^{(i)}), y^{(i)}) \quad (9.61)$$

with $h^{(i)}$ contains only trees that $\bar{x}^{(i)}$ is OOB of, eg., see Fig. 9.5,
 $h^{(m)} = \text{average}(h_2, h_3, h_T)$

Model Selection by OOB Error

previously: by best J_{val}

$$k^* = \arg \min_k J_{\text{val}}(A_k(D_{\text{train}})) \quad (9.62)$$

RF: by Best J_{oob}

$$k^* = \arg \min_k J_{\text{oob}}(A_k(D)) \quad (9.63)$$

- use J_{oob} for self-validation of RF parameters such n''
- no re-training needed
- J_{oob} often accurate in practice

9.4.3 Feature Selection

Feature Selection

remove redundant features and irrelevant features

advantages

- efficiency: simpler hypothesis and shorter prediction time
- generalization: ‘feature noise’ removed
- interpretability

disadvantages

- combinatorial optimization problem
- may overfit while selection
- mis-interpretability

decision tree: a rare model with built-in feature selection

Feature Selection by Importance

idea: select features of top- n' importance
importance by linear model

$$s = \vec{\theta}^T \vec{x} = \sum_{j=1}^n \theta_j x_j \quad (9.64)$$

- intuitive estimate: $\text{importance}(j) = |\theta_j|$ with some ‘good’ $\vec{\theta}$
- getting ‘good’ $\vec{\theta}$: learned from data
- non-linear models? often much harder

Feature Importance by Permutation Test

idea: random test –if feature j needed, ‘random’ values of $x_j^{(i)}$ degrades performance

which random values?

- uniform, Gaussian, ..., $P(\vec{x}^{(i)})$ changed
- bootstrap, permutation of $\{x_j^{(i)}\}_{i=1}^m$ to get \tilde{D} , $P(\vec{x}^{(i)})$ approximately remained

permutation test: a general statistical tool for arbitrary non-linear models like RF

$$\text{importance}(j) = \text{performance}(D) - \text{performance}(\tilde{D}) \quad (9.65)$$

- $\text{performance}(\tilde{D})$: needs re-training and validation in general
- ‘escaping’ validation? OOB in RF
- original RF solution:

$$\text{importance}(j) = J_{oob}(h \text{ with } D) - J_{oob}(h \text{ with } \tilde{D}) \quad (9.66)$$

where $J_{oob}(h \text{ with } \tilde{D})$ comes from replacing each request of $x_j^{(i)}$ by a permuted OOB value

RF feature selection via permutation + OOB: often efficient and promising in practice

9.4.4 Random Forest in Action

Random Forest in Action

a comparation of CART and random forest can be found in Fig. 9.6, 9.7

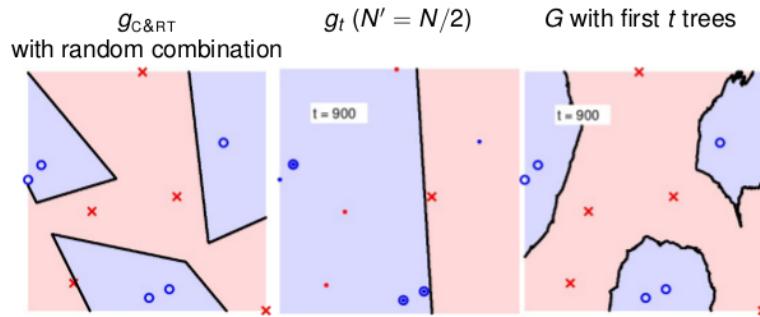


Figure 9.6: A comparation of CART and random forest on simple data.

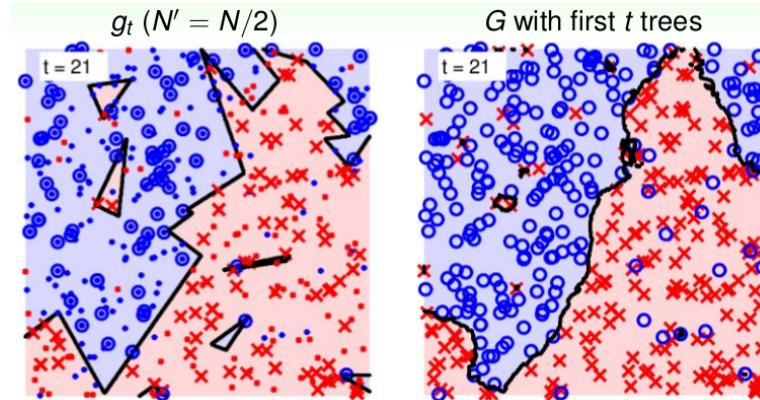


Figure 9.7: Random forest on noisy data.

random forest:

- smooth and large-margin-like boundary with many trees
- easy yet robust nonlinear model
- noise corrected by voting

How Many Trees Needed?

almost every theory: the more, the better
 assuming good $h = \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=1}^T h_t$
 cons of RF: may need lots of trees if the whole random process too unstable —should double-check stability of G to ensure enough trees

9.5 Gradient Boosted Decision Tree

9.5.1 Adaptive Boosted Decision Tree

From Random Forest to AdaBoost-DTree

Algorithm 46 AdaBoost-DTree

```

for  $t = 1 : T$ 
  reweight data by  $\vec{u}_t$ 
   $h_t \leftarrow \text{DTree}(D, \vec{u}_t)$ 
  calculate vote  $\alpha_t$  of  $h_t$ 
return  $h = \text{LinearHypo}(\{h_t, \alpha_t\})$ 

```

Weighted Decision Tree Algorithm

need weighted DTree(D, \vec{u}_t)

$$J_{in}^u(h) = \sum_{i=1}^m u^{(i)} \text{err}(h(\vec{x}^{(i)}), y^{(i)}) \quad (9.67)$$

if using existing algorithm as black box (no modifications), to get J_{in}^u approximately optimized

‘Weighted’ Algorithm in Bagging

- weights \vec{u} expressed by bootstrap-sampled copies —request size- m' data \hat{D} by bootstrapping with D
- weights \vec{u} expressed by sampling proportional to $u^{(i)}$ —request size- m' data \hat{D} by sampling $\propto \vec{u}$ on D

AdaBoost-DTree: often via AdaBoost + sampling $\propto \vec{u}_t$ + DTree(\hat{D}) without modifying DTree

Weak Decision Tree Algorithm

in AdaBoost

$$\alpha_t = \log \rho_t = \log \sqrt{\frac{1 - \varepsilon_t}{\varepsilon_t}} \quad (9.68)$$

if fully grown tree trained on all $\vec{x}^{(i)}$
 $\rightarrow J_{in}(h_t) = 0$ if all $\vec{x}^{(i)}$ diff.

$\rightarrow J_{in}^u(h_t) = 0$
 $\rightarrow \varepsilon_t = 0$
 $\rightarrow \alpha_t = \infty$ (autocracy!)
 need: pruned tree trained on some $\vec{x}^{(i)}$ to be weak

- pruned: usual pruning, or just limiting tree height
- some: sampling $\propto \vec{u}_t$

 AdaBoost-DTree: often via AdaBoost + sampling $\propto \vec{u}_t$ + pruned DTree(\hat{D})

AdaBoost with Extremely-Pruned Tree

what if DTree with height ≤ 1 (extremely pruned)?
branching criteria

$$b(\vec{x}) = b(\vec{x}) = \arg \min_b \sum_{q=1}^2 |D_q| \cdot \text{impurity}(D_q) \quad (9.69)$$

if impurity = binary classification error, \rightarrow just a decision stump
AdaBoost-Stump = special case of AdaBoost-DTree

9.5.2 Optimization View of AdaBoost

Example Weights of AdaBoost

$$u_{t+1}^{(i)} = \begin{cases} u_t^{(i)} \rho_t & \text{if incorrect: } y^{(i)} \neq h_t(\vec{x}^{(i)}) \\ \frac{u_t^{(i)}}{\rho_t} & \text{if correct: } y^{(i)} = h_t(\vec{x}^{(i)}) \end{cases} \quad (9.70)$$

$$= u_t^{(i)} \rho_t^{-y^{(i)} h_t(\vec{x}^{(i)})} \quad (9.71)$$

$$= u_t^{(i)} \exp \log \rho_t^{-y^{(i)} h_t(\vec{x}^{(i)})} \quad (9.72)$$

$$= u_t^{(i)} \exp(-y^{(i)} h_t(\vec{x}^{(i)}) \log \rho_t) \quad (9.73)$$

$$= u_t^{(i)} \exp(-y^{(i)} h_t(\vec{x}^{(i)}) \alpha_t) \quad (9.74)$$

$$(9.75)$$

$$u_{T+1}^{(i)} = u_1^{(i)} \prod_{t=1}^T \exp(-y^{(i)} \alpha_t h_t(\vec{x}^{(i)})) \quad (9.76)$$

$$= \frac{1}{m} \exp(-y^{(i)} \sum_{t=1}^T \alpha_t h_t(\vec{x}^{(i)})) \quad (9.77)$$

recall

$$h(\vec{x}) = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(\vec{x}^{(i)}) \right) \quad (9.78)$$

$s^{(i)} = \sum_{t=1}^T \alpha_t h_t(\vec{x}^{(i)})$: voting score of $h_t(\vec{x})$ on \vec{x}

Voting Score and Margin

linear blending = LinModel + hypotheses as transform

exponential error measure

$$h(\vec{x}^{(i)}) = \text{sign}\left(\sum_{t=1}^T \alpha_t h_t(\vec{x}^{(i)})\right) = \text{sign}\left(\sum_{j=1}^n \theta_j \phi(\vec{x})_j\right) \quad (9.79)$$

hard-margin SVM

$$\text{margin} = \frac{y^{(i)}(\vec{w}^T \phi(\vec{x}^{(i)}) + b)}{\|\vec{w}\|} \quad (9.80)$$

$y^{(i)} s^{(i)}$ = signed & unnormalized margin

want $y^{(i)} s^{(i)}$ positive & large

$\rightarrow \exp(-y^{(i)} s^{(i)})$ small

$\rightarrow u_{T+1}^{(i)}$ small

AdaBoost Error Function

Theorem 9.1. AdaBoost decreases $\sum_{i=1}^m u_t^{(i)}$ and thus somewhat minimizes

$$\sum_{i=1}^m u_{T+1}^{(i)} = \frac{1}{m} \sum_{i=1}^m \exp(-y^{(i)} \sum_{t=1}^T \alpha_t h_t(\vec{x}^{(i)})) \quad (9.81)$$

$$= \frac{1}{m} \sum_{i=1}^m \exp(-y^{(i)} s^{(i)}) \quad (9.82)$$

$$= \frac{1}{m} \sum_{i=1}^m \text{err}_{ada}(s^{(i)}, y^{(i)}) \quad (9.83)$$

this is *exponential error measure*

$\text{err}_{ada}(s, y)$ is an algorithmic error measure by convex upper bound of $\text{err}_{01}(s, y)$, see Fig. 9.8

$$\text{err}_{01}(s, y) = 1\{ys \leq 0\} \quad (9.84)$$

Learning Hypothesis as Optimization

at iteration t

$$J_{ada} = \frac{1}{m} \sum_{i=1}^m \text{err}_{ada}(s^{(i)}, y^{(i)}) = \frac{1}{m} \sum_{i=1}^m \exp(-y^{(i)} \sum_{\tau=1}^{t-1} \alpha_\tau h_\tau(\vec{x}^{(i)})) \quad (9.85)$$

for binary classification: $y^{(i)}, h(\vec{x}^{(i)}) \in \{-1, +1\}$

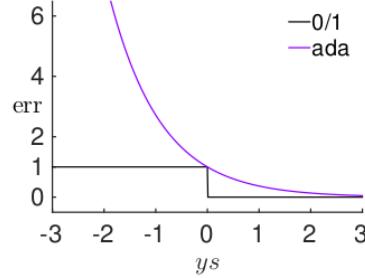


Figure 9.8: exponential error measure

to find h_t , solve

$$h_t^* = \arg \min_{h_t} J_{ada} \quad (9.86)$$

$$= \arg \min_{h_t} \frac{1}{m} \sum_{i=1}^m \left(-y^{(i)} \left(\sum_{\tau=1}^{t-1} \alpha_\tau h_\tau(\vec{x}^{(i)}) + \alpha_t h_t(\vec{x}^{(i)}) \right) \right) \quad (9.87)$$

$$= \arg \min_{h_t} \sum_{i=1}^m u_t^{(i)} \exp(-y^{(i)} \alpha_t h_t(\vec{x}^{(i)})) \quad (9.88)$$

$$\stackrel{\text{taylor}}{\approx} \arg \min_{h_t} \sum_{i=1}^m u_t^{(i)} (1 - y^{(i)} \alpha_t h_t(\vec{x}^{(i)})) \quad (9.89)$$

$$\// \exp(x) = 1 + x \quad (9.90)$$

$$= \arg \min_{h_t} \sum_{i=1}^m u_t^{(i)} - \alpha_t \sum_{i=1}^m u_t^{(i)} y^{(i)} h_t(\vec{x}^{(i)}) \quad (9.91)$$

$$= \arg \min_{h_t} - \sum_{i=1}^m u_t^{(i)} y^{(i)} h_t(\vec{x}^{(i)}) \quad (9.92)$$

$$= \arg \min_{h_t} \sum_{i=1}^m u_t^{(i)} \begin{cases} -1 & \text{if } y^{(i)} = h_t(\vec{x}^{(i)}) \\ 1 & \text{if } y^{(i)} \neq h_t(\vec{x}^{(i)}) \end{cases} \quad (9.93)$$

$$= \arg \min_{h_t} \sum_{i=1}^m u_t^{(i)} (2 \cdot 1\{y^{(i)} \neq h_t(\vec{x}^{(i)})\} - 1) \quad (9.94)$$

$$= \arg \min_{h_t} \left(2 \sum_{i=1}^m u_t^{(i)} 1\{y^{(i)} \neq h_t(\vec{x}^{(i)})\} - \sum_{i=1}^m u_t^{(i)} \right) \quad (9.95)$$

$$= \arg \min_{h_t} \sum_{i=1}^m u_t^{(i)} 1\{y^{(i)} \neq h_t(\vec{x}^{(i)})\} \quad (9.96)$$

$$= \arg \min_{h_t} J_{in}^{\vec{u}_t}(h_t) \quad (9.97)$$

who minimizes $J_{in}^{\vec{u}_t}(h)$? A in AdaBoost!

Deciding Blending Weight as Optimization

steepest descent

recall GD:

$$\min_{\|\vec{v}\|=1} J_{in}(\vec{\theta} + \alpha \vec{v}) \approx J_{in}(\vec{\theta}) + \alpha \vec{v}^T \nabla J_{in}(\vec{\theta}) \quad (9.98)$$

AdaBoost finds h_t by approximately

$$\min_{h_t} J_{ada} = \sum_{i=1}^m u_t^{(i)} \exp(-y^{(i)} \alpha_t h_t(\vec{x}^{(i)})) \quad (9.99)$$

after finding h_t , how about

$$\min_{\alpha_t} J_{ada} = \sum_{i=1}^m u_t^{(i)} \exp(-y^{(i)} \alpha_t h_t(\vec{x}^{(i)})) \quad (9.100)$$

optimal α_t somewhat ‘greedily faster’ than fixed (small) α : called **steepest descent** for optimization

$$\alpha_t^* = \arg \min_{\alpha_t} \sum_{i=1}^m u_t^{(i)} \exp(-y^{(i)} \alpha_t h_t(\vec{x}^{(i)})) \quad (9.101)$$

$$= \arg \min_{\alpha_t} \sum_{i=1}^m u_t^{(i)} \begin{cases} \exp(-\alpha_t) & \text{if } y^{(i)} = h_t(\vec{x}^{(i)}) \\ \exp(\alpha_t) & \text{if } y^{(i)} \neq h_t(\vec{x}^{(i)}) \end{cases} \quad (9.102)$$

$$\begin{aligned} &= \arg \min_{\alpha_t} \sum_{i=1}^m u_t^{(i)} 1\{y^{(i)} = h_t(\vec{x}^{(i)})\} \exp(-\alpha_t) \\ &\quad + \sum_{i=1}^m u_t^{(i)} 1\{y^{(i)} \neq h_t(\vec{x}^{(i)})\} \exp(\alpha_t) \end{aligned} \quad (9.103)$$

$$\begin{aligned} &= \arg \min_{\alpha_t} \sum_{i=1}^m u_t^{(i)} \left(\frac{\sum_{i=1}^m u_t^{(i)} 1\{y^{(i)} = h_t(\vec{x}^{(i)})\}}{\sum_{i=1}^m u_t^{(i)}} \exp(-\alpha_t) \right. \\ &\quad \left. + \frac{\sum_{i=1}^m u_t^{(i)} 1\{y^{(i)} \neq h_t(\vec{x}^{(i)})\}}{\sum_{i=1}^m u_t^{(i)}} \exp(\alpha_t) \right) \end{aligned} \quad (9.104)$$

$$= \arg \min_{\alpha_t} \sum_{i=1}^m u_t^{(i)} ((1 - \varepsilon_t) \exp(-\eta) + \varepsilon_t \exp(\eta)) \quad (9.105)$$

$$\frac{\partial J_{ada}}{\partial \alpha_t} = \sum_{i=1}^m u_t^{(i)} (-(1 - \varepsilon_t) \exp(-\eta) + \varepsilon_t \exp(\eta)) \stackrel{\text{set}}{=} 0 \quad (9.106)$$

$$\therefore \alpha_t = \log \sqrt{\frac{1 - \varepsilon_t}{\varepsilon_t}} \quad (9.107)$$

AdaBoost: steepest descent with approximate functional gradient

9.5.3 Gradient Boosting

Gradient Boosting for Arbitrary Error Function

AdaBoost

$$\min_{\alpha_t} \min_{h_t} \frac{1}{m} \sum_{i=1}^m \left(-y^{(i)} \left(\sum_{\tau=1}^{t-1} \alpha_\tau h_\tau(\vec{x}^{(i)}) + \alpha_t h_t(\vec{x}^{(i)}) \right) \right) \quad (9.108)$$

with binary-output hypothesis h_t

GradientBoost

$$\min_{\alpha_t} \min_{h_t} \frac{1}{m} \sum_{i=1}^m \text{err} \left(\sum_{\tau=1}^{t-1} \alpha_\tau h_\tau(\vec{x}^{(i)}) + \alpha_t h_t(\vec{x}^{(i)}), y^{(i)} \right) \quad (9.109)$$

with any hypothesis h_t (usually real-output hypothesis)

GradientBoost: allows extension to different err for regression / soft classification / etc.

GradientBoost for Regression

$$\text{err}(s, y) = (s - y)^2 \quad (9.110)$$

$$h_t^\star = \arg \min_{h_t} \frac{1}{m} \sum_{i=1}^m \text{err} \left(\sum_{\tau=1}^{t-1} \alpha_\tau h_\tau(\vec{x}^{(i)}) + \alpha_t h_t(\vec{x}^{(i)}), y^{(i)} \right) \quad (9.111)$$

$$\stackrel{\text{def}}{=} \arg \min_{h_t} \frac{1}{m} \sum_{i=1}^m \text{err} \left(s^{(i)} + \alpha_t h_t(\vec{x}^{(i)}), y^{(i)} \right) \quad (9.112)$$

$$\approx \arg \min_{h_t} \frac{1}{m} \sum_{i=1}^m \left(\text{err}(s^{(i)}, y^{(i)}) + \alpha_t h_t(\vec{x}^{(i)}) \frac{\partial \text{err}(s, y^{(i)})}{\partial s} \Big|_{s=s^{(i)}} \right) \quad (9.113)$$

$$= \arg \min_{h_t} \frac{1}{m} \sum_{i=1}^m \text{err}(s^{(i)}, y^{(i)}) + \frac{1}{m} \sum_{i=1}^m \alpha_t h_t(\vec{x}^{(i)}) \frac{\partial \text{err}(s, y^{(i)})}{\partial s} \Big|_{s=s^{(i)}} \quad (9.114)$$

$$= \arg \min_{h_t} \frac{\alpha_t}{m} \sum_{i=1}^m h_t(\vec{x}^{(i)}) 2(s^{(i)} - y^{(i)}) \quad (9.115)$$

$$= -\infty \text{ if no constraint on } h_t \quad (9.116)$$

Learning Hypothesis as Optimization

magnitude of h_t does not matter: because α_t will be optimized next
penalize large magnitude to avoid naive solution

$$h_t^\star = \arg \min_{h_t} \frac{\alpha_t}{m} \sum_{i=1}^m \left(h_t(\vec{x}^{(i)}) 2(s^{(i)} - y^{(i)}) + h(\vec{x}^{(i)})^2 \right) \quad (9.117)$$

$$\begin{aligned}
&= \arg \min_{h_t} \frac{\alpha_t}{m} \sum_{i=1}^m \left(-(s^{(i)} - y^{(i)})^2 + (s^{(i)} - y^{(i)})^2 + 2(s^{(i)} - y^{(i)})h_t(\vec{x}^{(i)}) + h(\vec{x}^{(i)})^2 \right) \\
&= \arg \min_{h_t} \frac{\alpha_t}{m} \sum_{i=1}^m \left(-(s^{(i)} - y^{(i)})^2 + (s^{(i)} - y^{(i)} + h(\vec{x}^{(i)}))^2 \right)
\end{aligned} \tag{9.119}$$

$$= \arg \min_{h_t} \frac{\alpha_t}{m} \sum_{i=1}^m \left(h(\vec{x}^{(i)}) - (y^{(i)} - s^{(i)}) \right)^2 \tag{9.120}$$

solution of penalized approximate functional gradient: squared-error regression on $\{\vec{x}^{(i)}, y^{(i)} - s^{(i)}\} = \{\vec{x}^{(i)}, \text{residual}\}$

Deciding Blending Weight as Optimization

after finding h_t

$$\alpha_t^* = \arg \min_{\alpha_t} \frac{1}{m} \sum_{i=1}^m \text{err} \left(\sum_{\tau=1}^{t-1} \alpha_\tau h_\tau(\vec{x}^{(i)}) + \alpha_t h_t(\vec{x}^{(i)}), y^{(i)} \right) \tag{9.121}$$

$$= \arg \min_{\alpha_t} \frac{1}{m} \sum_{i=1}^m \text{err} \left(s^{(i)} + \alpha_t h_t(\vec{x}^{(i)}) - y^{(i)} \right)^2 \tag{9.122}$$

$$= \arg \min_{\alpha_t} \frac{1}{m} \sum_{i=1}^m \text{err} \left((y^{(i)} - s^{(i)}) - \alpha_t h_t(\vec{x}^{(i)}) \right)^2 \tag{9.123}$$

$$\tag{9.124}$$

one-variable linear regression on $\{(h_t\text{-transformed input}, \text{residual})\}$

$$\alpha_t = \frac{\sum_{i=1}^m h_t(\vec{x})(y^{(i)} - s^{(i)})}{\sum_{i=1}^m h_t(\vec{x})^2} \tag{9.125}$$

Putting Everything Together

Algorithm 47 Gradient Boosted Decision Tree (GBDT)

```

for  $i = 1 : m$ 
     $s^{(i)} = 0$ 
for  $t = 1 : T$ 
     $h_t \leftarrow \text{LinReg}(\{\vec{x}^{(i)}, y^{(i)} - s^{(i)}\})$  using sq. err
     $\alpha_t \leftarrow \text{LinReg}(\{h_t(\vec{x}^{(i)}), y^{(i)} - s^{(i)}\})$  // one var. reg.
     $s^{(i)} \leftarrow s^{(i)} + \alpha_t h_t(\vec{x}^{(i)})$ 
return  $h(\vec{x}) = \sum_{t=1}^T \alpha_t h_t(\vec{x})$ 

```

GBDT: ‘regression sibling’ of AdaBoost-DTree — popular in practice

9.5.4 Summary of Aggregation Models

Map of Blending Models

blending: aggregate after getting diverse h_t , see Fig. 9.9

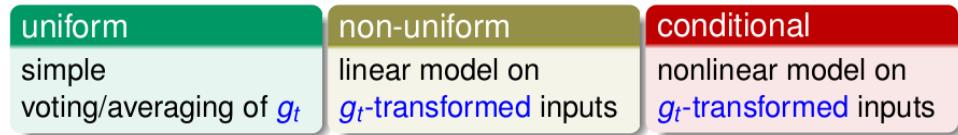


Figure 9.9: Map of Blending Models

uniform for ‘stability’; non-uniform / conditional carefully for ‘complexity’

Map of Aggregation-Learning Models

learning: aggregate as well as getting diverse h_t , see Fig. 9.10

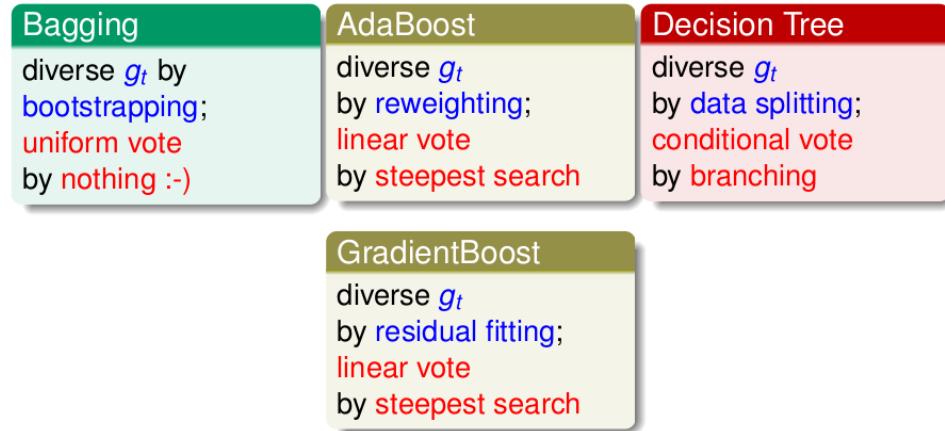


Figure 9.10: Map of Aggregation-Learning Models

boosting-like algorithms most popular

Map of Aggregation of Aggregation Models

all three frequently used in practice, see Fig. 9.11

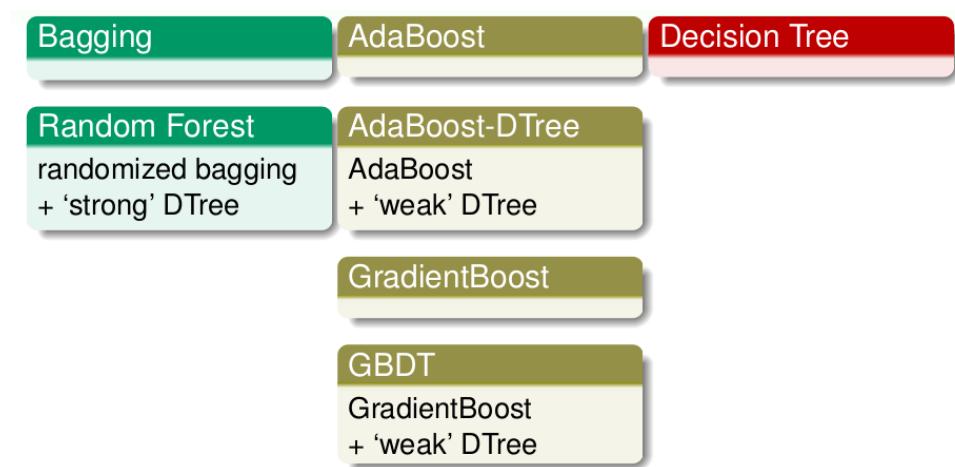


Figure 9.11: Map of Aggregation of Aggregation Models

Chapter 10

Distilling Implicit Features: Extraction Models

10.1 k -means Clustering Algorithm

10.1.1 Clustering Problem

goal: given dataset $\{\vec{x}^{(i)}\}_{i=1}^m$, we want to group the data into K cohesive clusters $c^{(1)}, c^{(2)}, \dots, c^{(K)}$, $\vec{\mu}_k$ is the cluster centroid repr. the center of the clusters, this is an **unsupervised learning** problem

hope $\vec{x}^{(i)}, \vec{x}^{(j)}$ both in $c^{(1)} \iff \vec{\mu}_1 \approx \vec{x}^{(i)} \approx \vec{x}^{(j)}$

10.1.2 Partition Optimization

Define 10.1 (Distortion Function). *It measures the sum of squared distances between each training example $\vec{x}^{(i)}$ and the cluster centroid $\vec{\mu}_{c^{(i)}}$ to which it has been assigned.*

$$J(c, \vec{\mu}) = \sum_{i=1}^m \|\vec{x}^{(i)} - \vec{\mu}_{c^{(i)}}\|^2 \quad (10.1)$$

$$\min_{c, \vec{\mu}} J(c, \vec{\mu}) \quad (10.2)$$

- hard to optimize joint combinatorial-numerical optimization
- two sets of variables: will optimize alternatingly
 if $\vec{\mu}$ fixed, for each $\vec{x}^{(i)}$
- $\|\vec{x}^{(i)} - \vec{\mu}_{c^{(i)}}\|^2$: choose one cluster
- opt. choose of $c^{(i)}$: the one with minimum $\|\vec{x}^{(i)} - \vec{\mu}_{c^{(i)}}\|^2$
 if c fixed

$$J = \sum_{i=1}^m \sum_{k=1}^K 1\{c^{(i)} = k\} \|\vec{x}^{(i)} - \vec{\mu}_k\|^2 \quad (10.3)$$

Elbow Method

$$\nabla_{\vec{\mu}_k} J = \sum_{i=1}^m 1\{c^{(i)} = k\} (2\vec{\mu}_k - 2\vec{x}^{(i)}) \quad (10.4)$$

$$= 2 \sum_{i=1}^m 1\{c^{(i)} = k\} \vec{\mu}_k - 2 \sum_{i=1}^m 1\{c^{(i)} = k\} \vec{x}^{(i)} \quad (10.5)$$

$$\stackrel{\text{set}}{=} \vec{0} \quad (10.6)$$

$$\vec{\mu}_k = \frac{\sum_{i=1}^m 1\{c^{(i)} = k\} \vec{x}^{(i)}}{\sum_{i=1}^m 1\{c^{(i)} = k\}} \quad (10.7)$$

moving each $\vec{\mu}_k$ to the mean of assigned points

10.1.3 k -means Algorithm

Algorithm 48 k -means Algorithm

init. $\vec{\mu}_1, \vec{\mu}_2, \dots, \vec{\mu}_K \in \mathbb{R}^n$ by set them equal to the vals. of K randomly chosen examples

repeat until convergence

```

for  $i = 1 : m$ 
     $c^{(i)} = \arg \min_k \| \vec{x}^{(i)} - \vec{\mu}_k \|^2$ 
for  $k = 1 : K$ 
     $\vec{\mu}_k = \frac{\sum_{i=1}^m 1\{c^{(i)} = k\} \vec{x}^{(i)}}{\sum_{i=1}^m 1\{c^{(i)} = k\}}$ 
```

- converge: no change of $c^{(1)}, c^{(2)}, \dots, c^{(K)}$ anymore
- guaranteed as J_{in} decreases during alternating minimization
- k -means is exactly coordinate descent on J
 - first, min. J wrt. c with $\vec{\mu}$ fixed
 - then, min. J wrt. $\vec{\mu}$ with c fixed
- J is non-convex, so coordinate descent on J is not guaranteed to converge to the global minimum
- one common thing to do is run k -means many times (using different random initial values for the cluster centroids $\vec{\mu}$). then, pick the one that gives the lowest distortion J

10.1.4 How to Choose K

Elbow Method, see Fig. 10.1

sometimes, you're running k -means to get clusters to use for some later / downstream purpose

evaluate k -means based on a metric for how well it performs for that later purpose

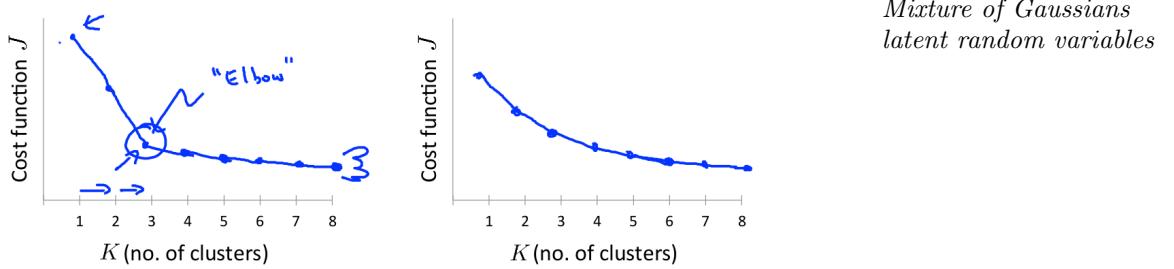


Figure 10.1: Elbow Method

10.2 Mixtures of Gaussians

10.2.1 Mixtures of Gaussians Model

goal: given $\{\vec{x}^{(i)}\}_{i=1}^m$, we want to model the data by mixtures of Gaussians

Define 10.2 (Mixture of Gaussians). *Randomly choosing $c^{(i)}$ from $\{1, 2, \dots, K\}$, and then $\vec{x}^{(i)}$ was generated from one of K Gaussians depending on $c^{(i)}$. $c^{(i)}$'s are **latent random variables***

$$p(\vec{x}^{(i)}, c^{(i)}) = p(\vec{x}^{(i)} | c^{(i)})p(c^{(i)}) \quad (10.8)$$

$$c^{(i)} \sim \text{Multinoulli}(\vec{\phi}) \quad (10.9)$$

$$\phi_k = \Pr(c^{(i)} = k) \quad (10.10)$$

$$\sum_{k=1}^K \phi_k = 1 \quad (10.11)$$

$$\vec{x}^{(i)} | c^{(i)} = k \sim N(\vec{\mu}_k, \Sigma_k) \quad (10.12)$$

10.2.2 Parameter Evaluation

the log-likelihood

$$l(\vec{\phi}, \vec{\mu}, \Sigma) = \sum_{i=1}^m \log p(\vec{x}^{(i)}) \quad (10.13)$$

$$= \sum_{i=1}^m \log \sum_{k=1}^K p(\vec{x}^{(i)}, c^{(i)} = k) \quad (10.14)$$

$$= \sum_{i=1}^m \log \sum_{k=1}^K p(\vec{x}^{(i)} | c^{(i)} = k)p(c^{(i)} = k) \quad (10.15)$$

$$= \sum_{i=1}^m \left(\log \sum_{k=1}^K p(\vec{x}^{(i)} | c^{(i)} = k) + \log p(c^{(i)} = k) \right) \quad (10.16)$$

if we set the derivatives wrt. the paras. to zero, we'll find that it is not possible to find the MLE of the paras. in closed form

10.2.3 Partition Optimization

note that if $c^{(i)}$'s are known, the log-likelihood is

$$l(\vec{\phi}, \vec{\mu}, \Sigma) = \sum_{i=1}^m \left(\log p(\vec{x}^{(i)} | c^{(i)}) + \log p(c^{(i)}) \right) \quad (10.17)$$

MLE gives

$$\phi_k = \frac{1}{m} \sum_{i=1}^m 1\{c^{(i)} = k\} \quad (10.18)$$

$$\vec{\mu}_k = \frac{\sum_{i=1}^m 1\{c^{(i)} = k\} \vec{x}^{(i)}}{\sum_{i=1}^m 1\{c^{(i)} = k\}} \quad (10.19)$$

$$\Sigma_k = \frac{\sum_{i=1}^m 1\{c^{(i)} = k\} (\vec{x}^{(i)} - \vec{\mu}_k)(\vec{x}^{(i)} - \vec{\mu}_k)^T}{\sum_{i=1}^m 1\{c^{(i)} = k\}} \quad (10.20)$$

this is like Gaussian discriminant analysis model, except that here $c^{(i)}$'s plays a role of class labels

but how to get $c^{(i)}$'s?

$$p(c^{(i)} = k | \vec{x}^{(i)}) = \frac{p(\vec{x}^{(i)} | c^{(i)} = k)p(c^{(i)} = k)}{\sum_{j=1}^K p(\vec{x}^{(i)} | c^{(i)} = j)p(c^{(i)} = j)} \quad (10.21)$$

this is a soft guess of $c^{(i)} = k$

10.2.4 EM for Mixtures of Gaussians

similar to k -means, it is also susceptible to local optima, so reinitializing at several different initial parameters may be a good idea

10.2.5 Proof it As a Result of General EM

E-step

$$w_k^{(i)} = q(c^{(i)} = k) = p(c^{(i)} = k | \vec{x}^{(i)}) \quad (10.22)$$

M-step

$$\sum_{i=1}^m \sum_{k=1}^K q(c^{(i)} = k) \log \frac{p(\vec{x}^{(i)}, c^{(i)} = k)}{q(c^{(i)} = k)} \quad (10.23)$$

Algorithm 49 EM with Mixtures of Gaussians

repeat until convergence

```
// E step
for  $i = 1 : m$ 
  for  $k = 1 : K$ 
     $w_k^{(i)} = p(c^{(i)} = k | \vec{x}^{(i)})$  // try to guess  $c^{(i)}$ 
// M step
for  $k = 1 : K$ 
   $\phi_k = \frac{1}{m} \sum_{i=1}^m w_k^{(i)}$ 
   $\vec{\mu}_k = \frac{\sum_{i=1}^m w_k^{(i)} \vec{x}^{(i)}}{\sum_{i=1}^m w_k^{(i)}}$ 
   $\Sigma_k = \frac{\sum_{i=1}^m w_k^{(i)} (\vec{x}^{(i)} - \vec{\mu}_k)(\vec{x}^{(i)} - \vec{\mu}_k)^T}{\sum_{i=1}^m w_k^{(i)}}$ 
```

$$\begin{aligned} &= \sum_{i=1}^m \sum_{k=1}^K w_k^{(i)} \log \frac{1}{w_k^{(i)}} p(\vec{x}^{(i)} | c^{(i)} = k) p(c^{(i)} = k) \\ &= \sum_{i=1}^m \sum_{k=1}^K w_k^{(i)} \log p(\vec{x}^{(i)} | c^{(i)} = k) + \sum_{i=1}^m \sum_{k=1}^K w_k^{(i)} \log \phi_k - \sum_{i=1}^m \sum_{k=1}^K w_k^{(i)} \log \phi_k^2 \end{aligned} \quad (10.24)$$

$$\vec{\mu}_k = \arg \max_{\vec{\mu}_k} \sum_{i=1}^m \sum_{k=1}^K w_k^{(i)} \log p(\vec{x}^{(i)} | c^{(i)} = k) \quad (10.26)$$

$$= \arg \max_{\vec{\mu}_k} \sum_{i=1}^m \sum_{k=1}^K w_k^{(i)} \left(-\frac{1}{2} (\vec{x}^{(i)} - \vec{\mu}_k)^T \Sigma_k^{-1} (\vec{x}^{(i)} - \vec{\mu}_k) \right) \quad (10.27)$$

$$\nabla_{\vec{\mu}_k} = \sum_{i=1}^m w_k^{(i)} \Sigma_k^{-1} (\vec{x}^{(i)} - \vec{\mu}_k) \quad (10.28)$$

$$= \sum_{i=1}^m w_k^{(i)} \Sigma_k^{-1} \vec{x}^{(i)} - \sum_{i=1}^m w_k^{(i)} \Sigma_k^{-1} \vec{\mu}_k \quad (10.29)$$

$$\stackrel{\text{set}}{=} \vec{0} \quad (10.30)$$

$$\vec{\mu}_k = \frac{\sum_{i=1}^m \sum_{k=1}^K w_k^{(i)} \vec{x}^{(i)}}{\sum_{i=1}^m \sum_{k=1}^K w_k^{(i)}} \quad (10.31)$$

$$\phi_k = \arg \max_{\phi_k} \sum_{i=1}^m \sum_{k=1}^K w_k^{(i)} \log \phi_k \quad (10.32)$$

note there is an additional constraint

$$\sum_{k=1}^K \phi_k = 1 \quad (10.33)$$

we don't need to add constraint $\phi_k \geq 0$, because the sol. already satisfy this constraint

construct Lagrangian

$$L(\vec{\phi}) = \sum_{i=1}^m \sum_{k=1}^K w_k^{(i)} \log \phi_k + \beta \left(\sum_{k=1}^K \phi_k - 1 \right) \quad (10.34)$$

$$\frac{\partial L}{\partial \phi_k} = \sum_{i=1}^m \frac{w_k^{(i)}}{\phi_k} + 1 \stackrel{\text{set}}{=} 0 \quad (10.35)$$

$$\phi_k = \frac{\sum_{i=1}^m w_k^{(i)}}{-\beta} \quad (10.36)$$

$$-\beta = \sum_{k=1}^K \sum_{i=1}^m w_k^{(i)} // -\beta \text{ is a normalizing const.} \quad (10.37)$$

$$= \sum_{i=1}^m \sum_{k=1}^K p(c^{(i)} = k | \vec{x}^{(i)}) \quad (10.38)$$

$$= \sum_{i=1}^m 1 \quad (10.39)$$

$$= m \quad (10.40)$$

$$\phi_k = \frac{1}{m} \sum_{i=1}^m w_k^{(i)} \quad (10.41)$$

10.3 Factor Analysis

10.3.1 Single Gaussian

model the data with a single multivariate Gaussian

$$\vec{x} \sim N(\mu, \Sigma) \quad (10.42)$$

$$l(\mu, \Sigma) = \sum_{i=1}^m \log p(\vec{x}; \vec{\mu}, \Sigma) \quad (10.43)$$

$$= \sum_{i=1}^m \log \frac{1}{\sqrt{2\pi}^n \sqrt{|\Sigma|}} \exp\left(-\frac{1}{2} (\vec{x} - \vec{\mu})^T \Sigma^{-1} (\vec{x} - \vec{\mu})\right) \quad (10.44)$$

MLE gives

$$\mu_j = \frac{1}{m} \sum_{i=1}^m \vec{x}^{(i)} \quad (10.45)$$

$$\Sigma = \frac{1}{m} \sum_{i=1}^m (\vec{x}^{(i)} - \vec{\mu})(\vec{x}^{(i)} - \vec{\mu})^T \quad (10.46)$$

when m are small, the estimate of Gaussian showed above are still poor

- if m data point span only a low-dim. subspace of \mathbb{R}^n
- the Gaussian will places all of its probability in the affine space spanned by that data
- when $m \leq n$, Σ is singular

want: fit a reasonable Gaussian model to the data, and perhaps capture some interesting covariance structure in the data

10.3.2 Restrictions on Σ

idea: restrict on Σ so we can fit it with small data

Diagonal Σ

choose Σ that is diagonal

$$x_j \sim N(\mu_j, \sigma_j^2), \forall j \quad (10.47)$$

Parameter Fitting

$$l(\mu, \sigma) = \sum_{i=1}^m \log p(\vec{x}^{(i)}) \quad (10.48)$$

$$= \sum_{i=1}^m \sum_{j=1}^n \log p(x_j) \quad (10.49)$$

$$= \sum_{i=1}^m \sum_{j=1}^n \log \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right) \quad (10.50)$$

MLE gives

$$\mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)} \quad (10.51)$$

$$\Sigma_{jj} = \sigma_j^2 = \frac{1}{m} \sum_{i=1}^m (x_j^{(i)} - \mu_j)^2 \quad (10.52)$$

- indep. estimate the mean and variance of each coordinate j
- diag. Σ corresponds to axis-aligned ellipse contours of Gaussian density

Choosing What Features to Use for non-Gaussian features, transform methods includes

- $x_j \leftarrow \log x_j$, see Fig. 10.2
- $x_j \leftarrow \log(x_j + c)$
- $x_j \leftarrow \sqrt{x_j}$
- $x_j \leftarrow x_j^c$

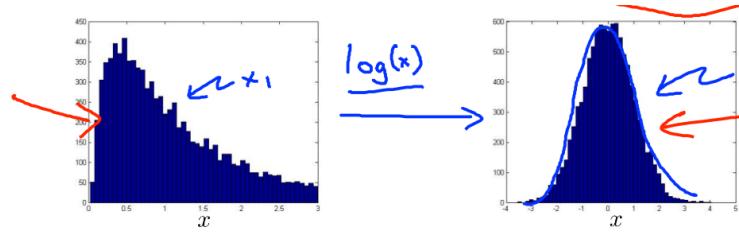


Figure 10.2: log transformation

Diagonal Σ with Equal Entries

choose Σ that is diagonal, and diagonal entries are equal, $\Sigma = \sigma^2 I$

$$\sigma^2 = \frac{1}{n} \sum_{j=1}^n \frac{1}{m} \sum_{i=1}^m (\bar{x}^{(i)} - \mu_j)^2 = \frac{1}{mn} \sum_{i=1}^m \sum_{j=1}^n (\bar{x}^{(i)} - \mu_j)^2 \quad (10.53)$$

- this corresponds to axis-aligned circle contours of Gaussian density
- Σ is non-singular if $m > 1$

Pros and Cons About Diagonal Σ

pros

- computational cheap
- OK even if m is small

cons

- modeling the diff. coordinates x_j, x_k of the data as being uncorrelated and indep.
- fail to capture correlation structure in the data
- need manually create features to capture anomalies where x_1, x_2 take unusual comb. of vals.

want

- use more parameters than the diagonal Σ and captures some correlations in the data
- without having to fit a full covariance matrix

10.3.3 The Factor Analysis Model

$$\vec{x} = \vec{\mu} + A\vec{z} + \vec{\varepsilon}, \vec{\mu} \in \mathbb{R}^n, A \in \mathbb{R}^{n \times k} \quad (10.54)$$

$$\vec{z} \sim N(\vec{0}, I), \vec{z} \in \mathbb{R}^k \quad (10.55)$$

$$\vec{\varepsilon} \sim N(\vec{0}, S), \vec{\varepsilon} \in \mathbb{R}^n \quad (10.56)$$

$$\mathbb{E}[\vec{x}] = \mathbb{E}[\vec{\mu}] + \mathbb{E}[A\vec{z}] + \mathbb{E}[\vec{\varepsilon}] \quad (10.57)$$

$$= \vec{\mu} + A\mathbb{E}[\vec{z}] + \vec{0} \quad (10.58)$$

$$= \vec{\mu} \quad (10.59)$$

$$\text{Cov}(\vec{x}) = \text{Cov}(\vec{\mu}) + \text{Cov}(A\vec{z}) + \text{Cov}(\vec{\varepsilon}) \quad (10.60)$$

$$= 0 + \mathbb{E}[A\vec{z}(A\vec{z})^T] + S \quad (10.61)$$

$$= \mathbb{E}[A\vec{z}\vec{z}^T A^T] + S \quad (10.62)$$

$$= A\mathbb{E}[\vec{z}\vec{z}^T]A^T + S \quad (10.63)$$

$$= AA^T + S \quad (10.64)$$

$$\vec{x} \sim N(\vec{\mu}, AA^T + S) \quad (10.65)$$

the log-likelihood

$$l(\vec{\mu}, A, S) = \sum_{i=1}^m \log p(\vec{x}^{(i)}) \quad (10.66)$$

$$= \sum_{i=1}^m \log N(\vec{\mu}, AA^T + S) \quad (10.67)$$

MLE explicitly is hard, we instead use EM

10.3.4 EM for Factor Analysis

$$\text{Cov}(\vec{z}, \vec{x}) = \mathbb{E}[\vec{z}(\vec{\mu} + A\vec{z} + \vec{\varepsilon} - \vec{\mu})^T] \quad (10.68)$$

$$= \mathbb{E}[\vec{z}(A\vec{z})^T + \vec{z}\vec{\varepsilon}^T] \quad (10.69)$$

$$= \mathbb{E}[\vec{z}\vec{z}^T]A^T + \mathbb{E}[\vec{z}]\mathbb{E}[\vec{\varepsilon}^T] // \vec{z} \text{ and } \vec{\varepsilon} \text{ are indep.} \quad (10.70)$$

$$= A^T \quad (10.71)$$

$$\begin{bmatrix} \vec{z} \\ \vec{x} \end{bmatrix} \sim N\left(\begin{bmatrix} \vec{0} \\ \vec{\mu} \end{bmatrix}, \begin{bmatrix} I & A^T \\ A & AA^T + S \end{bmatrix}\right) \quad (10.72)$$

E-step

$$q(\vec{z}^{(i)}) = p(\vec{z}^{(i)} | \vec{x}^{(i)}) = N(A^T(AA^T + S)^{-1}(\vec{x}^{(i)} - \vec{\mu}), I - A^T(A^TA + S)^{-1}A) \quad (10.73)$$

M-step

$$\sum_{i=1}^m \int_{\vec{z}^{(i)}} q(\vec{z}^{(i)}) \log \frac{p(\vec{x}^{(i)}, \vec{z}^{(i)})}{q(\vec{z}^{(i)})} d\vec{z}^{(i)} \quad (10.74)$$

$$= \sum_{i=1}^m \int_{\vec{z}^{(i)}} q(\vec{z}^{(i)}) \left(\log p(\vec{x}^{(i)} | \vec{z}^{(i)}) + \log p(\vec{z}^{(i)}) - \log q(\vec{z}^{(i)}) \right) \quad (10.75)$$

$$= \sum_{i=1}^m \mathbb{E}_{\vec{z}^{(i)}} [\log p(\vec{x}^{(i)} | \vec{z}^{(i)}) + \log p(\vec{z}^{(i)}) - \log q(\vec{z}^{(i)})] \quad (10.76)$$

MLE gives

$$A = \left(\sum_{i=1}^m (\vec{x}^{(i)} - \vec{\mu}) \vec{\mu}_{\vec{z}^{(i)} | \vec{x}^{(i)}}^T \right) \left(\sum_{i=1}^m \vec{\mu}_{\vec{z}^{(i)} | \vec{x}^{(i)}} \vec{\mu}_{\vec{z}^{(i)} | \vec{x}^{(i)}}^T + \Sigma_{\vec{z}^{(i)} | \vec{x}^{(i)}} \right)^{-1} \quad (10.77)$$

- this looks like in linReg, $\vec{\theta}^T = (\vec{y}^T X)(X^T X)^{-1}$
- \vec{x} are linear functions of \vec{z} , given the guess for \vec{z} in E-step, we will now try to estimate the unknown linearity A relating \vec{x} and \vec{z}

$$\vec{\mu} = \frac{1}{m} \sum_{i=1}^m \vec{x}^{(i)} \quad (10.78)$$

$$D = \frac{1}{m} \sum_{i=1}^m \left(\vec{x}^{(i)} \vec{x}^{(i)T} - \vec{x}^{(i)} \vec{\mu}_{\vec{z}^{(i)} | \vec{x}^{(i)}}^T A^T - A \vec{\mu}_{\vec{z}^{(i)} | \vec{x}^{(i)}} \vec{x}^{(i)T} + A (\vec{\mu}_{\vec{z}^{(i)} | \vec{x}^{(i)}} \vec{\mu}_{\vec{z}^{(i)} | \vec{x}^{(i)}}^T + \Sigma_{\vec{z}^{(i)} | \vec{x}^{(i)}}) A^T \right) \quad (10.79)$$

S be the diag. mat. containing only the diag. entries of D

$$S_{jj} = D_{jj}, \forall j \quad (10.80)$$

10.3.5 Application: Anomaly Detection

Anomaly Detection Problem

given $\{\vec{x}^{(i)}\}_{i=1}^m$, estimate $p(\vec{x})$, further check whether $p(\vec{x}) < \varepsilon$

Evaluation of Anomaly Detection

when developing a learning algorithm, making decisions is much easier if we have a way of some real-number evaluating our learning system

fit model $p(\vec{x})$ on training set

on a val. set, predict

- $y = 1$ if $p(\vec{x}) < \varepsilon$: anomaly
- $y = 0$ if $p(\vec{x}) \geq \varepsilon$: normal

possibly use Precision / Recall, F_1 score for evaluation
can also use val. to choose ε

Anomaly Detection vs. Supervised Learning

anomaly detection

- very small # of pos. examples
- large # of negative examples
- many diff. types of anomalies, hard for any alg. to learn from pos. examples what the anomalies look like
- future anomalies may look nothing like any of the anomalies examples we've seen so far

supervised learning

- large # of pos. and neg. examples
- enough pos. examples for alg. to get a sense of what pos. examples are like
- future pos. examples likely to be similar to ones in training set

10.4 Principle Component Analysis

10.4.1 PCA Problem

suppose you are training your algorithm on 16×16 grayscale images

- $\vec{x} \in \mathbb{R}^{256}$ are 256 dimensional vectors
- the values of adjacent pixels in an image are highly correlated
- then the input will be somewhat redundant

want: approximate the input with a much lower dimensional one, remove correlations, while incurring very little error

10.4.2 Solving PCA

Mean and Variance Normalization

normal data first prior to PCA

$$\vec{\mu} = \frac{1}{m} \sum_{i=1}^m \vec{x}^{(i)} \quad (10.81)$$

$$\vec{x}^{(i)} \leftarrow \vec{x}^{(i)} - \vec{\mu}, \forall i \quad (10.82)$$

these steps can be omitted for data known to have zero mean (eg., times series corresponding to speech or other acoustic signals)

$$\sigma_j^2 = \frac{1}{m} \sum_{i=1}^m x_j^{(i)2}, \forall j \quad (10.83)$$

$$x_j^{(i)} \leftarrow \frac{x_j^{(i)}}{\sigma_j^2}, \forall i, j \quad (10.84)$$

these steps can be omitted if we had apriori knowledge that diff. features are all on the same scale (eg., if each data point repr. a grayscale img., and $x_j^{(i)} \in \{0, 1, \dots, 255\}$ corresponding to the intensity val. of pixel j in img. i)

One Dimensional Case

PCA is finding the unit vector \vec{u} st. when the data is proj. onto the direction corresponding to \vec{u} , the variance of the projected data is maximized
the length of the proj. of \vec{x} onto \vec{u} is given by $\vec{x}^T \vec{u}$

$$\max_{\|\vec{u}\|=1} \frac{1}{m} \sum_{i=1}^m (\vec{x}^{(i)T} \vec{u})^2 \quad (10.85)$$

$$= \max_{\|\vec{u}\|=1} \frac{1}{m} \sum_{i=1}^m \vec{u}^T \vec{x}^{(i)} \vec{x}^{(i)T} \vec{u} \quad (10.86)$$

$$= \max_{\|\vec{u}\|=1} \vec{u}^T \left(\frac{1}{m} \sum_{i=1}^m \vec{x}^{(i)} \vec{x}^{(i)T} \right) \vec{u} \quad (10.87)$$

$$= \max_{\|\vec{u}\|=1} \vec{u}^T \Sigma \vec{u} \quad (10.88)$$

where

$$\Sigma = \frac{1}{m} \sum_{i=1}^m \vec{x}^{(i)} \vec{x}^{(i)T} \in \mathbb{S}_{++}^n \quad (10.89)$$

if \vec{x} has zero mean, then Σ is exactly the covariance matrix of \vec{x}
so the opt. problem is

$$\min_{\vec{u}} -\vec{u}^T \Sigma \vec{u} \quad (10.90)$$

$$\text{st. } \|\vec{u}\|^2 = \vec{u}^T \vec{u} = 1 \quad (10.91)$$

Lagrangian

$$L(\vec{u}) = -\vec{u}^T \Sigma \vec{u} + \lambda(\vec{u}^T \vec{u} - 1) \quad (10.92)$$

$$\nabla_{\vec{u}} L(\vec{u}) = -2\Sigma \vec{u} + 2\lambda \vec{u} \stackrel{\text{set}}{=} \vec{0} \quad (10.93)$$

$$\Sigma \vec{u} = \lambda \vec{u} \quad (10.94)$$

which implies \vec{u} is an eig.-vec. of Σ , with eig.-val. λ
the objective becomes

$$-\vec{u}^T \Sigma \vec{u} = -\lambda \vec{u}^T \vec{u} = -\lambda \quad (10.95)$$

thus, we select the top eig.-vec. of Σ as \vec{u}

k Dimensional Case

Reducing the Data Dimension PCA will find a lower-dimensional subspace onto which to project our data

suppose that subspace is spanned by $\vec{u}_1, \vec{u}_2, \dots, \vec{u}_k$, the are orthonormal, let

$$U = [\vec{u}_1 \ \vec{u}_2 \ \cdots \ \vec{u}_k] \quad (10.96)$$

thus, we can proj. \vec{x} in the u -basis by computing

$$\vec{z} = U^T \vec{x} = \begin{bmatrix} \vec{u}_1^T \vec{x} \\ \vec{u}_2^T \vec{x} \\ \vdots \\ \vec{u}_k^T \vec{x} \end{bmatrix} \in \mathbb{R}^k \quad (10.97)$$

$$\max_U \frac{1}{m} \sum_{i=1}^m \|U^T \vec{x}^{(i)}\|^2 = \max_U \frac{1}{m} \sum_{i=1}^m (U^T \vec{x}^{(i)})^T (U^T \vec{x}^{(i)}) \quad (10.98)$$

$$= \max_U \frac{1}{m} \sum_{i=1}^m \text{tr}(U^T \vec{x}^{(i)}) (U^T \vec{x}^{(i)})^T \quad (10.99)$$

$$= \max_U \frac{1}{m} \sum_{i=1}^m \text{tr} U^T \vec{x}^{(i)} \vec{x}^{(i)T} U \quad (10.100)$$

$$= \max_U \text{tr} \frac{1}{m} \sum_{i=1}^m U^T \vec{x}^{(i)} \vec{x}^{(i)T} U \quad (10.101)$$

$$\text{// } \text{tr}(A + B) = \text{tr } A + \text{tr } B, \text{tr } cA = c \text{tr } A$$

$$= \max_U \text{tr} U^T \left(\frac{1}{m} \sum_{i=1}^m \vec{x}^{(i)} \vec{x}^{(i)T} \right) U \quad (10.102)$$

$$= \max_U \text{tr} U^T \Sigma U \quad (10.103)$$

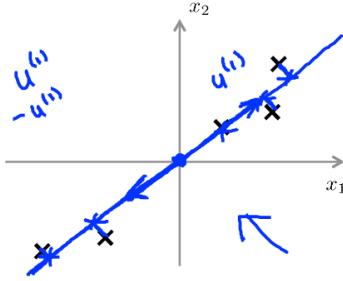
$$\min_{U \in \mathbb{R}^{n \times k}} -\text{tr} U^T \Sigma U \quad (10.104)$$

$$\text{st. } U^T U = I \quad (10.105)$$

it turns out that $\vec{u}_1, \vec{u}_2, \dots, \vec{u}_k$ correspond to top- k eig.-vecs. of Σ

then $\vec{u}_j^T \vec{x}$ is the length (magnitude) of the projection of \vec{x} onto the vector \vec{u}_j

another way of explaining PCA is finding a lower-dimensional subspace onto which to project our data, so as to minimize the projection error, see Fig. 10.3

Figure 10.3: PCA is used to reduce from n -dim. to k -dim.

Recovering an Approximation of the Data now, $\vec{z} \in \mathbb{R}^k$ is a lower-dimensional, “compressed” representation of the original $\vec{x} \in \mathbb{R}^n$ given \vec{z} , how can we recover an approximation $\hat{\vec{x}}$ to the original value of \vec{x} ?

$$\hat{\vec{x}} = U\vec{z} = \sum_{j=1}^k z_j \vec{u}_j \quad (10.106)$$

n Dimensional Case

particularly, if $k = n$

- U is the exact eig.-vecs. of Σ
- U satisfies $U^T U = U U^T = I$
- $\vec{z} = U^T \vec{x}$ is rotating the data to u 's basis
- can get the original data by $\vec{x} = U U^T \vec{x} = U \vec{z}$

10.4.3 Number of components to retain

How do we set k ?

- If k is too large, then we won't be compressing the data much
- if k is too small, then we might be using a very bad approximation to the data.

we will usually look at the ”percentage of variance retained” for diff. k

$$\lambda_j = \sum_{i=1}^m x_{red,j}^2 \quad (10.107)$$

if $\lambda_j \approx 0$, that shows that $x_{rot,j}$ is usually near 0 anyway, and we lose relatively little by approximating it with a constant 0

one common heuristic is to choose k so as to retain 99% of the variance. In other words, we pick the smallest value of k that satisfies

$$\frac{\sum_{j=1}^k \lambda_j}{\sum_{j=1}^n \lambda_j} \geq 99\% \quad (10.108)$$

values in the 90-98% range are also sometimes used eigenfaces

Applications on PCA

compression

visualize the data by set $k = 2, 3$

preprocess a dataset to reduce its dim. before running a learning alg. in order to speed up your alg. while introducing very little approximation error noise reduction, eg. **eigenfaces** methods

- the PCs retain the interesting, systematic variations betw. faces that capture what a person really looks like
- but not the noise in the imgs. introduced by minor lighting variations, slightly diff. img. conditions, and so on
- then measure betw. faces i and j by working in the reduced dim., computing $\|\vec{z}^{(i)} - \vec{z}^{(j)}\|$

bad use of PCA: to prevent overfitting, this might work OK, but isn't a good way to address overfilng, use regularization instead.

before implementing PCA, first try running whatever you want to do with the original / raw data $\vec{x}^{(i)}$, only if that doesn't do what you want, then implement PCA and consider using $\vec{z}^{(i)}$

10.4.4 Implementing PCA Whitening

for $X \in \mathbb{R}^{m \times n}$

First, we need to ensure that the data has (approximately) zero-mean. For natural images, we achieve this (approximately) by subtracting the mean value of each image patch.

```

1 X = X';      % make it n by m, one training example per col.
2 Mu = mean(X);
3 X = X - ones(n, 1) * Mu;    % ensure data has zero-mean
4 Sigma = X * X' / m;
5 [U, D, V] = svd(Sigma); % compute eig.-vecs. of Sigma
6 Z = U(:, 1:k)' * X;
```

10.5 Spectral Clustering

10.5.1 Spectral Clustering Algorithm

Algorithm 50 Spectral Clustering Algorithm

use Laplacian Eigenmap to find a low dim. repr. of X as Z
apply k -means to Z for clustering

next, we will see how to use Laplacian Eigenmap

graph Laplacian
generalized
eigenproblem

10.5.2 Graph Construction

- a node is defined for every $\vec{x}^{(i)}$
- an edge exists betw. $\vec{x}^{(i)}$ and $\vec{x}^{(j)}$
- if $dist(\vec{x}^{(i)}, \vec{x}^{(j)}) \leq \varepsilon$
- if either one is a k -NN of the other
- the weight on the edge
- 1
- Gaussian kernel: $\exp(-\frac{dist(\vec{x}^{(i)}, \vec{x}^{(j)})}{2\sigma^2})$

10.5.3 One-Dimensional Case

$$z^{(i)} \in \mathbb{R} \quad (10.109)$$

$$\vec{z} = \begin{bmatrix} z^{(1)} \\ z^{(2)} \\ \vdots \\ z^{(m)} \end{bmatrix} \quad (10.110)$$

The Objective Function

$$J = \sum_{i=1}^m \sum_{j=1}^m w_{ij} (x^{(i)} - x^{(j)})^2 = 2\vec{z}^T L \vec{z} \quad (10.111)$$

$$L = D - W \in \mathbb{S}_{++}^n \quad (10.112)$$

- L is the **graph Laplacian**
- $W \in \mathbb{R}^{n \times n}$ is the similarity mat.
- $D \in \mathbb{R}^{n \times n}$ is the diag. mat. with $D_{ii} = \sum_{j=1}^n w_{ij}$

The Optimization Problem

add a constraint to remove scaling factor, D is introduced for normalization

$$\min_{\vec{z}} \vec{z}^T L \vec{z} \quad (10.113)$$

$$\text{st. } \vec{z}^T D \vec{z} = 1 \quad (10.114)$$

sol.

$$L \vec{z} = \lambda D \vec{z} \quad (10.115)$$

- **generalized eigenproblem**
- the smallest eig.-vec. is $\vec{z} = \vec{1}$, this is useless
- use the second smallest eig.-vec.

10.5.4 k -Dimensional Case

$$Z = \begin{bmatrix} \vec{z}^{(1)T} \\ \vec{z}^{(2)T} \\ \vdots \\ \vec{z}^{(m)T} \end{bmatrix} \quad (10.116)$$

$$J = \sum_{i=1}^m \sum_{j=1}^m w_{ij} \|\vec{x}^{(i)} - \vec{x}^{(j)}\|^2 = 2 \operatorname{tr} Z^T L Z \quad (10.117)$$

the opt. problem

$$\min_{\vec{z}} \operatorname{tr} Z^T L Z \quad (10.118)$$

$$\text{st. } Z^T D Z = I \quad (10.119)$$

sol.

$$L Z = \lambda D Z \quad (10.120)$$

- use $Z = [\vec{z}_2 \ \vec{z}_3 \ \cdots \ \vec{z}_{k+1}]$ as the opt. sol.
- \vec{z}_k is the k -th generalized eig.-vec.
- the new repr. of $\vec{x}^{(i)}$ is the i -th row of Z
- don't forget the normalization $Z^T D Z = I$

10.6 Non-Negative Matrix Factorization (NMF)

10.6.1 Non-Negative Matrix Factorization

non-negative data mat.

$$X = [\vec{x}^{(1)} \ \vec{x}^{(2)} \ \cdots \ \vec{x}^{(m)}] \in \mathbb{R}^{n \times m} \quad (10.121)$$

want approx X as prod. of two non-negative mat.

$$X \approx U Z \quad (10.122)$$

$$U = [\vec{u}_1 \ \vec{u}_2 \ \cdots \ \vec{u}_k] \in \mathbb{R}^{n \times k} \quad (10.123)$$

$$Z = [\vec{z}^{(1)} \ \vec{z}^{(2)} \ \cdots \ \vec{z}^{(m)}] \in \mathbb{R}^{k \times m} \quad (10.124)$$

10.6.2 Interpretation of NMF

$$\vec{x}^{(i)} = U \vec{z}^{(i)} = \sum_{j=1}^k z_j^{(i)} \vec{u}_j \quad (10.125)$$

cocktail party problem

- $\vec{u}_1, \vec{u}_2, \dots, \vec{u}_k$ can be treated as basis vectors
they may be not orthonormal
they are non-negative
- \vec{u}_j can be treated as an repr. of the j -th cluster
- $\vec{z}^{(i)}$ can be treated as a new k -dim. repr. of $\vec{x}^{(i)}$
the cluster label for $\vec{x}^{(i)}$

$$c^{(i)} = \arg \max_j z_j^{(i)} \quad (10.126)$$

an example can be seen in Fig. 10.4

2	2	1	2	0	0
2	3	3	3	0	0
1	1	1	1	0	0
2	2	2	3	1	1
0	0	0	1	1	1
0	0	0	2	1	2

≈

2	0
3	0
1	0
2	1
0	1
0	2

×

1	1	1	1	0	0
0	0	0	1	1	1

Figure 10.4: Discover both row and col. clusters

10.6.3 Optimization in NMF

$$U \leftarrow U * (XZ^T) ./ (UZZ^T) \quad (10.127)$$

$$Z \leftarrow Z * (X^T U) ./ (Z^T U^T U) \quad (10.128)$$

10.7 Independent Components Analysis

10.7.1 ICA Problem

consider the **cocktail party problem**

- n speakers are speaking simultaneously at a party: $\vec{a} \in \mathbb{R}^n$, a_j is the sound that speaker j was uttering
- n microphones placed in the room recording only an overlapping combination of the n speakers' voices: $\vec{x} \in \mathbb{R}^n$, x_j is the acoustic reading recorded by microphone j
- each microphone is at a different distance from each of the speakers, it records a different combination of the speakers' voices
- using these microphone recordings, can we separate out the original n speakers' speech signals?

to formalize this problem, what we observe is

$$\vec{x} = U\vec{a} \quad (10.129)$$

mixing matrix
unmixing matrix

where U is an unknown square matrix called the **mixing matrix**

repeated observations gives us a dataset $\{\vec{x}^{(i)}\}_{i=1}^m$, and our goal is to recover the sources $\vec{a}^{(i)}$ uttering at time i that had generated our data $\vec{x}^{(i)} = U\vec{a}^{(i)}$ uttering at time i .

note matrix U was for mapping speaker source \vec{a} to microphone record \vec{x} , the **unmixing matrix** $W = U^{-1}$ works in the opposite direction, mapping record \vec{x} to sources \vec{a} instead

$$\vec{a} = W\vec{x} = \begin{bmatrix} \vec{w}_1^T \\ \vec{w}_2^T \\ \vdots \\ \vec{w}_n^T \end{bmatrix} \vec{x} = \begin{bmatrix} \vec{w}_1^T \vec{x} \\ \vec{w}_2^T \vec{x} \\ \vdots \\ \vec{w}_n^T \vec{x} \end{bmatrix} \in \mathbb{R}^n \quad (10.130)$$

thus, the j -th source can be recovered by computing $a_j = \vec{w}_j^T \vec{x}$

10.7.2 ICA ambiguities

if we have no prior knowledge about the sources and the mixing matrix, there are some inherent ambiguities in U that are impossible to recover, given only the $\vec{x}^{(i)}$'s

let $P \in \mathbb{R}^{n \times n}$ be any permutation matrix

- there will be no way to distinguish between W and PW
- the permutation of the original sources is ambiguous
- this does not matter for most applications

there is no way to recover the correct scaling of the w_j 's

- if a single column of U were scaled by a factor of α , and the corresponding source were scaled by a factor of $\frac{1}{\alpha}$
- given only the $\vec{x}^{(i)}$'s, we cannot determine that this had happened
- scaling a speaker's speech signal a_j by some positive factor α affects only the volume of that speaker's speech.
- also, a_j and $-a_j$ sound identical when played on a speaker
- for the applications that we are concerned with, this ambiguity also does not matter

it turns out that they are the only sources of ambiguity in ICA, so long as the sources \vec{a} are non-Gaussian

for Gaussian data, consider an example $\vec{a} \sim N(\vec{0}, I)$, note that the contours of the density of the \vec{a} are circles centered on the origin, which is rotationally symmetric

suppose the mixing mat. is U , we observe

$$\vec{x} = U\vec{a} \quad (10.131)$$

$$\sim N(\vec{0}, \mathbb{E}[\vec{x}\vec{x}^T]) \quad (10.132)$$

$$= N(\vec{0}, \mathbb{E}[U\vec{a}\vec{a}^TU^T]) \quad (10.133)$$

$$= N(\vec{0}, U\mathbb{E}[\vec{a}\vec{a}^T]U^T) \quad (10.134)$$

$$= N(\vec{0}, UIU^T) \quad (10.135)$$

$$= N(\vec{0}, UU^T) \quad (10.136)$$

let R be an arbitrary orthogonal (less formally, a rotation /reflection) matrix, so that $RR^T = R^TR = I$, and let $U' = UR$. if the data has been mixed according to U'

$$\vec{x}' = U'\vec{a} \quad (10.137)$$

$$\sim N(\vec{0}, U'U'^T) \quad (10.138)$$

$$= N(\vec{0}, URR^TU^T) \quad (10.139)$$

$$= N(\vec{0}, UU^T) \quad (10.140)$$

- whether the mixing matrix is U or U' , we would observe data from a $N(\vec{0}, UU^T)$ distribution
- there is no way to tell if the sources were mixed using U or U'
- there is an arbitrary rotational component in the mixing matrix that cannot be determined from the data, and we cannot recover the original sources

10.7.3 ICA Algorithm

ICA Model

assume the joint distr. of the sources \vec{a} is given by

$$p_{\vec{a}}(\vec{a}) = \prod_{j=1}^n p_a(a_j) // which\ assume\ the\ sources\ are\ indep. \quad (10.141)$$

$$= \prod_{j=1}^n p_a(\vec{w}_j^T \vec{x}) |W| // \vec{x} = U\vec{a}, \vec{a} = W\vec{x} \quad (10.142)$$

$$(10.143)$$

all that remains is to specify a density for the individual sources $p_a(a)$, also, the density of a can be found from the cdf

$$p_a(a) = F'(a) \quad (10.144)$$

all we need to do is to specify some cdf for all a_j 's

- a cdf has to be a monotonic function that increases from zero to one
- we cannot choose the cdf to be the cdf of the Gaussian, as ICA doesn't work on Gaussian data

- a reasonable “default” function that slowly increases from 0 to 1, is the sigmoid function

$$F(a_j) = g(a_j) = \frac{1}{1 + \exp(-a_j)} \quad (10.145)$$

- the presentation here assumes that a_j has zero-mean $\rightarrow \vec{x}^{(i)}$ has been preprocessed to have zero mean
- this assumption is natural because acoustic signals are expected to have zero-mean

$$p_a(a_j) = g'(a_j) = g(a_j)g(-a_j) \quad (10.146)$$

MLE

$$W^* = \arg \max_W l(W) \quad (10.147)$$

$$= \arg \max_W \log \prod_{i=1}^m p(\vec{x}^{(i)}; W) \quad // \text{assume } \vec{x}^{(i)} \text{'s are indep.} \quad (10.148)$$

$$= \arg \max_W \log \prod_{i=1}^m \prod_{j=1}^n p_a(\vec{w}_j^T \vec{x}^{(i)}) |W| \quad (10.149)$$

$$= \arg \max_W \sum_{i=1}^m \sum_{j=1}^n \log p_a(\vec{w}_j^T \vec{x}^{(i)}) + \sum_{i=1}^m \log |W| \quad (10.150)$$

$$= \arg \max_W \sum_{i=1}^m \sum_{j=1}^n \log g'(\vec{w}_j^T \vec{x}^{(i)}) + \sum_{i=1}^m \log |W| \quad (10.151)$$

$$= \arg \min_W - \sum_{i=1}^m \sum_{j=1}^n \log g'(\vec{w}_j^T \vec{x}^{(i)}) - \sum_{i=1}^m \log |W| \quad (10.152)$$

$$= \arg \min_W \sum_{i=1}^m \text{err}(W, \vec{x}^{(i)}, y^{(i)}) \quad (10.153)$$

note this assumption is clearly incorrect

- for speech data and other time series where the $\vec{x}^{(i)}$'s are dep.
 - but if we have sufficient data, it can be shown that having correlated training examples will not hurt the performance of the algorithm
- the SGD rule is

$$W \leftarrow W - \alpha \nabla_W \text{err}(W, \vec{x}^{(i)}, y^{(i)}) \quad (10.154)$$

$$= W + \alpha \left(\begin{bmatrix} 1 - 2g(\vec{w}_1^T \vec{x}^{(i)}) \\ 1 - 2g(\vec{w}_2^T \vec{x}^{(i)}) \\ \vdots \\ 1 - 2g(\vec{w}_n^T \vec{x}^{(i)}) \end{bmatrix} \vec{x}^{(i)T} + W^{-T} \right) \quad (10.155)$$

categorical features

for problems where successive training examples are correlated, when implementing SGD, it also sometimes helps accelerate convergence if we visit training examples in a randomly permuted order

10.8 Recommender Systems

10.8.1 Content-Based Recommendations

predicting movie ratings

- data: how “many users” have rated “some movies”
- skill: predict how a user would rate an unrated movie formulation
 - m : # movies
 - n : # users
 - $r(i, j) = 1$ if user j has rated movie i
 - $y^{(i,j)} \in \{0, 1, 2, 3, 4, 5\}$: rating given by user j to movie i (defined only if $r(i, j) = 1$)

feature are very abstract: user # j rates movie # i with $y^{(i,j)}$ scores
problem: how to learn user preference from data?

Encoding of Categorical Feature

user id, movie id are **categorical features**

many ML models operate on numerical features except decision tree
need: encoding (transform) from categorical to numerical

for each user j , learn a para. $\vec{\theta}^{(j)}$, predict user j as rating movie with $\vec{\theta}^{(j)T}\vec{x}^{(i)}$ scores, $\vec{x}^{(i)}$ is the feature vec. for movie $\vec{x}^{(i)}$

- $\vec{x}^{(i)}$, eg., x_1 measures how much romance in that movie, x_2 measures how much action in that movie
- $\vec{\theta}^{(j)}$, eg., θ_1 measures how that user likes romance movie, θ_2 measures how that user likes action movie

10.8.2 Collaborative Filtering

Optimization Objective

$\min. \vec{x}^{(1)}, \vec{x}^{(2)}, \dots, \vec{x}^{(m)}$ and $\vec{\theta}^{(1)}, \vec{\theta}^{(2)}, \dots, \vec{\theta}^{(n)}$ simultaneously

$$\min_{\vec{x}, \vec{\theta}} J(\vec{x}, \vec{\theta}) \quad (10.156)$$

$$= \min_{\vec{x}, \vec{\theta}} \sum_{i=1}^m \sum_{j:r(i,j)=1} (\vec{\theta}^{(j)T}\vec{x}^{(i)} - y^{(i,j)})^2 + \lambda \sum_{i=1}^m \Omega(\vec{x}^{(i)}) + \lambda \sum_{j=1}^n \Omega(\vec{\theta}^{(j)}) \quad (10.157)$$

Collaborative Filtering

coordinate descent on J by $\vec{x}, \vec{\theta}$
given $\vec{\theta}^{(1)}, \vec{\theta}^{(2)}, \dots, \vec{\theta}^{(n)}$, to learn $\vec{x}^{(1)}, \vec{x}^{(2)}, \dots, \vec{x}^{(m)}$: para. for users

$$\min_{\vec{x}^{(1)}, \vec{x}^{(2)}, \dots, \vec{x}^{(m)}} \sum_{i=1}^m \sum_{j:r(i,j)=1} (\vec{\theta}^{(j)T} \vec{x}^{(i)} - y^{(i,j)})^2 + \lambda \sum_{i=1}^m \Omega(\vec{x}^{(i)}) \quad (10.158)$$

given $\vec{x}^{(1)}, \vec{x}^{(2)}, \dots, \vec{x}^{(m)}$, to learn $\vec{\theta}^{(1)}, \vec{\theta}^{(2)}, \dots, \vec{\theta}^{(n)}$: para. for users

$$\min_{\vec{\theta}^{(1)}, \vec{\theta}^{(2)}, \dots, \vec{\theta}^{(n)}} \sum_{j=1}^n \sum_{i:r(i,j)=1} (\vec{\theta}^{(j)T} \vec{x}^{(i)} - y^{(i,j)})^2 + \lambda \sum_{j=1}^n \Omega(\vec{\theta}^{(j)}) \quad (10.159)$$

Collaborative Filtering Algorithm

Algorithm 51 Collaborative Filtering Algorithm

idea: alternating opt. to extract hidden user / movie features

init. $\vec{x}^{(1)}, \dots, \vec{x}^{(m)}, \vec{\theta}^{(1)}, \dots, \vec{\theta}^{(n)}$ to small random values

repeat until convergence

for $i = 1 : m$
 $\vec{x}^{(i)} \leftarrow \vec{x}^{(i)} - 2\alpha \sum_{j:r(i,j)=1} (\vec{\theta}^{(j)T} \vec{x}^{(i)} - y^{(i,j)}) \vec{\theta}^{(j)} - \alpha \lambda \nabla_{\vec{x}^{(i)}} \Omega(\vec{x}^{(i)})$

for $j = 1 : n$
 $\vec{\theta}^{(j)} \leftarrow \vec{\theta}^{(j)} - 2\alpha \sum_{i:r(i,j)=1} (\vec{\theta}^{(j)T} \vec{x}^{(i)} - y^{(i,j)}) \vec{x}^{(i)} - \alpha \lambda \nabla_{\vec{\theta}^{(j)}} \Omega(\vec{\theta}^{(j)})$

return $\vec{\theta}^T \vec{x}$

how to find movie j that is related to movie i ?

find $\vec{x}^{(j)}$ with smallest $\|\vec{x}^{(i)} - \vec{x}^{(j)}\|^2$

10.8.3 Low Rank Matrix Factorization

$$Y = \begin{bmatrix} \vec{\theta}^{(1)T} \vec{x}^{(1)} & \vec{\theta}^{(2)T} \vec{x}^{(1)} & \dots & \vec{\theta}^{(n)T} \vec{x}^{(1)} \\ \vec{\theta}^{(1)T} \vec{x}^{(2)} & \vec{\theta}^{(2)T} \vec{x}^{(2)} & \dots & \vec{\theta}^{(n)T} \vec{x}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ \vec{\theta}^{(1)T} \vec{x}^{(m)} & \vec{\theta}^{(2)T} \vec{x}^{(m)} & \dots & \vec{\theta}^{(n)T} \vec{x}^{(m)} \end{bmatrix} \quad (10.160)$$

$$= \begin{bmatrix} \vec{x}^{(1)T} \\ \vec{x}^{(2)T} \\ \vdots \\ \vec{x}^{(m)T} \end{bmatrix} [\vec{\theta}^{(1)} \ \vec{\theta}^{(2)} \ \dots \ \vec{\theta}^{(n)}] \quad (10.161)$$

$$= X \Theta^T \quad (10.162)$$

10.8.4 Implementational Detail

Mean Normalization

if user j has not seen any movies before

- $r(i, j) = 0, \forall i$
 - the objective function relating j remains the regularization term
 - then the alg. will return $\vec{\theta}^{(j)} = \vec{0}$
 - it means we will predict any movie's score made by user j is 0
- mean normalization

$$\mu_i = \frac{\sum_{j:r(i,j=1)} y^{(i,j)}}{\sum_{j=1}^n r(i,j)}, \forall i \quad (10.163)$$

- extract mean from corresponding Y
- use that Y for training
- predict as $\vec{\theta}^{(j)} \vec{x}^{(i)} + \mu_i$
- this will avoid predicting all zeros

Emphasize Latter Examples

use time-deterministic SGD that visits latter examples last

10.8.5 Summary of Extraction Models

Four Different Methods

see Tab. 10.1

Table 10.1: The four different methods

Probabilistic	Not Probabilistic
Find Subspace	Factor Analysis
Find Groups	Mixture of Gaussian
	k -means

Pros and Cons of Extraction Models

pros

- easy: reduces human burden in designing features
- powerful: if enough hidden units considered

cons

- hard: non-convex opt. problems in general
- overfitting: needs proper regularization / validation

Part III

Neural Network and Deep Learning

Chapter 11

Introduction

11.1 AI, Machine Learning and Deep Learning

11.1.1 Rule-Based Systems

solve problems that can be described by a list of formal, mathematical rules

- abstract and formal tasks are intellectually difficult for human beings
- but relatively straight-forward for computers
- eg., IBMs Deep Blue chess-playing system
 - chess is a very simple world, containing only sixty-four locations and thirty-two pieces that can move in only rigidly circumscribed ways
 - it can be completely described by formal rules, easily provided ahead of time by the programmer

Knowledge Based Approach

hard-code knowledge about the world in formal languages

computer can reason about statements in these formal languages automatically using logical inference rules

Challenge

a person's everyday life requires an immense amount of knowledge about the world

- computers need to capture this same knowledge in order to behave in an intelligent way
- much of this knowledge is subjective and intuitive
- therefore difficult to articulate in a formal way

problems that are easy for people to perform but hard for people to describe formally

representation

- problems that we solve intuitively, that feel automatic
- eg., recognizing spoken words or faces in images

11.1.2 Classic Machine Learning

learning from data

- avoids the need for human formally specifying all of the knowledge that the computer needs
- computer acquire their own knowledge, by extracting patterns from raw data

Challenge

depends heavily on the **representation** of the data they are given

- for many tasks, it is difficult to know what features should be extracted
- eg., detect cars in photographs
we may use presence of a wheel as a feature
but it is difficult to describe exactly what a wheel looks like in terms of pixel values, because of the shallow, sun glaring off, obscuration and so on

11.1.3 Representation Learning

use machine learning to discover not only the mapping from representation to output but also the representation itself

- separate the factors of variation that explain the observed data
eg., detect cars in photographs, the factors of variation include the position of the car, its color, and the angle and brightness of the sun
- better performance than hand-designed representations
- also allow AI systems to rapidly adapt to new tasks
- eg., autoencoder. It converts the input data into a different representation and make the new representation have various nice properties

Challenge

many of the factors of variation influence every single piece of data we are able to observe

- the individual pixels in an image of a red car might be very close to black at night, the shape of the cars silhouette depends on the viewing angle.
- it can be very difficult to extract such high-level, abstract features from raw data

11.1.4 Deep Learning

introducing representations that are expressed in terms of other, simpler representations

- the hierarchy of concepts allows the computer to learn complicated concepts by building them out of simpler ones
- information in a layer's representation of the input necessarily encodes factors of variation that explain the input. It is also used to store state information that helps to execute a program that can make sense of the input.

see Fig. 11.1 for an example

- it is difficult for a computer to understand the meaning of raw sensory input data such as pixel values
- learning or evaluating the direct mapping from a set of pixels to an object identity is very difficult
- deep learning resolves this difficulty by breaking the desired complicated mapping into a series of nested simple mappings, each described by a different layer of the model
 - 1st hidden layer identity edges by comparing the brightness of neighboring pixels
 - 2nd hidden layer search for corners and contours as collection of edges
 - 3rd hidden layer detect object parts by finding specific collections of contours and corners
 - finally recognize objects present in the img.

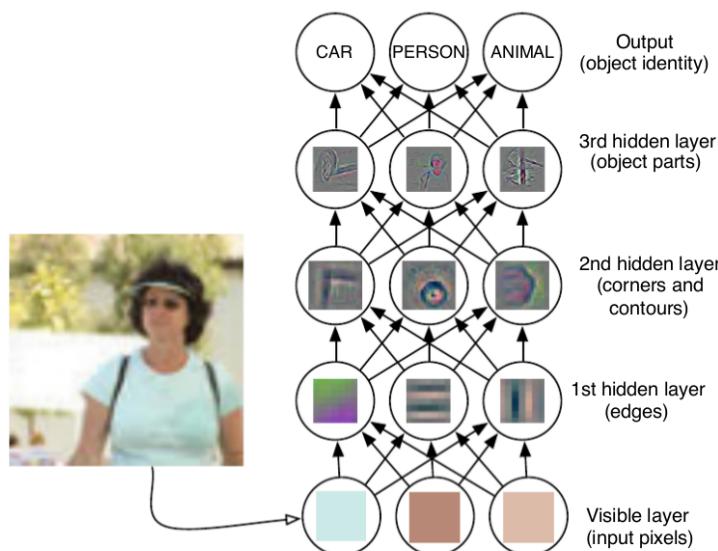


Figure 11.1: Illustration of a deep learning model

kernel engineering

this is the philosophy of connectionism: while an individual biological neuron or an individual feature in a machine learning model is not intelligent, a large population of these neurons or features acting together can exhibit intelligent behavior

Measuring the Depth of a Model

First View number of sequential instructions that must be executed to evaluate the architecture

- is the length of the longest path through a flow chart that describes how to compute each of the model's outputs given its inputs
- it depends on which functions we allow to be used as individual steps

Second View depth of the graph describing how concepts are related to each other

- it may be shallower than the prev. model
- different people choose different sets of smallest elements from which to construct their graphs, there is no single correct value for the depth of an architecture

the 4 different parts of an AI system can be see in Fig. 11.2

11.2 Feature / Representation Learning

11.2.1 Linear Models

linear models such as perceptron, linear regression and logistic regression

- training them involves a convex optimization problem
- Convex optimization comes with some convergence guarantees towards a global optimum, irrespective of initial conditions
- limitation: many tasks, for a given choice of input representation \vec{x} (the raw input features), cannot be solved by using only a linear predictor what are our options to avoid that limitation?

11.2.2 Kernel Methods

use a fixed mapping from $\vec{x} \mapsto \vec{\phi}(\vec{x})$ to a higher dim. space

- optimization remains convex (or even analytic)
- higher $VC(\mathcal{H})$, but at the risk of overfitting
- the choice of $\vec{\phi}$ (or kernel) is a prior (**kernel engineering**)
- Gaussian kernel $k(\vec{x}, \vec{x}') = \exp(-\frac{\|\vec{x}-\vec{x}'\|^2}{2\sigma^2})$ has very broad prior, such as smoothness, but it is sensitive to the curse of dimensionality

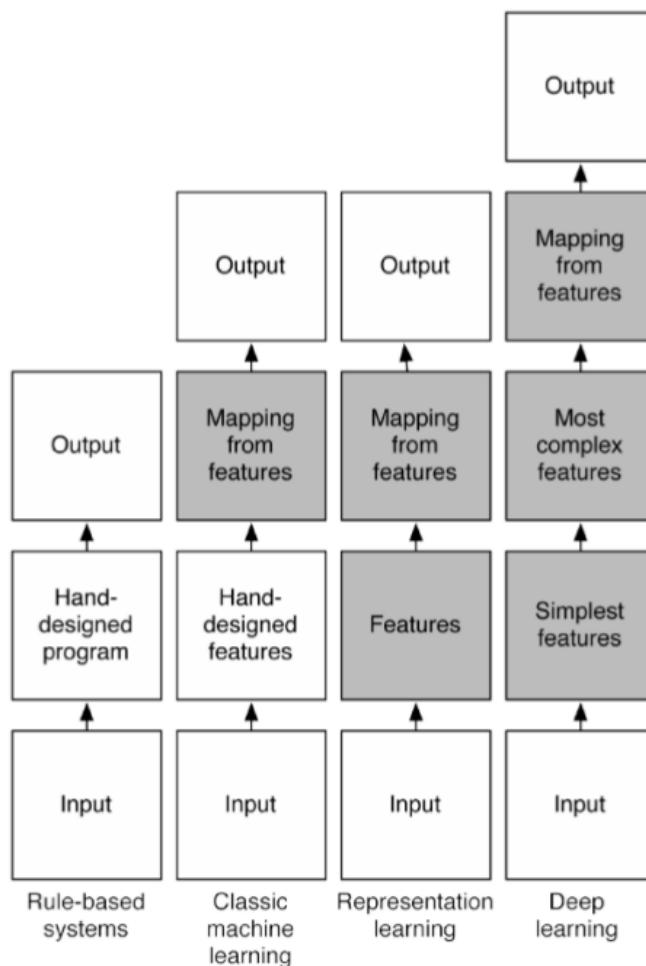


Figure 11.2: The 4 different parts of an AI system. Shaded boxes indicate components that are able to learn from data.

11.2.3 Manually Engineer the Representation or Features

designing new features $\vec{\phi}(\vec{x})$ that are most appropriate to the task at hand

- the prior knowledge can be very useful
- it involves a very task-specific engineering work and a laborious never-ending effort to improve systems by designing better features

we want some more general feature learning approaches that could be applied to a large set of related tasks

11.2.4 Learn the Features / Representation

neural networks allow the feature function $\vec{\phi}(\vec{x})$ to be very rich

- smoothness, depth are prior

11.3 Historical Trends in Deep Learning

11.3.1 The Many Names and Changing Fortunes of Neural Networks

see Fig. 11.3

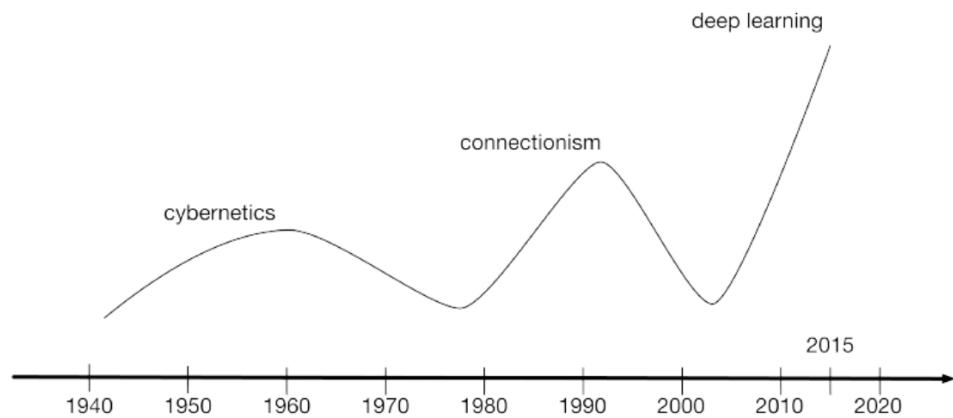


Figure 11.3: Historical Trends in Deep Learning

cybernetics

- perceptron
- first dip: because the limitation of linear models

connectionism

- parallel distributed processing
- BP with 1 or 2 hidden layers
- second dip

failure of excessive promise

improvements of kernels method and graphical models

deep learning

- very deep networks
- greedy layer-wise pretraining

11.3.2 Increasing Dataset Sizes

deep learning has become more useful as the amount of available training data has increased

- the amount of skill required reduces as the amount of training data increases

- the age of “Big Data” has made machine learning much easier because the key burden of statistical estimation—generalizing well to new data after observing only a small amount of data—has been considerably lightened

11.3.3 Increasing Model Sizes

deep learning models have grown in size over time as computer hardware and software infrastructure for deep learning has improved

- since the introduction of hidden units, artificial neural networks have doubled in size roughly every 2.4 years

11.3.4 Increasing Accuracy, Application Complexity and Real-World Impact

deep learning has solved increasingly complicated applications with increasing accuracy over time

- object recognition
- speech recognition
- pedestrian detection
- image segmentation
- machine translation
- self-programming

Chapter 12

Feedforward Deep Network

Feedforward Deep Networks, also known as **Multilayer Perceptrons (MLPs)**

12.1 MLPs from 1980's

Linear Aggregation of Perceptrons: Pictorial View

see Fig. 12.1

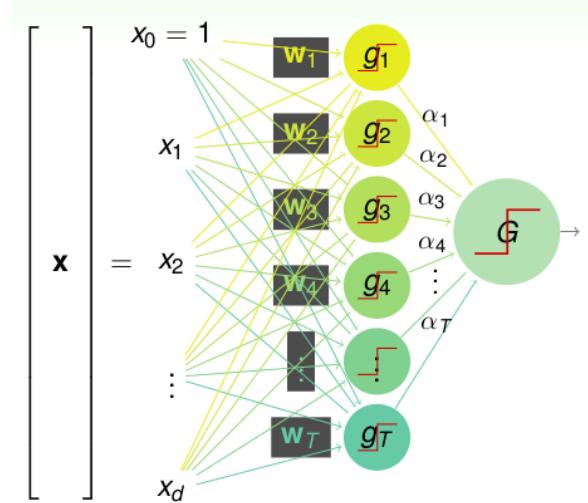


Figure 12.1: Linear Aggregation of Perceptrons: Pictorial View

$$h(\vec{x}) = \text{sign} \left(\sum_{k=1}^K \alpha_k h_k(\vec{x}) \right) = \text{sign} \left(\sum_{k=1}^K \alpha_k \text{sign}(\vec{w}_k^T \vec{x} + b_k) \right) \quad (12.1)$$

- two layers of weights: \vec{w}_k 's and α_k 's

- two layers of sign fcns.: in h_k and h

Logic Operations with Aggregation

And

$$h_1(\vec{x}), h_2(\vec{x}) \in \{-1, 1\}, h(\vec{x}) = h_1(\vec{x}) \wedge h_2(\vec{x}) \quad (12.2)$$

$$h(\vec{x}) = \text{sign}(-1 + h_1(\vec{x}) + h_2(\vec{x})) \quad (12.3)$$

see Tab. 12.1, Fig. 12.2, 12.3

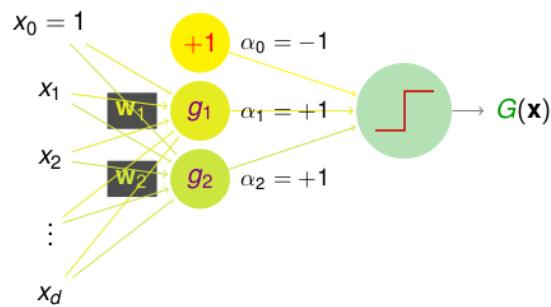


Figure 12.2: And Operation

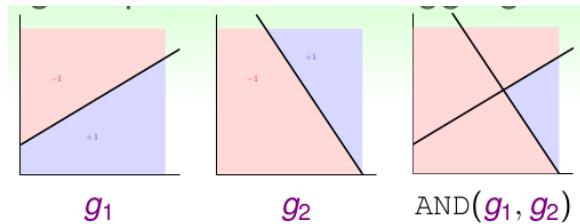


Figure 12.3: And Operation

Table 12.1: And Operation

$h_1(\vec{x})$	$h_2(\vec{x})$	$h(\vec{x})$
1	1	$\text{sign}(1) = 1$
1	-1	$\text{sign}(-1) = -1$
-1	1	$\text{sign}(-1) = -1$
-1	-1	$\text{sign}(-3) = -1$

Or

$$h_1(\vec{x}), h_2(\vec{x}) \in \{-1, 1\}, h(\vec{x}) = h_1(\vec{x}) \vee h_2(\vec{x}) \quad (12.4)$$

$$h(\vec{x}) = \text{sign}(1 + h_1(\vec{x}) + h_2(\vec{x})) \quad (12.5)$$

see Tab. 12.2

Table 12.2: Or Operation

$h_1(\vec{x})$	$h_2(\vec{x})$	$h(\vec{x})$
1	1	$\text{sign}(3) = 1$
1	-1	$\text{sign}(1) = 1$
-1	1	$\text{sign}(1) = 1$
-1	-1	$\text{sign}(-1) = -1$

Powerfulness and Limitation

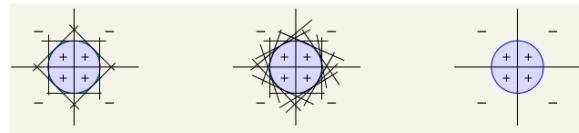


Figure 12.4: Powerfulness

powerfulness:

- can implement cvx. set, see Fig. 12.4: $VC(\mathcal{H}) = \infty$
 - enough perceptrons \approx smooth boundary
- limitation: XOR not ‘lin. sep.’ under

$$h(\vec{x}) = \text{sign}\left(\sum_{k=1}^K \alpha_k \text{sign}(\vec{w}_k^T \vec{x} + b_k)\right) \quad (12.6)$$

Multi-Layer Perceptrons: Basic Neural Network

non-sep. data: can use more transform, see Fig. 12.5

$$h(\vec{x}) = h_1(\vec{x}) \oplus h_2(\vec{x}) = (-h_1(\vec{x}) \wedge h_2(\vec{x})) \vee (h_1(\vec{x}) \wedge -h_2(\vec{x})) \quad (12.7)$$

perceptrons: simple

aggregation of perceptrons: powerful

multi-layer perceptrons: more powerful

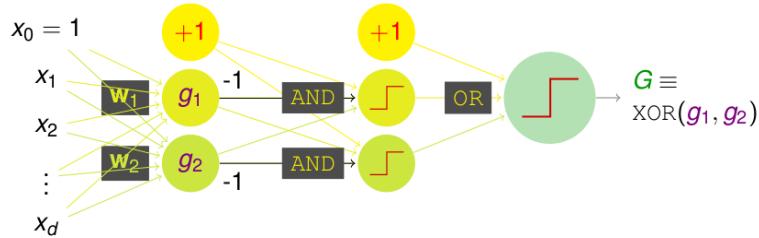


Figure 12.5: Multi-Layer Perceptrons

Non-linear Hypothesis

consider computer vision car detection problem

input: 50×50 pixel images $\rightarrow \vec{x} \in \mathbb{R}^{2500}$

how about using feature transform?

- recall computation / storage price to transform to \tilde{n} -th order: $O(\tilde{n}^n)$
- consider quad. features $x_i \times x_j$: very large when $n = 2500$

Connection to Biological Neurons

origins: algorithms that try to mimic the brain, see Fig. 12.6

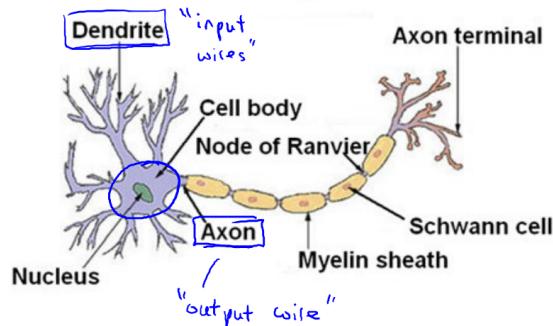


Figure 12.6: Neuron in the brain

was very widely used in 80s and early 90s; popularity diminished in late 90s.

recent resurgence: State-of-the-art technique for many applications

12.2 Parametrizing a Learned Predictor

Sigmoid
Hyperbolic Tangent

12.2.1 Family of Functions

Output

simply a lin. model with score

$$s = \vec{\theta}^T \vec{\phi}^{(2)}(\vec{\phi}^{(1)}(\vec{x})) \quad (12.8)$$

any lin. model can be used

Transformation

how about s ? whole network linear & thus less useful
how about

$$a(s) = \text{sign}(s) \quad (12.9)$$

- first compute a weighted sum of the inputs
- then send out a fixed size spike of activity if the weighted sum exceeds a threshold
- each spike is like the truth value of a proposition and each neuron combines truth values to compute the truth value of another proposition
- discrete & thus hard to optimize for W, \vec{b}

Define 12.1 (Sigmoid).

$$a(s) = \sigma(s) = \frac{1}{1 + \exp(-s)} \in [0, 1] \quad (12.10)$$

$$g'(s) = g(s)(1 - g(s)) \quad (12.11)$$

- ‘analog’ approximation of $\text{sign}(s)$
- easy to opt.
- somewhat closer to biological neuron
- treat the output as the prob. of producing a spike in a short time window

Define 12.2 (Hyperbolic Tangent).

$$a(s) = \tanh s = \frac{\exp(s) - \exp(-s)}{\exp(s) + \exp(-s)} = 2\sigma(2s) - 1 \in [-1, 1] \quad (12.12)$$

$$a'(s) = 1 - a^2(s) \quad (12.13)$$

- linearly related to sigmoid
- easy to opt.

*Rectified Linear
Neurons, ReLU,
Positive Part*

Leaky ReLU

Parametric ReLU,
PReLU

Define 12.3 (Rectified Linear Neurons, ReLU, Positive Part).

$$a(s) = \max(0, s) \stackrel{\text{def}}{=} s^+ \quad (12.14)$$

$$a'(s) = \begin{cases} 0 & \text{if } s < 0 \\ 1 & \text{if } s > 0 \end{cases} \quad (12.15)$$

- non-lin. fcn. of the lin. weighted sum of their inputs
- it is not bounded or continuously differentiable
- the gradient is undefined at $s = 0$, though this doesn't cause problems in practice because we average the gradient over many training examples during optimization.
- works better in practice for deep neural networks, most popular
- variant of ReLU

$$a(s) = \max(0, s) + \alpha \min(0, s) \quad (12.16)$$

Leaky ReLU : fixes α be a small value like 0.01

Parametric ReLU, PReLU : treats α as a learnable parameter

- the output, if its input is above zero, is treated as the Poisson rate for spikes
- once we've figured out these rate of producing spikes, the actual times at which spikes are produced is a random process
- see Fig. 12.7 for comparation

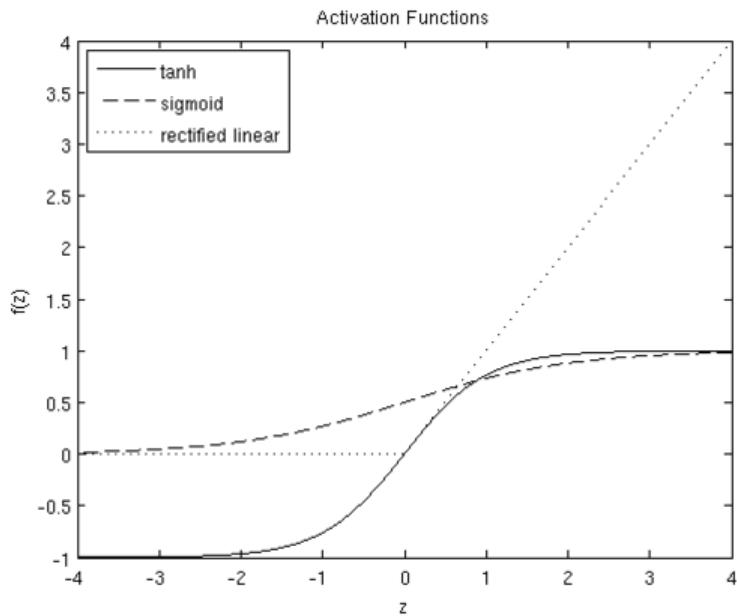


Figure 12.7: plots of the sigmoid, tanh and rectified linear functions

Define 12.4 (Hard tanh).

$a(s) = \max(-1, \min(1, s))$ <ul style="list-style-type: none"> • has similar shape to tanh • but it is bounded 	<i>Hard tanh</i> <i>Softplus</i> <i>Absolute Value Rectification</i> <i>Maxout</i> <i>Softmax</i> <i>Radial Basis Function, RBF</i>
--	--

Define 12.5 (Softplus).

$$a(s) = \zeta(s) = \log(1 + \exp(s)) \quad (12.18)$$

- smooth version of ReLU
- but ReLU has better results

Define 12.6 (Absolute Value Rectification).

$$a(s) = |s| \quad (12.19)$$

- used for object recognition from images
- it makes sense to seek features that are invariant under a polarity reversal of the input illumination

Define 12.7 (Maxout). *It uses multiple weight vectors \vec{w}_j (called filters) for each hidden unit*

$$a = \max_j(\vec{w}_j^T \vec{x} + b_j) \quad (12.20)$$

Define 12.8 (Softmax).

$$a_j = \frac{\exp(s_j)}{\sum_{k=1}^n \exp(s_k)} \in (0, 1) \quad (12.21)$$

$$\sum_{j=1}^n a_j = 1 \quad (12.22)$$

- can be considered as a probability distribution over a finite set of outcomes
- mostly used as output non-linearity for predicting discrete probabilities over output categories
- softmax's output is invariant to adding a scalar to all of its inputs

$$\text{softmax}(\vec{s}) = \text{softmax}(\vec{s} + c), \forall c \quad (12.23)$$

by set $c = -\max_j s_j$, this allows us to evaluate softmax with only small numerical errors even when \vec{s} contains extremely large or extremely negative numbers

Define 12.9 (Radial Basis Function, RBF).

$$a_j = \exp\left(-\frac{\|\vec{w}_j - \vec{x}\|^2}{\sigma_j}\right) \quad (12.24)$$

- heavily used in kernel SVM
- such \vec{w}_j 's can be easily initialized as a random subset of the input examples

catastrophic forgetting

12.2.2 Piecewise Linear Hidden Units

use piecewise linear units, such as absolute value rectifiers and rectified linear units

- such units consist of two linear pieces and their behavior is driven by a single weight vector
- it is easy for an optimization algorithm to tell how to improve the behavior of a unit
- easy to regularize
- traditional units such as sigmoids are more prone to discarding information due to saturation both in forward propagation and in back-propagation

but piecewise linear units cannot learn via gradient-based methods on examples for which their activation is zero

- can be mitigated by initializing the biases to a small positive number, but it is still possible for a rectified linear unit to learn to de-activate and then never be activated again
- maxout units are also piecewise linear, but each piece of the linear function has its own weight vector, so whichever piece is active can always learn
 - but maxout has greater number of weight vectors, so extra regularization such as dropout is needed
 - in practise works well if the training set is large and the number of pieces per unit is kept low
 - if the features captured by n different linear filters can be summarized without losing information by taking the max over each group of n' features, then the next layer can get by with n' times fewer weights
 - because each unit is driven by multiple filters, maxout units have some redundancy that helps them to resist forgetting how to perform tasks that they were trained on in the past
- neural networks trained with stochastic gradient descent are generally believed to suffer from a phenomenon called **catastrophic forgetting**, but maxout units tend to exhibit only mild forgetting
 - with large enough n' , a maxout unit can learn to approximate any convex function with arbitrary fidelity
 - in particular, maxout with two pieces can learn to implement the rectified linear activation function or the absolute value rectification function
- leaky or parametric ReLU: by having a small slope rather than a zero slope when the argument of the rectifier is negative, gradients pass all the time

12.2.3 Error Function and Conditional Log-Likelihood

Define 12.10 (Squared Error). *For regression problem,*

$$\text{err}(\vec{\theta}, \vec{x}, y) = (h(\vec{x}) - y)^2 \quad (12.25)$$

min. the expected value of the err. function over some data-generating distribution p yields $h(\vec{x}) = E[y|\vec{x}]$.

<i>Squared Error</i>
<i>Absolute Error</i>
<i>Cross Entropy Error, Bernoulli Negative Log-Likelihood</i>
<i>Softmax Error</i>
competition winner takes all

Define 12.11 (Absolute Error).

$$\text{err}(\vec{\theta}, \vec{x}, y) = |h(\vec{x}) - y| \quad (12.26)$$

min. the expected value of the err. function over some data-generating distribution p yields $h(\vec{x}) = \text{median}(y|\vec{x})$.

Define 12.12 (Cross Entropy Error, Bernoulli Negative Log-Likelihood). *For classification problem, $y \in \{0, 1\}$, the output layer is sigmoid,*

$$\text{err}(\vec{\theta}, \vec{x}, y) = -y \log h(\vec{x}) - (1 - y) \log(1 - h(\vec{x})) \quad (12.27)$$

min. the expected value of the err. function over some data-generating distribution p yields $h(\vec{x}) = \Pr(y = 1|\vec{x})$.

Define 12.13 (Softmax Error). *For multi-label classification problem, $y \in \{1, 2, \dots, K\}$*

$$\text{err}(\vec{\theta}, \vec{x}, y) = -\log h(\vec{x})_y \quad (12.28)$$

$$h(\vec{x})_j = \frac{\exp(s_j)}{\sum_{k=1}^K \exp(s_k)} \quad (12.29)$$

we often set $s_K = 0$ in order to specify only $K - 1$ of the output. Min. the expected value of the err. function over some data-generating distribution p yields $h(\vec{x})_j = \Pr(y = j|\vec{x})$ and $h(\vec{x})_K = 1 - \sum_{j=1}^{K-1} \Pr(y = j|\vec{x})$.

$$\min \text{err}(\vec{\theta}, \vec{x}, y) = \max \frac{\exp(s_y)}{\sum_{k=1}^K \exp(s_k)}$$

→ numerator↑, denominator↓

↔ $s_y \uparrow, s_j \downarrow, \forall j \neq y$, it is like a **competition** betw. units

in the extreme case, s_y is very large, $s_j, \forall j \neq y$ is very small, it is called **winner takes all**

$$\frac{\partial \text{err}(\vec{\theta}, \vec{x}, y)}{\partial s_j} = -\frac{\partial}{\partial s_j} \log h(\vec{x})_y \quad (12.30)$$

$$= -\frac{\partial}{\partial s_j} \left(\log \frac{\exp(s_y)}{\sum_{k=1}^K \exp(s_k)} \right) \quad (12.31)$$

$$= -\frac{\partial}{\partial s_j} (s_y - \log \sum_{k=1}^K \exp(s_k)) \quad (12.32)$$

$$= -1\{y = j\} + \frac{\exp(s_j)}{\sum_{k=1}^K \exp(s_k)} \quad (12.33)$$

$$= h(\vec{x})_j - 1\{y = j\} \quad (12.34)$$

$$\nabla_{\vec{s}} \text{err}(\vec{\theta}, \vec{x}, y) = \vec{h}(\vec{x}) - \vec{e}_y \quad (12.35)$$

- $\vec{e}_y = [0 \ 0 \ \dots \ 1 \ \dots \ 0]^T$ with a 1 at position y
 - for $j \neq y$, $\frac{\partial \text{err}(\vec{\theta}, \vec{x}, y)}{\partial s_j} = h(\vec{x})_j$
if predict is correct $\iff h(\vec{y})_j \approx 0 \rightarrow \text{grad.} \approx 0$
if predict is wrong $\iff h(\vec{y})_j$ is high $\rightarrow \text{grad.} \approx 1 \rightarrow$ there is a strong push to reduce s_j
 - for $j = y$, $\frac{\partial \text{err}(\vec{\theta}, \vec{x}, y)}{\partial s_j} = h(\vec{x})_j - 1$
if predict is correct $\iff h(\vec{y})_j \approx 1 \rightarrow \text{grad.} \approx 0$
if predict is wrong $\iff h(\vec{y})_j$ is low $\rightarrow \text{grad.} \approx -1 \rightarrow$ there is a strong push to increase s_j
- how about multi-label classification with squared error?
the last layer is still the softmax layer
encode the label as

$$\vec{y} \in \mathbb{R}^K, \text{eg., } \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \quad (12.36)$$

$$\text{err}(\vec{\theta}, \vec{x}, y) = \|\vec{h}(\vec{x}) - \vec{y}\|^2 \quad (12.37)$$

$$\frac{\partial \text{err}(\vec{\theta}, \vec{x}, y)}{\partial s_j} = (\frac{\partial \text{err}(\vec{\theta}, \vec{x}, y)}{\partial \vec{h}(\vec{x})})^T (\frac{\partial \vec{h}(\vec{x})}{\partial s_j}) \quad (12.38)$$

$$= (\nabla_{\vec{h}(\vec{x})} (\vec{h}(\vec{x})^T \vec{h}(\vec{x}) - 2\vec{y}^T \vec{h}(\vec{x}) + \vec{y}^T \vec{y}))^T \nabla_{s_j} \vec{h}(\vec{x}) \quad (12.39)$$

$$= 2(\vec{h}(\vec{x}) - \vec{y})^T \nabla_{s_j} \vec{h}(\vec{x}) \quad (12.40)$$

$$= \sum_{k=1}^K 2(h(\vec{x})_k - y_k) h(\vec{x})_k (1\{k=j\} - h(\vec{x})_j) \quad (12.41)$$

for $j = y$, if predict is wrong

$\iff h(\vec{y})_j$ is low

$$\rightarrow \frac{\partial \text{err}(\vec{\theta}, \vec{x}, y)}{\partial s_j} = \sum_{k=1}^K 2(h(\vec{x})_k - y_k) h(\vec{x})_k (1\{k=j\} - h(\vec{x})_j) = 2(h(\vec{x})_j - 1) h(\vec{x})_j \approx 0$$

\rightarrow it does not push s_j up

for this reason, softmax error is more preferred

12.3 Flow Graphs and Back-Propagation

12.3.1 Neural Network Hypothesis

Neural Network Hypothesis

see Fig. 12.8

$W^{(l)} \in \mathbb{R}^{n^{(l+1)} \times n^{(l)}}$, $b^{(l)} \in \mathbb{R}^{n^{(l+1)}}$: mat. of weights controlling mapping from layer l to $l + 1$

- $1 \leq l \leq L - 1$: layers
- $1 \leq j \leq n^{(l)}$: inputs
- $1 \leq k \leq n^{(l+1)}$: outputs
- $s^{(l)} \in \mathbb{R}^{n^{(l)}}$: score
- $1 \leq l \leq L$: layers
- $1 \leq k \leq n^{(l)}$: unit

$$s_k^{(l+1)} = W_{k1}^{(l)} a_1^{(l)} + \dots + W_{kn^{(l)}}^{(l)} a_{n^{(l)}}^{(l)} + b_k^{(l)} \quad (12.42)$$

$$= \sum_{j=1}^{n^{(l)}} W_{kj}^{(l)} a_j^{(l)} + b_k^{(l)} \quad (12.43)$$

$$= W_k^{(l)} \vec{a}^{(l)} + b_k^{(l)} \quad (12.44)$$

$$\vec{s}^{(l+1)} = W^{(l)} \vec{a}^{(l)} + \vec{b} \quad (12.45)$$

$a^{(l)} \in \mathbb{R}^{n^{(l)}}$: activation

- $1 \leq l \leq L$: layers
- $1 \leq j \leq n^{(l)}$: units
- $\vec{a}^{(1)} = \vec{x}$
- $\vec{h}(\vec{x}) = \vec{a}^{(L)}$
- the non-linearity may be diff. on diff. layers

$$\vec{a}^{(l)} = \vec{g}(s^{(l)}) \quad (12.46)$$

Physical Interpretation

each layer: transformation to be learned from data

$$\vec{\phi}^{(l)}(\vec{x}) = \vec{a}^{(l)} = \vec{g}(\vec{s}^{(l)}) \quad (12.47)$$

NNet: learns its own features with layers of connection weights

12.3.2 Neural Network Learning

Forward Propagation

- Ω is a regularizer
- when in practise, implement these equations on minibatches

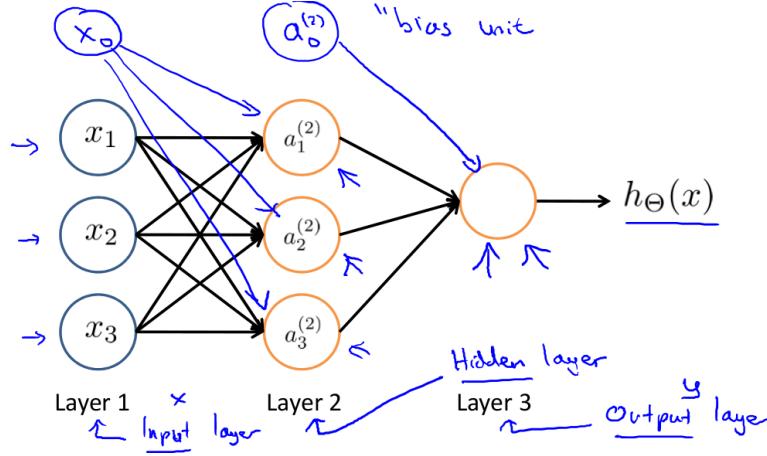


Figure 12.8: Neural Network Hypothesis

Algorithm 52 Forward Propagation

```

 $\vec{a}^{(1)} = \vec{x}$ 
for  $l = 1 : L - 1$ 
     $\vec{s}^{(l+1)} = W^{(l)}\vec{a}^{(l)} + \vec{b}^{(l)}$ 
     $\vec{a}^{(l+1)} = \vec{g}(\vec{s}^{(l+1)})$ 
return  $\vec{h}(\vec{x}) = \vec{a}^{(L)}$ , err( $W, \vec{b}, \vec{x}, y$ ) +  $\lambda\Omega$ 

```

- rather than using a vector $\vec{s}^{(l)}, \vec{a}^{(l)}$, an efficient implementation should use a matrix with the additional dimension representing the index of an example within the minibatch
- the err. function will become cost function J which average err. across examples in the minibatch
- reasonable default architecture 1 hidden layer
or if > 1 hidden layers, have same # of hidden units in every layer

Computing the gradient goal: learn all $W_{kj}^{(l)}, b_k^{(l)}$'s to min. $\text{err}(W, \vec{b}, \vec{x}, y) + \lambda\Omega$

for hidden layer $1 \leq l \leq L - 1$

$$\frac{\partial(\text{err} + \lambda\Omega)}{\partial W_{kj}^{(l)}} = \frac{\partial \text{err}}{\partial s_k^{(l+1)}} \frac{\partial s_k^{(l+1)}}{\partial W_{kj}^{(l)}} + \lambda \frac{\partial \Omega}{\partial W_{kj}^{(l)}} \quad (12.48)$$

$$\stackrel{\text{def}}{=} \delta_k^{(l+1)} \frac{\partial(W_{k*}^{(l)} \vec{a}^{(l)} + b_k^{(l)})}{\partial W_{kj}^{(l)}} + \lambda \frac{\partial \Omega}{\partial W_{kj}^{(l)}} \quad (12.49)$$

$$= \delta_k^{(l+1)} a_j^{(l)} + \lambda \frac{\partial \Omega}{\partial W_{kj}^{(l)}} \quad (12.50)$$

$$\nabla_{W^{(l)}} (\text{err} + \lambda\Omega) = \delta^{(l+1)} (a^{(l)})^T + \lambda \nabla_{W^{(l)}} \Omega \quad (12.51)$$

$$\frac{\partial(\text{err} + \lambda\Omega)}{\partial b_k^{(l)}} = \frac{\partial\text{err}}{\partial s_k^{(l+1)}} \frac{\partial s_k^{(l+1)}}{\partial b_k^{(l)}} + \lambda \frac{\partial\Omega}{\partial b_k^{(l)}} \quad (12.52)$$

$$\stackrel{\text{def}}{=} \delta_k^{(l+1)} \frac{\partial(W_{k*}^{(l)} \vec{a}^{(l)} + b_k^{(l)})}{\partial b_k^{(l)}} + \lambda \frac{\partial\Omega}{\partial b_k^{(l)}} \quad (12.53)$$

$$= \delta_k^{(l+1)} + \lambda \frac{\partial\Omega}{\partial b_k^{(l)}} \quad (12.54)$$

$$\nabla_{\vec{b}^{(l)}}(\text{err} + \lambda\Omega) = \delta^{(l+1)} + \lambda\nabla_{\vec{b}^{(l)}}\Omega \quad (12.55)$$

$$(12.56)$$

Computing $\delta_k^{(l+1)}$

$$s_j^{(l)} \xrightarrow{g} a_j^{(l)} \xrightarrow{W^{(l)}} \begin{bmatrix} s_1^{(l+1)} \\ s_2^{(l+1)} \\ \vdots \\ s_n^{(l+1)} \end{bmatrix} \rightarrow \dots \rightarrow \text{err}(W, \vec{b}, \vec{x}, y) \quad (12.57)$$

$$\delta_j^{(l)} = \frac{\partial\text{err}}{\partial s_j^{(l)}} \quad (12.58)$$

$$= \frac{\partial\text{err}}{\partial \vec{s}^{(l+1)}} \frac{\partial \vec{s}^{(l+1)}}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial s_j^{(l)}} \quad (12.59)$$

$$= (\vec{\delta}^{(l+1)})^T W_{*j}^{(l)} g'(s_j^{(l)}) \quad (12.60)$$

$$= (W_{*j}^{(l)})^T \vec{\delta}^{(l+1)} g'(s_j^{(l)}) \quad (12.61)$$

$$\vec{\delta}^{(l)} = (W^{(l)})^T \vec{\delta}^{(l+1)} * \vec{g}'(\vec{s}^{(l)}) \quad (12.62)$$

$\vec{\delta}^{(l)}$ can be computed backwards from $\vec{\delta}^{(l+1)}$

Backpropagation Algorithm

Algorithm 53 Back Propagation

$$\vec{\delta}^{(L)} = \frac{\partial\text{err}}{\partial s^{(L)}}$$

for $l = L - 1 : 1$

$$\nabla_{W^{(l)}}(\text{err} + \lambda\Omega) = \delta^{(l+1)}(a^{(l)})^T + \lambda\nabla_{W^{(l)}}\Omega$$

$$\nabla_{\vec{b}^{(l)}}(\text{err} + \lambda\Omega) = \delta^{(l+1)} + \lambda\nabla_{\vec{b}^{(l)}}\Omega$$

$$\vec{\delta}^{(l)} = (W^{(l)})^T \vec{\delta}^{(l+1)} * \vec{g}'(\vec{s}^{(l)})$$

when in practise, implement these equations on minibatches

Flow Graph

12.3.3 Learning in a General Flow Graph

Define 12.14 (Flow Graph). Suppose the network formed a DAG, we call it a flow graph. The nodes satisfy a partial order which dictates in what order the computation can proceed. Each node a_k of the graph depends on $a_j, a_j \in a_k.parent$, also denote those a_j 's as $a_{k,i}$. The final node $a_N = \text{err}(W, \vec{b}, \vec{x}, y)$. See Fig. 12.9

$$a_k = g(\{a_k.parent\}) \quad (12.63)$$

the non-linearity function g may be diff. on diff. nodes.

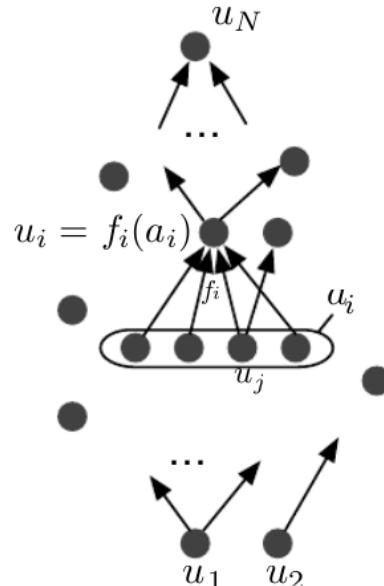


Figure 12.9: A flow graph.

Forward Propagation

Algorithm 54 Forward Propagation in a Flow Graph

for $j = 1 : n$

$a_j = x_j$

for $k = n + 1 : N$

$a_k = g(\{a_k.parent\})$

return a_N

Computing the Gradient

note that if a_j is a parent of a_k , a_j could influence a_k through multiple paths

- $\frac{\partial a_k}{\partial a_j}$ denote the total gradient adding up all of these influences
- $\frac{\partial a_k}{\partial a_{k,i}}$ only denotes the derivative of a_k wrt. its specific i -th argument, keeping the other arguments fixed, ie., only considering the influence through the arc from a_k to $a_{k,i}$

Backward Propagation

Algorithm 55 Backward Propagation in a Flow Graph

```
 $\frac{\partial a_N}{\partial a_N} = 1$ 
for  $j = N - 1 : 1$ 
 $\frac{\partial a_N}{\partial a_j} = \sum_{k: a_j \in a_k.parent} \frac{\partial a_N}{\partial a_k} \frac{\partial a_k}{\partial a_{k,i}}$  //  $a_{k,i}$  is  $a_j$  as a parent of  $a_k$ 
```

- this is dynamic programming, memorize all the $\frac{\partial a_N}{\partial a_k}$
- the cost of the overall computation is proportional to the number of arcs in the graph
- this is of the same order as the number of computations for the forward propagation
- when a_k 's are vectors, all equations holds true

12.3.4 Implantation Notes

Unrolling Parameters

unroll $W^{(l)}, \vec{b}^{(l)}$, and the gradients DW, Db into vectors
eg. $n_1 = 10, n_2 = 10, n_3 = 10, n_4 = 1$
 $W^{(1)} \in \mathbb{R}^{10 \times 10}, W^{(2)} \in \mathbb{R}^{10 \times 10}, W^{(3)} \in \mathbb{R}^{1 \times 10}$
 $\vec{b}^{(1)} \in \mathbb{R}^{10}, \vec{b}^{(2)} \in \mathbb{R}^{10}, \vec{b}^{(3)} \in \mathbb{R}$
 $DW^{(1)} \in \mathbb{R}^{10 \times 10}, DW^{(2)} \in \mathbb{R}^{10 \times 10}, DW^{(3)} \in \mathbb{R}^{1 \times 10}$
 $Db^{(1)} \in \mathbb{R}^{10}, Db^{(2)} \in \mathbb{R}^{10}, Db^{(3)} \in \mathbb{R}$

```
1 % init. W1, W2, W3, b1, b2, b3
2 theta = [W1(:); b1(:); W2(:); b2(:); W3(:); b3(:)];
3 fminunc(@costFcn, thetaVec, options)
```

```
1 function [J, grad] = costFcn(theta)
2     W1 = reshape(theta(1: 100), 10, 10);
3     b1 = reshape(theta(101: 110), 10, 1);
4     W2 = reshape(theta(111: 210), 10, 10);
5     b2 = reshape(theta(211: 220), 10, 1);
6     W3 = reshape(theta(221: 230), 1, 10);
7     b3 = reshape(theta(230: 231), 1, 1);
8     % use forward-prop / back-prop to compute
```

reinforce learning

```

9   % J and DW1, DW2, DW3, Db1, Db2, Db3
10  grad = [DW1(:); Db1(:); DW2(:); Db2(:); DW3(:); Db3(:)]

```

12.4 Back-Propagation Through Random Operations and Graphical Models

think of the network as defining a sampling process that deterministically transforms some random values

consider the operation consisting of drawing samples x from a Gaussian distribution

$$x \sim N(\mu, \sigma^2) \quad (12.64)$$

because an individual sample of x is not produced by a function, but rather by a sampling process, it may seem counterintuitive to take the derivatives of x wrt. to the parameters μ, σ

however, we can rewrite the sampling process as a deterministic transform of an underlying random value to obtain a sample from the desired distribution

$$z \sim N(0, 1) \quad (12.65)$$

$$x = \mu + \sigma z \quad (12.66)$$

we can introduce para. $\vec{\theta}$ that shape the distr. μ, σ , then

$$x = f(z, \vec{\theta}) \quad (12.67)$$

in NNet, typically choose z to be drawn from some simple distribution, such as a unit uniform or unit Gaussian distribution

it can achieve more complex distributions by allowing the deterministic portion of the network to reshape its input

12.4.1 Back-Propagating Through Discrete Stochastic Operations

when x is discrete, for a given sampled value z , a small change of $\vec{\theta}$ would generally not change $x \rightarrow \text{err}$ will not change \rightarrow straightforward back-prop. is not applicable

idea: use **reinforce learning**

- err will become continuous of $\vec{\theta}$ when we average over the possible samples of z

$$\mathbb{E}[\text{err}(x)] = \sum_x \text{err}(x)p(x) \quad (12.68)$$

- then use SGD or other SGD based opt. tech.

$$\frac{\partial \mathbb{E}[\text{err}(x)]}{\partial \vec{\theta}} = \sum_x \text{err}(x) \frac{\partial p(x)}{\partial \vec{\theta}} \quad // x \text{ is discrete} \rightarrow \frac{\partial f(z, \vec{\theta})}{\partial \vec{\theta}} = \vec{0} \quad (12.69)$$

variance reduction methods

$$= \sum_x \text{err}(x) p(x) \frac{\partial \log p(x)}{\partial \vec{\theta}} \quad (12.70)$$

$$\approx \frac{1}{N} \sum_{x_i \sim p(x)} \text{err}(x_i) \frac{\partial \log p(x_i)}{\partial \vec{\theta}} \quad (12.71)$$

// this is an unbiased Monte-Carlo estimator of the (12.72)

eg., if x consists of a set of x_i 's where $x_i \sim \text{Ber}(\phi_i)$

$$\phi_i = \Pr(x_i = 1; \vec{\theta}) = \sigma(\vec{\theta}) \quad (12.73)$$

$$\frac{\partial \log p(x_i)}{\partial \vec{\theta}} = x_i(1 - \phi_i) + (1 - x_i)\phi_i \quad (12.74)$$

it is the Bernoulli cross-entropy grad.

but this method has a very high variance

- so that many samples of x need to be drawn to obtain a good estimator of the gradient
- or equivalently, if only one sample is drawn, SGD will converge very slowly and will require a smaller learning rate

or use **variance reduction methods**

- compute a baseline that is used to offset $\text{err}(x)$, then the grad. estimator wrt. $\vec{\theta}$ is

$$(\text{err}(x) - b(\vec{\theta})) \frac{\partial \log p(x)}{\partial \vec{\theta}} \quad (12.75)$$

- any offset $b(\vec{\theta})$ that does not depend on x would not change the expectation of the estimated grad.

$$\mathbb{E}[(\text{err}(x) - b(\vec{\theta})) \frac{\partial \log p(x)}{\partial \vec{\theta}}] \quad (12.76)$$

$$= \mathbb{E}[\text{err}(x) \frac{\partial \log p(x)}{\partial \vec{\theta}}] - b(\vec{\theta}) \mathbb{E}[\frac{\partial \log p(x)}{\partial \vec{\theta}}] \quad (12.77)$$

$$= \mathbb{E}[\text{err}(x) \frac{\partial \log p(x)}{\partial \vec{\theta}}] - b(\vec{\theta}) \sum_x p(x) \frac{\partial \log p(x)}{\partial \vec{\theta}} \quad (12.78)$$

$$= \mathbb{E}[\text{err}(x) \frac{\partial \log p(x)}{\partial \vec{\theta}}] - b(\vec{\theta}) \sum_x \frac{\partial p(x)}{\partial \vec{\theta}} \quad (12.79)$$

$$= \mathbb{E}[\text{err}(x) \frac{\partial \log p(x)}{\partial \vec{\theta}}] - b(\vec{\theta}) \frac{\partial}{\partial \vec{\theta}} \sum_x p(x) \quad (12.80)$$

$$= \mathbb{E}[\text{err}(x) \frac{\partial \log p(x)}{\partial \vec{\theta}}] - b(\vec{\theta}) \frac{\partial 1}{\partial \vec{\theta}} \quad (12.81)$$

$$= \mathbb{E}[\text{err}(x) \frac{\partial \log p(x)}{\partial \vec{\theta}}] - b(\vec{\theta}) \vec{0} \quad (12.82)$$

Radial Basis Function
(RBF) kernel

$$= E[\text{err}(x) \frac{\partial \log p(x)}{\partial \vec{\theta}}] \quad (12.83)$$

- find the opt. $b(\vec{\theta})$ by min. the variance of $(\text{err}(x) - b(\vec{\theta})) \frac{\partial \log p(x)}{\partial \vec{\theta}}$

$$b(\vec{\theta})_j = \frac{E[(\text{err}(x) \frac{\partial \log p(x)}{\partial \theta_j})^2]}{E[(\frac{\partial \log p(x)}{\partial \theta_j})^2]} \quad (12.84)$$

- this can be done by using additional outputs for the NNet that compute $\vec{\theta}$
it outputs an estimator of $E[(\text{err}(x) \frac{\partial \log p(x)}{\partial \theta_j})^2]$ and $E[(\frac{\partial \log p(x)}{\partial \theta_j})^2]$
these extra outputs can be trained with the mean squared error objective, using respectively $\text{err}(x) \frac{\partial \log p(x)}{\partial \theta_j}$ and $\frac{\partial \log p(x)}{\partial \theta_j}$ as targets when x is sampled from $p(x)$, for a given $\vec{\theta}$

12.5 Radial Basis Function Network

12.5.1 RBF Network Hypothesis

Gaussian SVM Revisited

$$h(\vec{x}) = \text{sign} \left(\sum_{SV} \alpha_i y^{(i)} \exp \left(-\frac{\|\vec{x} - \vec{x}^{(i)}\|^2}{2\sigma^2} \right) + b \right) \quad (12.85)$$

Gaussian kernel: also called **Radial Basis Function (RBF) kernel**

- radial: only depends on distance between \vec{x} and center $\vec{x}^{(i)}$
- basis function: to be combined

let

$$h_i(\vec{x}) = y^{(i)} \exp \left(-\frac{\|\vec{x} - \vec{x}^{(i)}\|^2}{2\sigma^2} \right) \quad (12.86)$$

$$h(\vec{x}) = \text{sign} \left(\sum_{SV} \alpha_i h_i(\vec{x}) + b \right) \quad (12.87)$$

RBF Network: linear aggregation of selected radial hypotheses

From Neural Network to RBF Network

see Fig. 12.10

- hidden layer different: (inner-product + non-linear function) versus (distance to center $\vec{x}^{(i)}$ + Gaussian)
- output layer same: just linear aggregation

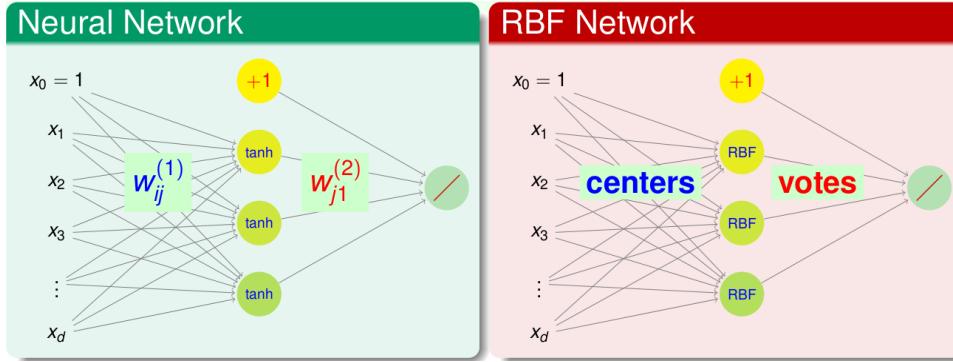


Figure 12.10: From Neural Network to RBF Network

RBF Network Hypothesis

$$h(\vec{x}) = g\left(\sum_{k=1}^K \beta_k \text{RBF}(\vec{x}, \vec{\mu}_k) + b\right) \quad (12.88)$$

- μ_k : centers
- β_k : (signed) votes
- $h(\vec{x})$ for Gaussian-SVM
- RBF: Gaussian
- g : sign (binary classification)
- $K = \#SV$
- $\vec{\mu}_k$: SVM SVs $\vec{x}^{(k)}$
- β_k : $\alpha_k y^{(k)}$ from SVM Dual

RBF and Similarity

kernel: similarity via Z -space inner product

$$\text{Poly}(\vec{x}, \vec{x}') = (1 + \vec{x}^T \vec{x}')^2 \quad (12.89)$$

$$\text{Gaussian}(\vec{x}, \vec{x}') = \exp\left(-\frac{\|\vec{x} - \vec{x}'\|^2}{2\sigma^2}\right) \quad (12.90)$$

RBF: similarity via X -space distance

$$\text{Truncated}(\vec{x}, \vec{x}') = 1\{\|\vec{x} - \vec{x}'\| \leq 1\}(1 - \|\vec{x} - \vec{x}'\|^2) \quad (12.91)$$

often monotonically non-increasing to distance

RBF Network: distance similarity-to-centers as feature transform
general similarity function between \vec{x} and \vec{x}'

$$\text{Neuron}(\vec{x}, \vec{w}) = \sigma(\vec{x}^T \vec{w} + b) \quad (12.92)$$

$$\text{DNASim}(\vec{x}, \vec{x}') = \text{EditDistance}(\vec{x}, \vec{x}') \quad (12.93)$$

nearest neighbor model
 k nearest neighbor

12.5.2 RBF Network Learning

Full RBF Network

$$h(\vec{x}) = g\left(\sum_{i=1}^m \beta_i \text{RBF}(\vec{x}, \vec{x}^{(i)})\right) \quad (12.94)$$

physical meaning: each $\vec{x}^{(i)}$ influences similar \vec{x} by β_i
 eg. uniform influence with $\beta_i = y^{(i)}$ for binary classification

$$h(\vec{x}) = \text{sign}\left(\sum_{i=1}^m y^{(i)} \exp\left(-\frac{\|\vec{x} - \vec{x}^{(i)}\|^2}{2\sigma^2}\right)\right) \quad (12.95)$$

aggregate each example's opinion subject to similarity
 full RBF Network: lazy way to decide $\vec{\mu}_i$

Nearest Neighbor

$$\exp\left(-\frac{\|\vec{x} - \vec{x}^{(i)}\|^2}{2\sigma^2}\right) \quad (12.96)$$

- maximum when \vec{x} closest to $\vec{x}^{(i)}$
- maximum one often dominates the $\sum_{i=1}^m$ term
 take $y^{(i)}$ of maximum $\exp(\cdot)$ instead of voting of all $y^{(i)}$ — selection instead of aggregation
- physical meaning: $h(\vec{x}) = y^{(i)}$ such that \vec{x} closest to $\vec{x}^{(i)}$
- called **nearest neighbor model**
- can uniformly aggregate k neighbors also: **k nearest neighbor** — also lazy but very intuitive

Interpolation by Full RBF Network

full RBF Network for squared error regression:

$$h(\vec{x}) = \sum_{i=1}^m \beta_i \text{RBF}(\vec{x}, \vec{x}^{(i)}) \quad (12.97)$$

- just linear regression on RBF-transformed data

$$\vec{z}^{(i)} = \vec{\phi}(\vec{x}^{(i)}) = \begin{bmatrix} \text{RBF}(\vec{x}^{(i)}, \vec{x}^{(1)}) \\ \text{RBF}(\vec{x}^{(i)}, \vec{x}^{(2)}) \\ \vdots \\ \text{RBF}(\vec{x}^{(i)}, \vec{x}^{(m)}) \end{bmatrix} \quad (12.98)$$

- opt. $\vec{\beta}$? $\vec{\beta} = (Z^T Z)^{-1} Z^T \vec{y}$, if $Z^T Z$ invertible

- $Z \in \mathbb{S}^m$
- theoretical fact: if $\vec{x}^{(i)}$ all different, Z with Gaussian RBF invertible
- optimal $\vec{\beta}$ with invertible Z : $\vec{\beta} = Z^{-1}\vec{y}$

exact interpolation for
function
approximation
*Universal
Approximation*

Regularized Full RBF Network

full Gaussian RBF Network for regression

$$\vec{\beta} = Z^{-1}\vec{y} \quad (12.99)$$

$$h(X) = Z\vec{\beta} = ZZ^{-1}\vec{y} = \vec{y} \quad (12.100)$$

which means $J_{in}(h) = 0$

- called **exact interpolation for function approximation**
- but overfitting for learning
- how about regularization? eg. ridge regression for $\vec{\beta}$ instead

$$\vec{\beta} = (Z^T Z + \lambda I)^{-1} Z^T \vec{y} \quad (12.101)$$

- $Z_{ij} = \text{RBF}(\vec{x}^{(i)}, \vec{x}^{(j)})$, when RBF is Gaussian, $Z = \text{Gaussian kernel matrix } K$, kernel ridge regression

$$\vec{\beta} = (K + \lambda I)^{-1} \vec{y} \quad (12.102)$$

Fewer Centers as Regularization

in SVM, only $\ll m$ SVs needed in network

- therefore: use $m' \ll m$ instead of $m' = m$
- effect: regularization by constraining number of centers and voting weights
- physical meaning of centers $\vec{\mu}_i$: prototypes
- how to extract prototypes? k -means and so on

12.6 Universal Approximation Properties and Depth

linear model

- mapping from features to outputs via matrix multiplication
 - has the advantage of being easy to train because many cost functions result in a convex optimization problem
- but we often want to learn non-linear functions

Theorem 12.1 (Universal Approximation). *A feedforward network with a linear output layer, at least one hidden layer with any “squashing” activation function (such as the logistic sigmoid activation function) and enough hidden units can approximate any continuous function on a closed and bounded subset \mathbb{R}^n from one finite-dimensional space to another with any desired non-zero amount of error. The derivatives of the feedforward network can also approximate the derivatives of the function arbitrarily well.*

it means that, regardless of what function we are trying to learn, we know that a large MLP will be able to *represent* this function

- roughly, $VC(\mathcal{H}) = O(VD)$
- V : # of neurons
- D : # of weights

but, we are not guaranteed that the training algorithm will be able to *learn* that function

- the optimization algorithm used for training may not be able to find the value of the parameters that corresponds to the desired function
- the training algorithm might choose the wrong function due to overfitting

also, the universal approximation theorem does not say how large this network will be

we want to use deep models

- a feedforward network with a single layer is sufficient to represent any function, but it may be infeasibly large (exponential number of hidden units) and may fail to learn and generalize correctly
- any time we choose a specific machine learning algorithm, we are implicitly stating some set of prior beliefs we have about what kind of function the algorithm should learn, choosing a deep model encodes a very general belief that the function we want to learn should involve composition of several simpler functions, indeed, it generalizes well

Chapter 13

Regularization of Deep Models

13.1 Ways to Reduce Overfitting

weight-decay: try to keep many of the weights are zero

weight-sharing: many of the weights have the exactly same val. as each other

early stopping: once the perf. on the fake test set starts getting worse, you stop training

model averaging: train diff. NNet and avg.

Bayesian fitting of neural nets:

dropout: try to make your model more robust by randomly emitting hidden units when training

generative pre-training

13.2 Weight Decay

basice choice, L_2 regularizer

$$\frac{\lambda}{m} \sum_{l=1}^{L-1} \sum_{k=1}^{n^{(l+1)}} \sum_{j=1}^{n^{(l)}} (W_{kj}^{(l)})^2 \quad (13.1)$$

shrink weights:

- large weight \rightarrow large shrink
- small weight \rightarrow small shrink

want $\Theta_{kj}^{(l)} = 0$ (sparse) to effectively decrease $VC(\mathcal{H})$

- L_1 regularizer: but not dofferentiable

$$\sum_{l=1}^{L-1} \sum_{k=1}^{n^{(l+1)}} \sum_{j=1}^{n^{(l)}} |W_{kj}^{(l)}| \quad (13.2)$$

weight-elimination
(scaled L_2) regularizer
hyperparameter
selection algorithm

- **weight-elimination (scaled L_2) regularizer :**

large weight \rightarrow median shrink

small weight \rightarrow median shrink

$$\sum_{l=1}^{L-1} \sum_{k=1}^{n^{(l+1)}} \sum_{j=1}^{n^{(l)}} \frac{(W_{kj}^{(l)})^2}{1 + (W_{kj}^{(l)})^2} \quad (13.3)$$

$$\frac{\partial \text{err}}{\partial W_{kj}^{(l)}} = \sum_{i=1}^m \delta_k^{(l+1)} a_k^{(l)} + \frac{2W_{kj}^{(l)}}{1 + (W_{kj}^{(l)})^2} \quad (13.4)$$

13.3 Early Stopping as a Form of Regularization

in NNet, when training large models with high $VC(\mathcal{H})$, we often observe that training error decreases steadily over time, but validation set error begins to rise again, see Fig. 13.1

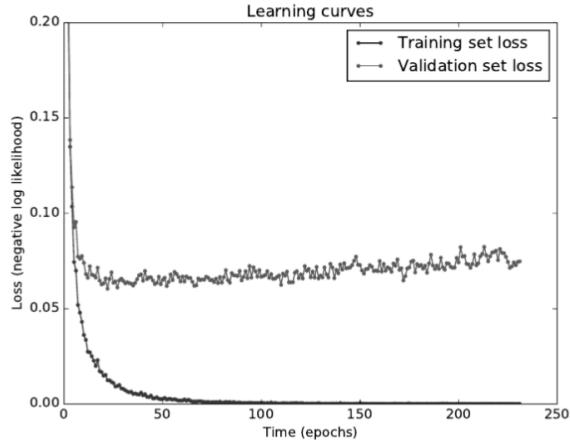


Figure 13.1: Learning curves showing how the negative log-likelihood loss changes overtime.

13.3.1 Early Stopping

idea: run the alg. until the error on the val. set has not improved for some amount of time to get the point in time with lowest validation err.

- this is a meta-alg., or **hyperparameter selection algorithm** , # iterations is a hyperparameter
- effective, simple
- unlike regularization as weight decay, it does not change the training procedure
- often init. the model again and retrain on the whole dataset for T^* iterations

Algorithm 56 Early Stopping

input: T_A : # iterations betw. evaluation

 δ : patience time, max. times to observe worsening validation set error before giving up

```

init.  $t = 0, \delta_t = 0$ 
while  $\delta_t < \delta$ 
     $\vec{\theta} \leftarrow A(D_{train}, T_A)$  // update  $\vec{\theta}$  by  $A$  for  $T_A$  iterations
     $t \leftarrow t + T_A$ 
    if  $J_{val}(\vec{\theta})$  is the smallest
         $\vec{\theta}^* \leftarrow \vec{\theta}$ 
         $T^* \leftarrow t$ 
         $\delta_t = 0$ 
    else
         $\delta_t += \delta$ 
return  $\vec{\theta}^*, T^*$  // return best para.  $\vec{\theta}^*$ , best # iterations  $T^*$ 

```

13.3.2 Early Stopping and the Use of Surrogate Error Function

early stopping can mitigate the problems caused by a mismatch between the surrogate error function whose gradient we follow downhill and the underlying performance measure that we actually care about

- eg., 0/1 error is difficult to opt., so we train with a surrogate CE error
- but 0/1 error is inexpensive to compute, so it can easily be used as an early stopping criterion on val. set

early stopping can reduce the computational cost of the training procedure, it does not change the error function, so no additional gradient computations are needed

13.3.3 Early Stopping Acts as a Regularizer

early stopping has the effect of restricting the optimization procedure to a relatively small volume of parameter space in the neighborhood of the initial parameter value $\vec{\theta}_0$

in order to compare with L_2 regularization, assume the only para. are linear weights \vec{w} , and init. it $\vec{w}_0 = \vec{0}$

$$\vec{w}^* = \arg \min_{\vec{w}} J(\vec{w}) \quad (13.5)$$

$$J(\vec{w}) = J(\vec{w}^*) + \frac{1}{2}(\vec{w} - \vec{w}^*)H(\vec{w} - \vec{w}^*) \quad (13.6)$$

$$\nabla J(\vec{w}) = H(\vec{w} - \vec{w}^*) \quad (13.7)$$

$$\vec{w}_T = \vec{w}_{T-1} - \alpha \nabla J(\vec{w}_t) \quad (13.8)$$

parameter sharing

$$= \vec{w}_{T-1} - \alpha H(\vec{w}_{T-1} - \vec{w}^*) \quad (13.9)$$

$$\vec{w}_T - \vec{w}^* = (I - \alpha H)(\vec{w}_{T-1} - \vec{w}^*) \quad (13.10)$$

$$= (I - \alpha U \Lambda U')(\vec{w}_{T-1} - \vec{w}^*) \quad (13.11)$$

$$Q'(\vec{w}_T - \vec{w}^*) = (I - \alpha \Lambda)Q'(\vec{w}_{T-1} - \vec{w}^*) \quad (13.12)$$

$$= (I - \alpha \Lambda)^T Q'(-\vec{w}^*) \quad (13.13)$$

$$Q' \vec{w}_T = (I - (I - \alpha \Lambda)^T)Q' \vec{w}^* \quad (13.14)$$

// recall in L_2 regularization

$$Q' \vec{w} = (\Lambda + \frac{\lambda}{m} I)^{-1} \Lambda Q' \vec{w}^* \quad (13.15)$$

$$= (I - (\Lambda + \frac{\lambda}{m} I)^{-1} \frac{\lambda}{m})Q' \vec{w}^* \quad (13.16)$$

if

$$(I - \alpha \Lambda)^T = (\Lambda + \frac{\lambda}{m} I)^{-1} \frac{\lambda}{m} \quad (13.17)$$

then early stopping and L_2 are equiv.

$$T \approx \frac{m}{\alpha \lambda} \quad (13.18)$$

$$\frac{1}{\alpha T} \approx \frac{\lambda}{m} \quad (13.19)$$

$\frac{1}{\alpha T}$ plays a role of the weight decay coefficient

13.4 Parameter Tying and Parameter Sharing

A common type of model parameters' dependency that we often want to express is that certain parameters should be close to one another

when regularized the parameters of one model, trained as a classifier in a supervised paradigm, with the parameters of another model, trained in an unsupervised paradigm

- the parameters in the classifier model $\vec{\theta}$ could be paired to corresponding parameters in the unsupervised model $\vec{\theta}'$
- we can use a para. norm penalty $\Omega(\vec{\theta}, \vec{\theta}') = \|\vec{\theta} - \vec{\theta}'\|_2^2$

the more popular way is to use constraints: to force sets of parameters to be equal, this is **parameter sharing**

13.5 Sparse Representations

- L_1 penalization introduces a sparse parametrization: many zeros in $\vec{\theta}$
- sometimes, it is hard to specify a penalty or constraint on the model parameters $\vec{\theta}$ that matches out prior knowledge of the function being learning

sparse representation : many zeros in representation

sparse representation

$$J_{aug}(\vec{\theta}) = J(\vec{\theta}) + \frac{\lambda}{m} \Omega(\vec{a}) \quad (13.20)$$

$\Omega(\vec{a})$: representational penalty

- L_1 : representational sparsity
- Student- t
- KL divergence

13.6 Dropout

13.6.1 Bagging and Other Ensemble Methods

model averaging is an extremely powerful and reliable method for reducing generalization error

NNets reach a wide enough variety of solution points that they can often benefit from model averaging even if all of the models are trained on the same dataset

- same model, different initializations
- top 10 models say, discovered during cross-validation
- different checkpoints of a single model
- running average of parameters during training

13.6.2 Dropout

dropout can be thought of as a method of making bagging practical for large neural networks

- dropout trains the ensemble consisting of all sub-networks that can be formed by removing units from an underlying base network with prob. $1 - p$, see Fig. 13.2
- remove a unit from a network is by multiplying its output value \vec{a} by 0
- only update the para. of the sampled network based on the input data
- very computational cheap
- works well for nearly every model

```

1 a2 = max(0, W1 * a1 + b1);
2 u2 = rand(n2, 1) < p;
3 a2 = a2 .* u2;
```

13.6.3 Testing

in testing there is no dropout applied

we want the outputs of neurons at test time to be identical to their expected outputs at training time

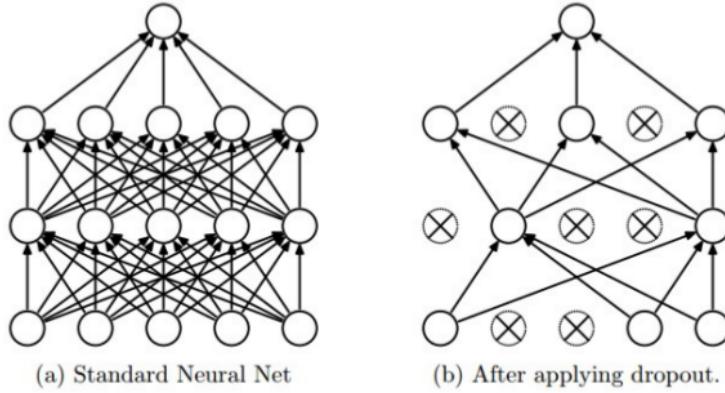


Figure 13.2: Dropout.

- consider an output of a neuron \vec{a} (before dropout)
- with dropout, the expected output from this neuron will become $p\vec{a} + (1-p)\vec{0} = p\vec{a}$
- so we are performing a scaling of both hidden layer outputs by p
- performing this attenuation at test time can be related to the process of iterating over all the possible binary masks (and therefore all the exponentially many sub-networks) and computing their ensemble prediction.

```
1 a2 = max(0, W1 * a1 + b1) * p;
```

13.6.4 Inverted Dropout

scaling at train time, leaving the forward pass at test time untouched

```
1 a2 = max(0, W1 * a1 + b1);
2 u2 = (rand(n2, 1) < p) / p;
3 a2 = a2 .* u2;
```

in practice, $p = \frac{1}{2}$ is a reasonable default

13.6.5 Dropout For Other Units

for many classes of models that do not have nonlinear hidden units, the weight scaling inference rule is exact eg., consider a softmax classifier, $\vec{a} \in \mathbb{R}^n$

$$\Pr(y = k | \vec{a}) = \frac{\exp(W_k \cdot \vec{a} + b_k)}{\sum_{j=1}^K \exp(W_j \cdot \vec{a} + b_j)} = g(W \vec{a} + \vec{b})_k \quad (13.21)$$

we can index into the family of sub-models by element-wise multiplication of the input with a binary vector \vec{v}

$$\Pr(y = k | \vec{a}) = g(W(\vec{a} \cdot \vec{v}) + \vec{b})_k \quad (13.22)$$

the ensemble predictor is defined by re-normalizing the geometric mean over all ensemble members' predictions

$$\tilde{\Pr}(y = k|\vec{a}) = \frac{\sqrt[2^n]{\prod_{\vec{v} \in \{0,1\}^n} \Pr(y = k|\vec{a}; \vec{v})}}{\sum_{j=1}^K \sqrt[2^n]{\prod_{\vec{v} \in \{0,1\}^n} \Pr(y = j|\vec{a}; \vec{v})}} \quad (13.23)$$

$$\sqrt[2^n]{\prod_{\vec{v} \in \{0,1\}^n} \Pr(y = k|\vec{a}; \vec{v})} = \sqrt[2^n]{\prod_{\vec{v} \in \{0,1\}^n} \frac{\exp(W_{k*}(\vec{a} \cdot * \vec{v}) + b_k)}{\sum_{j=1}^K \exp(W_{j*}(\vec{a} \cdot * \vec{v}) + b_j)}} \quad (13.24)$$

$$= \frac{\sqrt[2^n]{\prod_{\vec{v} \in \{0,1\}^n} \exp(W_{k*}(\vec{a} \cdot * \vec{v}) + b_k)}}{\sqrt[2^n]{\sum_{j=1}^K \exp(W_{j*}(\vec{a} \cdot * \vec{v}) + b_j)}} \quad (13.25)$$

$$= \sqrt[2^n]{\prod_{\vec{v} \in \{0,1\}^n} \exp(W_{k*}(\vec{a} \cdot * \vec{v}) + b_k)} \quad (13.26)$$

$$\text{// because it will be normalized, const. term can be ignored} \\ = \exp\left(\frac{1}{2^n} \sum_{\vec{v} \in \{0,1\}^n} W_{k*}(\vec{a} \cdot * \vec{v}) + b_k\right) \quad (13.27)$$

$$= \exp\left(\frac{1}{2} \sum_{\vec{v} \in \{0,1\}^n} W_{k*}\vec{a} + b_k\right) \quad (13.28)$$

13.7 Multi-Task Learning

multi-task learning is a way to improve generalization by pooling the examples arising out of several tasks

- assume tasks have some stat. relationship
- the tasks share a common lower layers but involve diff. target rv., see Fig. 13.3
generic parameters: shared across all the tasks
task-specific parameters: upper layers of a NNet
- the part of shared is more constrained towards vals. for better generalization, compared with single task, more examples are used to train the generic parameters

13.8 Adversarial Learning

by adding very small variation to the originale example, the network can make highly diff. predictions, see Fig. 13.4

one of the primary causes of these adversarial examples is **excessive linearity**

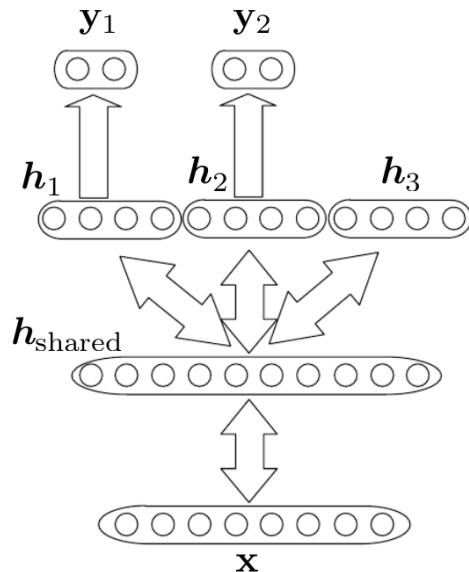


Figure 13.3: Multi-task learning

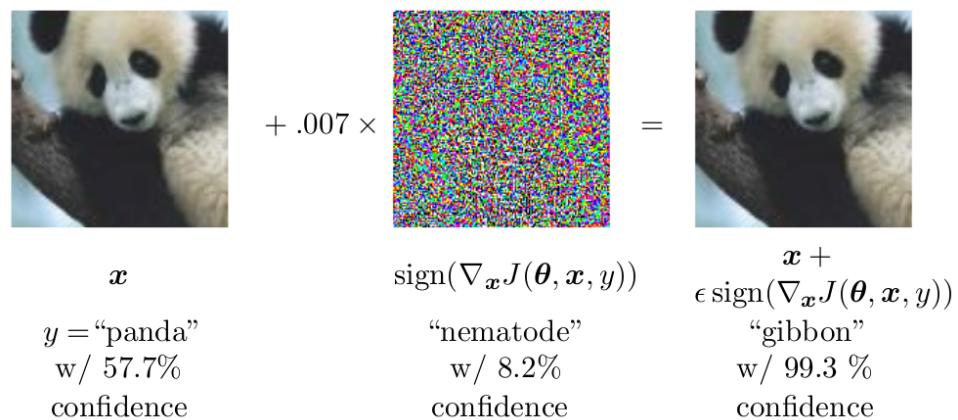


Figure 13.4: A demonstration of adversarial example.

- NNet are built out of primarily linear building blocks
- in some experiments the overall function they implement proves to be highly linear as a result
- these linear functions are easy to optimize
- the output changes rapidly even for small change of the input
- adversarial training discourages this highly sensitive locally linear behavior by encouraging the network to be locally constant in the neighborhood of the training data
- this can be seen as a way of introducing explicitly the local smoothness

prior into supervised neural nets

purely linear models, like logistic regression, are not able to resist adversarial examples because they are forced to be linear

neural networks are able to represent functions that can range from nearly linear to nearly locally constant and thus have the flexibility to capture linear trends in the training data while still learning to resist local perturbation

Chapter 14

Optimization for Training Deep Models

14.1 Challenges in Neural Network Optimization

14.1.1 Ill-Conditioning

consider

$$\vec{\theta}_{t+1} = \vec{\theta}_t - \alpha \nabla J(\vec{\theta}_t) \quad (14.1)$$

$$J(\vec{\theta}_{t+1}) \approx J(\vec{\theta}_t) - \alpha (\nabla J(\vec{\theta}))^T \nabla J(\vec{\theta}) + \frac{1}{2} \alpha^2 (\nabla J(\vec{\theta}))^T H \nabla J(\vec{\theta}) \quad (14.2)$$

(14.3)

- $-\alpha (\nabla J(\vec{\theta}))^T \nabla J(\vec{\theta}) < 0$
- $\frac{1}{2} \alpha^2 (\nabla J(\vec{\theta}))^T H \nabla J(\vec{\theta})$: can be pos. or neg. depending on the eig.-vals. of H and the alignment of the corresponding eig.-vecs. with $\nabla J(\vec{\theta})$
- on steps where $\nabla J(\vec{\theta})$ aligns closely with large, pos. eig.-vals. of H , α must be small, or the second-order term will result in GD accidentally moving uphill

see Fig. 14.1, most second-order methods, as well as momentum or gradient averaging methods are meant to address that problem

- by increasing the step size in the direction of the valley (where it is most paying in the long run to go)
- decreasing it in the directions of steep rise, which would otherwise lead to oscillations (blue full arrows)
- the objective is to smoothly go down, staying at the bottom of the valley (green dashed arrow)

Newton's method is an excellent tool for minimizing convex functions with poorly conditioned Hessian matrices, but it requires significant modification before it can be applied to neural networks

Model Identifiability
weight space symmetry

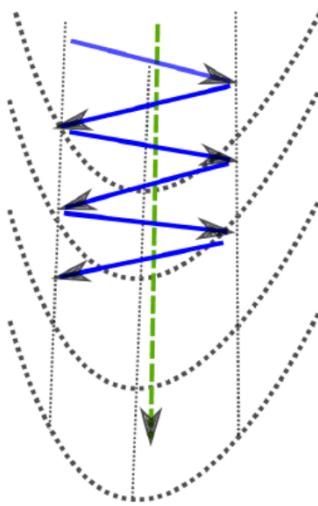


Figure 14.1: Some directions have a high curvature (second derivative), corresponding to the quickly rising sides of the valley (going left or right), and other directions have a low curvature, corresponding to the smooth slope of the valley (going down, dashed arrow).

14.1.2 Local Minima

Define 14.1 (Model Identifiability). *A model is said to be identifiable if a sufficiently large training set can rule out all but one setting of the model's parameters.*

- neural networks and any models with multiple equivalently parameterized latent variables are often not identifiable because we can obtain equivalent models by exchanging latent variables with each other, this is **weight space symmetry**
- in ReLU, or maxout network, we can scale all of the incoming weights and biases of a unit by c , also scale all of the outgoing weights by $\frac{1}{c}$, the output is equiv.
- because these local min. are equiv., they are not a problematic form of non-convexity
- local minima can be problematic if they have high cost

14.1.3 Plateaus, Saddle Points and Other Flat Regions

we can think of a saddle point as being a local minimum along one cross-section of the cost function and a local maximum along another cross-section

many random functions is that the eigenvalues become more likely to be positive as we reach regions of lower cost, this means that local minima are much more likely to have low cost than high cost, critical points with high cost are far more likely to be saddle points

gradient descent is designed to move “downhill” and is not explicitly designed to seek a critical point. Newtons method, however, is designed to solve for a point where the gradient is zero

14.1.4 Cliffs and Exploding Gradients

when the objective function does not have “vallery” like in Fig. 14.1, instead, there are cliffs where the gradient rises sharply, see Fig. 14.2

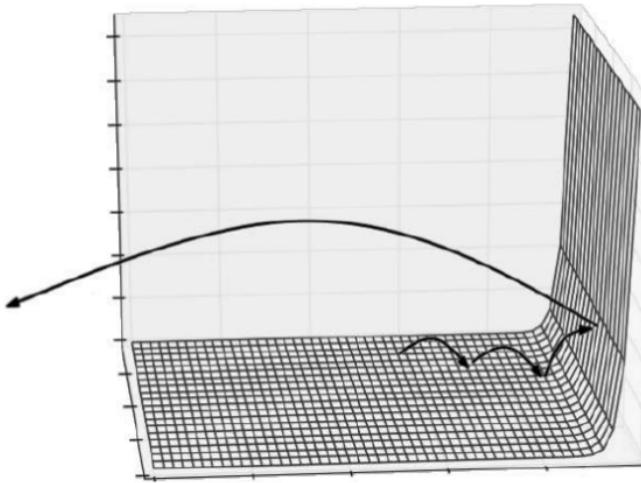


Figure 14.2: There are sharp non-linearities that give rise to very high derivatives in some places. When the parameters get close to such a cliff region, a gradient descent update can catapult the parameters very far, possibly ruining a lot of the optimization work that had been done

a useful heuristic method is limit the size of jumps that one would made, see Fig. 14.3

- GD is valid if update is of infinitesimal moves
- in the presence of a cliff, the decrease in the objective function expected from going in the direction of the gradient is only valid for a very small step
- not trust the gradient too much when it is very large in magnitude
- clip the magnitude of the gradient, only keeping its direction if its magnitude is above a threshold

14.1.5 Inexact Gradients

in practice, we usually only have a noisy or even biased estimate of gradient or Hessian

the obj. function we want to min. is actually intractable, typically its grad. is intractable as well

momentum

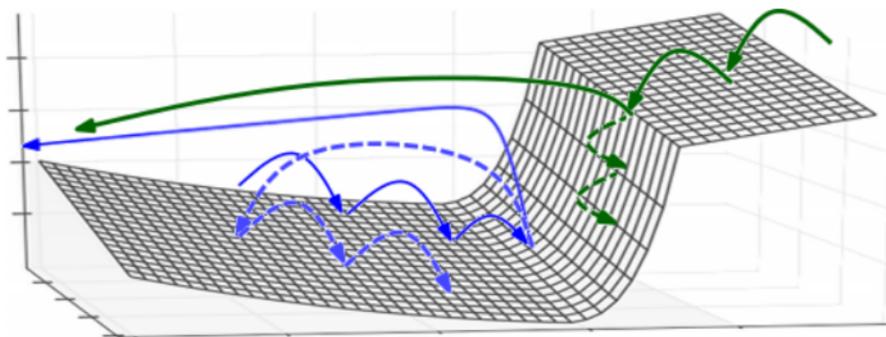


Figure 14.3: Using such a gradient clipping heuristic (dotted arrows trajectories) helps to avoid the destructive big moves which would happen when approaching the cliff, either from above or from below (bold arrows trajectories)

14.1.6 Theoretical Limits of Optimization

theoretical analysis of whether an opt. alg. can accomplish this goal is extremely difficult

developing more realistic bounds on the perf. of opt. alg. remains an important goal for ML research

14.2 Optimization Algorithm I: Basic Algorithm

14.2.1 Momentum

learning using SGD can sometimes be slow when grad. is small

idea: when the grad. is small and consistent across consecutive mini-batches, we can afford to take larger steps in this direction

eg., a ball is on a gentle slope

- the potential energy $J = mgh$, the init. velocity is 0
- the instantaneous force $\vec{F} = -\nabla J$ pulling the ball downhill, this is the grad. term
- $\vec{F} = m\vec{a}$, grad. only directly influences the velocity, which in turn has an effect on the position
- the downhill velocity of the ball gradually begins to increase over time

momentum : accumulate gradient contributions over training iterations, directions that consistently have positive contributions to the gradient will be augmented, see Fig. 14.4

it is useful when the objective has the form of a long shallow ravine leading to the optimum and steep walls on the sides

- standard SGD will tend to oscillate across the narrow ravine since the

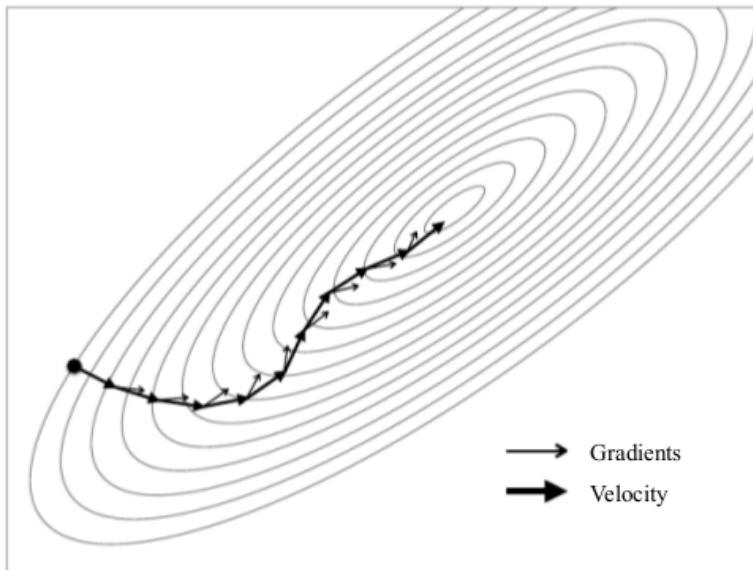


Figure 14.4: Momentum

negative gradient will point down one of the steep sides rather than along the ravine towards the optimum

- the objectives of deep architectures have this form near local optima and thus standard SGD can lead to very slow convergence particularly after the initial steep gains
- momentum is one method for pushing the objective more quickly along the shallow ravine

Algorithm 57 Momentum

init. $\vec{v} = \vec{0}$

repeat until convergence

$$\vec{v} \leftarrow \gamma \vec{v} - \alpha \nabla J(\vec{\theta})$$

$$\vec{\theta} \leftarrow \vec{\theta} + \vec{v}$$

- velocity \vec{v} : accumulates the grad. ∇J
- coefficient of friction γ : determines for how many iterations the previous gradients are incorporated into the current update
- if γ is larger to $\alpha \rightarrow$ more previous gradients affect the current direction
- α may need to be smaller since the magnitude of the gradient will be larger
- γ is often set 0.5, 0.9, 0.95, 0.99, a typical setting is to start with momentum of about 0.5 and anneal it to 0.99 or so over multiple epochs

14.2.2 Nesterov Momentum

a variant of momentum

Algorithm 58 Nesterov Momentum

```

init.  $\vec{v} = \vec{0}$ 
repeat until convergence
   $\vec{v} \leftarrow \gamma\vec{v} - \alpha\nabla J(\vec{\theta} + \gamma\vec{v})$ 
   $\vec{\theta} \leftarrow \vec{\theta} + \vec{v}$ 

```

Nesterov momentum incorporates the gradient after the velocity is already applied, see Fig. 14.5

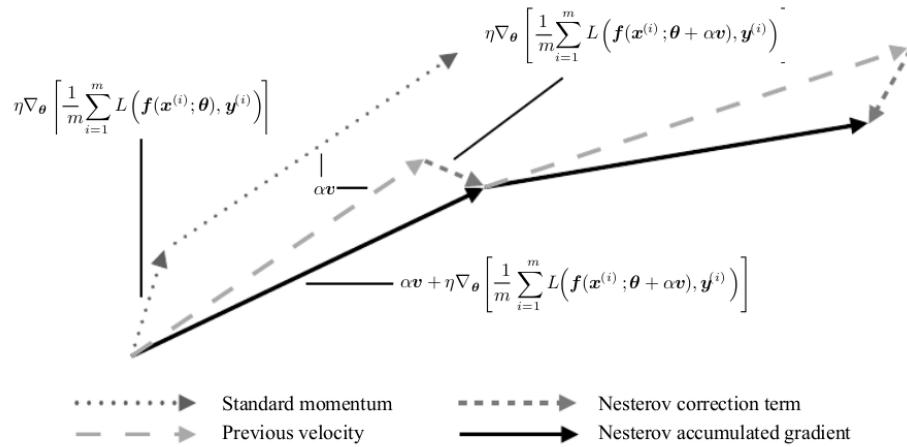


Figure 14.5: Difference betw. Nesterov momentum and std. momentum

in practice, often express the update to look as similar to SGD

Algorithm 59 Nesterov Momentum

```

init.  $\vec{v} = \vec{0}$ 
repeat until convergence
   $\vec{v}_p \leftarrow \vec{v}$ 
   $\vec{v} \leftarrow \gamma\vec{v} - \alpha\nabla J(\vec{\theta})$ 
   $\vec{\theta} \leftarrow \vec{\theta} - \gamma\vec{v}_p + (1 + \gamma)\vec{v}$ 

```

14.3 Optimization Algorithm II: Adaptive Learning

α has significant impact on model perm. and it is hard to set

momentum can alleviate this issue but γ is also very hard to set
want: α automatically set

14.3.1 AdaGrad

Algorithm 60 AdaGrad

```
init.  $\vec{r} = \vec{0}$ 
repeat until convergence
   $\vec{r} \leftarrow \vec{r} + (\nabla J)^2$  // square is element-wise
   $\vec{\theta} \leftarrow \vec{\theta} - \alpha \nabla J / \sqrt{\vec{r} + 10^{-8}}$ 
```

- accumulate grad. over all iterations
 - scale grad. inversely prop. to that sum
 - net effect is greater progress in the more gently sloped direction
- although it has some desirable theoretical properties, if accumulate from the beginning, the monotonic learning rate usually proves too aggressive and stops learning too early

14.3.2 RMSprop

directions in with strong grad. early in training may flatten out as training process

Algorithm 61 RMSprop

```
init.  $\vec{r} = \vec{0}$ 
repeat until convergence
   $\vec{r} \leftarrow \rho \vec{r} + (1 - \rho)(\nabla J(\vec{\theta}))^2$  // square is element-wise
   $\vec{\theta} \leftarrow \vec{\theta} - \alpha \nabla J(\vec{\theta}) / \sqrt{\vec{r} + 10^{-8}}$ 
```

- changing the grad. accumulation into an exponentially weighted moving avg.
- this adapts to the changing local topology of the error function
- ρ is often set as 0.9, 0.99, 0.999

Algorithm 62 RMSprop with Nesterov Momentum

```
init.  $\vec{r} = \vec{0}$ 
repeat until convergence
   $\vec{r} \leftarrow \rho \vec{r} + (1 - \rho)(\nabla J(\vec{\theta} - \gamma \vec{v}))^2$  // square is element-wise
   $\vec{\theta} \leftarrow \gamma \vec{\theta} - \alpha \nabla J(\vec{\theta} - \gamma \vec{v}) / \sqrt{\vec{r}}$ 
```

in practice, RMSprop has shown to be effective, it does not appear to be a great sensitivity to the alg.'s hyperpara.

14.3.3 Adam

Algorithm 63 Adam

init. $\vec{s} = \vec{0}, \vec{r} = \vec{0}$
for $t = 1 : T$
 $\quad \vec{s} \leftarrow \frac{\rho_1 \vec{s} + (1 - \rho_1) \nabla J(\vec{\theta})}{1 - \rho_1^t}$ // 1st-order moment
 $\quad \vec{r} \leftarrow \frac{\rho_2 \vec{r} + (1 - \rho_2) (\nabla J(\vec{\theta}))^2}{1 - \rho_2^t}$ // 2nd-order moment
 $\quad \vec{\theta} \leftarrow \vec{\theta} - \alpha \vec{s} \cdot \nabla J(\vec{\theta}) / (\sqrt{\vec{r}} + \epsilon)$

- momentum is incorporated directly as an estimate of the 1st-order moment
- unlike Adam, the RMSprop 2nd-order moment estimate may have high bias early in training

14.3.4 AdaDelta

consider Newton's method on a single para.

$$v = \Delta\theta = -\frac{\frac{dJ}{d\theta}}{\frac{d^2J}{d\theta^2}} \quad (14.4)$$

$$\frac{1}{\frac{d^2J}{d\theta^2}} = \frac{v}{\frac{dJ}{d\theta}} \quad (14.5)$$

assuming H diag. and a Newton's update, its inverse could be estimated as the ratio of increment over gradient

Algorithm 64 AdaDelta

init. $\vec{s} = \vec{0}, \vec{r} = \vec{0}$
repeat until convergence
 $\quad \vec{r} \leftarrow \rho \vec{r} + (1 - \rho) (\nabla J(\vec{\theta}))^2$
 $\quad \vec{v} \leftarrow \nabla J \cdot \sqrt{\vec{s} + \epsilon} / \sqrt{\vec{r} + \epsilon}$
 $\quad \vec{s} \leftarrow \rho \vec{s} + (1 - \rho) \vec{v}^2$
 $\quad \vec{\theta} \leftarrow \vec{\theta} - \vec{v}$

14.3.5 Choosing the Right Optimization Algorithm

RMSprop and AdaDelta perm. fairly robustly

popular choice

- SGD
- SGD+momentum
- RMSprop
- RMSprop+momentum

- AdaDelta
- Adam

14.4 Optimization Algorithm III: Approximate Second-Order Methods

14.4.1 Newton's Method

Newton's method is only appropriate when the local quad. approx. holds
in deep learning, the surface of obj. function is typically non-convex and
with many saddle points

if the eig.-vals. of H are not all pos., Newton's method can actually cause
updates to move in the wrong direction

add regularization

$$\vec{\theta} \leftarrow \vec{\theta} - (H + \lambda I)^{-1} \nabla J(\vec{\theta}) \quad (14.6)$$

- $\lambda \uparrow \rightarrow$ better suits for extreme directions of curvature, direction more like std. GD
- need inverse a $n \times n$ mat., computational complexity is $O(n^3)$, it need to be compute in every iteration

14.4.2 Conjugate Gradients

Problem with Steepest Descent

steepest descent: when applied in a quad. bowl, it zig-zag because at the min. of the obj. along a given direction, the grad. is orthogonal to that direction, see Fig. 14.6

let prev. search direction \vec{d}_t , at min.

$$\nabla J(\vec{\theta}) \perp \vec{d}_t \quad (14.7)$$

$$\vec{d}_{t+1} \perp \vec{d}_t \quad (14.8)$$

as see in Fig. 14.6, the choice of orthogonal directions of descent do not preserve the min. along the prev. search directions, this gives rise to zig-zag pattern of progress

Conjugate Direction

in the conjugate gradients, we add back the prev. direction \vec{d}_t to the current search direction

$$\vec{d}_{t+1} = \nabla J(\vec{\theta}) + \beta_{t+1} \vec{d}_t \quad (14.9)$$

Conjugate

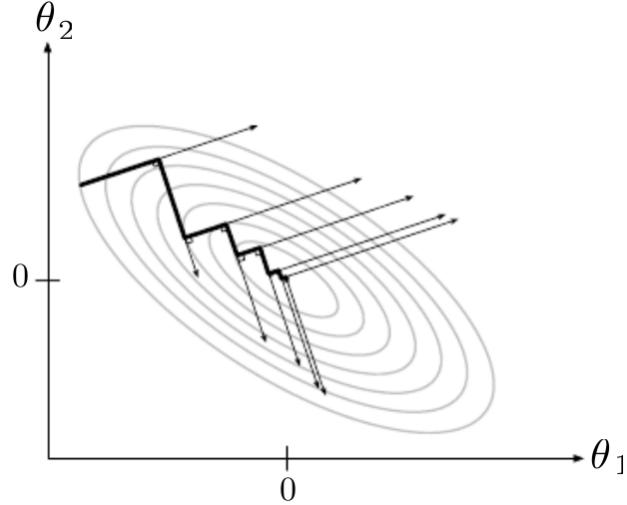


Figure 14.6: Problem with steepest descent, the arrow shows directions of the gradient

Define 14.2 (Conjugate). *Two directions \vec{d}_{t+1} and \vec{d}_t are conjugate if*

$$\vec{d}_{t+1}^T H \vec{d}_t = 0 \quad (14.10)$$

that is

$$\vec{d}_{t+1}^T H \vec{d}_t = \vec{d}_{t+1}^T U \Lambda U^T \vec{d}_t \quad (14.11)$$

$$= (\Lambda^{\frac{1}{2}} U^T \vec{d}_{t+1})^T (\Lambda^{\frac{1}{2}} U^T \vec{d}_t) \quad (14.12)$$

$$\stackrel{\text{def}}{=} \vec{d}_{t+1}^T \vec{d}_t \quad (14.13)$$

$$= 0 \quad (14.14)$$

where $\vec{d}_t' = \Lambda^{\frac{1}{2}} U^T \vec{d}_t$ is the direction \vec{d}_t in $\vec{\phi}$ space.

conjugate gradient can be interpreted as steepest descent in $\vec{\phi}$ space
we want to avoid compute \vec{d}_t by $\Lambda^{\frac{1}{2}} U^T \vec{d}_t$ because we want to avoid computing H

there are two methods

- Fletcher-Reeves

$$\beta_{t+1} = \frac{(\nabla J(\vec{\theta}_{t+1}))^T \nabla J(\vec{\theta}_{t+1})}{(\nabla J(\vec{\theta}_t))^T \nabla J(\vec{\theta}_t)} \quad (14.15)$$

- Polak-Ribiere

$$\beta_{t+1} = \frac{(\nabla J(\vec{\theta}_{t+1}) - \nabla J(\vec{\theta}_t))^T \nabla J(\vec{\theta}_{t+1})}{(\nabla J(\vec{\theta}_t))^T \nabla J(\vec{\theta}_t)} \quad (14.16)$$

for a quad. surface

- the conjugate directions ensure that the grad. along the prev. direction does not increase in magnitude, ie., we stay at the min. along the previous directions
- in a n -dim. para. space, conjugate grad. only requires n line searches to achieve the min., see Fig. 14.7

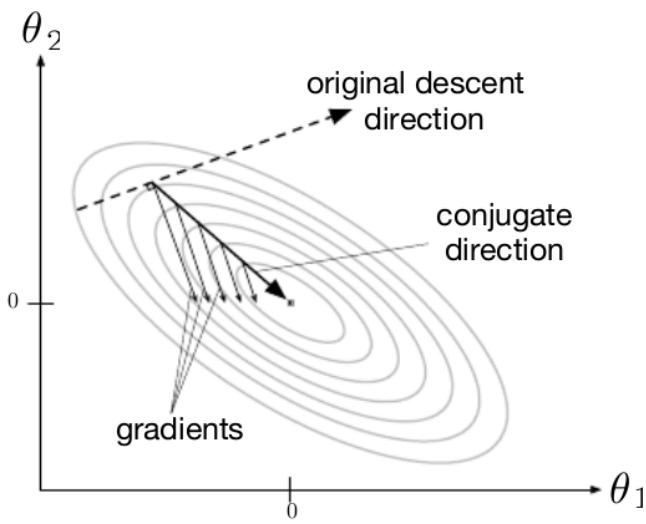


Figure 14.7: Conjugate gradient

Conjugate Gradient

Algorithm 65 Conjugate Gradient

```

init.  $\vec{d}_0 = \vec{0}$ 
for  $t = 0 : T$ 
     $\beta_{t+1} = \frac{(\nabla J(\vec{\theta}_{t+1}) - \nabla J(\vec{\theta}_t))^T \nabla J(\vec{\theta}_{t+1})}{(\nabla J(\vec{\theta}_t))^T \nabla J(\vec{\theta}_t)}$ 
     $\vec{d}_{t+1} = \nabla J(\vec{\theta}_{t+1}) + \beta_{t+1} \vec{d}_t$ 
     $\alpha = \arg \min_{\alpha} J(\vec{\theta}_{t+1})$ 
     $\vec{\theta}_{t+2} \leftarrow \vec{\theta}_{t+1} - \alpha \vec{d}_{t+1}$ 

```

Nonlinear Conjugate Gradient

when obj. function is not quad.

- the conjugate directions are no longer assured to remain at the min. of the obj. for prev. directions

- the nonlinear conjugate gradients alg. includes occasional resets where the method of conjugate gradients is restarted with line search along the unaltered gradient
in practice often init. the opt. with a few iterations of SGD

14.4.3 BFGS

approximate the H^{-1} with M
from secant condition (quasi-Newton condition)

$$\vec{\theta}_{t+1} - \vec{\theta}_t = -H^{-1}(\nabla J(\vec{\theta}_{t+1}) - \nabla J(\vec{\theta}_t)) \quad (14.17)$$

M is updated according to

$$M_t = M_{t-1} + \left(1 + \frac{(\Delta\vec{g})^T M_t (\Delta\vec{g})}{(\Delta\vec{\theta})^T \Delta\vec{g}}\right) \frac{(\Delta\vec{g})^T \Delta\vec{g}}{(\Delta\vec{\theta})^T \Delta\vec{g}} - \frac{\Delta\vec{\theta} (\Delta\vec{g})^T M_t + M_t \Delta\vec{g} (\Delta\vec{\theta})^T}{(\Delta\vec{\theta})^T \Delta\vec{g}} \quad (14.18)$$

where

$$\Delta\vec{g} = \nabla J(\vec{\theta}_t) - \nabla J(\vec{\theta}_{t-1}) \quad (14.19)$$

$$\Delta\vec{\theta} = \vec{\theta}_t - \vec{\theta}_{t-1} \quad (14.20)$$

- this is rank 1 approx. of H
 - so the computational complexity is $O(n^2)$
- the update rule

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha M_t \nabla J(\vec{\theta}_t) \quad (14.21)$$

Algorithm 66 BFGS

init. $M_0 = I$

for $t = 1 : T$

$$\Delta\vec{g} = \nabla J(\vec{\theta}_t) - \nabla J(\vec{\theta}_{t-1})$$

$$\Delta\vec{\theta} = \vec{\theta}_t - \vec{\theta}_{t-1}$$

$$M_t = M_{t-1} + \left(1 + \frac{(\Delta\vec{g})^T M_t (\Delta\vec{g})}{(\Delta\vec{\theta})^T \Delta\vec{g}}\right) \frac{(\Delta\vec{g})^T \Delta\vec{g}}{(\Delta\vec{\theta})^T \Delta\vec{g}} - \frac{\Delta\vec{\theta} (\Delta\vec{g})^T M_t + M_t \Delta\vec{g} (\Delta\vec{\theta})^T}{(\Delta\vec{\theta})^T \Delta\vec{g}}$$

$$\alpha = \arg \min_{\alpha} J(\vec{\theta}_t + \alpha M_t \nabla J(\vec{\theta}_t))$$

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha M_t \nabla J(\vec{\theta}_t)$$

- incorporates line search, but not heavily dep. on the line search
- M need be stored in memory, which needs $O(n^2)$

14.4.4 Limited Memory BFGS (L-BFGS)

idea: do not store complete M

by replace $M_{t-1} = I$

$$\vec{d}_t = -\nabla J(\vec{\theta}_t) + b\Delta\vec{\theta} + a\Delta\vec{g} \quad (14.22)$$

$$a = - \left(1 + \frac{(\Delta \vec{g})^T (\Delta \vec{g})}{(\Delta \vec{\theta})^T \Delta \vec{g}} \right) \frac{(\Delta \vec{\theta})^T \nabla J(\vec{\theta}_t)}{(\Delta \vec{\theta})^T \Delta \vec{g}} + \frac{(\Delta \vec{g})^T \nabla J(\vec{\theta}_t)}{(\Delta \vec{\theta})^T \Delta \vec{g}} \quad (14.23)$$

$$b = \frac{(\Delta \vec{\theta})^T \nabla J(\vec{\theta}_t)}{(\Delta \vec{\theta})^T \Delta \vec{g}} \quad (14.24)$$

- if use exact line searches, the directions are mutually conjugate
- unlike conjugate gradient method, this procedure remains well behaved when the min. of the line search is reached only approx.

14.5 Optimization Algorithm IV: Natural Gradient Methods

natural finite difference

$$\Delta_N \vec{\theta} = \arg \min_{\Delta \vec{\theta}} E[-\log p(\vec{x}; \vec{\theta} + \Delta \vec{\theta})] \quad (14.25)$$

$$\text{st. } KL(p(\vec{x}; \vec{\theta}) || p(\vec{x}; \vec{\theta} + \Delta \vec{\theta})) = \Delta KL \quad (14.26)$$

$\Delta_N \vec{\theta}$ is the direction that min. the obj. function under constraint of KL divergence ΔKL

$$KL(p(\vec{x}; \vec{\theta}) || p(\vec{x}; \vec{\theta} + \Delta \vec{\theta})) \quad (14.27)$$

$$= \sum_{\vec{x}} p(\vec{x}; \vec{\theta}) \log p(\vec{x}; \vec{\theta}) - \sum_{\vec{x}} p(\vec{x}; \vec{\theta}) \log p(\vec{x}; \vec{\theta} + \Delta \vec{\theta}) \quad (14.28)$$

$$\approx \sum_{\vec{x}} p(\vec{x}; \vec{\theta}) \log p(\vec{x}; \vec{\theta}) - \sum_{\vec{x}} p(\vec{x}; \vec{\theta}) \left(\log p(\vec{x}; \vec{\theta}) + (\nabla \log p(\vec{x}; \vec{\theta}))^T \Delta \vec{\theta} + \frac{1}{2} (\Delta \vec{\theta})^T (\nabla^2 \log p(\vec{x}; \vec{\theta})) \Delta \vec{\theta} \right) \quad (14.29)$$

$$= - \sum_{\vec{x}} p(\vec{x}; \vec{\theta}) (\nabla \log p(\vec{x}; \vec{\theta}))^T \Delta \vec{\theta} - \frac{1}{2} \sum_{\vec{x}} p(\vec{x}; \vec{\theta}) (\Delta \vec{\theta})^T (\nabla^2 \log p(\vec{x}; \vec{\theta})) \Delta \vec{\theta} \quad (14.30)$$

$$= -(\Delta \vec{\theta})^T \left(\sum_{\vec{x}} p(\vec{x}; \vec{\theta}) \nabla \log p(\vec{x}; \vec{\theta}) \right) - \frac{1}{2} (\Delta \vec{\theta})^T E[\nabla^2 \log p(\vec{x}; \vec{\theta})] \Delta \vec{\theta} \quad (14.31)$$

$$= -(\Delta \vec{\theta})^T \sum_{\vec{x}} \nabla p(\vec{x}; \vec{\theta}) - \frac{1}{2} (\Delta \vec{\theta})^T E[\nabla^2 \log p(\vec{x}; \vec{\theta})] \Delta \vec{\theta} \quad (14.32)$$

$$= -(\Delta \vec{\theta})^T \nabla \sum_{\vec{x}} p(\vec{x}; \vec{\theta}) - \frac{1}{2} (\Delta \vec{\theta})^T E[\nabla^2 \log p(\vec{x}; \vec{\theta})] \Delta \vec{\theta} \quad (14.33)$$

$$= -(\Delta \vec{\theta})^T \nabla 1 - \frac{1}{2} (\Delta \vec{\theta})^T E[\nabla^2 \log p(\vec{x}; \vec{\theta})] \Delta \vec{\theta} \quad (14.34)$$

$$= -\frac{1}{2} (\Delta \vec{\theta})^T E[\nabla^2 \log p(\vec{x}; \vec{\theta})] \Delta \vec{\theta} \quad (14.35)$$

$$E[\nabla^2 \log p(\vec{x}; \vec{\theta})] + E[(\nabla \log p(\vec{x}; \vec{\theta}))^T \nabla \log p(\vec{x}; \vec{\theta})] \quad (14.36)$$

Fisher information matrix
 coordinate descent
 block coordinate descent

$$= \sum_{\vec{x}} p(\vec{x}; \vec{\theta}) \nabla^2 \log p(\vec{x}; \vec{\theta}) + \sum_{\vec{x}} p(\vec{x}; \vec{\theta}) (\nabla \log p(\vec{x}; \vec{\theta}))^T \nabla \log p(\vec{x}; \vec{\theta}) \quad (14.37)$$

$$= \sum_{\vec{x}} \nabla(p(\vec{x}; \vec{\theta}) \nabla \log p(\vec{x}; \vec{\theta})) \quad (14.38)$$

$$= \nabla^2 \sum_{\vec{x}} p(\vec{x}; \vec{\theta}) \quad (14.39)$$

$$= \nabla^2 1 \quad (14.40)$$

$$= 0 \quad (14.41)$$

$$- E[\nabla^2 \log p(\vec{x}; \vec{\theta})] = E[(\nabla \log p(\vec{x}; \vec{\theta}))^T \nabla \log p(\vec{x}; \vec{\theta})] \quad (14.42)$$

the rhs. is **Fisher information matrix**
 the Lagrangian

$$\mathcal{L}(\vec{\theta}, \Delta \vec{\theta}, \beta) = E[-\log p(\vec{x}; \vec{\theta})] + E[-\nabla \log p(\vec{x}; \vec{\theta})]^T \Delta \vec{\theta} + \frac{\beta}{2} (\Delta \vec{\theta})^T E[-\nabla^2 \log p(\vec{x}; \vec{\theta})] \Delta \vec{\theta} \quad (14.43)$$

$$\nabla_{\Delta \vec{\theta}} \mathcal{L} \stackrel{\text{set}}{=} \vec{0} \quad (14.44)$$

$$\vec{\theta}_{t+1} = \vec{\theta}_t + (E[-\nabla^2 \log p(\vec{x}; \vec{\theta})])^{-1} E[-\log p(\vec{x}; \vec{\theta})] \quad (14.45)$$

natural grad. methods and Newton's methods

- Newton's methods scaled the grad. by inv. of H , natural grad. methods scaled the grad. by inv. of Fisher information matrix
- Newton's methods makes 2nd-order approx. and then attempts to jump directly to the min. of this approx., natural grad. finds the direction of steepest descent in the space of prob. distr.
- Newton's methods compute over the dataset to analogous expectation wrt. $p_{EMP}(\vec{\theta})$, natural grad. is taken wrt. model distr. $p(\vec{\theta})$

14.6 Optimization Strategies and Meta-Algorithms

14.6.1 Coordinate Descent

coordinate descent : opt. one coordinate at a time

block coordinate descent : opt. a subset of variables simultaneously
 it is useful when opt. wrt. one group of var. is significantly more efficient
 coordinate descent is not a very good strategy when the val. of one variable strongly influences the opt. val. of another var.

14.6.2 Initialization Strategies

when training deep learning models, most alg. are strongly affected by the choice of initialization

Symmetry Breaking

how about init. $W_{kj}^{(l)} = 0, b_k^{(l)} = 0, \forall l, k, j?$

after each update, parameters corresponding to inputs going into each of two hidden units are identical

eg.,

$$s_1^{(2)} = s_2^{(2)} \quad (14.46)$$

$$\rightarrow \delta_1^{(2)} = \delta_2^{(2)} \quad (14.47)$$

$$\rightarrow \frac{\partial \text{err}}{\partial b_1^{(1)}} = \frac{\partial \text{err}}{\partial b_2^{(1)}} \quad (14.48)$$

$$\rightarrow b_1^{(1)} = b_2^{(1)} \quad (14.49)$$

$$\rightarrow a_1^{(2)} = a_2^{(2)} \quad (14.50)$$

How to Set Initial W

it is usually best to init. each unit to compute a diff. function from all of the other units

- this may help to make sure that no input patterns are lost in the nullspace of forward prop. and no grad. patterns are lost in the nullspace of backward prop.
- we could explicitly search for a large set of basis functions that are mutually diff. from each other, e.g., use Gram-Schmidt orthogonalization on init. W , but this often incurs a noticeable computational cost
- random init. from Gaussian or uniform distr. is computational cheaper
- it is reasonable to assume that approximately half of the weights will be positive and half of them will be negative

large init. W

- stronger symmetry breaking, avoid redundant units
- avoid losing signal during forward or backward prop.
- may result in exploding gradient values
- may result in extreme values that cause the g function saturate, causing complete loss of gradient through saturated units

from optimization perspective: larger W to prop. info. successfully

from regularization perspective: encourage making W smaller

we can think of init. the para. $\vec{\theta}$ to $\vec{\theta}_0$ as imposing a Gaussian prior $p(\vec{\theta})$ with mean $\vec{\theta}_0$

it makes sense to choose $\vec{\theta}_0$ to be near 0

heuristic normalized initialization

$$\vec{\theta} \sim \frac{2}{\sqrt{n}} \cdot N(\vec{0}, I) \quad (14.51)$$

this ensures that all neurons in the network initially have approximately the same output distribution and empirically improves the rate of convergence

*Pre-training
Greedy Supervised
Pre-training*

when computational resources allow it, it is usually a good idea to treat the init. scale of the weights as a hyperpara., and choose these scales using a hyperpara. search alg.

a good rule of thumb for choosing the initial scales is to look at the range or standard deviation of activations or gradients on a single minibatch of data

- if the weights are too small, the range of activations across the mini-batch will shrink as the activations propagate forward through the network
- by repeatedly identifying the first layer with unacceptably small activations and increasing its weights, it is possible to eventually obtain a network with reasonable initial activations throughout
- If learning is still too slow at this point, it can be useful to look at the range or standard deviation of the gradients as well as the activations

Sparse Initialization

another way to address the uncalibrated variances problem is to set all weight matrices to zero, but to break symmetry every neuron is randomly connected (with weights sampled from a small gaussian as above) to a fixed number of neurons below it

a typical number of neurons to connect to may be as small as 10

How to Set Initial b

usually set \vec{b} for each unit chosen const., often is 0, sometimes non-zero when

- in output layer, set b according to the stat. of the output, eg., set \vec{b} to be the stat. distr. of each class
- avoid saturation at init., eg., in ReLU set $b = 0.01$ rather than 0

14.6.3 Greedy Supervised Pre-training

Define 14.3 (Pre-training). *Training simple models on simple tasks before confronting the challenge of training the desired model to perform the desired task.*

Define 14.4 (Greedy Supervised Pre-training). *Algorithms that break supervised learning problems into other simpler supervised learning algorithms. Each stage consists in a supervised learning training task involving only a subset of the layers in the final neural network.*

procedure, see Fig. 14.8

- starts by training a sufficiently shallow architecture (a)
- discard the hidden-to-output layer (b)

- send the output of hidden layer as input to another supervised single hidden layer (c)
- repeat for as many layers as desired (d)
- to further improve the opt., jointly fine-tuning all the already pre-trained layers

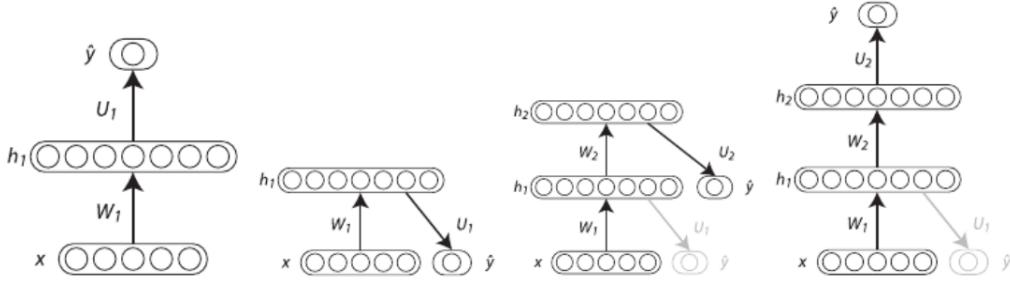


Figure 14.8: Greedy Supervised Pre-training

pre-training helps to provide better guidance to the intermediate levels of a deep hierarchy

14.6.4 Designing Models to Aid Optimization

it is more important to choose a model family that is easy to opt. than to use a powerful opt. alg.

ReLU, maxout units make opt. easier

14.6.5 Continuation Methods and Curriculum Learning

Continuation Methods

Define 14.5 (Continuation Methods). *A family of strategies that can make opt. easier by choosing init. points to ensure that local opt. spends most of its time in well-behaved regions of space.*

the idea is to construct a serious cost. functions $\{J_0(\vec{\theta}), J_1(\vec{\theta}), \dots, J_K(\vec{\theta})\}$

- these cost functions are designed to be increasingly difficult, eg., $J_0(\vec{\theta})$ is fairly easy to min., and $J_K(\vec{\theta})$, the most difficult, being $J(\vec{\theta})$
- easier means that it is well behaved over $\vec{\theta}$ space, a random init. is more likely to land in the region where local descent can mean the cost function successfully
- the sol. to one is a good init. point of the next

continuation methods are mostly designed with the goal of overcoming the challenge of local min.

to do so, it constructs easier cost functions by “blurring” the original cost function by approximating via sampling

$$J_k(\vec{\theta}) = \underset{\hat{\vec{\theta}} \sim N(\vec{\theta}, \sigma_k^2)}{\text{E}} [\hat{\vec{\theta}}] \quad (14.52)$$

- the intuition for this approach is that some non-convex functions become approximately convex when blurred
- often, this blurring preserves info. about the location of a global min., see Fig. 14.9
- it finds better local min. than local descent with random init.

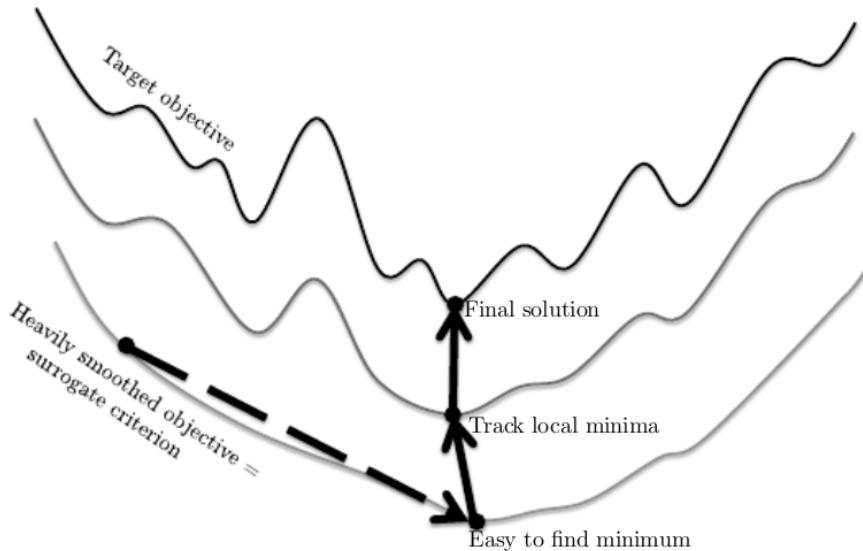


Figure 14.9: Continuation Methods

two problems

- blurred function is still non-convex
- min. of the blurred function may track to local rather than a global min.

Simulated Annealing

the blurring occurs by injecting noise

with a lot of noise, the effective obj. function (avg. over the noise) is flatter and convex

if the amount of noise is reduced sufficiently slowly, then one can show conv. to global min.

but rate at which the noise level is decreased might need to be extremely slow

Curriculum Learning

begin by learning simple concepts and progress to learning more complex concepts that dep. on these simpler concepts

earlier $J_k(\vec{\theta})$ are made easier by increasing the influence of simpler examples

- by assigning their contributions to the cost function larger coefficients
- or by sampling them more frequently

Chapter 15

Convolutional Neural Networks

Define 15.1 (Convolutional Neural Networks, CNN). *Convolutional neural networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.*

15.1 The Convolution Operation

suppose we are tracking the location of a spaceship with a laser sensor

- laser sensor provides a single output $x(t)$, the position of the spaceship at time t
- laser sensor is noisy, we weighted avg. several measurements to obtain a less noisy estimate
- more recent measurements are more relevant, more weight to recent measurement
- weight function $w(a)$, a is the age of a measurement
- the smoothed estimate of position s is

$$s(t) = \int x(a)w(t-a)da \stackrel{\text{def}}{=} x * a \quad (15.1)$$

discrete case

$$s(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \quad (15.2)$$

two-dimensional case

$$s(i, j) = \sum_a \sum_b x(a, b)w(i - a, j - b) \quad (15.3)$$

cross-correlation , see Fig. 15.1

$$s(i, j) = \sum_a \sum_b x(a, b)w(i + a, j + b) \quad (15.4)$$

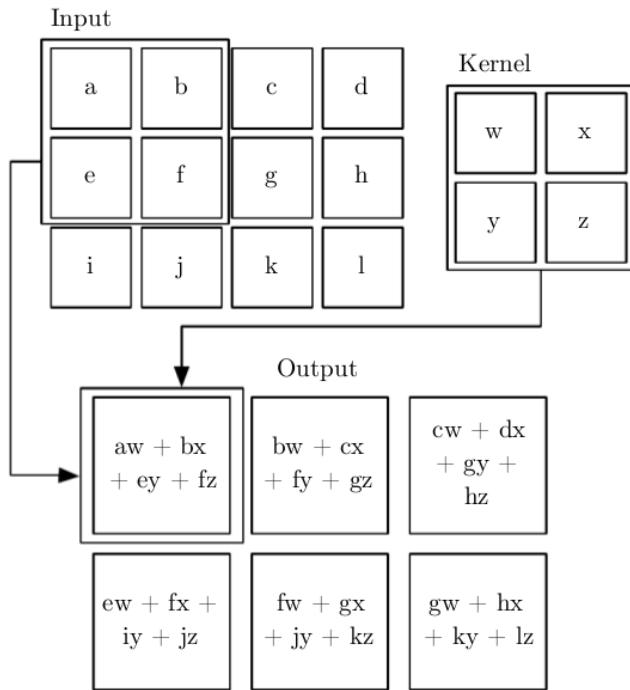


Figure 15.1: Cross-correlation, also called convolution in ML field, the restricted output is a “valid” convolution

discrete convolution can be viewed as multiplication by a (very sparse) matrix

- this is because kernel is usually much smaller than the input img.
- view conv. as mat. multiplication usually does not help to implement conv. op., but it is useful for understanding and designing NNet

15.2 Motivation

15.2.1 Sparse Interactions

traditional NNet layers use a mat. multiplication

- interaction betw. each input and each output
- every output unit interacts with every input unit

CNN use sparse interaction (sparse connectivity)

- making the kernel smaller than the input
- limit the # connections each output may have, see Fig. 15.2, 15.3
- eg., img. can have mlns. of pixels, but we can detect small, meaningful features such as edges with kernels that occupy only hundreds of pixels

- the receptive field of units in the deeper layers of a CNN is larger than the receptive field of the units in the shallow layers
- this allows the network to efficiently describe complicated interactions between many variables by constructing such interactions from simple building blocks that each describe only sparse interactions, see Fig. 15.4

Parameter Sharing

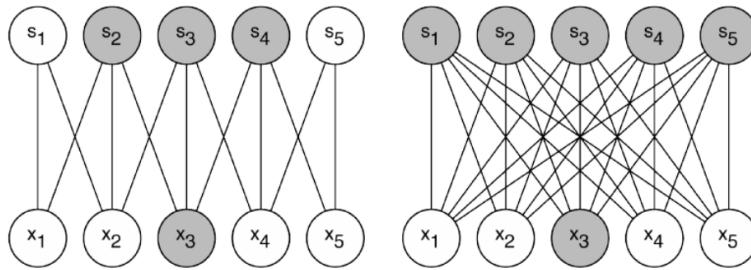


Figure 15.2: Sparse connectivity, diff. # of outputs are affected by one input unit

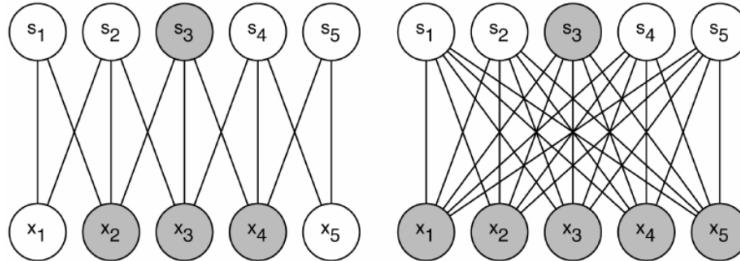


Figure 15.3: Sparse connectivity, diff. # of inputs (receptive field) affect one output unit

15.2.2 Parameter Sharing

Define 15.2 (Parameter Sharing). *Using the same parameter for more than one function in a model.*

in traditional NNet, each elem. of W is used exactly once when computing the output of a layer, it is multiplied by one element of the input and then never revisited

in CNN, each member of the kernel is used at every position of the input (may except some boundary pixels), it further reduce the storage requirements of the model, see Fig. 15.5

Equivariant Function

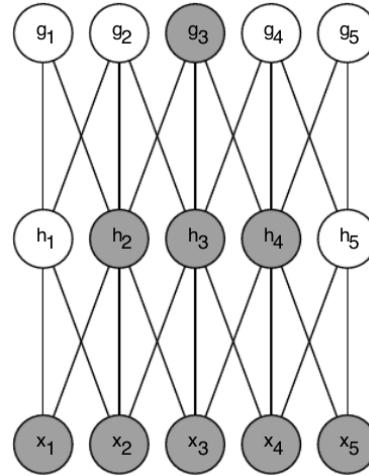


Figure 15.4: Units in the deeper layers may indirectly interact with a larger portion of the input

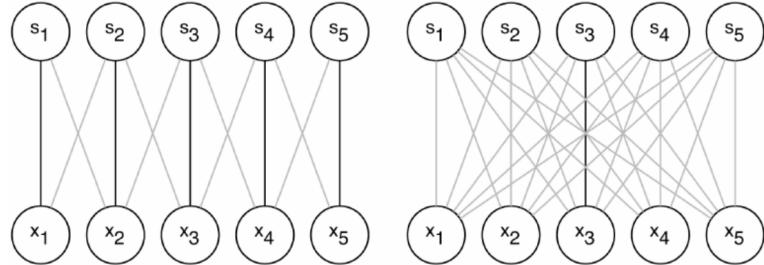


Figure 15.5: Parameter sharing: We highlight the connections that use a particular parameter in two different models

Fig. 15.6 shows how sparse connectivity and parameter sharing can dramatically improve the efficiency of a linear function for detecting edges in an image

15.2.3 Equivariant Representations

parameter sharing causes the layer to have property called equivariant to translation

Define 15.3 (Equivariant Function). *If the input changes, the output changes the same way. A function f is equivalent if for any function g*

$$f(g(x)) = g(f(x)) \quad (15.5)$$

In case of convolution, f is the convolution function, let g be any function that translate the input, the convolution is equivalent to g .

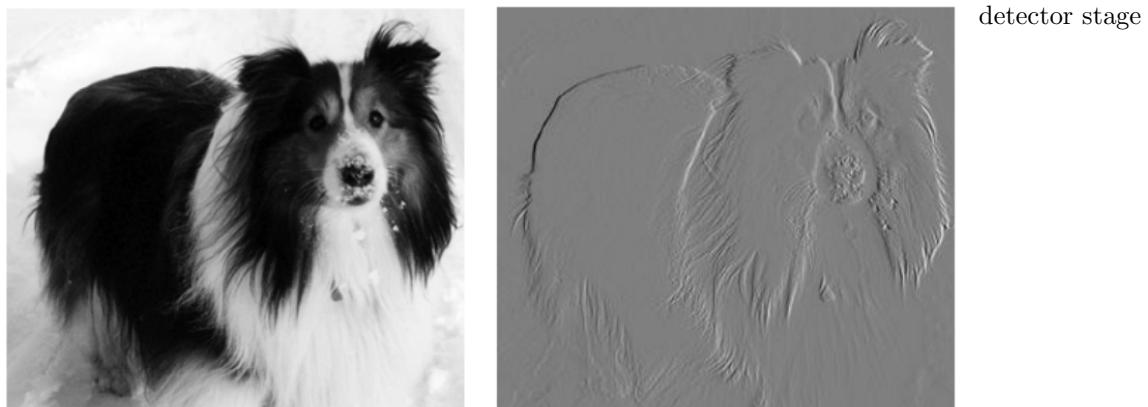


Figure 15.6: Detect vertical edges by computing $x(i, j) - x(i - 1, j)$, it is extremely efficient than matrix multiplication

eg., if we shift every element of x one position to the right

- $g(x(i, j)) = x(i - 1, j)$
- transform then convolution, the result will be the same as convolution then transform
- if we move the object in the input, its repr. will move the same amount in the output

but in some case, we don't want to share para., eg., if we are processing images that are cropped to be centered on an individual's face, we probably want to extract different features at different locations

conv. is not equivariant to some other transformations, such as scale and rotate

15.3 Pooling

a typical layer of CNN consists of 3 stages, see Fig. 15.7

- several convolutions in parallel to produce a set of presynaptic activations
- each presynaptic activation is run through a nonlinear activation function, such as reLU, called **detector stage**
- use a pooling function to modify the output of the layer further

a pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs, including

- maximum output within a rectangular neighborhood
- average of a rectangular neighborhood
- L_2 norm of a rectangular neighborhood
- weighted average based on the distance from the central pixel

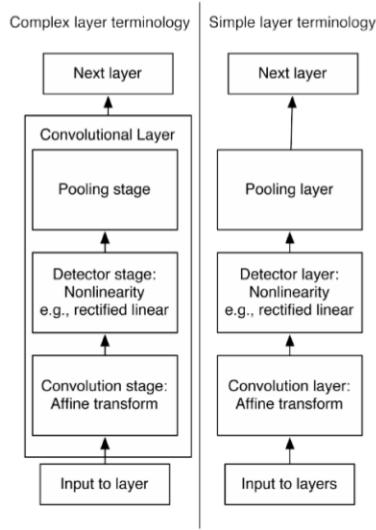


Figure 15.7: A typical layer of CNN. Left views CNN as a small number of relatively complex layers, with each layer having many stages. Right views CNN as a larger number of simple layers, not every “layer” has parameters.

pooling helps to make the repr. become invariant to small local translation of the input

- if we translate the input by a small amount, the values of most of the pooled outputs do not change, see Fig. 15.8
- this is useful if we care more about whether some feature is present than exactly where it is

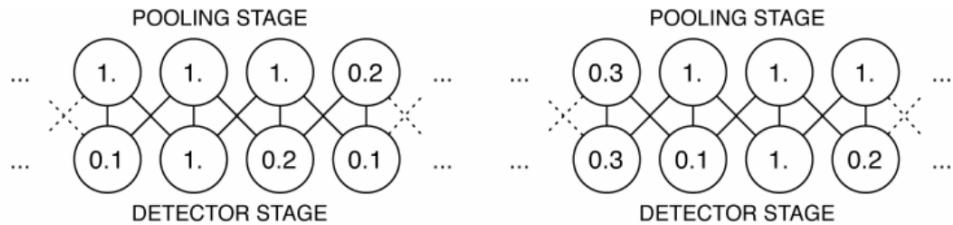


Figure 15.8: Max pooling introduces invariance. After the input has been shifted to the right by 1 pixel, every value in the bottom row has changed, but only half of the values in the top row have changed, because the max pooling units are only sensitive to the maximum value in the neighborhood, not its exact location.

if we pool over the outputs of separately parametrized convolutions, the features can learn which transformations to become invariant to, see Fig. 15.9

pooling can also reduces the representation size, reduces the computa-



Figure 15.9: By learning to have each filter be a different rotation of the “5” template, this max. pooling unit has learned to be invariant to rotation. This is in contrast to translation invariance, which is usually achieved by hard-coding the net to pool over shifted versions of a single learned filter.

tional and statistical burden on the next layer , note that the final pool has a smaller size, but must be included if we do not want to ignore some of the detector units, see Fig. 15.10

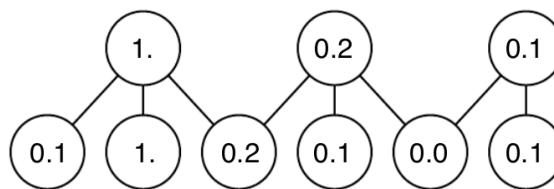


Figure 15.10: Pooling with downsampling

pooling is essential for handling inputs of varying size, eg., if we want to classify images of variable size, the input to the classification layer must have a fixed size, this is usually accomplished by varying the size of and offset between pooling regions so that the classification layer always receives the same number of summary statistics regardless of the input size

15.4 Convolution and Pooling as an Infinitely Strong Prior

Define 15.4 (Infinitely Strong Prior). *It places zero probability on some parameters and says that these parameter values are completely forbidden, regardless of how much support the data gives to those values.*

we can imagine a CNN as being similar to a fully connected net, but with an infinitely strong prior over its weights

- this infinitely strong prior says that the weights for one hidden unit must be identical to the weights of its neighbor, but shifted in space

- the prior also says that the weights must be zero, except for in the small, spatially contiguous receptive field assigned to that hidden unit convolution and pooling can cause underfitting
- convolution and pooling are only useful when the assumptions made by the prior are reasonably accurate
- if a task relies on preserving precision spatial information, then using pooling on all features can cause underfitting
- when a task involves incorporating information from very distant locations in the input, then the prior imposed by convolution may be inappropriate

we should only compare convolutional models to other convolutional models in benchmarks of statistical learning performance, eg., models that do not use convolution would be able to learn even if we permuted all of the pixels in the image

15.5 Variants of the Basic Convolutional Functions

15.5.1 Parallel Convolution

in practical, often convolution in parallel

- convolution with a single kernel can only extract one kind of feature, albeit at many spatial locations
- usually we want each layer of our network to extract many kinds of features, at many locations

15.5.2 Tensor Convolution

the input is usually not just a grid of real values, rather, it is a grid of vector-valued observations

assume input $A^{(1)} : r^{(1)} \times c^{(1)} \times n^{(1)}$

- $r^{(1)}$: rows
 - $c^{(1)}$: cols.
 - $n^{(1)}$: channels, eg., RGB img. has $n^{(1)} = 3$
- kernel tensor $W^{(1)} : \tau_1^{(1)} \times \tau_2^{(1)} \times n^{(1)} \times n^{(2)}$
- $\tau_1^{(1)}$: rows
 - $\tau_2^{(1)}$: cols.
 - $n^{(1)}$: input channels
 - $n^{(2)}$: # parallel kernels (output channels)
- output $A^{(2)} : r^{(2)} \times c^{(2)} \times n^{(2)}$

$$A^{(2)} = W^{(1)} * A^{(1)} \quad (15.6)$$

software implementations usually work in batch mode, so they will actually use 4-d tensors, with the fourth axis indexing different examples in the batch

because convolutional networks usually use multi-channel convolution, the linear operations they are based on are not guaranteed to be commutative, even if kernel-flipping is used, these multi-channel operations are only commutative if each operation has the same number of output channels as input channels

Strided Convolution

we want to skip over some positions of the img. in order to reduce the computational cost, if we want to sample only every u pixels in each direction in the output

$$S_{r,c,k}^{(2)} = \sum_{r'} \sum_{c'} \sum_{j=1}^{n^{(1)}} W_{r',c',j,k}^{(1)} A_{ur+r'-1,uc+c'-1,j}^{(1)} \quad (15.7)$$

- if $u = 1$, we conv. the img. with 1 spatial unit apart, this will lead to heavily overlapping **receptive fields** (partial extent of the local connectivity of conv.) between the columns, and also to large output volumes
- if we use higher strides, the receptive fields will overlap less and the resulting output volume will have smaller dimensions spatially

Valid Convolution

the convolution kernel is only allowed to visit positions where the entire kernel is contained entirely within the image

$$r^{(2)} = r^{(1)} - \tau_1^{(1)} + 1 \quad (15.8)$$

$$c^{(2)} = c^{(1)} - \tau_2^{(1)} + 1 \quad (15.9)$$

$$S_{r,c,k}^{(2)} = \sum_{r'=1}^{\tau_1^{(1)}} \sum_{c'=1}^{\tau_2^{(1)}} \sum_{j=1}^{n^{(1)}} W_{r',c',j,k}^{(1)} A_{r+r'-1,c+c'-1,j}^{(1)} \quad (15.10)$$

Same Convolution

valid conv.: output's spatial extent will shrink or we must use small kernels—both will significantly limit the expressive power of the network

can zero-pad of A to make conv. wider, to make an arbitrarily deep CNN, see Fig. 15.11

$$r^{(2)} = r^{(1)} \quad (15.11)$$

$$c^{(2)} = c^{(1)} \quad (15.12)$$

the input pixels near the border influence fewer output pixels than the input pixels near the center, this can make the border pixels somewhat underrepresented in the model

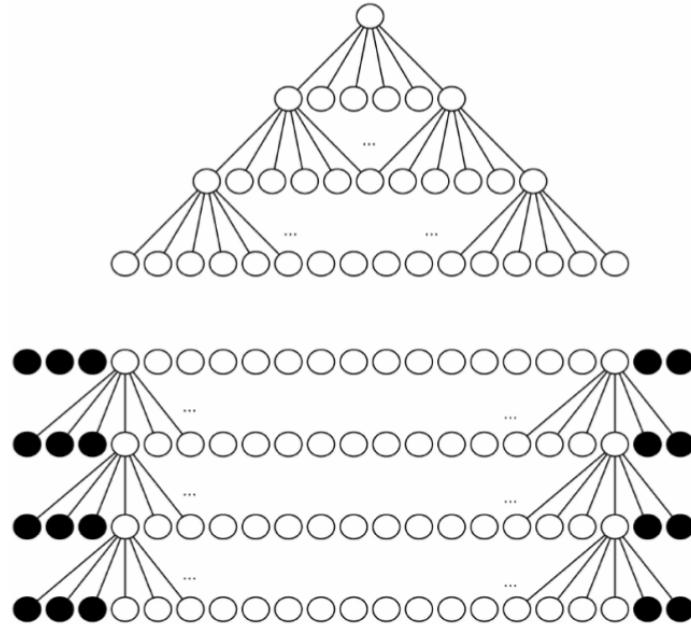


Figure 15.11: The effect of zero-padding. Top does not use zero-padding. Bottom adding five implicit zeroes to each layer.

Full Convolution

enough zeroes are added for every pixel to be visited $\tau^{(1)}$ times in each direction

$$r^{(2)} = r^{(1)} + \tau^{(1)} - 1 \quad (15.13)$$

$$c^{(2)} = c^{(1)} + \tau^{(1)} - 1 \quad (15.14)$$

the output pixels near the border are a function of fewer pixels than the output pixels near the center, this can make it difficult to learn a single kernel that performs well at all positions in the convolutional feature map

usually the optimal amount of zero padding (in terms of test set classification accuracy) lies somewhere between “valid” and “same” convolution

15.5.3 Locally Connected Layers

the adjacency mat. in the graph of our MLP is the same, but every connection has its own weigh

$$W^{(1)} :: \tau^{(1)} \times \tau^{(1)} \times n^{(1)} \times n^{(2)} \times r^{(2)} \times c^{(2)}$$

- $\tau^{(1)}$: rows / cols.
- $n^{(1)}$: input channels
- $n^{(2)}$: output channels
- $r^{(2)}$: output rows
- $c^{(2)}$: output cols.

$$S_{r,c,k}^{(2)} = \sum_{r'} \sum_{c'} \sum_{j=1}^{n^{(1)}} W_{r',c',j,k,r,c}^{(1)} A_{r+r'-1,c+c'-1,j}^{(1)} \quad (15.15)$$

unshared convolution

this is also called **unshared convolution**

- similar operation to discrete convolution with a small kernel
- without sharing parameters across locations

locally connected layers are useful when we know that each feature should be a function of a small part of space, but there is no reason to think that the same feature should occur across all of space, eg., if we want to tell if an image is a picture of a face, we only need to look for the mouth in the bottom half of the image

it can also be useful to make versions of convolution or locally connected layers in which the connectivity is further restricted, eg., constraint that each output channel k be a function of only a subset of the input channels

15.5.4 Tiled Convolution

compromise between a convolutional layer and a locally connected layer
we learn a set of kernels that we rotate through as we move through space, but the that immediately neighboring locations will have different filters

15.6 Back Propagation on Convolutional Layers

suppose multi-channel input img. is $A^{(l)}$, kenerl $W^{(l)}$, with stride u , output $S^{(l+1)}$

suppose we recieve a tensor Δ , st.

$$\Delta_{r,c,k}^{(l+1)} = \frac{\partial J(W, \vec{b})}{\partial S_{r,c,k}^{(l+1)}} \quad (15.16)$$

then

$$\frac{\partial J(W, \vec{b})}{\partial W_{r,c,j,k}^{(l)}} = \sum_{r'} \sum_{c'} \Delta_{r',c',k}^{(l+1)} A_{ur'+r,uc'+c,j}^{(l)} \quad (15.17)$$

$$\Delta_{r,c,j}^{(l)} = \frac{\partial J(W, \vec{b})}{\partial S_{r,c,j}^{(l+1)}} = \quad (15.18)$$

15.7 Structured Outputs

CNN can also be used to produce as output a high-dimensional structured object, such as the set of pixel-wise segmentation label predictions, in that case, both the input and the output of the network have an image topology

Separable Kernel

15.8 Data Types

CNN can process inputs with varying spatial extents

- these inputs can not be repr. by traditional mat. multiplication
- kernel is simply applied a diff. # of times depending on the size of the input
- the output of the conv. scales accordingly

15.9 Efficient Convolutional Algorithms

15.9.1 Fourier Transform

converting both the input and the kernel to the frequency domain using a Fourier transform

performing point-wise multiplication

converting back to the time domain using an inverse Fourier transform for some problem size, this can be faster than naive discrete convolution

15.9.2 Separable Kernels

Define 15.5 (Separable Kernel). *d-Dimensional kernel that can be expressed as the out product of d vectors, one vector per dimension.*

when kernel is separable, naive conv. is equiv. to compose d one-dim. conv. with each of these vectors, it is more faster and takes fewer paras.

15.10 Random or Unsupervised Features

obtain conv. kernels w/o supervised training random filters

- often works surprisingly well
- learning filters using unsupervised way—use greedy layer-wise unsupervised pretraining
 - train the first layer in isolation
 - then extract all features from the first layer only once
 - then train the second layer in isolation given those features, and so on

Chapter 16

Practical Methodology

16.1 Default Baseline Models

begin w/o using deep learning, eg., Naive Bayes

then use general models like ReLU (or their generalizations like Leaky ReLU, PreLus, maxout)

reasonable choice of opt. alg. is SGD with momentum with a decaying learning rate, including

- step decay: reduce the learning rate by some factor every few epochs typical values might be reducing the learning rate by a half every 5 epochs, or by 0.1 every 20 epochs

one heuristic you may see in practice is to watch the validation error while training with a fixed learning rate, and reduce the learning rate by a constant (e.g. 0.5) whenever the validation error stops improving

- exponential decay: $\alpha = \alpha_0 \exp(-kt)$
- $\frac{1}{t}$ decay: $\alpha = \frac{\alpha_0}{1+kt}$

or use Adam

unless m is in mlns. or more, you should include some mild regularization from the start

- early stopping should be used almost universally
- dropout is an excellent regularizer that is easy to implement and compatible with many models and training algorithms
- batch normalization is another excellent addition to standard neural networks and it has been shown to help substantially and is becoming commonly used

if your task is similar to another task that has been studied extensively, you will probably do well by first copying the model and algorithm that is already known to perform best on the previously studied task, eg., it is common to use the features from a convolutional network trained on ImageNet to solve other computer vision tasks

whether to begin by using unsupervised pretraining

- in CV, current unsupervised learning techniques do not bring a benefit, except in the semi-supervised setting
- the brute force way to practically guarantee success is to continually increase model VC dim. and training set size until the task is solved

16.2 Data Preprocessing

16.2.1 Mean Subtraction

centering the cloud of data around the origin along every dimension

$$\vec{\mu} = \frac{1}{m} \sum_{i=1}^m \vec{x}^{(i)} \quad (16.1)$$

$$\vec{x}^{(i)} \leftarrow \vec{x}^{(i)} - \vec{\mu}, \forall i \quad (16.2)$$

for img.

- subtract a single value from all pixels
 - or do so separately across the three color channels
- mean must be computed only over the training data and then subtracted equally from all splits (train / val. / test)

16.2.2 Normalization

normalizing the data dimensions so that they are of approximately the same scale

divide each dimension by its standard deviation, once it has been zero-centered

$$\sigma_j^2 = \frac{1}{m} \sum_{i=1}^m x_j^{(i)2}, \forall j \quad (16.3)$$

$$x_j^{(i)} \leftarrow \frac{x_j^{(i)}}{\sigma_j}, \forall i, j \quad (16.4)$$

another way is to normalize each dimension so that the min and max along the dimension is -1 and 1 respectively

normalization only makes sense if you have a reason to believe that different input features have different scales (or units), but they should be of approximately equal importance to the learning algorithm

for img., the relative scales of pixels are already approximately equal (and in range from 0 to 255), so it is not strictly necessary to perform this additional preprocessing step

see Fig. 16.1 for an example

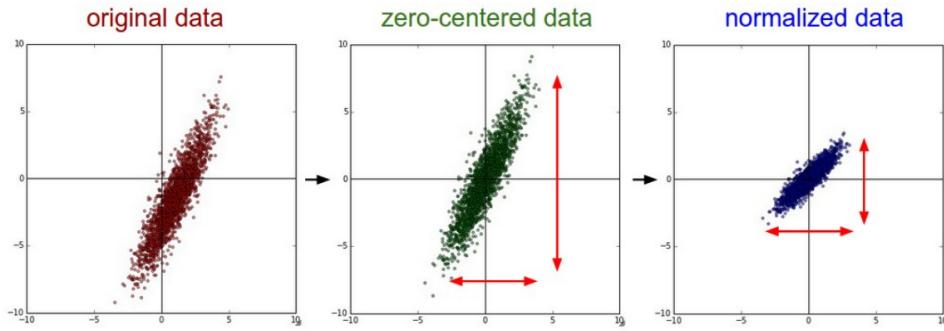


Figure 16.1: Mean-subtraction and normalization.

16.2.3 PCA and Whitening

PCA

data is first centered as described above
then PCA, consider $X \in \mathbb{R}^{n \times m}$

$$\Sigma = \frac{1}{m} XX^T = UDV^T \quad (16.5)$$

$$Z = U_{red}^T X \in \mathbb{R}^{K \times m} \quad (16.6)$$

- the cov. mat. of Z is diag.
- it is very often the case that you can get very good performance by training linear classifiers or neural networks on the PCA-reduced datasets, obtaining savings in both space and time

PCA Whitening

last step is whitening: divides every dimension by the eigenvalue to normalize the scale to have the cov. mat. equals I

$$z_j^{(i)} \leftarrow \frac{z_j^{(i)}}{\sqrt{\sigma_j + \varepsilon}} \quad (16.7)$$

- add ε say 10^{-5} to prevent division by zero
- one weakness of this transformation is that it can greatly exaggerate the noise in the data since it stretches all dimensions
- it can in practice be mitigated by stronger smoothing, ie., increase 10^{-5} to be a larger number

see Fig. 16.2 for an example

stretch img. into a vector and apply PCA and whitening, the result can be see in Fig. 16.3

```
1 Z = diag(1./sqrt(diag(D) + EPSILON)) * U' * X;
```

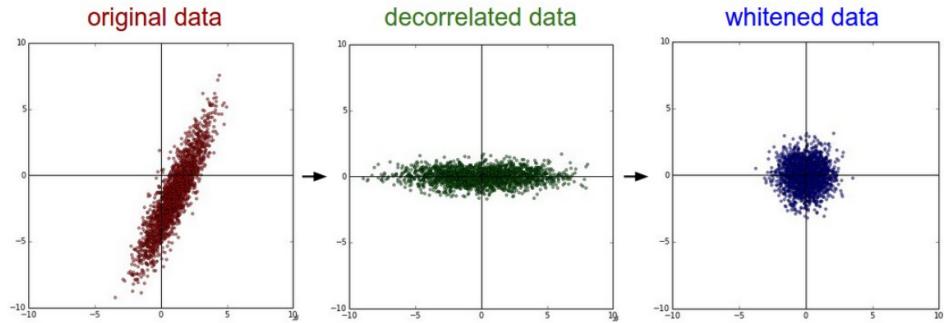


Figure 16.2: PCA and whitening.

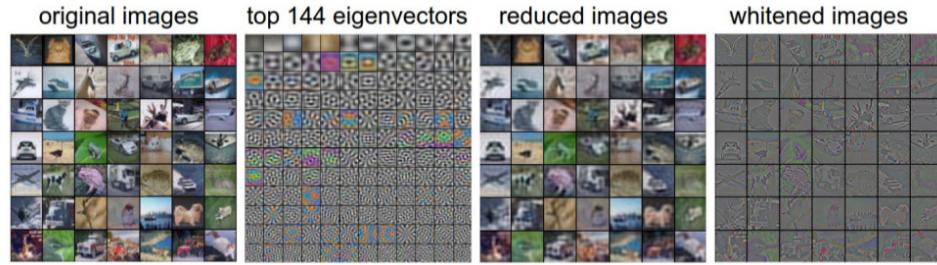


Figure 16.3: PCA and whitening on img. The top eigenvectors account for most of the variance in the data, and we can see that they correspond to lower frequencies in the images. After whitening, the lower frequencies (which accounted for most variance) are now negligible, while the higher frequencies like edges (which account for relatively little variance originally) become exaggerated.

ZCA Whitening

the way of getting the data to have covariance mat. I isn't unique

- if R is any orthogonal matrix: $RR^T = R^TR = I$ (less formally, if R is a rotation / reflection matrix)
- RZ will also have identity covariance
- in ZCA whitening, we choose $R = U$

it can be shown that out of all possible choices for R , this choice of rotation causes RZ to be as close as possible to the original input data X

when using ZCA whitening (unlike PCA whitening), we usually keep all n dimensions of the data, and do not try to reduce its dimension

```
1 X_ZCAwhite = U * diag(1./sqrt(diag(D) + EPSILON)) * U' * X;
```

Final Note

PCA / Whitening in these notes for completeness, but these transformations are not used with CNN, however, it is very important to zero-center the data, and it is common to see normalization of every pixel as well

16.3 Selecting Hyperparameters

16.3.1 Manual Hyperparameter Tuning

requires understanding what the hyperparameters do and how machine learning models achieve good generalization

the generalization error typically follows a U-shaped curve when plotted as a function of one of the hyperparameters

- at one extreme, the hyperparameter value corresponds to low VC dim., and generalization error is high because training error is high — underfitting
- at the other extreme, the hyperparameter value corresponds to high VC dim., and the generalization error is high because the gap between training and test error is high
- in the middle lies the optimal VC dim.

Learning Rate

learning rate is the most important hyperparameter, see Fig. 16.4, 16.5

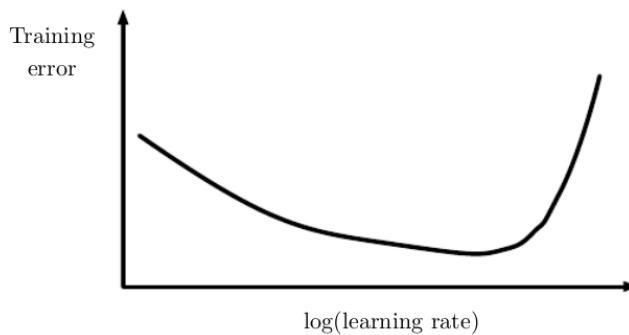


Figure 16.4: Learning rate and training error

- low learning rates the improvements will be linear
- high learning rates they will start to look more exponential, higher learning rates will decay the loss faster, but they get stuck at worse values of loss
- the right fig. of 16.5 shows a typical loss function over time,

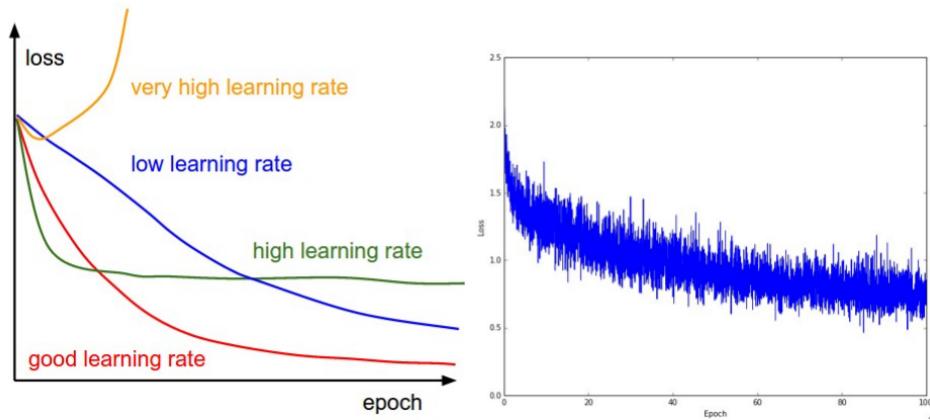


Figure 16.5: Different learning rate. X-axis is epochs, which measure how many times every example has been seen during training in expectation (e.g. one epoch means that every example has been seen once). It is preferable to track epochs rather than iterations since the number of iterations depends on the arbitrary setting of batch size.

the amount of “wiggle” in the loss is related to the batch size

- when the batch size is 1, the wiggle will be relatively high
 - when the batch size is the full dataset, the wiggle will be minimal because every gradient update should be improving the loss function monotonically (unless the learning rate is set too high)
- some people prefer to plot their loss functions in the log domain
- the exp. shape will be straight line
 - if multiple cross-validated models are plotted on the same loss graph, the differences between them become more apparent

tuning the parameters other than the learning rate requires monitoring both training and test error to diagnose whether your model is overfitting or underfitting, see Fig. 16.6

16.3.2 Automatic Hyperparameter Optimization algorithms

reduce the need to understand these ideas, but they are often much more computationally costly

Grid Search

- when # hyperparameters is 1, 2 or 3, grid search is commonly used
the picked vals. approx. on a log-scale
- learning rate with $0.1, 0.01, 10^{-3}, 10^{-4}, 10^{-5}$
 - # hidden units taken with 50, 100, 200, 500, 1000, 2000

Convolution kernel width	increased	Increasing the kernel width increases the number of parameters in the model	A wider kernel results in a narrower output dimension, reducing model capacity unless you use implicit zero padding to reduce this effect. Wider kernels require more memory for parameter storage and increase runtime, but a narrower output reduces memory cost.
Implicit zero padding	increased	Adding implicit zeros before convolution keeps the representation size large	Increased time and memory cost of most operations. Can be used to offset the output narrowing effect of wide kernels.

Figure 16.6: The effect of hyperparameter with VC dim.

often do it repeatedly, zooming in on one or more regions in the space of hyperparameters.

grid search wastes an amount of computation that is exponential in the number of non-influential hyperparameters

- run it in parallel
- but still may not provide a satisfactory size of search

Random Search

it is computationally more efficient than grid search

define a marginal distribution for each hyperparameter

- a Bernoulli or multinomial for binary or symbolic hyperparameters
- a uniform distribution on a log-scale
- e.g., $\log \alpha \sim U(-1, -5)$

one should not discretize or bin the values of the hyperparameters, this allows one to explore a larger set of values, and does not incur additional computational cost

performing random search rather than grid search allows you to much more precisely discover good values for the important ones, see Fig. 16.7

Model-Based Hyperparameter Optimization

build a model of the validation set error, then propose new hyperparameter guesses by performing optimization within this model, e.g., use Bayesian regression model

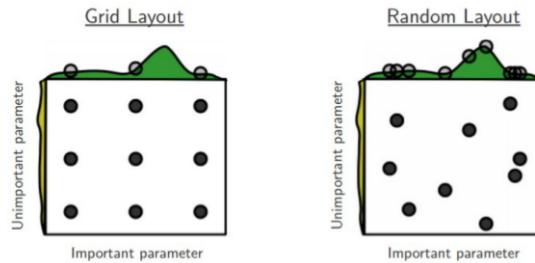


Figure 16.7: Grid search vs. random search.

Stage Your Search From Coarse to Fine

it can be helpful to perform the initial coarse search while only training for 1 epoch or even less, because many hyperparameter settings can lead the model to not learn at all, or immediately explode with infinite cost
 the second stage could then perform a narrower search with 5 epochs
 the last stage could perform a detailed search in the final range for many more epochs

16.4 Debugging Strategies

16.4.1 Visualize the Model in Action

if the task is object detection, view some images with the bounding boxes on them
 directly observing the machine learning model performing its task will help you to determine whether the quantitative performance numbers you're achieving seem reasonable
 plot the first-layer features visually, see Fig. 16.8

16.4.2 Visualize the Worst Mistakes

if use softmax, it gives the prob. of each class
 by viewing the training set examples that are the hardest to model correctly, you can often discover problems with the way the data has been preprocessed or labeled

16.4.3 Compare Train and Test error

evaluate if overfitting or underfitting, see Fig. 16.9
 increase T (which will increase VC dim.) to see the val. err.
 • val. err $\uparrow \rightarrow$ we are overfitting
 • val. err $\downarrow \rightarrow$ we are underfitting

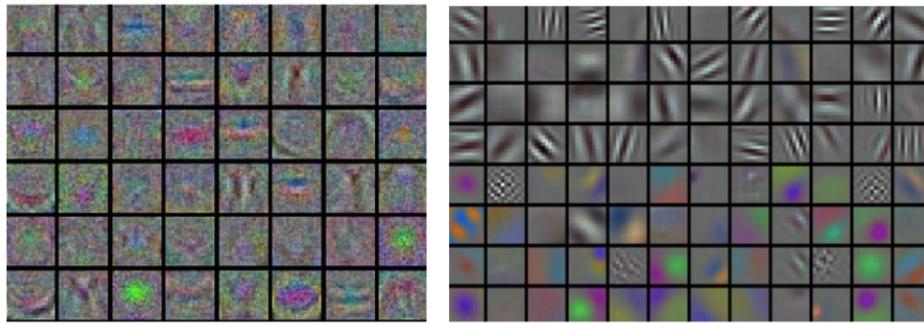


Figure 16.8: Left: Noisy features indicate could be a symptom: Unconverged network, improperly set learning rate, very low weight regularization penalty. Right: Nice, smooth, clean and diverse features are a good indication that the training is proceeding well.

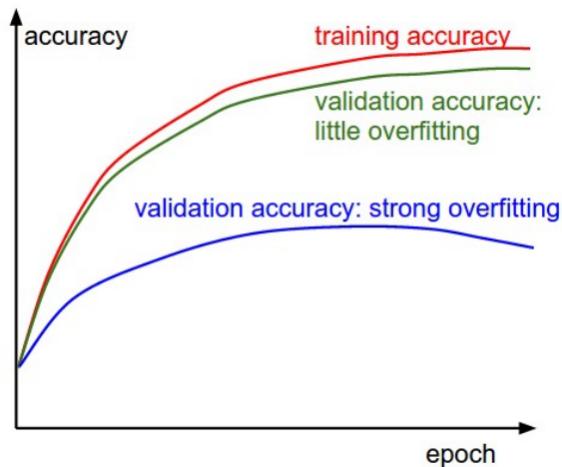


Figure 16.9: The gap between the training and validation accuracy indicates the amount of overfitting.

16.4.4 Fit a Tiny Dataset

if training set error is high, determine whether it is due to genuine underfitting or a bug

usually even small models can be guaranteed to fit a sufficiently small dataset

if cannot fit small dataset correctly, there is some bug in the alg.

when trying it out with just a few examples, more of the parameters are exercised and this tests the settings of the optimization hyperparameters, like learning rate

16.4.5 Numerical Gradient

numerical estimation of grad.
for $J(\theta), \theta \in \mathbb{R}$

$$\frac{d}{d\theta} J(\theta) \approx \frac{J(\theta + \varepsilon) - J(\theta - \varepsilon)}{2\varepsilon} \quad (16.8)$$

$\varepsilon = 10^{-5}$, or $10^{-4}, 10^{-6}$ is okay, but cannot be too small because of the finite-precision computation

for $J(\vec{\theta}), \vec{\theta} \in \mathbb{R}^n$ (eg., $\vec{\theta}$ is the unrolled version of $W^{(l)}, \vec{b}^{(l)}$)

$$\frac{\partial}{\partial \theta_1} J(\vec{\theta}) \approx \frac{J(\theta_1 + \varepsilon, \theta_2, \dots, \theta_n) - J(\theta_1 - \varepsilon, \theta_2, \dots, \theta_n)}{2\varepsilon} \quad (16.9)$$

$$\frac{\partial}{\partial \theta_2} J(\vec{\theta}) \approx \frac{J(\theta_1, \theta_2 + \varepsilon, \dots, \theta_n) - J(\theta_1, \theta_2 - \varepsilon, \dots, \theta_n)}{2\varepsilon} \quad (16.10)$$

$$\vdots \quad \vdots \quad (16.11)$$

$$\frac{\partial}{\partial \theta_n} J(\vec{\theta}) \approx \frac{J(\theta_1, \theta_2, \dots, \theta_n + \varepsilon) - J(\theta_1, \theta_2, \dots, \theta_n - \varepsilon)}{2\varepsilon} \quad (16.12)$$

$$(16.13)$$

```

1  for j = 1: n
2      thetaPlus = thetaVec;
3      thetaPlus(j) = thetaPlus(j) + EPSILON;
4      thetaMinus = thetaVec;
5      thetaMinus(j) = thetaVec(j) - EPSILON;
6      gradApprox = (J(thetaPlus) - J(thetaMinus)) / (2*EPSILON);
7  end

```

check $\text{gradApprox} \approx \text{grad}$

```

1  sum(abs(gradApprox - grad)) / max(sum(abs(gradApprox)), sum(abs
    (grad)));

```

- the relative error is 10^{-7} , and you will be happy
- if the function has kinks (non-differentiable parts), less than 10^{-4} is okay
- the deeper the network, the higher the relative errors will be
note: after checking that they have similar vals., turn off gradient checking, ow., your code will be very slow

16.4.6 Monitor Histograms of Activations and Gradient

- collect a large chunk of data, and stat. the activations or gradients
- eg., for ReLU, it can tell us how often are they off, how often they do
 - an incorrect initialization will let the variance of activations vanish extremely quickly in the higher layers of the network

we would like the magnitude of parameter updates (not grad., eg., this would be $\alpha \nabla J$) over a minibatch to represent something like 0.001 of the magnitude of the parameter, see Fig. 16.10

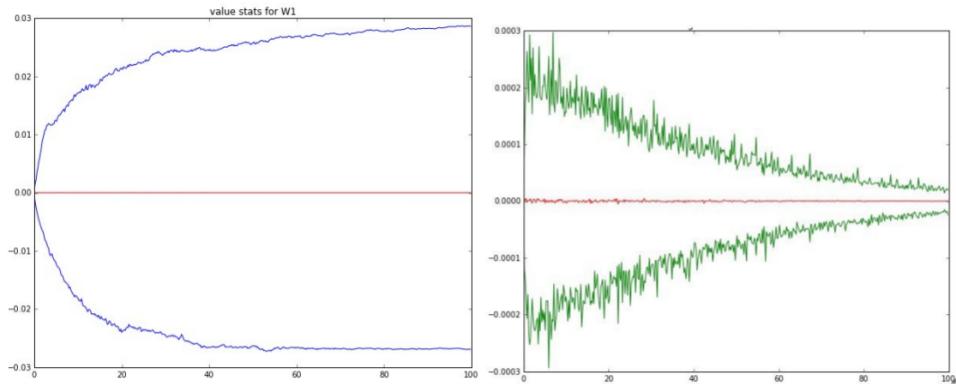


Figure 16.10: Example of a cross-validated 2-layer neural network where the learning rate is set relatively well. Left: stat. for weights. Right: stat. for updates of weights.

- if lower than 0.001 $\rightarrow \alpha$ might be too low
- if higher than 0.001 $\rightarrow \alpha$ might be too high

Chapter 17

Applications

17.1 Large Scale Deep Learning

the # neurons are large

17.1.1 Fast CPU Implementations

careful specialization of numerical computation routines can yield a large payoff

other strategies, besides choosing whether to use fixed or floating point, including optimizing data structures to avoid cache misses and using vector instructions

17.1.2 GPU Implementations

Define 17.1 (Graphics Processing Units, GPUs). *GPUs are specialized hardware components that were originally developed for graphics application. Video game rendering requires performing many operations in parallel quickly, like convert these 3-D coordinates into 2-D on-screen coordinates. The computations are fairly simple and do not involve much branching compared to the computational workload that a CPU usually encounters. GPUs are designed to have a high degree of parallelism and high memory bandwidth, at the cost of having a lower clock speed and less branching capability relative to traditional CPUs.*

neural networks also benefit from the same performance characteristics: neural networks usually involve large and numerous buffers of parameters, activation values, and gradient values, each of which must be completely updated during every step of training

Define 17.2 (General Purpose GPUs, GP-GPUs). *These GP-GPUs could execute arbitrary code, not just rendering subroutines. NVIDIA's CUDA*

model parallelism
 data parallelism
 asynchronous SGD
 parameter server
Model Compression

programming language provided a way to write this arbitrary code in a C-like language

- most writable memory locations are not cached, so it can actually be faster to compute the same value twice, rather than compute it once and read it back from memory
- GPU code is also inherently multi-threaded and the different threads must be coordinated with each other carefully
- make sure that each thread in a group executes the same instruction simultaneously, this means that branching can be difficult on GPU
 use ML library to avoid the design work on GPU

17.1.3 Large Scale Distributed Implementations

model parallelism : multiple machines work together on a single datapoint, each running a different part of the model

data parallelism : increase the size of the minibatch used for SGD, but usually get less than linear return

asynchronous SGD : several processor cores share the memory repr. the para.

- each core reads parameters without a lock, then computes a gradient, then increments the parameters without a lock
- this reduces the average amount of improvement that each gradient descent step yields, because some of the cores overwrite each other's progress
- but the increased rate of production of steps causes the learning process to be faster overall

para. are managed by a **parameter server** rather than stored in shared memory

17.1.4 Model Compression

goal: training a powerful model, but test on a tiny one

Define 17.3 (Model Compression). *Replace the original, expensive model with a smaller model that requires less resources to store and evaluate. It is applicable when the size of the orig. model is driven primarily by a need to prevent overfitting.*

the large model learn function $h(\vec{x})$, once we have the $h(\vec{x})$, we can generate a training set containing inf. many examples by applying h to randomly sampled points \vec{x} , then train the new, smaller model to match $h(\vec{x})$ on these points

best to sample the new \vec{x} points from a distr. resembling the actual test inputs that will be supplied to the model later

also can train with orig. points but copy other features of the model, conditional computation such as its posterior distr. over the incorrect classes

17.1.5 Dynamic Structure

data processing systems can dynamically determine which subset of many neural networks should be run on a given input

individual neural networks can also exhibit dynamic structure internally by determining which subset of features (hidden units) to compute given information from the input

this form of dynamic structure inside neural networks is sometimes called **conditional computation**

the system can run faster by computing these features only when they are needed

a venerable strategy for accelerating inference in a classifier is to use a cascade of classifiers

- to know for sure that a rare object is present, we must use a sophisticated classifier with high VC dim., that is expensive to run
- because the object is rare, we can usually reject inputs as not containing the object with much less computation
- we can train a sequence of classifiers
the first classifiers in the sequence have low VC dim., and are trained to have high recall, ie., they are trained to make sure we do not wrongly reject an input when the object is present
the final classifier is trained to have high precision
- at test time, we run inference by running the classifiers in a sequence, abandoning any example as soon as any one element in the cascade rejects it
- overall, this allows us to verify the presence of objects with high confidence, using a high VC dim., but does not force us to pay the cost of inference in a high VC dim. model for every example.

17.1.6 Specialized Hardware Implementations of Deep Networks

recently using FPGA

8 or 16 bits precision can suffice for using or training deep NNet

17.2 Computer Vision

17.2.1 Preprocessing

preprocessing of this kind is usually designed to remove some kind of variability in the input data that is easy for a human designer to describe

Constrast
global constrast
normalization, GCN

and that the human designer is confident has no relevance to the task

when training with large datasets and large models, this kind of pre-processing is often unnecessary, and it is best to just let the model learn which kinds of variability it should become invariant to

the img. should be standardized st. that is lie in $[0, 1]$ or $[-1, 1]$

17.2.2 Contrast Normalization

one of the most obvious sources of variation that can be safely removed for many tasks is the amount of contrast in the image

Define 17.4 (Contrast). *Contrast refers to the magnitude of the difference between the bright and the dark pixels in an image.*

in deep learning, suppose the img. is $A : r \times c \times n$, define contrast

$$\mu = \frac{1}{rcn} \sum_{r'=1}^r \sum_{c'=1}^c \sum_{j'=1}^n A_{r',c',j} \quad (17.1)$$

$$C = \sqrt{\frac{1}{rcn} \sum_{r'=1}^r \sum_{c'=1}^c \sum_{j'=1}^n (A_{r',c',j} - \mu)^2} \quad (17.2)$$

global contrast normalization, GCN : prevent images from having varying amounts of contrast by subtracting the mean from each image, then rescaling it so that the standard deviation across its pixels is equal to some constant

when img. has very low but non-zero contrast (all pixels have nearly same intensity)

- often this img. has little info. content
- dividing by the std. will amplify sensor noise or compression artifacts use instead

$$A'_{r,c,j} = \frac{A_{r,c,j} - \mu}{\max(\varepsilon, \sqrt{\lambda + \frac{1}{rcn} \sum_{r'=1}^r \sum_{c'=1}^c \sum_{j'=1}^n (A_{r',c',j} - \mu)^2})} \quad (17.3)$$

in practise, if the img. is carefully chosen st. they are unlikely to contain nearly const. intensity

- it is safe to set $\lambda = 0$
- set ε to extremely low val. like 10^{-8}

when imgs. are cropped randomly and very small, it is likely to have nearly const. intensity

- can use $lambda = 10, \varepsilon = 0$
- GCN maps examples onto a sphere, see Fig. 17.1
- a. raw input data may have any norm
- b. GCN with $\lambda = 0$ maps all non-zero examples perfectly onto a sphere

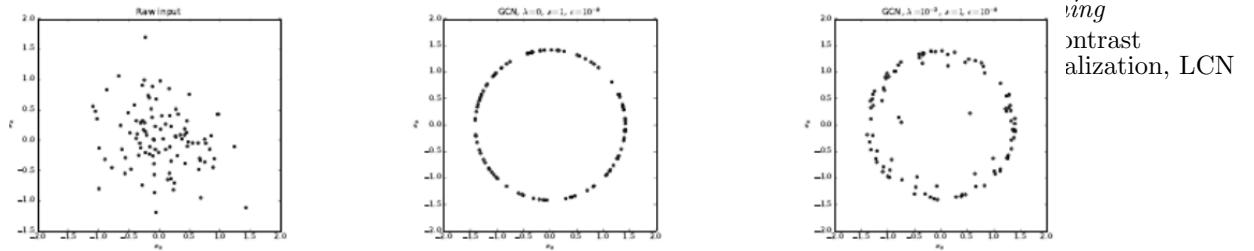


Figure 17.1: GCN maps examples onto a sphere.

- Regularized GCN, with $\lambda > 0$, draws examples toward the sphere but does not completely discard the variation in their norm
 - responding to multiple distances in the same direction requires hidden units with collinear weight vectors but different biases, such coordination can be difficult for the learning algorithm to discover
 - GCN reduces each example to a direction rather than a direction and a distance

Define 17.5 (Whitening). *It does not refer to making the data lie on a spherical shell, but rather to rescaling the principal components to have equal variance, so that the multivariate normal distribution used by PCA has spherical contours, it is not the same operation as GCN.*

Global contrast normalization will often fail to highlight image features we would like to stand out, such as edges and corners

- if we have a scene with a large dark area and a large bright area (such as a city square with half the image in the shadow of a building)
- then GCN will ensure there is a large difference between the brightness of the dark area and the brightness of the light area
- it will not, however, ensure that edges within the dark region stand out

local contrast normalization, LCN ensures that the contrast is normalized across each small window, rather than over the image as a whole, see Fig. 17.2

- can use mean or Gaussian weighted mean window
- it can be used as a nonlinearity to the hidden layers of a network, as well as a preprocessing operation applied to the input
- smaller windows are more likely to contain values that are all nearly the same as each other, we typically need to regularize local contrast normalization to avoid division by zero

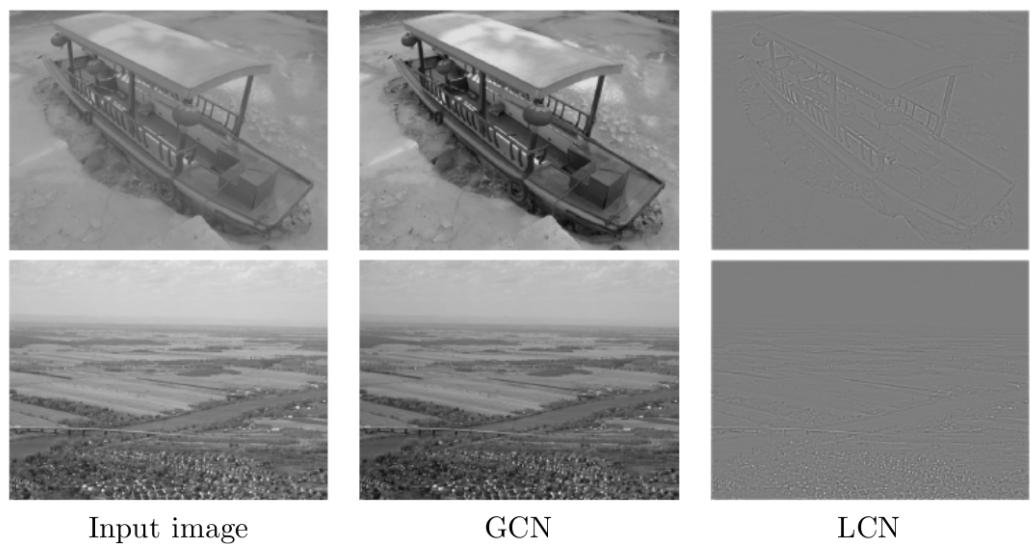


Figure 17.2: GCN and LCN. GCN places all images on roughly the same scale. LCN modifies the image much more, discarding all regions of constant intensity, this allows the model to focus on just the edges, regions of fine texture, such as the houses in the second row, may lose some detail due to the bandwidth of the normalization kernel being too high.

Part IV

Finale

Chapter 18

Finale

18.1 Feature Exploitation Techniques

18.1.1 Exploiting Numerous Features via Kernel

numerous features within some ϕ : embedded in kernel with inner product operation, see Fig. 18.1

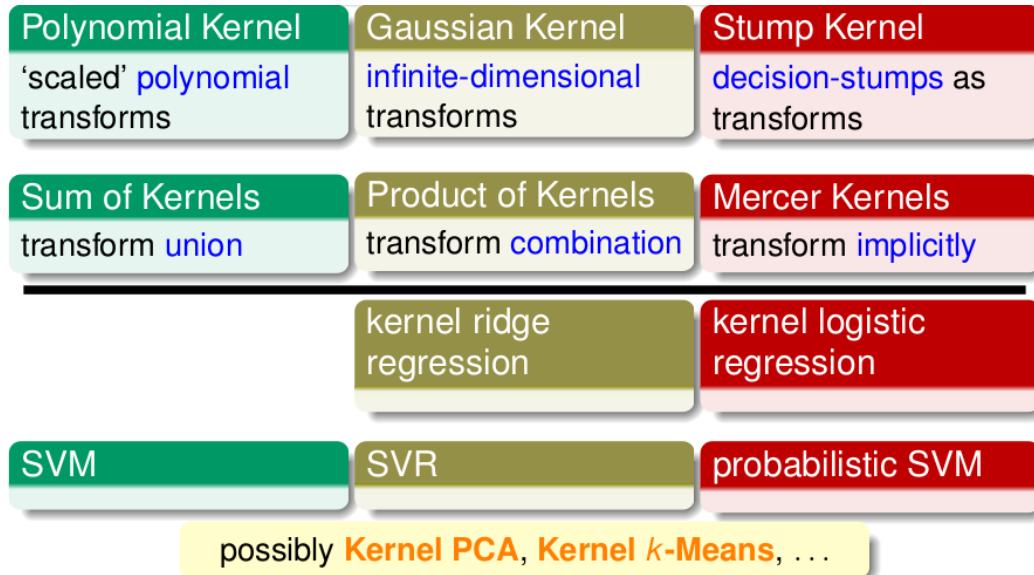


Figure 18.1: Exploiting Numerous Features via Kernel

18.1.2 Exploiting Predictive Features via Aggregation

predictive features within some ϕ : $\phi(\vec{x})_t = h(\vec{x})_t$, see Fig. 18.2

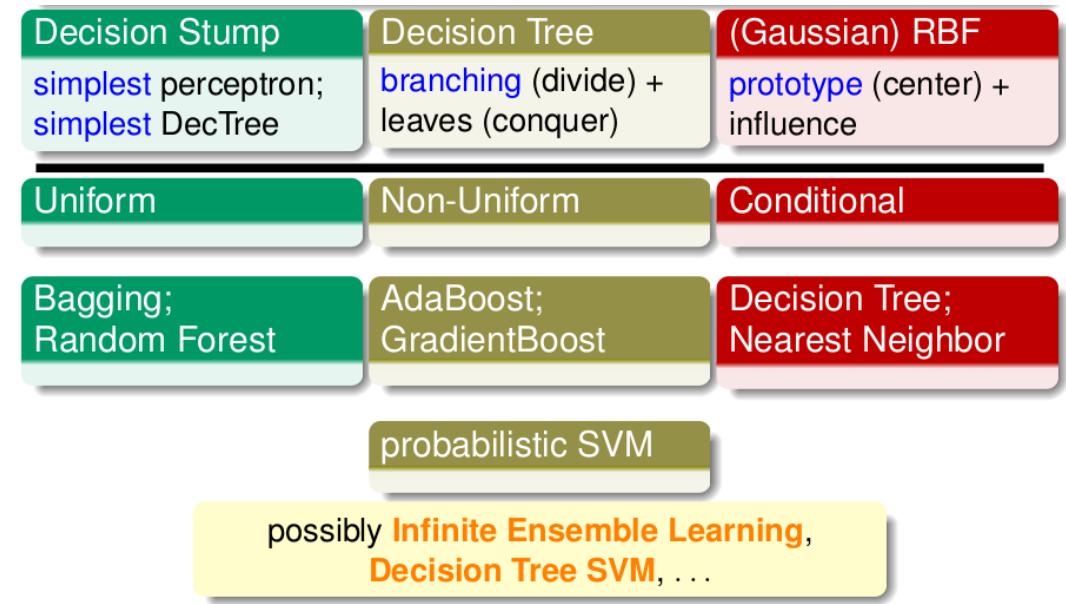


Figure 18.2: Exploiting Predictive Features via Aggregation

18.1.3 Exploiting Hidden Features via Extraction

hidden features within some ϕ : as hidden variables to be ‘jointly’ optimized with usual weights

possibly with the help of unsupervised learning, see Fig. 18.3

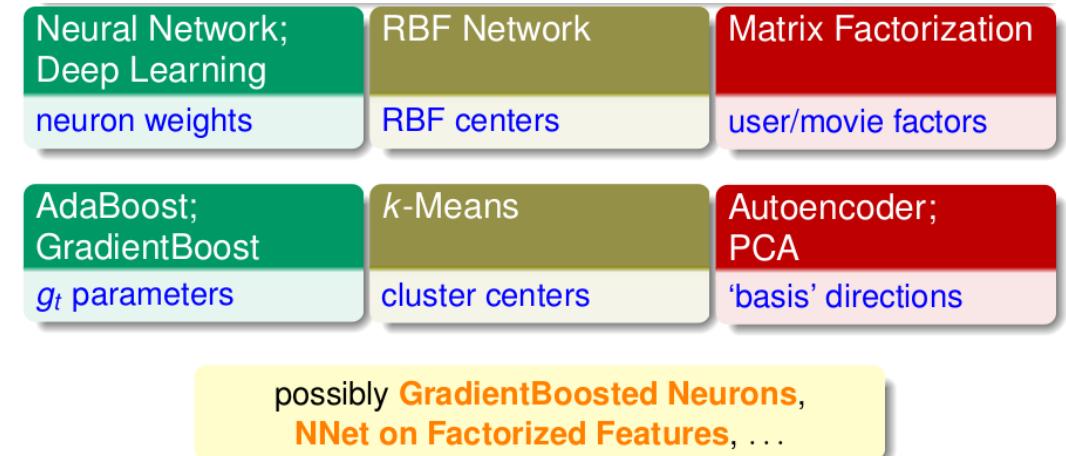


Figure 18.3: Exploiting Hidden Features via Extraction

18.1.4 Exploiting Low-Dimensional Features via Compression

low-dimensional features within some ϕ : compressed from original features, see Fig. 18.4

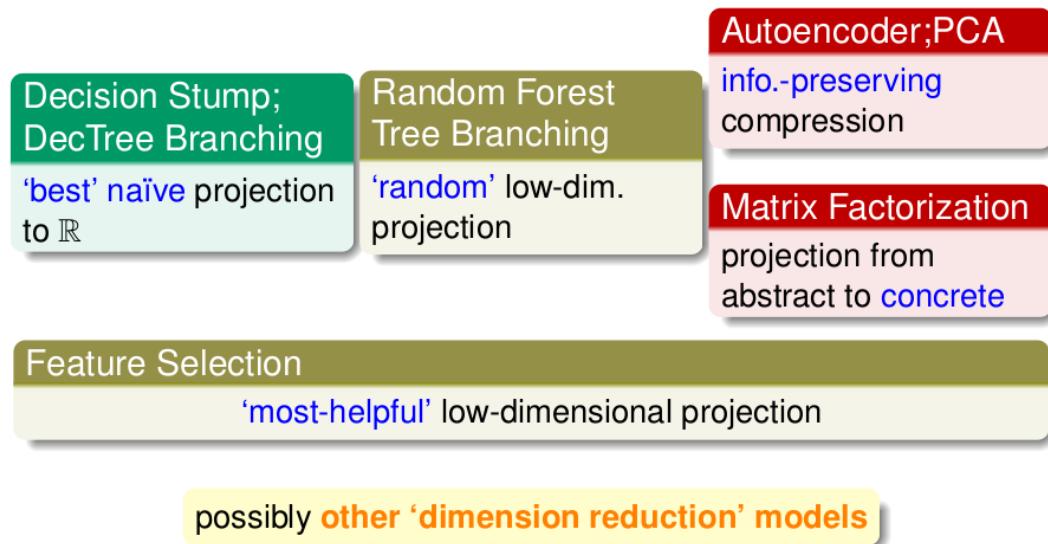


Figure 18.4: Exploiting Low-Dimensional Features via Compression

18.2 Error Optimization Techniques

18.2.1 Numerical Optimization via Gradient Descent

when ∇J ‘approximately’ define, use it for 1-st order approximation, see Fig. 18.5

$$\vec{\theta} \leftarrow \vec{\theta} - \alpha \nabla J \quad (18.1)$$

18.2.2 Indirect Optimization via Equivalent Solution

when difficult to solve original problem, seek for equivalent solution, see Fig. 18.6

18.2.3 Complicated Optimization via Multiple Steps

when difficult to solve original problem, seek for ‘easier’ sub-problems, see Fig. 18.7

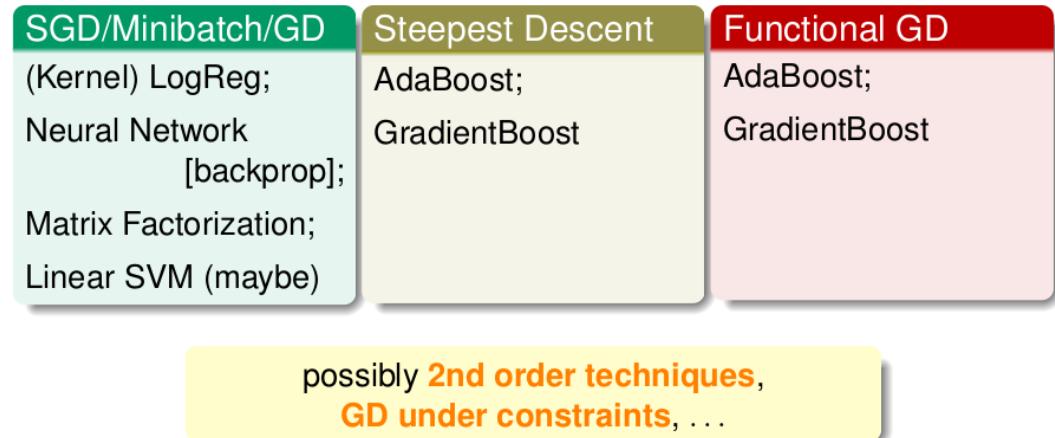


Figure 18.5: Numerical Optimization via Gradient Descent

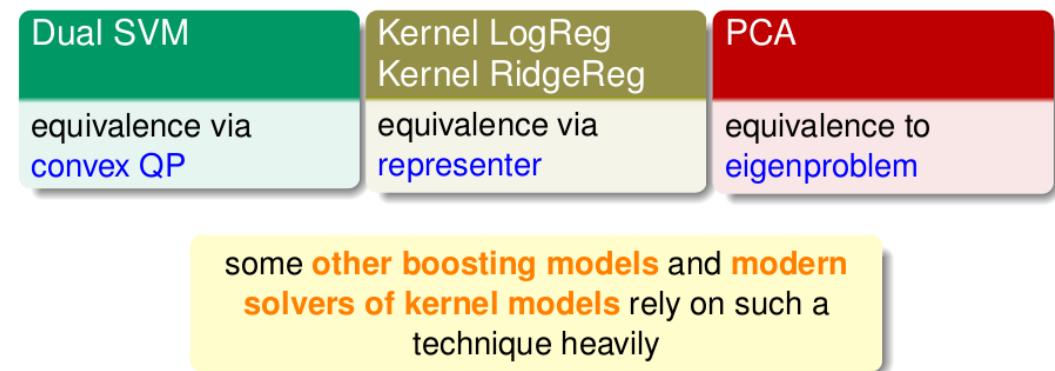


Figure 18.6: Numerical Optimization via Gradient Descent

18.3 Overfitting Elimination Techniques

18.3.1 Overfitting Elimination via Regularization

when model too ‘powerful’: add brakes somewhere, see Fig. 18.8

18.3.2 Overfitting Elimination via Validation

when model too ‘powerful’: check performance carefully and honestly, see Fig. 18.9

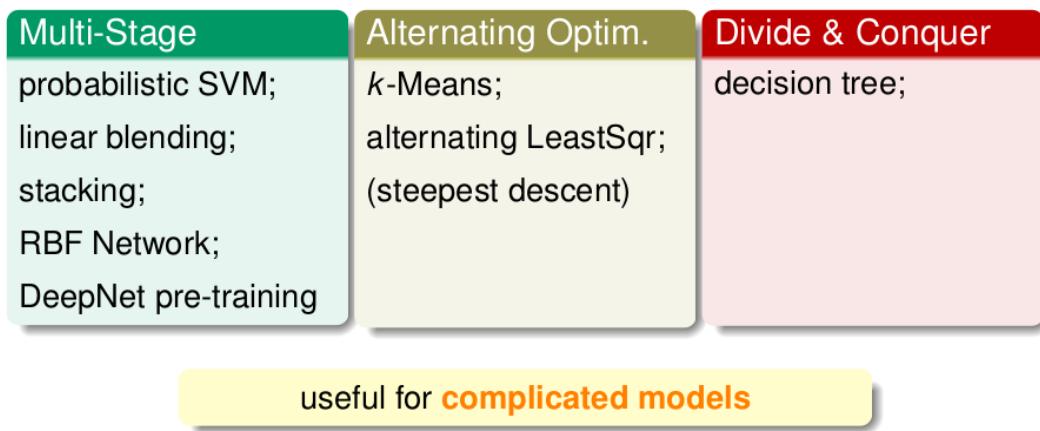


Figure 18.7: Numerical Optimization via Gradient Descent

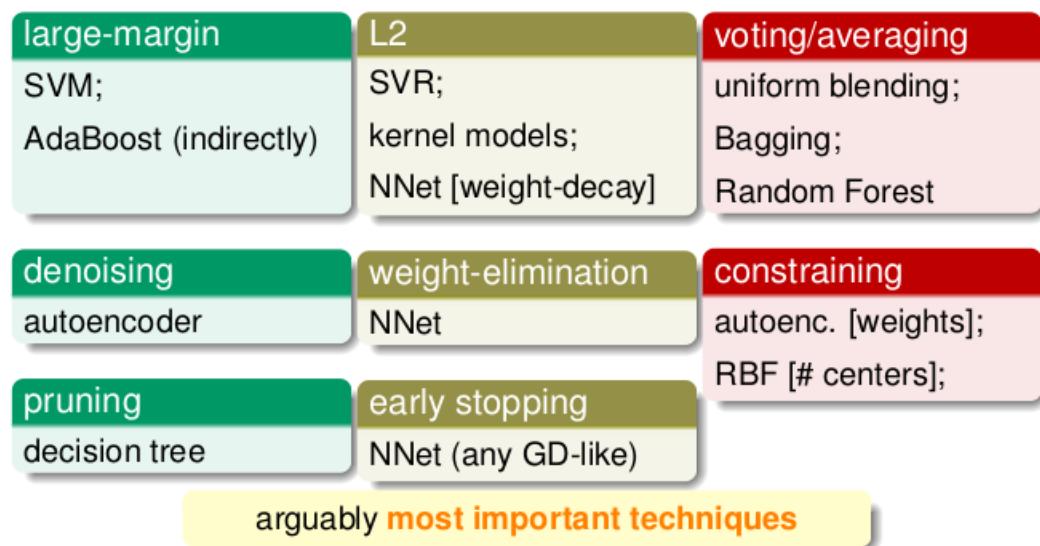


Figure 18.8: Overfitting Elimination via Regularization

18.4 Machine Learning in Practice

18.4.1 NTU KDDCup 2010

- feature engineering and classifier ensemble
- linear blending of
- Logistic Regression + many rawly encoded features
- Random Forest + human-designed features

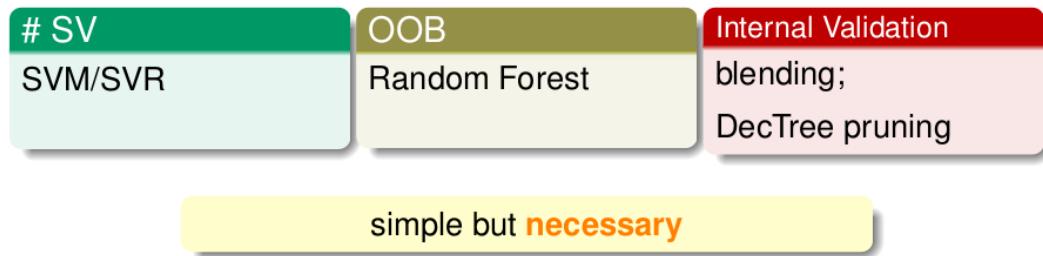


Figure 18.9: Overfitting Elimination via Regularization

18.4.2 NTU KDDCup 2011

a linear ensemble of individual and blended models for music rating prediction

NNet, DecTree-like, and then linear blending of

- Matrix Factorization variants, including probabilistic PCA
- Restricted Boltzmann Machines: an 'extended' autoencoder
- k Nearest Neighbors
- Probabilistic Latent Semantic Analysis: an extraction model that has 'soft clusters' as hidden variables
- linear regression, NNet, & GBDT

18.4.3 NTU KDDCup 2012

a two-stage ensemble of diverse models for advertisement ranking
NNet, GBDT-like, and then linear blending of

- Linear Regression variants, including linear SVR
 - Logistic Regression variants
 - Matrix Factorization variants
- 'key' is to blend properly without overfitting

18.4.4 NTU KDDCup 2013

combination of feature engineering and ranking models for paper- author identification

linear blending of

- Random Forest with many many many trees
- GBDT variants

with tons of efforts in designing features

'another key' is to construct features with domain knowledge

18.4.5 ICDM 2006 Top 10 Data Mining Algorithms

C4.5: another decision tree

k-Means

SVM

Apriori: for frequent itemset mining

EM: 'alternating optimization' algorithm for some models

PageRank: for link-analysis, similar to matrix factorization

AdaBoost

k Nearest Neighbor

Naive Bayes: a simple linear model with 'weights' decided by data statistics

CART

personal view of five missing ML competitors

- LinReg
- LogReg
- Random Forest
- GBDT
- NNet

Index

- L_1 regularizer, 198
- L_2 regularizer, weight decay, 197
- θ -sublevel set, 90
- ε -insensitive error, 246
- k nearest neighbor, 338
- k -means Algorithm, 284
- A Mixture Distribution, 53
- Absolute Error, 325
- Absolute Value Rectification, 322
- AdaBoost Algorithm, 263
- AdaBoost-DTree, 274
- adjoint, 17
- asymptotically unbiased, 57
- Axioms of Probability, 34
- Back Propagation, 330
- Backward Propagation in a Flow Graph
 - 331
- backward search, 203
- bagging, 259
- Basic CART Algorithm, 267
- Bayes's Rule, 41
- Bayesian Probability, 34
- Bayesian Statistics, 63
- Bernoulli, 49
- Bias, 57
- Binomial, 49
- Boltzmann machines, 354
- Bootstrapp Aggregation, 259
- bootstrapping, 259
- catastrophic forgetting, 325
- categorical features, 302
- Central Limit Theorem, 52
- Chebyshev's Inequality, 68
- Chi-Squared, 52
- clique, 40
- cocktail party problem, 298
- Collaborative Filtering Algorithm, 303
- column rank, 11
- competition, 326
- complementary slackness, 98
- complimentary slackness, 223
- concatenate, 386
- concave, 88
- Condition Number, 73
- Conditional Distribution, 39
- Conditional Entropy, 46
- Conditional Expectation, 43
- Conditional Probability, 35
- Conditioning, 73
- Consistency, 61
- constraint qualifications, 98
- Continuous Random Variable, 36
- convex combination, 86
- Convex Functions, 88
- Convex Set, 86
- Convolutional Neural Network (CNN), 356
- Coordinate Ascent, 235
- coordinate ascent, 67
- Covariance, 44
- Covariance Matrix, 44
- covariates, 57
- Critical Point / Stationary Point, 74
- Cross Entropy, 48
- Cross Entropy Error, Bernoulli Negative Log-Likelihood, 325
- Cumulative Distribution Functions (CDF), 36
- dead units, 199

- definiteness, 8
degree of belief, 34
Delta Method, 56
determinant, 16
diagonalizable, 18
differential entropy, 46
Dirac, 52
Directed Graphical Model, 40
Discrete Random Variable, 35
Distortion Function, 283
Dual Hard-Margin SVM, 225
Dual SVM, 223
Dual SVM with QP solver, 224
Dual SVR, 249

Early Stopping, 343
Effective Number of Parameters, 195
eigenfaces, 298
Elbow Method, 284
EM Algorithm, 66, 67
EM with Mixtures of Gaussians, 287
Empirical, 52
empirical distribution, 62
Event Space \mathcal{F} , 34
events, 34
exact interpolation for function approximation, 339
excessive linearity, 348
Expectation, 42
Exponential, 50
exponential error measure, 276

feed-forward neural network, 353
Feedforward Deep Networks, 317
filters, 357
first-order approximation, 88
Fisher scoring, 85
Flow Graph, 330
for, 159, 256, 371
Forward Propagation, 328
Forward Propagation in a Flow Graph, 330
Forward Search, 203
Frequentist Probability, 34

Frequentist Statistics, 62
Frobenius norm, 10
full rank, 11
Function Estimation, 57

Gamma, 51
Gaussian (Normal), 51
Gaussian Mixture Model, 53
Geometric, 49
Globally Optimal, 93
Gradient Boosted Decision Tree (GBDT), 281
gradient descent, 73
Gradient Descent on Linear Regression, 160
Gram matrix, 17

Haar basis, 21
Hard tanh, 322
hinge error measure, 240
homogeneity, 8
Hopfield nets, 354
Hyperbolic Tangent, 321
hyperparameter selection algorithm, 343

ICA, 384
idea, 159
if, 159
ill-conditioning, 73
in, 105, 159, 179
Independence, 41
Independent, 35
independent component analysis (ICA), 383
input, 269

Jacobian matrix, 76
Joint Cumulative Distribution Function, 36
Joint Probability Density Function, 37
Joint Probability Mass Function, 37

kernel engineering, 313

- kernel function, 226
 Kernel Hard-Margin Dual SVM with QP solver, 227
 kernel trick, 227
 KL Divergence, 47
 KL divergence, 203
 Laplace (Double Exponential), 51
 LASSO, 198
 latent random variables, 65, 285
 latent variable, 53
 Law of Total Expectation (Law of Iterated Expectation), 43
 Law of Total Probability, 35
 Law of Total Variance, 43
 Leaky ReLU, 322
 learning rate, 76
 Least-Squares SVM (LSSVM), 246
 Level Curves, Isocountours, 53
 Lin. Reg. Alg., 148
 Lin. Reg. for Classification, 154
 line search, 76
 linear, 20
 Linear Blending and Any Blending (Stacking), 258
 Linearity of Expectation, 43
 Locally Optimal, 93
 Logistic Regression Alg., 159, 179
 Logistic Sigmoid Function, 69
 Map-reduce, 81
 Marginal Cumulative Distribution Functions, 36
 Marginal Probability Density Function, 38
 Marginal Probability Mass Function, 37
 Markov's Inequality, 68
 max norm, 9
 Maximum A Posterior (MAP) Point Estimate, 64
 Maximum Likelihood Estimation, 61
 Maximum Likelihood Estimation on Conditional Probability, 62
 Maxout, 323
 mean, 42
 Mean Square Error, 60
 Mercers condition, 231
 method of steepest descent, 76
 Mini-batch Logistic Regression Alg., 81
 mixing matrix, 299
 Mixture of Gaussians, 285
 Model Identifiability, 350
 Moment Generating Function (MGF), 45
 Momentum, 367
 Moore-Penrose Pseudoinverse, 23
 Multilayer Perceptrons (MLPs), 317
 Multinomial, 50
 Multivariate Gaussian, 51
 Mutual Information, 47
 mutual information, 203
 Mutually Independent, 35
 nearest neighbor model, 338
 non-negative, 8
 Nuclear Norm, 9
 One-Versus-All (OVA) Decomposition, 181
 One-Versus-One (OVO) Decomposition, 182
 orthogonal, 11
 orthogonal complements, 14
 orthogonal matrix, 11
 orthonormal, 11
 out-of-bag (OOB), 270
 outcome, 34
 Overflow, 71
 parameter sharing, 344
 Parametric ReLU, PReLU, 322
 PCA whitened, 377
 Pearson's Correlation Coefficient, 44
 PLA, 105, 159, 179
 Platt's Model of Probabilistic SVM for Soft Binary Classification, 242

- Pocket Alg., 108
- Point Estimator, 56
- Poisson, 49
- pooling, 360
- posterior probability distribution, 63
- Pre-Training with Autoencoders, 371
- precision / inverse variance, 51
- precision matrix, 52
- Primal Hard-Margin SVM, 225
- Primal SVR, 248
- prior probability distribution, 63
- Probability Density Functions (PDF), 37
- Probability Mass Functions (PMF), 37
- Probability Measure Pr, 34
- projection, 12
- Proof, 12, 14, 15, 17, 243
- pruned decision tree, 267
- QP, 219, 224
- Radial Basis Function (RBF) kernel, 229, 336
- Radial Basis Function, RBF, 324
- Random Forest, 269
- Random Variable, 35
- Random Vector, 36
- rank, 11
- Reconstruction ICA (RICA), 384
- Rectified Linear Neurons, ReLU, Positive Part, 322
- recurrent networks, 353
- Reg. for Classification, 178
- reinforce learning, 334
- replace, 386
- representation, 310
- return, 219, 227, 269
- ridge regression, 192
- Rounding Error, 71
- row rank, 11
- Sample Mean, 57
- Sample Space Ω , 34
- Sample Variance, 58
- sampling distribution, 56
- Self-Information, 46
- self-taught learning, 386, 387
- semi-supervised learning, 387
- Shannon Entropy, 46
- Sigmoid, 321
- Slater's condition, 98
- SMO, 237
- SMO (sequential minimal optimization), 235
- Soft-Margin Dual SVM, 235
- Soft-Margin Primal SVM, 233
- Softmax, 323
- Softmax Error, 325
- softmax regression, 165
- Softplus, 322
- Softplus Function, 69
- span, 12
- sparse representation, 345
- sparsity, 198, 380
- spectral decomposition, 19
- Spectral Norm, 10
- sphering, 376
- Squared Error, 325
- Standard Error, 58
- stationary, 357
- steepest descent, 279
- Stochastic Logistic Regression Alg., 179
- strictly concave, 88
- strictly convex, 88
- strong duality, 98
- Structured Probabilistic Model (Graphical Model), 40
- svd, 379
- SVM, 218, 221
- SVM with QP solver, 219
- symmetrically connected networks, 354
- symmetry breaking, 332
- The Nonlinear Transform Steps, 183
- triangle inequality, 8
- tube regression, 246

unbiased, 57
uncorrelated, 44
undercomplete), 384
Underflow, 71
Undirected Graphical Model, 40
Uniform, 50
Union Bound, 34
Universal Approximation, 340
unmixing matrix, 299
unsupervised feature learning, 386
unsupervised learning, 283

Variance, 43, 57
variance reduction methods, 335

weak duality, 98
weight space symmetry, 350
weight-elimination (scaled L_2) regularizer, 342
whitening, 376
Whitening combined with dimensionality reduction, 377
winner takes all, 326

ZCA whitened, 384
ZCA whitening, 378

Happy Learning!