Deep Learning 读书笔记

黄帅 [1] [2] [3]

March 24, 2018

[1] https://github.com/ShuaiHuang/DeepLearningNotes
[2] http://shuaihuang.github.io/
[3] shuaihuang.sjtu@gmail.com

Contents

1 Introduction **3**

1.1 Who shoule read this book? 4

1.2 Historical Treands in Deep Learning 4

2 Linear Algebra **5**

2.8 Singular Value Decomposition 5

2.9 The Moore-Penrose Pesudoinverse 6

3 Probability and Information Theory **7**

3.9 Common Probability Distributions 7

3.9.1 Bernoulli Distribution 7

3.9.2 Multinoulli Distribution 7

3.9.3 Gaussian Distribution 7

3.9.4 Exponential and Laplace Distribution 7

3.9.5 The Dirac Distribution and Empirical Distribution . . 8

3.9.6 Mixtures of Distributions 8

3.10 Useful Properties of Common Functions 8

3.13 Information Theory 8

4 Numerical Computation **11**

4.1 Overflow and Unferflow 11

4.2 Poor Condition 11

4.3 Gradient-Based Optimization 11

4.3.1 Beyond the Gradient: Jacobian and Hessian Matrics . 12

4.4 Constrained Optimization 13

5 Machine Learning Basics **15**

5.1 Learning Algorithms 15
5.1.1 The Task, T 15
5.1.2 The Performance Measure, P 16
5.1.3 The Experience, E 16
5.2 Capacity, Overfitting and Underfitting 17
5.2.1 The No Free Lunch Theorem 18
5.2.2 Regularization 19
5.3 Hyperparameters and Validation Sets 19
5.3.1 Cross-Validation 19
5.4 Estimators, Bias, and Variance 19
5.4.1 Point Estimation 20
5.4.2 Bias 20
5.4.3 Variance and Standard Error 20
5.4.4 Trading off Bias and Variance to Minimize Mean Squared Error 21
5.4.5 Consistency 21
5.5 Maximum Likelihood Estimation 21
5.5.1 Conditional Log-Likelihood and Mean Squared Error . 22
5.5.2 Properties of Maximum Likelihood 22
5.6 Bayesian Statistics 23
5.6.1 Maximum A *Posteriori* (MAP) Estimation 23
5.7 Supervised Learning Algorithms 23
5.7.1 Probabilistic Supervised Learning 24
5.7.2 Support Vector Machine 24
5.7.3 Other Simple Supervised Learning Algorithms 24
5.8 Unsupervised Learning Algorithms 25
5.8.1 Principal Component Analysis 25
5.8.2 k-means Clustering 25
5.9 Stochastic Gradient Descent 26
5.10 Building a Machine Learning Algorithm 26

5.11 Challenges Motivating Deep Learning 26
5.11.1 The Curse of Dimensionality 27
5.11.2 Local Constancy and Smoothness Regularization . . . 27
5.11.3 Manifold Learning 27

6 Deep Feedforward Networks 29
6.1 Example: Learning XOR 29
6.2 Gradient-Based Learning 30
6.2.1 Cost Function 30
6.2.2 Output Units 31
6.3 Hidden Units 33
6.3.1 Rectified Linear Units for Their Generalizations 33
6.3.2 Logistic Sigmoid and Hyperbolic Tangent 34
6.3.3 Other Hidden Units 35
6.4 Architecture Design 35
6.4.1 Universal Approximation Properties and Depth 35
6.4.2 Other Architectural Considerations 36
6.5 Back-Propagation and Other Differentiation Algorithms . . . 36
6.5.1 Computational Graphs 37
6.5.2 Chain Rule of Calculus 37
6.5.3 Recursively Applying the Chain Rule to Obtain Back-prop 37
6.5.4 Back-Propagation Computation in Fully-Connected MLP 37
6.5.5 Symbol-to-Symbol Derivatives 37
6.5.6 General Back-Propagation 38
6.5.7 Example: Back-Propagation for MLP Training 38
6.5.8 Complications 38
6.5.9 Differentiation outside the Deep Learning Community 39
6.5.10 Higher-Order Derivatives 39
6.6 Historical Notes 39

7 Regularization for Deep Learning **41**

7.1 Parameter Norm Penalties 41
7.1.1 L^2 Parameter Regularization 42
7.1.2 L^1 Regularization 42
7.2 Norm Penalties as Constrained Optimization 43
7.3 Regularization and Under-Constrained Problems 44
7.4 Dataset Augmentation 44
7.5 Noise Robustness 45
7.5.1 Injection Noise at the Output Targets 45
7.6 Semi-Supervised Learning 45
7.7 Multi-Task Learning 45
7.8 Early Stopping 46
7.9 Parameter Tying and Parameter Sharing 46
7.10 Sparse Representation 47
7.11 Bagging and Other Ensemble Methods 47
7.12 Dropout 48
7.13 Adversarial Training 50
7.14 Tangent Distance, Tangent Prop, and Manifold Tangent Classifier 50

8 Optimization for Training Deep Models **53**

8.1 How Learning Differs from Pure Optimization 53
8.1.1 Empirical Risk Minimization 54
8.1.2 Surrogate Loss Functions and Early Stopping 54
8.1.3 Batch and Minibatch Algorithms 54
8.2 Challenges and Neural Network Optimization 55
8.2.1 Ill-Conditioning 55
8.2.2 Local Minima 55
8.2.3 Plateaus, Saddle Points and Other Flat Regions 56
8.2.4 Cliffs and Exploding Gradients 56
8.2.5 Long-Term Dependencies 57
8.2.6 Inexact Gradients 57

8.2.7 Poor Correspondence between Local and Global Structure . . . 57
8.2.8 Theoretical Limits of Optimization . . . 57
8.3 Basic Algorithms . . . 58
8.3.1 Stochastic Gradient Descent . . . 58
8.3.2 Momentum . . . 58
8.3.3 Nesterov Momentum . . . 59
8.4 Parameter Initialization Strategies . . . 60
8.5 Algorithms with Adaptive Learning Rates . . . 62
8.5.1 AdaGrad . . . 62
8.5.2 RMSProp . . . 62
8.5.3 Adam . . . 62
8.5.4 Choosing the Right Optimization Algorithm . . . 62
8.6 Approximate Second-Order Methods . . . 63
8.6.1 Newton's Method . . . 63
8.6.2 Conjugate Gradients . . . 63
8.6.3 BFGS . . . 64
8.7 Optimization Strategies and Meta-Algorithms . . . 64
8.7.1 Batch Normalization . . . 64
8.7.2 Coordinate Descent . . . 65
8.7.3 Polyak Averaging . . . 65
8.7.4 Supervised Pretraining . . . 65
8.7.5 Designing Models to Aid Optimization . . . 66
8.7.6 Continuation Methods and Curriculum Learning . . . 66

9 Convolutional Networks 69
9.1 The Convolution Operation . . . 69
9.2 Motivation . . . 70
9.3 Pooling . . . 70
9.4 Convolution and Pooling as Infinitely Strong Prior . . . 71
9.5 Variants of the Basic Convolution Function . . . 72
9.6 Structured Outputs . . . 73

9.7 Data Types 73
9.8 Efficient Convolution Algorithms 74
9.9 Random or Unsupervised Feature 74
9.10 The Neuroscientific Basis for Convolutional Networks 75
9.11 Convolutional Networks and the History of Deep Learning . . 76

10 Sequence Modeling: Recurrent and Recursive Nets 77
10.1 Unfolding Computational Graphs 77
10.2 Recurrent Neural Networks 78
10.2.1 Teacher Forcing and Networks with Output Recurrence 78
10.2.2 Computing the Gradient in a Recurrent Neural Network 79
10.2.3 Recurrent Networks as Directed Graphical Models . . 79
10.2.4 Modeling Sequences Conditioned on Context with RNNs 79
10.3 Bidirectional RNNs 80
10.4 Encoder-Decoder Sequence-to-Sequence Architectures 80
10.5 Deep Recurrent Networks 80
10.6 Recursive Neural Networks 81
10.7 The Challenge of Long-Term Dependencies 81
10.8 Echo State Networks 81
10.8.1 Adding Skip Connections through Time 82
10.8.2 Leaky Units and a Specturm of Different Time Scales . 82
10.8.3 Removing Connections 82
10.9 The Long Short-Term Memory and Other Gated RNNs 83
10.9.1 LSTM 83
10.9.2 Other Gated RNNs 84
10.10 Optimization for Long-Term Dependencies 84
10.10.1 Clipping Gradients 84
10.10.2 Regularizing to Encourage Information Flow 84
10.11 Explicit Memory 85

11 Practical Methodology 87
11.1 Performance Metrics 87

11.2 Default Baseline Models 87
11.3 Determining Whether to Gather More Data 88
11.4 Selecting Hyperparamters 88
11.4.1 Manual Hyperparameter Tuning 88
11.4.2 Automatic Hyperparameter Optimization Algorithm . 89
11.4.3 Grid Search 89
11.4.4 Random Search 89
11.4.5 Model-Based Hyperparameter Optimization 89
11.5 Debugging Strategies 90
11.6 Example: Multi-Digit Number Recognition 91

12 Applications **93**
12.1 Large-Scale Deep Learning 93
12.1.1 Fast CPU Implementations 93
12.1.2 GPU Implementations 93
12.1.3 Large-Scale Distributed Implementations 93
12.1.4 Model Compression 94
12.1.5 Dynamic Structure 94
12.1.6 Specialized Hardware Implementations of Deep Networks 95
12.2 Computer Vision 95
12.2.1 Preprocessing 95
12.3 Speech Recognition 96
12.4 Natural Language Processing 97
12.4.1 n-grams 97
12.4.2 Neural Language Models 98
12.4.3 High-Dimensional Outputs 98
12.4.4 Combining Neural Language Models with n-grams . . 99
12.4.5 Neural Machine Translation 99
12.4.6 Historical Perspective 100
12.5 Other Applications 100
12.5.1 Recommender Systems 100

12.5.2 Knowledge Representation, Reasoning and Question Answering . . . 101

13 Linear Factor Models 103

13.1 Probabilistic PCA and Factor Analysis . . . 103
13.2 Independent Component Analysis (ICA) . . . 104
13.3 Slow Feature Analysis . . . 105
13.4 Sparse Coding . . . 105
13.5 Manifold Interpretation of PCA . . . 106

14 Autoencoders 107

14.1 Undercomplete Autoencoders . . . 107
14.2 Regularized Autoencoders . . . 107
14.2.1 Denoising Autoencoders . . . 108
14.2.2 Regularizing by Penalizing Derivatives . . . 108
14.3 Representation Power, Layer Size and Depth . . . 108
14.4 Stochastic Encoders and Decoders . . . 109
14.5 Denoising Autoencoders . . . 109
14.5.1 Estimating the Score . . . 109
14.6 Learning Manifolds with Autoencoders . . . 110
14.7 Contractive Autoencoders . . . 110
14.8 Predictive Sparse Decomposition . . . 111
14.9 Application of Autoencoders . . . 111

15 Representation Learning 113

15.1 Greedy Layer-Wise Unsupervised Pretraining . . . 113
15.1.1 When and Why Does Unsupervised Pretraining Work? 114
15.2 Transfer Learning and Domain Adaption . . . 115
15.3 Semi-Supervised Distangling of Causal Factors . . . 115
15.4 Distributed Representation . . . 116
15.5 Exponential Gains from Depth . . . 117
15.6 Providing Clues to Discover Underlying Causes . . . 117

16 Structured Probabilistic Models for Deep Learning 119

16.1 The Challenge of Unstructured Modeling 119

16.2 Using Graphs to Describe Model Structure 120

16.2.1 Directed Models 120

16.2.2 Undirected Models 120

16.2.3 The Partition Function 120

16.2.4 Energy-Based Models 121

16.2.5 Separation and D-Separation 121

16.2.6 Converting between Undirected and Directed Graph . 121

16.2.7 Factor Graph 122

16.3 Sampling from Graphical Models 122

16.4 Advantages of Structured Modeling 122

16.5 Learning about Dependencies 123

16.6 Inference and Approximate Inference 123

16.7 The Deep Learning Approach to Structured Probabilistic Models 123

16.7.1 Example: The Restricted Boltzmann Machine 124

17 Monte Carlo Methods 125

17.1 Sampling and Monte Carlo Methods 125

17.1.1 Why Sampling? 125

17.1.2 Basics of Monte Carlo Sampling 125

17.2 Importance Sampling 126

17.3 Markov Chain Monte Carlo Methods 126

17.4 Gibbs Sampling 127

17.5 The Challenge of Mixing between Separated Modes 127

17.5.1 Tempering to Mix between Modes 128

17.5.2 Depth May Help Mixing 128

18 Confronting the Partition Function 129

18.1 The Log-Likelihood Gradient 129

18.2 Stochastic Maximum Likelihood and Contrastive Divergence . 130

18.3 Pseudolikelihood . . . 131
18.4 Score Matching and Ratio Matching . . . 132
18.5 Denoising Score Matching . . . 132
18.6 Noise-Contrastive Estimation . . . 133
18.7 Estimating the Partition Function . . . 133
18.7.1 Annealed Importance Sampling . . . 134
18.7.2 Bridge Sampling . . . 134

19 Approximate Inference **137**
19.1 Inference as Optimization . . . 137
19.2 Expectation Maximization . . . 137
19.3 MAP Inference and Sparse Coding . . . 138
19.4 Variational Inference and Learning . . . 138
19.4.1 Discrete Latent Variables . . . 139
19.4.2 Calculus of Variations . . . 139
19.4.3 Continuous Latent Variables . . . 140
19.4.4 Interactions between Learning and Inference . . . 140
19.5 Learned Approximate Inference . . . 140
19.5.1 Wake-Sleep . . . 140
19.5.2 Other Forms of Learned Inference . . . 141

20 Deep Generative Models **143**
20.1 Boltzmann Machines . . . 143
20.2 Restricted Boltzmann Machines . . . 143
20.2.1 Conditional Distributions . . . 144
20.2.2 Training Restricted Boltzmann Machines . . . 144
20.3 Deep Belief Networks . . . 144
20.4 Deep Boltzmann Machines . . . 145
20.4.1 Interesting Properties . . . 146
20.4.2 DBM Mean Field Inference . . . 146
20.4.3 DBM Parameter Learning . . . 146
20.4.4 Layer-Wise Pretraining . . . 147

20.4.5 Jointly Training Deep Boltzmann Machines 147
20.5 Boltzmann Machines for Real-Valued Data 148
20.5.1 Gaussian-Bernoulli RBMs 148
20.5.2 Undirected Models of Conditional Covariance 148
20.6 Convolutional Boltzmann Machines 149
20.7 Boltzmann Machine for Structured or Sequential Outputs . . 150
20.8 Other Boltzmann Machines 150
20.9 Back-Propagation through Random Operation 150
20.9.1 Back-Propagating through Discrete Stochastic Operations 150
20.10 Directed Generative Nets 151
20.10.1 Sigmoid Belief Nets 151
20.10.2 Differentiable Generator Nets 151
20.10.3 Variational Autoencoders 152
20.10.4 Generative Adversarial Networks 152
20.10.5 Generative Moment Matching Networks 153
20.10.6 Convolutional Generative Networks 153
20.10.7 Auto-Regressive Networks 153
20.10.8 Linear Auto-Regressive Networks 154
20.10.9 Neural Auto-Regressive Networks 154
20.10.10 NADE 154
20.11 Drawing Samples from Autoencoders 154
20.11.1 Markov Chain Associated with any Denoising Autoencoder 154
20.11.2 Clamping and Conditional Sampling 154
20.11.3 Walk-Back Training Procedure 155
20.12 Generative Stochastic Networks 155
20.12.1 Discriminant GSNs 155
20.13 Other Generation Schemes 155
20.14 Evaluating Generative Models 155
20.15 Conclusion 156

Chapter 1 Introduction

本章从 AI 发展的角度阐述了深度学习是什么，从哪里来，可以用来解决什么问题。

自人工智能发展之初的符号逻辑时代起，人们就借助于规则去解决一些常规方法所不能解决的问题。虽然符号逻辑可以解决一些简单的问题，但是对于一些复杂的情况，如何用符号逻辑去表示这些情况，却比解决问题本身更加困难。之后发展到机器学习时代，人们直接从原始数据中提取出特征，用特征训练模型来解决问题。这就相当于从符号逻辑时代向前更进了一步。但是如何从原始数据中选择合适的特征是一个充满经验性和技巧性的环节。从这一点出发，就发展出了表示学习（repersentation learning)。表示学习通过学习的手段从原始数据中提取出特征，而非传统的手工选取以及手工加工组合特征的方式。深度学习就是表示学习的一种。

深度学习中的深度是指模型的深度较大，具体可以从两个方面进行阐述：

- 模型的算法流程执行环节较多
- 描述各个概念关联的图的深度较大

但是并没有一个确切的指标表示达到什么样的标准才叫做深度模型。所以深度模型可以泛指包含大量通过学习而得的函数或者通过学习而得的概念的模型，这里的大量是与传统的机器学习中的函数或者概念相对而言的。

基于以上，机器学习就是：

- 向 AI 方向更近一步的方式
- 一种机器学习的方法

1.1 Who shoule read this book?

本书的结构为

- **Part I** 一些基本概念
- **Part II** 一些基本的深度学习算法，以及相应的训练方法
- **Part III** 一些前瞻性的想法，并且这些想法极有可能对未来深度学习的发展起到推动性的作用。

由于时间有限，本书的重点将放在 Part II，其他的部分快速带过。

1.2 Historical Treands in Deep Learning

本节主要介绍了深度学习的发展历史。

1. 深度学习有着长久的历史；
2. 深度学习近年来有广泛的应用场景是因为可供使用的学习数据增多了；
3. 深度学习模型的复杂度也在不断增加；
4. 深度学习方法解决的问题复杂度也在不断增加，同时准确率也在不断增加。

Chapter 2 Linear Algebra

本书的第二章内容主要介绍了深度学习相关的线性代数和矩阵理论基础知识，其中大部分知识比较简单，可以快速带过，这里重点记录一下自己不太熟悉的奇异值分解（Singular Value Decomposition）以及线性代数知识在主成分分析（Principal Components Analysis）推导过程中的实际应用[1]。

2.8 Singular Value Decomposition

一般来说，一个方形矩阵可以分解为特征值与特征向量相乘的形式。特征值分解从提供了将矩阵分解为相乘形式的另一种角度。

根据特征值的性质，有

$$\boldsymbol{A} = \boldsymbol{V}\text{diag}(\boldsymbol{\lambda})\boldsymbol{V}^{-1} \tag{2.1}$$

其中 $\boldsymbol{V}$ 是 $\boldsymbol{A}$ 所有特征向量组成的矩阵；$\text{diag}(\boldsymbol{\lambda})$ 是 $\boldsymbol{V}$ 中与特征向对应的特征值组成的对角矩阵。

同样的，有奇异值分解

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^{T} \tag{2.2}$$

其中 $\boldsymbol{A}$ 是一个 $m \times n$ 矩阵；$\boldsymbol{U}$ 是一个 $m \times m$ 矩阵；$\boldsymbol{V}$ 是一个 $n \times n$ 矩阵；$\boldsymbol{D}$ 是一个 $m \times n$ 矩阵。$\boldsymbol{U}$ 和 $\boldsymbol{V}$ 是正交矩阵，$\boldsymbol{D}$ 是对角矩阵。$\boldsymbol{D}$ 对角线上元素被称为奇异值；$\boldsymbol{U}$ 中的每一列被称为左奇异值向量；$\boldsymbol{V}$ 中的每一列被称为右奇异值向量。

$\boldsymbol{A}$ 的左奇异值向量是 $\boldsymbol{A}\boldsymbol{A}^{T}$ 对应的特征向量；$\boldsymbol{A}$ 的右奇异值向量是 $\boldsymbol{A}^{T}\boldsymbol{A}$ 对应的特征向量；非零奇异值是 $\boldsymbol{A}\boldsymbol{A}^{T}$ 或 $\boldsymbol{A}^{T}\boldsymbol{A}$ 特征值的平方根。

[1]前面 7 节内容比较简单，这里略去

2.9 The Moore-Penrose Pesudoinverse

$\boldsymbol{A}$ 的 *Moore-Penrose* 广义逆被定义为

$$\boldsymbol{A}^{+} = \lim_{\alpha \to 0} (\boldsymbol{A}^{T}\boldsymbol{A} + \alpha \boldsymbol{I})^{-1}\boldsymbol{A}^{T} \tag{2.3}$$

但是通常情况下采用如下的方法进行计算

$$\boldsymbol{A}^{+} = \boldsymbol{V}\boldsymbol{D}^{+}\boldsymbol{U}^{T} \tag{2.4}$$

其中，$\boldsymbol{U}$，$\boldsymbol{D}$，$\boldsymbol{V}$ 的含义同式2.2中的定义。$\boldsymbol{D}$ 的广义逆 $\boldsymbol{D}^{+}$ 求取方法为对 $\boldsymbol{D}$ 中非零元素取倒数再转置。

使用广义逆求解 $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}$ 即 $\boldsymbol{x} = \boldsymbol{A}^{+}\boldsymbol{y}$。如果 $\boldsymbol{A}$ 的列数大于行数，则 $\|\boldsymbol{x}\|_2$ 在所有可能解中最小；如果 $\boldsymbol{A}$ 的行数大于列数，则 $\boldsymbol{A}\boldsymbol{x}$ 非常近似于 $\boldsymbol{y}$ 即 $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2$ 最小。

Chapter 3 Probability and Information Theory

3.9 Common Probability Distributions

3.9.1 Bernoulli Distribution

$$\begin{aligned} P(x=1) &= \phi \\ P(x=0) &= 1-\phi \end{aligned} \tag{3.1}$$

3.9.2 Multinoulli Distribution

Bernoulli 分布的扩展，假设离散变量有 k 种不同状态。Bernoulli 分布和 Multinoulli 分布在各自的定义域内可以描述任何分布。

3.9.3 Gaussian Distribution

$$\mathcal{N}(\boldsymbol{x};\boldsymbol{\mu},\boldsymbol{\beta^{-1}}) = \sqrt{\frac{\det(\boldsymbol{\beta})}{(2\pi)^n}}\exp\Big(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\beta}(\boldsymbol{x}-\boldsymbol{\mu})\Big) \tag{3.2}$$

其中 $\boldsymbol{\beta}$ 是精度矩阵，即协方差矩阵的逆矩阵。

3.9.4 Exponential and Laplace Distribution

指数分布：

$$p(x;\lambda) = \lambda\mathbf{1}_{x\geq 0}\exp(-\lambda x) \tag{3.3}$$

其中 $\mathbf{1}_{x\geq 0}$ 是指示函数。

拉普拉斯分布

$$\text{Laplace}(x;\mu,\gamma) = \frac{1}{2\gamma}\exp\Big(-\frac{|x-\mu|}{\gamma}\Big) \tag{3.4}$$

3.9.5 The Dirac Distribution and Empirical Distribution

狄利克雷分布:

$$p(x) = \delta(x-\mu) \tag{3.5}$$

经验分布:

$$\hat{p}(x) = \frac{1}{m}\sum_{i=1}^{m}\delta(\boldsymbol{x}-\boldsymbol{x^{(i)}}) \tag{3.6}$$

其中，$\boldsymbol{x}^{(i)}$ 是从数据集中采样得到的样本集。

3.9.6 Mixtures of Distributions

$$P(x) = \sum_i P(c=i)P(x|c=i) \tag{3.7}$$

3.10 Useful Properties of Common Functions

logistic sigmoid:

$$\sigma(x) = \frac{1}{1+\exp(-x)} \tag{3.8}$$

softplus:

$$\zeta(x) = \log(1+\exp(x)) \tag{3.9}$$

3.13 Information Theory

事件 $\mathbf{x}=\boldsymbol{x}$ 的自信息定义为

$$I(\boldsymbol{x}) = -\log P(\boldsymbol{x}) \tag{3.10}$$

我们通常使用 *Shannon entropy* 量化**整个概率分布的不确定性**

$$H(\mathbf{x}) = \mathbb{E}_{\mathbf{x}\sim P}[I(x)] = \mathbb{E}_{\mathbf{x}\sim P}[\log P(x)] \tag{3.11}$$

也就是说，一个概率分布的 Shannon 熵也就是基于该分布的事件所包含信息的期望值。

使用 *Kullback-Leibler divergence* 度量两个分布 $P(\mathbf{x})$ 和 $Q(\mathbf{x})$ 的差异程度

$$D_{\mathrm{KL}}(P\|Q) = \mathbb{E}_{\mathbf{x}\sim P}\left[\log \frac{P(x)}{Q(x)}\right] = \mathbb{E}_{\mathbf{x}\sim P}[\log P(x) - \log Q(x)] \tag{3.12}$$

需要注意的是 $D_{\mathrm{KL}}(P\|Q) \neq D_{\mathrm{KL}}(Q\|P)$.

与 *Kullback-Leibler divergence* 类似的还有 *cross-entropy*

$$H(P, Q) = H(P) + D_{\mathrm{KL}}(P\|Q) = -\mathbb{E}_{\mathbf{x}\sim P} \log Q(x) \tag{3.13}$$

通常情况下，规定 $\lim_{x\to 0} x \log x = 0$.

Chapter 4 Numerical Computation

机器学习算法中涉及到大量的计算过程. 通常情况下, 机器学习算法没有解析解, 只有通过迭代优化过程得到次优解. 本章重点介绍机器学习的数值计算过程相关的知识点.

4.1 Overflow and Unferflow

数值上溢 (Overflow) 和下溢 (Underflow) 是两种不同的取整错误 (rounding error). 其中, 下溢发生在数值接近于 0 的时候. 有些函数对于接近于 0 和 0 的数字非常敏感. 上溢发生在数值的绝对值非常大的时候, 会被近似认为是 ∞ 或者 $-\infty$. 在实际过程中, 涉及到机器学习底层算法库开发时, 必须考虑上溢和下溢的问题, 并且在设计优化方法时进行规避.

4.2 Poor Condition

Conditioning 是指一个函数的输入发生微小变化时, 其对应的输出变化程度, 即对输入变化的敏感程度.Poor Condition 会放大计算误差.

4.3 Gradient-Based Optimization

大多数深度学习算法都会涉及到某些种类的优化过程. 我们通常沿着目标函数的梯度下降方向搜索全局最小值但是如果参数选取不当的话, 会收敛到局部极小值或者鞍点上.

4.3.1 Beyond the Gradient: Jacobian and Hessian Matrics

对于输入和输出都是向量的函数, 如果要计算其偏导数, 就需要用 **Jacobian 矩阵**进行表示. 设 $f : \mathbb{R}^m \to \mathbb{R}^n$, 则有 Jacboian 矩阵 $\boldsymbol{J} \in \mathbb{R}^{m\times n}$. 其中,

$$J_{i,j} = \frac{\partial}{\partial x_j} f(\boldsymbol{x})_i \tag{4.1}$$

如果求二次偏导, 则有 **Hessian 矩阵**, 即

$$\boldsymbol{H}(f)(\boldsymbol{x})_{i,j} = \frac{\partial^2}{\partial x_i \partial x_j} f(\boldsymbol{x}) \tag{4.2}$$

如果想计算某一个方向上的二阶偏导, 则有 $\boldsymbol{d}^T\boldsymbol{H}\boldsymbol{d}$ 其中,$\boldsymbol{d}$ 是一单位向量. 特别的, 如果 $\boldsymbol{d}$ 是 $\boldsymbol{H}$ 的特征向量, 则 $\boldsymbol{d}$ 方向上的二阶偏导就是 $\boldsymbol{H}$ 对应于 $\boldsymbol{d}$ 的特征值. 在任何方向上的二阶偏导都在最大特征值和最小特征值的范围之间.

二阶导数可以告诉我们梯度下降的程度. 对 $f(\boldsymbol{x})$ 做二阶 Taylor 展开, 有

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}^{(0)}) + (\boldsymbol{x} - \boldsymbol{x}^{(0)})^T\boldsymbol{g} + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^{(0)})^T\boldsymbol{H}(\boldsymbol{x} - \boldsymbol{x}^{(0)}) \tag{4.3}$$

其中,$\boldsymbol{g}$ 和 $\boldsymbol{H}$ 分别是 $f(\boldsymbol{x})$ 在 $\boldsymbol{x}^{(0)}$ 点的梯度和 Hessian 矩阵. 设学习率为 ϵ, 则新到达的点为 $\boldsymbol{x}^{(0)} - \epsilon\boldsymbol{g}$, 在该点上取值为

$$f(\boldsymbol{x}^{(0)} - \epsilon\boldsymbol{g}) \approx f(\boldsymbol{x}^{(0)}) - \epsilon\boldsymbol{g}^T\boldsymbol{g} + \frac{1}{2}\epsilon^2\boldsymbol{g}^T\boldsymbol{H}\boldsymbol{g} \tag{4.4}$$

上式中最后一项不能太大, 否则新的数值将会比原值还要大. 当 $\boldsymbol{g}^T\boldsymbol{H}\boldsymbol{g}$ 等于或者小于 0 时,ϵ 可以取到很大的值; 当 $\boldsymbol{g}^T\boldsymbol{H}\boldsymbol{g}$ 大于 0 时, 有

$$\epsilon^* = \frac{\boldsymbol{g}^T\boldsymbol{g}}{\boldsymbol{g}^T\boldsymbol{H}\boldsymbol{g}} \tag{4.5}$$

优化算法可以按照一阶偏导和二阶偏导进行分类. 只用到梯度的优化算法称为一阶优化算法, 涉及到 Hessian 矩阵的优化算法称为二阶优化算法. 深度学习的情况复杂, 一般的优化算法无法保证结果, 因此必须做出一些限制. 通常用的限制为 **Lipschitz continuous**, 即

$$\forall\boldsymbol{x}, \forall\boldsymbol{y}, |f(\boldsymbol{x}) - f(\boldsymbol{y})| \leq \mathcal{L}\|\boldsymbol{x} - \boldsymbol{y}\|_2 \tag{4.6}$$

其中,$\mathcal{L}$ 是 **Lipschitz 常数**.

凸优化由于有了更多的限制, 因此其优化结果可以保证收敛. 因此机器学习中在一些特定条件下使用凸优化算法进行优化.

4.4 Constrained Optimization

在机器学习中通常遇到的优化问题是有限制条件的优化问题. 为了解决优化问题, 一个简单的方法是在梯度下降时将限制条件考虑进去. 另外一种更加成熟的方案就是将限制条件考虑进问题中, 构造一个与原问题同解的无限制的优化问题.

Karush-Kuhn-Tucker 就提供了一种有限制优化问题的解决方法. 有 Lagrangian

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) = f(\boldsymbol{x}) + \sum_i \lambda_i g^{(i)}(\boldsymbol{x}) + \sum_j \alpha_j h^{(j)}(\boldsymbol{x}) \tag{4.7}$$

其中,$g^{(i)}$ 和 $h^{(j)}$ 分别是等式限制和不等式限制. 则原问题与

$$\min_{\boldsymbol{x}} \max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq 0} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) \tag{4.8}$$

同解.

Chapter 5 Machine Learning Basics

5.1 Learning Algorithms

Mitchell 将机器学习定义为

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

本节分别从上述定义中的 T,P,E 三个角度对机器学习的相关知识点进行介绍.

5.1.1 The Task, T

机器学习所关注的任务是人类很难通过具体规则进行编程实现进行解决的任务. 学习过程本身不是任务, 它只是我们赋予计算机解决问题能力的过程. 可以通过机器学习得到解决的任务有

- 分类问题
- 有缺失值的分类问题
- 回归问题
- 转译 (Transcription): 输入非结构化表示的数据, 输出离散化文字化的数据.
- 机器翻译
- 结构化输出 (Structured Output)

- 异常检测
- 合成与采样
- 缺失值处理
- 去噪
- 概率密度估计或概率分布函数估计

5.1.2 The Performance Measure, P

为了可以量化地衡量机器学习算法的表现, 通常根据不同的任务类型设计相应的性能度量方式. 最常见的是度量模型的准确率 (accuracy). 与此等价的度量指标还有错误率 (error rate) 和0-1 损失期望 (expected 0-1 loss). 对于概率密度估计任务, 通常使用在某些样本上的对数概率均值去衡量.

机器学习算法的表现衡量的是算法泛化能力, 通常是通过在测试集上的表现去近似估计. 选择机器学习算法的度量方法很有技巧性, 度量系统的哪一个方面很难决定. 即使知道了度量系统的哪些因素, 如何量化这些因素也是一大挑战.

5.1.3 The Experience, E

根据学习过程中获取到的经验种类, 机器学习可以分为有监督学习和无监督学习两大类. 并且机器学习所获取到的经验是从整个数据集上获取到的. 从数学模型上来看, 无监督学习所学到的模型是样本的分布 $p(\boldsymbol{x})$, 有监督学习所学到的模型是根据样本 $\boldsymbol{x}$ 求其对应样本标记 $\boldsymbol{y}$ 的分布 $p(\boldsymbol{y}|\boldsymbol{x})$.

有监督学习和无监督学习并没有严格的区分界限. 假设有向量 $\boldsymbol{x} \in \mathbb{R}^n$, 根据概率链式法则

$$p(\boldsymbol{x}) = \prod_{i=1}^{n} p(x_i|x_1, \cdots, x_{i-1}) \tag{5.1}$$

则可以得出, 无监督学习可以分解为有监督学习的子问题. 同样地, 我们可以使用无监督学习的方法学得联合概率分布 $p(\boldsymbol{x}, \boldsymbol{y})$, 进而求得 $p(\boldsymbol{y}|\boldsymbol{x})$.

$$p(\boldsymbol{y}|\boldsymbol{x}) = \frac{p(\boldsymbol{x}, \boldsymbol{y})}{\sum_{\boldsymbol{y}'} p(\boldsymbol{x}, \boldsymbol{y}')} \tag{5.2}$$

除了有监督学习和无监督学习, 还有诸如*多样例学习* (multi-instance learning) 和*强化学习* (reinforcement learning) 等不同形式的学习算法.

绝大部分机器学习算法是从数据集中获取到经验的. 通常使用*设计矩阵* (design matrix) 对数据集进行表示, 其中设计矩阵的每一行对应一个样本, 每一列对应一个特征值. 这就要求每一个样本所对应的特征维度必须相同.[1]对于有监督学习, 样本还需要包含对应的标签. 需要注意的是, 标签可能不只使用一个数字进行表示.

5.2 Capacity, Overfitting and Underfitting

机器学习算法最终是在未知样本上执行的, 算法在未知样本上也能够取得良好表现效果的能力被称为*泛化能力* (generalization). 通常我们通过计算机器学习算法在测试集上的错误率来评估其泛化能力. **但是如何能够仅仅通过观察算法在训练集上的表现来估计其在训练集上的表现呢?**

通常可以假设训练集和测试集是由同一个数据分布生成的, 即训练集和测试集是独立同分布, 这被称为*独立同分布假设* (i.i.d. assumptions). 假设有一个随机的分布模型, 由此分布模型进行数据采样, 分别得到训练集合测试集, 则期望训练误差和期望测试误差相等. 但是在实际中, 模型的参数未定, 所以测试误差大于或等于训练误差. 因此, 决定机器学习算法泛化能力的因素有:

1. 训练误差足够小;
2. 训练误差和测试误差之间的差异足够小.

上述两点分别对应机器学习的两大挑战: 欠拟合 (underfitting) 和过拟合 (overfitting).

通过对机器学习算法的容量 (capacity)[2]进行调整, 可以对机器学习算法的过拟合或欠拟合情况进行调整. 调整容量的方法有很多, 其中一种是选择机器学习算法的*假设空间* (hypothesis space), 即机器学习算法允许选择的函数集合. 在容量和实际任务以及训练集规模匹配的情况下, 机器学

[1]9.7 和 10 涉及到异构特征的组织方式.

[2]The ability to fit a wide variety of functions.

习算法会取得最好的效果. 但是这是一个依赖经验和技巧的选择过程. 除此之外, 给定模型函数, 通过改变函数参数达到训练目标的过程被称为模型表示容量 (representational capacity), 通过对模型添加限制形成模型有效容量 (effective capacity), 有效能量将会比表示容量小得多.

提升机器学习模型泛化能力的理论是一个逐步完善的过程. 今天的统计学习理论是对奥卡姆剃刀 (Occam's razor) 的进一步完善. **统计学习理论提供了量化模型容量的多种方法.** 其中最有名的是 *VC* 维 (Vapnik-Chervonenkis dimension)[3]. 通过量化模型的容量, 统计学习理论可以量化出训练误差和泛化误差差值的上限. 但是在实际中, 很少通过这种方式对训练误差和泛化误差的差值进行评估. 从整体上看, 尽管简单模型有较好的泛化能力, 但是还是需要选择足够复杂的模型去降低测试集上的误差[4].

产生数据集的最本质分布和通过数据集观察到的真实分布 $P(\boldsymbol{x}, y)$ 之间的误差被称为贝叶斯误差 (Bayes error)[5]对于无参模型, 可以通过增加样本集规模使得泛化能力提升并使错误率趋近贝叶斯误差. 但是对于通常的有参数模型, 通常可以达到的错误率下限是一个比贝叶斯错误率更高的下限.

5.2.1 The No Free Lunch Theorem

机器学习算法可以从有限规模训练集中得到泛化能力较好的模型, 从统计学习理论的角度看来是很违背常理的. 机器学习算法仅仅给出一个能够使得我们所关注的集合中绝大多数样本近似正确的规则.

机器学习中的没有免费午餐定义 (no free lunch theorem) 说明了, 对于所有可能的**数据生成分布**做同等重要性考虑, 任何一种分类算法在未被观测到的新样本上都有同样的错误率. 在实际中, 我们通常只针对特定的数据生成分布感兴趣, 并且机器学习的目的不是对所有任务寻找通解.

---

[3]被假设空间打散的最大示例集大小.

[4]书中用无参模型的极端例子说明复杂度较高的模型可以有较低的训练误差.

[5]开个脑洞: 真理与认识之间的差异.

5.2.2 Regularization

通常的机器学习算法中都包含了归纳偏好. 机器学习算法所需要关注的不仅仅只是其表示容量, 函数的可选种类也非常重要. 在假设空间中, 机器学习算法被赋予一个偏好, 使其对特定的函数有所侧重.

偏好还有的目的之一是为了控制机器学算法的过拟合和欠拟合的程度. 一般情况下, 偏好以正则化项的形式进行表示. 偏好还有一个目的是控制模型假设空间的容量. **正则化是表达机器学习算法偏好的所有方法统称**.

NFL 定理从本质上表明没有普适的正则化标准, 需要根据不同的任务制定不同的正则化方法.

5.3 Hyperparameters and Validation Sets

机器学习算法通过超参数 (hyperparameter) 控制学习算法的行为, 超参数一般是手动调整的. 超参数一般是难以通过学习算法进行优化的参数, 同时也是难以通过训练集学习得到的参数. 需要靠人工经验设定或者网格搜索.

超参数一般通过验证集 (validation set) 进行选取. 需要进行区分的是, 测试集用来估计泛化误差, 不能用来评价超参数的选取或者进行模型选择.

5.3.1 Cross-Validation

当数据集规模较小时, 将其进行固定分割会使得测试集规模较小, 带来统计误差. 交叉验证可以解决这一问题.

5.4 Estimators, Bias, and Variance

统计学可以帮助机器学习从训练集上学得具有较好泛化能力的模型. 参数估计, 偏差, 方差被用来描述模型的泛化能力, 欠拟合和过拟合.

5.4.1 Point Estimation

点估计是根据训练样本的分布对兴趣点或者向量提供一个 “最好” 的预测结果, 即

$$\hat{\boldsymbol{\theta}}_m = g(\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(m)}) \tag{5.3}$$

这个定义比较宽泛. 一个好的估计可以产生非常接近数据生成的参数 $\boldsymbol{\theta}$. 需要强调的是, 这个是频率主义的观点, 即认为模型的参数是固定且未知的.

点估计还可以用来估计样本和标记之间的关系, 这被称为*函数估计* (Function Estimation). 即假设有 $y = f(\boldsymbol{x}) + \epsilon$, 其中 ϵ 是 y 不能从 $\boldsymbol{x}$ 中预测到的部分. 函数估计是根据模型估计出 f 的估计量 $\hat{f}$.

5.4.2 Bias

偏差的定义为

$$\text{bias}(\hat{\boldsymbol{\theta}}_m) = \mathbb{E}(\hat{\boldsymbol{\theta}}_m) - \boldsymbol{\theta} \tag{5.4}$$

其中 $\boldsymbol{\theta}$ 是产生数据模型的真实参数;$\hat{\boldsymbol{\theta}}$ 是估计量. 如果有 $\text{bias}(\hat{\boldsymbol{\theta}}_m) = 0$, 则称估计是*无偏的* (unbiased). 如果有 $\lim_{m\to\infty} \text{bias}(\hat{\boldsymbol{\theta}}_m) = 0$, 则称估计是*渐进无偏的* (asymptotically unbiased).

通过书中的举例可以看出, 无偏估计符合大多数算法的要求, 但是他们并不是 “最好” 的估计. 我们通常使用有偏估计去满足算法的其他特性.

5.4.3 Variance and Standard Error

方差 (variance) 反映了数据的散布范围, 其平方根被称为*标准差* (standard error). 它们分别被定义为 $\text{Var}(\hat{\boldsymbol{\theta}})$ 和 $\text{SE}(\hat{\boldsymbol{\theta}})$. 同一个分布产生的不同数据集会带来统计量的差异, 这种方差的期望值是一个误差来源, 如果能量化这个误差来源, 对研究机器学习算法也有帮助.

对于标准差, 无论是对样本的方差开根号还是对方差的无偏估计开根号, 其结果都不是无偏估计, 但是它有很重要的用途. 在机器学习中, 标准差常用来计算平均值的置信区间, 并根据置信区间的大小评价算法的好坏.

5.4.4 Trading off Bias and Variance to Minimize Mean Squared Error

偏差和方差描述了两种不同类型的统计量, 在实际的机器学习算法中需要对这两者进行适当的权衡. 常用的两种方式为: 交叉验证和均方误差比较.

$$\begin{aligned} \mathrm{MSE} &= \mathbb{E}[(\hat{\boldsymbol{\theta}}_m - \boldsymbol{\theta})^2] \\ &= \mathrm{Bias}(\hat{\boldsymbol{\theta}}_m)^2 + \mathrm{Var}(\hat{\boldsymbol{\theta}}_m) \end{aligned} \tag{5.5}$$

理想估计的偏差和方差都很小, 直接反映就是均方差小.

偏差和方差与模型的容量有关, 模型容量上升会导致偏差下降方差上升.

5.4.5 Consistency

通常我们比较关注的情形是, 随着训练样本的数量上升, 估计量是否趋近于真实值? 即对于 $\forall \epsilon > 0$, 当 $m \to 0$ 时, 有 $P(|\hat{\boldsymbol{\theta}}_m - \boldsymbol{\theta}| > \epsilon) \to 0$.

一致性保证了样本数量增加的情况下, 估计量偏差会逐渐降低. 但是反过来不一定成立.

5.5 Maximum Likelihood Estimation

通常使用*极大似然估计* (Maximum Likelihood Estimation) 对待估计参数进行估计. 假设 $p_{\mathrm{model}}(\boldsymbol{x}; \boldsymbol{\theta})$ 是由样本估计出的整体分布, 以 $\boldsymbol{\theta}$ 为参数. 其目的就是将 $\boldsymbol{x}$ 映射到真实的分布 $p_{\mathrm{data}}(\boldsymbol{x})$. 极大似然度估计定义为

$$\begin{aligned} \boldsymbol{\theta}_{ML} &= \arg\max_{\boldsymbol{\theta}} p_{\mathrm{model}}(\mathbb{X}; \boldsymbol{\theta}) \\ &= \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p_{\mathrm{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \end{aligned} \tag{5.6}$$

进行适当的尺度变换和归一化后, 可以得到

$$\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x} \sim \hat{p}_{\mathrm{data}}} \log p_{\mathrm{model}}(\boldsymbol{x}; \boldsymbol{\theta}) \tag{5.7}$$

其中,$\hat{p}_{\text{data}}$ 是样本在训练集上的经验分布 (empirical distribution). 定性地看, 极大似然估计是通过选择 $\boldsymbol{\theta}$ 使得 p_{model} 逐步逼近 $\hat{p}_{\text{data}}$. 上述两个分布间差异程度可以通过 KL 散度进行量化:

$$D_{KL} = (\hat{p}_{\text{data}} \| p_{\text{model}}) = \mathbb{E}_{\boldsymbol{x}\sim\hat{p}_{\text{data}}}[\log \hat{p}_{\text{data}}(\boldsymbol{x}) - \log p_{\text{model}}(\boldsymbol{x})] \tag{5.8}$$

令上式最小, 即最小化

$$-\mathbb{E}_{\boldsymbol{x}\sim\hat{p}_{\text{data}}} \log p_{\text{model}}(\boldsymbol{x}) \tag{5.9}$$

也就是最小化交叉熵. 需要强调的是, 任何一个包含负对数似然的损失函数都是训练集经验分布和模型真实分布之间的交叉熵.

实际上, 我们可以让模型分布逼近经验分布, 但是无法逼近真实分布.

5.5.1 Conditional Log-Likelihood and Mean Squared Error

极大似然估计可以推广到更加一般的条件分布的情形.

$$\begin{aligned} \boldsymbol{\theta}_{ML} &= \arg\max_{\boldsymbol{\theta}} P(\boldsymbol{Y}|\boldsymbol{X};\boldsymbol{\theta}) \\ &= \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log P(\boldsymbol{y}^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta}) \end{aligned} \tag{5.10}$$

需要注意的是, 第二行等式成立的条件是所有样本满足 i.i.d 条件.

5.5.2 Properties of Maximum Likelihood

如果极大似然估计满足

1. 真实分布 p_{data} 被包含在模型分布族 $p_{model}(\cdot;\boldsymbol{\theta})$ 中;
2. 真实分布 p_{data} 对应的 $\boldsymbol{\theta}$ 有唯一值.

则估计量满足5.4.5中提到的一致性. 由一致性还可以推导出其他估计量性质, 不同的估计量有不同的统计效率 (statistic efficiency). 最小均方误差 (MSE) 最为一致性估计具有比较好的性质, 被广泛应用于各种算法中, 有时还被加上各种归一化项.

5.6 Bayesian Statistics

上节中提到的极大似然估计是频率学派的观点. 与之对应的是贝叶斯学派 (Baysian) 认为知识的确定性需要用概率来进行表示[6].

确定参数 $\boldsymbol{\theta}$ 的过程首先选取一个不确定性最大即熵最大的分布 $p(\boldsymbol{\theta})$, 然后通过观测样本, 利用贝叶斯公式求取 $\boldsymbol{\theta}$ 的后验概率分布. 从本质上来说, 这是一个熵降低的过程.

贝叶斯估计与极大似然估计有两点不同:

1. 极大似然估计对 $\boldsymbol{\theta}$ 进行点估计, 而贝叶斯估计根据 $\boldsymbol{\theta}$ 的全分布去做预测;

2. 贝叶斯估计中的先验分布暗含了参数偏好.

一般情况下, 如果训练样本较少, 贝叶斯估计会有较好的泛化能力, 但是它也存在着计算代价较高的缺点.

5.6.1 Maximum A *Posteriori* (MAP) Estimation

由于贝叶斯统计量估计出的是参数的全分布, 但是一般希望导出的是一个点估计. 所以使用 MAP 得到一个对应的点估计.

$$\boldsymbol{\theta}_{MAP} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\boldsymbol{x}) = \arg\max_{\boldsymbol{\theta}} \log p(\boldsymbol{x}|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) \tag{5.11}$$

由于 $p(\boldsymbol{\theta})$ 是预先选定的, 所以只要关心 $p(\boldsymbol{x}|\boldsymbol{\theta})$ 即可.

MAP 贝叶斯推断利用了训练集中所不包含的先验知识进行推断, 可以减少方差, 但同时也会增加偏差.

许多正则化策略在本质上就是 MAP 贝叶斯推断 (在估计过程中添加先验知识 $p(\boldsymbol{\theta})$). 并且由于 MAP 贝叶斯推断提供了一个设计复杂可解释正则化项的方法.

5.7 Supervised Learning Algorithms

本节介绍了几种典型的有监督学习模型.

[6]用概率表征预先知道的信息

5.7.1 Probabilistic Supervised Learning

有监督学习可以等价为求取后验概率 $p(y|\boldsymbol{x})$ 的问题. 线性回归问题可以进行扩展用来解决分类任务. 在 logistic regression 中, 借助于 logistic sigmoid 函数, 可以将线性回归进行扩展以解决二分类问题.

由于上述方法没有闭式解, 所以需要通过使用梯度下降法最小化负对数似然进行迭代求解. 这也是其他有监督学习的求解方法.

5.7.2 Support Vector Machine

logistic regression 与 SVM 都被用来解决二分类任务. 但是 LR 给出的是一个概率,SVM 直接输出标签.

SVM 引入 kernel 方法后, 其性能得到大大提升. 核化方法有两大优势:

1. 将非线性模型进行映射, 借助于凸优化求解问题;

2. 比计算原始高维向量更具有优势.

最常用的核函数是*高斯核* (Gaussian kernel), 也被称为*径向基函数核* (radial basis function kernel).

$$k(\boldsymbol{u}, \boldsymbol{v}) = \mathcal{N}(\boldsymbol{u} - \boldsymbol{v}; 0, \sigma^2 \boldsymbol{I}) \tag{5.12}$$

其中 $\mathcal{N}(\boldsymbol{x}; \mu, \Sigma)$ 是标准正态密度.RBF 实质上就是一种模板匹配, 算法训练的过程就是如何生成模板.

除了 SVM, 其他的算法也可以使用核方法. 同时核方法的缺点也很明显, 损失评估函数是训练样本的线性叠加. 但是 SVM 可以稀疏地叠加部分训练样本. 此外, 核方法的计算代价也比较高.

5.7.3 Other Simple Supervised Learning Algorithms

k-近邻可用于回归任务或者分类任务, 其在大规模数据集上可以达到很高的准确率, 并趋近于贝叶斯错误率. 但是它的计算代价很高, 并且不能学习或者选择出区分度高的特征.

5.8 Unsupervised Learning Algorithms

有监督学习和无监督学习之间并没有严格的区分界限. 最典型的无监督学习是找到数据 “最佳” 的表示方式. 最佳表示方式一般遵循三条原则:

- 低维度表示 lower dimensional representations
- 稀疏表示 sparse representations
- 独立成分表示 independent representations

这三条原则并不是互相排斥的. 特征表示是深度学习的一个核心思想, 其他的特征表示学习算法也以不同的方式去运用上述三种不同的原则.

5.8.1 Principal Component Analysis

PCA 是一种非监督特征学习方法, 利用到低维度表示独立成分表示两条原则.PCA 只是去除了特征间的线性关系, 如果要使得特征间完全独立, 还需要去除变量间的非线性关系.PCA 在本质上是将原空间中的主成分与新空间的基进行对齐. 为进一步去除数据元素间的耦合关系, 需要由线性变换向前更进一步.

5.8.2 k-means Clustering

k-均值聚类的样本标记可以用 one-hot code 进行表示, 这是一种稀疏表示方式.one-hot code 在统计和计算上有很大的优势.

k-均值聚类训练过程是一个迭代过程, 但它也是一个病态问题, 因为没有一个衡量标准去度量聚类结果与真实情况之间的契合程度. 实际中可能会存在多种聚类方式从不同方面契合真实情况.

从表示方式上来看, 我们更加偏向于*分布式表示* (distributed representations). 分布式表示可以减轻算法选择人类喜好特征的负担.

5.9 Stochastic Gradient Descent

随机梯度下降法 (SGD) 是梯度下降法的延伸, 对于深度学习的发展起到推动作用.

大规模的训练数据集会带来比较强的泛化性能, 但同时也会增加计算代价. 一般的损失函数会计算所有样本在损失函数下的和, 梯度下降法会在损失函数上计算梯度.

SGD 将梯度视为一个期望, 即可以通过样本子集的梯度加和估计整体的梯度.

梯度下降对于非凸问题是低效并且是不可靠的, 但它可以让损失函数在极短时间内下降, 从而使得模型可用. 在深度学习领域更是需要借助梯度下降法快速获得可用的模型.

随机梯度下降法还被用于深度学习以外的领域. 当训练集样本数量上升时,SGD 会逐渐收敛, 但其复杂度被认为是 $O(1)$.

5.10 Building a Machine Learning Algorithm

深度学习算法的组成元素很简单: 数据集的特征组合方式, 代价函数, 优化方法和模型. 通过替换各个元素, 可以组合出多种算法. 对于非监督学习, 这个方法同样适用.

对于代价函数, 只要可以计算损失函数的梯度, 不计算损失函数也可以进行优化.

大部分机器学习算法都可以从上述几个部分进行归纳[7], 只不过有些算法的组成部分并不明显.

5.11 Challenges Motivating Deep Learning

传统算法在 AI 领域内处理高维数据的能力和泛化能力都存在缺陷, 深度学习有效地克服了这一问题.

[7]TODO: 抽空总结一下

5.11.1 The Curse of Dimensionality

涉及到高维特征的机器学习算法会变得异常困难, 这被称为维度诅咒 (curse of dimensionality).

5.11.2 Local Constancy and Smoothness Regularization

通过先验知识和学习所得到的分布函数决定了模型的泛化能力. 分布函数可以通过显式或者隐式的方式进行选择. 其中, 隐式地选择方式有: 平滑先验 (smoothness prior) 或者局部一致性先验 (local constancy prior). 但是仅仅依靠上述的先验而排除掉其他类型的先验是不够的.

平滑先验和非参数学习算法可以再样本分布符合一定条件的情况下取得较好的泛化能力. 但是在高维度情况下, 平滑的分布函数可能在某个维度上变得不平滑. 如果分布函数比较复杂, 这个结论是否还能够成立? 答案是肯定的. 在限定了数据分布的情况下, 平滑假设依然成立.

深度学习算法通过提供不同任务上的通用合理假设以利用这些便利条件. 另外一种方法是针对特定任务设定不同的假设, 但是这些假设不会被嵌入到深度学习模型之中. 深度学习的核心思想是假设样本数据是在多种因素在分层次作用生成的, 这就使得样本数量和样本分布空间呈指数关系.

5.11.3 Manifold Learning

在机器学习领域中, 流形用以指代小范围内的点集组成的连通域. 在流形上, 每一个点的维度都有可能不一样.

许多机器学习的任务在全局内学习分布函数的代价很大, 流形学习通过添加空间限制克服了这一困难. 由于流形是嵌入在高维空间内的, 为何不直接使用低维空间或者高维对数据进行表示有两方面的原因:

1. AI 领域内的任务中, 图像, 文字和声音的分布相对集中, 同时噪声在整个空间上的分布很分散, 均匀.

2. 这些样本的领域和变换方式很直观, 虽然可能不严谨.

使用流形的坐标轴去度量流形具有很深刻的现实意义, 并且对于提升机器学习算法很有帮助. 但同时也具有很大的挑战性.

Chapter 6 Deep Feedforward Networks

深度前向神经网络 (Deep Feedforward Network) 是深度神经网络的经典模型, 它通过学习参数 $\boldsymbol{\theta}$ 来估计映射函数 $y = f(\boldsymbol{x})$. 在前向神经网络中, 信息向前流动没有反馈, 有反馈的网络被称为 *recurrent neural network*(RNN). 从工程角度看, 前向神经网络是许多工业用途的神经网络的基础, 因而它十分重要. 前向深度神经网络通过有向无环图直观表示如何将不同的函数进行嵌套组合, 进而形成网络.

在发展之初, 神经网络是一个神经科学模型, 但如今它在数学和工程领域内得以发展, 也就脱离了原始的神经科学的范畴.

如果想要理解神经网络, 最好从线性模型及其不足之处开始, 逐步过渡到神经网络领域中. 通常如果要将线性模型扩展到非线性领域, 常用映射 ϕ 为原始数据提供一种新的表示方法. 常见的映射有三种形式:

1. 使用通用的映射 ϕ(如 RBF)

2. 人工设计映射 ϕ

3. 使用机器学习的方法学习出映射 ϕ

使用特征学习来提升模型性能的方法不仅仅只是局限于前向神经网络. 特征学习是深度学习中一个广泛的课题.

6.1 Example: Learning XOR

本节通过举例使用不同的机器学习方法学习出 XOR 的映射函数.

首先将该任务视为一个回归任务, 使用最小均方误差作为损失函数. 但是可以看出最小均方误差维持在 0.5 并且很难继续下降. 其原因是线性模型难以将样本进行正确划分.

接着引入单隐层的前向神经网络. 大多数神经网络单元对输入做仿射变换后, 还使用激活函数做非线性变换, 因而可以完美的拟合 XOR 映射函数. 在现代的神经网络中, 常用 *rectified linear unit*(ReLU) 作为激活函数. 其定义为

$$g(z) = \max\{0, z\} \tag{6.1}$$

6.2 Gradient-Based Learning

神经网络使用梯度下降法进行优化. 由于神经网络的损失函数非凸, 没有闭式解, 通常使用梯度下降法进行迭代, 是损失函数达到一个很小的值. 需要注意的是, 这个很小的值一般不是全局最小值, 但是这个值可以使模型达到实际可用的程度. 深度学习算法使用随机梯度下降法 (stochastic gradient descent, SGD) 提升计算效率, 但是 SGD 不保证收敛且计算结果与初始值的选取有很大关联性. 深度学习领域内的优化过程均是梯度下降法基础上发展出来的变体. 与其他模型相比, 深度神经网络的训练过程并无实质性差别, 尽管梯度下降法可能使得训练过程稍显复杂, 但是整体上来说还是比较高效准确的. 此外, 与其他模型一样, 必须结合深度神经网络的特殊场景设计损失函数.

6.2.1 Cost Function

如果模型定义了分布 $p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$, 则使用交叉熵作为损失函数 (6.2.1). 如果模型是为了估计 $\boldsymbol{y}$ 的某些统计量, 则使用针对任务特别设计的损失函数. 上述损失函数的基本型还需要加上正则化项形成完整的损失函数 (6.2.1).

Learning Conditional Distributions with Maximum Likelihood

交叉熵损失函数定义为

$$J(\boldsymbol{\theta}) = -\mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim\hat{p}_{\text{data}}} \log p_{\text{model}}(\boldsymbol{y}|\boldsymbol{x}) \tag{6.2}$$

其中 $\log p_{\text{model}}(\cdot)$ 的具体形式与模型本身有关.

交叉熵损失函数的优点是, 不用设计损失函数, 根据模型分布 $p(\boldsymbol{y}|\boldsymbol{x})$ 就可以确定损失函数的具体形式. 负对数似然项满足损失函数梯度值域区间

大的要求. 缺点是损失函数不一定存在最小值点, 但是可以通过添加正则项来规避这一问题.

Learning Conditional Statistics

如果模型的目的是为了学习全分布相关的统计量而非全分布本身, 那么可以使用根据实际任务设计损失函数. 深度学习模型本身可以看做一个函数族, 并且函数族中每一个函数的参数取值都不是固定值. 那么就可以把损失函数视为 *functional*. 解决上述问题的工具被称为 *calculus of variations*.

如果通过最小化均方误差的方法求取 $\boldsymbol{y}$ 的均值 $\boldsymbol{x}$, 则可以解下面的问题

$$f^* = \arg\min_f \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim p_{\text{data}}} \|\boldsymbol{y} - f(\boldsymbol{x})\|^2 \tag{6.3}$$

等价于

$$f^*(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{y}\sim p_{\text{data}}(\boldsymbol{y}|\boldsymbol{x})}[\boldsymbol{y}] \tag{6.4}$$

通过最小化绝对均值误差 (mean absolute error) 求取 $\boldsymbol{y}$ 的中值 $\boldsymbol{x}$, 可以使用下面的优化问题

$$f^* = \arg\min_f \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim p_{\text{data}}} \|\boldsymbol{y} - f(\boldsymbol{x})\|_1 \tag{6.5}$$

上述上述两种问题的缺点是,MSE 和 MAE 使用梯度优化的表现并不好.

6.2.2 Output Units

本节主要介绍各种类型的输出函数. 损失函数与输出函数关系紧密. 需要注意的是, 本节介绍的输出单元也可用作隐层单元.

Linear Units for Gaussian Distributions

线性单元只对输入做放射变换

$$\hat{\boldsymbol{y}} = \boldsymbol{W}^T\boldsymbol{h} + \boldsymbol{b} \tag{6.6}$$

常用来估计条件正态分布的均值. 并且由于在正态分布条件下,MSE 等价于极大对数似然, 损失函数为 MSE. 如果要估计条件正态分布的协方差, 就需要添加限制条件.

线性单元的优点是不会达到 saturate 状态, 用梯度优化算法没有太大困难.

Sigmoid Units for Bernoulli Output Distributions

使用 Sigmoid 函数替代原始 Bernoulli 分布以便使用梯度优化.Sigmoid 单元定义为

$$\hat{y} = \sigma(\boldsymbol{w}^T \boldsymbol{h} + b) \tag{6.7}$$

则有分布

$$P(y) = \frac{\exp(yz)}{\sum_{y'=0}^{1} \exp(y'z)} \tag{6.8}$$

相应的损失函数为

$$J(\boldsymbol{\theta}) = -\log P(y|\boldsymbol{x}) = \zeta((1-2y)z) \tag{6.9}$$

如果使用其他的损失函数,Sigmoid 可能会 saturate. 定量来看,Sigmoid 的对数完备且有限.

Softmax Units for Multinoulli Output Distributions

如果使用 softmax 单元,sigmoid 可以视为其一般形式. 当然,softmax 也可用在隐层做 n 选 1 的选择.Sigmoid 定义为

$$\text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)} \tag{6.10}$$

其中 z_i 为向量 $\boldsymbol{z} = \boldsymbol{W}^T \boldsymbol{h} + \boldsymbol{b}$ 第 i 个元素.

Softmax 单元使用对数最大似然进行训练. 如果不使用对数似然作为损失函数的话, 很难处理 softmax 函数.softmax 函数也可能 saturate.

Softmax 函数中参数 $\boldsymbol{z}$ 的产生方法有两种. 第一种是由前一层神经元产生 n 个参数, 第二种是通过添加限制产生 $n-1$ 个参数. 这两种方法并没有太大的差别.

从神经科学角度看 softmax 单元, 可以将其视为竞争关系. 极端情况下就是 *winner-take-all* 形式.

Softmax 名称可以解释为 *"softened" version of the* $\arg\max$.

Other Output Types

本节主要介绍的是估计 $\boldsymbol{y}$ 相关统计量是用到的输出层单元类型. 其核心思想是, 线性,sigmoid 和 softmax 可以作为任何神经网络的输出层, 最大似然度准则提供了设计损失函数的方法. 最大似然度准则使用对数似然作为损失函数的具体实现形式. 一般的, **神经网络表示函数 $f(\boldsymbol{x};\boldsymbol{\theta})$, 与预测值并无直接关系. 但它为 $\boldsymbol{y}$ 的分布提供参数.** 假设有 $f(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{w}$, 则损失函数可以被写成 $-\log p(\boldsymbol{y};\boldsymbol{w}(\boldsymbol{x}))$.

6.3 Hidden Units

怎样选择前向神经网络中隐层神经元类型并无定式, 需要根据具体情况去分析.

某些隐层神经元甚至不能满足处处可导的性质, 但是仍然使用梯度优化算法是因为大部分情况下, 训练过程并不能达到不可导位置, 即使达到了不可导位置, 也可以对到达不可导点位置的样本进行忽略.

6.3.1 Rectified Linear Units for Their Generalizations

Rectified 单元的激活函数为

$$g(z) = \max\{0, z\} \tag{6.11}$$

它与线性单元很像, 但是非常便于求梯度.Rectified 单元通常用在仿射变换上

$$\boldsymbol{h} = g(\boldsymbol{W}^{\boldsymbol{T}}\boldsymbol{x} + \boldsymbol{b}) \tag{6.12}$$

从经验上来说, 通常把 $\boldsymbol{b}$ 中每个元素的初始值都设置的非常小, 可以取得较好的效果.

Rectified 单元的缺点很明显, 当样本映射变换后的值为 0 时, 函数不可导, 不可进行梯度优化.

为了避免上述缺陷, 当 $z_i < 0$ 时, 使用非 0 斜率 α_i

$$h_i = g(z, \alpha)_i = \max(0, z_i) + \alpha_i \min(0, z_i) \tag{6.13}$$

并在此基础上有三种变体:

- *Absolute value rectification*: 固定 $\alpha_i = -1$, 获得 $g(z) = |z|$;
- *Leaky ReLU*: 固定 α_i 到一个非常小的值;
- *Parametric ReLU(PReLU)*: 将 α_i 当做一个可学习的参数.

更进一步, 有 *maxout units*

$$g(z)_i = \max_{j \in \mathbb{G}^{(i)}} z_j \tag{6.14}$$

其中 $\mathbb{G}^{(i)}$ 第 i 组输入下标集合,$\{(i-1)k+1, \cdots, ik\}$.Maxout 的本质是在空间上拟合出分段线性方程, 也可视为自己学习激活函数. 相比于 rectified,maxout 需要更多正则化项, 但是数据较多且 k 较小时, 可以省去正则化项.

Maxout 的优势为, 参数少统计便利; 可以用作无损特征融合或降维; 有冗余, 避免灾难性遗忘 (catastrophic forgetting).

上述 rectified 及其变体的特性为, 如果它们的行为接近线性, 就会非常容易进行优化.

6.3.2 Logistic Sigmoid and Hyperbolic Tangent

许多神经网络单元使用 sigmoid 激活函数

$$g(z) = \sigma(z) \tag{6.15}$$

或者 *hyper tangent* 激活函数

$$g(z) = \tanh(z) \tag{6.16}$$

这两种激活函数很类似, 因为有 $\tanh(z) = 2\sigma(2z) - 1$.

Sigmoid 函数大部分时候会 saturate, 只有在 0 附近时才会 sensitive, 不适宜使用梯度优化. 因此不推荐使用 sigmoid 单元作为前向神经网络的隐层单元.Sigmoid 在前向神经网络之外的网络中使用比较多.

Hyperbolic tangent 单元在前向神经网络的隐层中表现稍微好一些.

6.3.3 Other Hidden Units

其他种类的隐层单元使用很少, 只有表现特别突出的隐层单元才会受到关注. 但是通常情况下, 表现平平的隐层单元很多, 它们并没有受到太多的关注. 这里列举几种比较特别的隐层单元类型:

- 没有激活函数的隐层单元
- softmax 隐层单元
- 径向基函数隐层单元 $h_i = \exp\left(-\frac{1}{\delta^2}\|\boldsymbol{W}_{:,i} - \boldsymbol{x}\|^2\right)$
- Softplus 隐层单元 $g(a) = \zeta(a) = \log(1 + e^a)$(a smooth version of the rectifier)
- Hard $\tanh g(a) = \max(-1, \min(1, a))$

隐层单元的类型到目前为止还是一个比较热门的研究方向.

6.4 Architecture Design

网络的结构包括两个方面: 网络的层数和神经元之间的连接方式. 绝大部分的神经网络是按层组织的, 层数的选择包括每一层中的神经元数目, 这个是通过验证集上的错误率决定的.

6.4.1 Universal Approximation Properties and Depth

大多数情况下, 神经网络学习的目标是一个非线性函数. **前向神经网络有一个通用结构估计框架 (universal approximation theorem)**. 假如一

个神经网络有线性输出层以及任意一种隐层, 就可以学习出一个从有限维输入空间到有限维输出空间的 *Borel 可度量函数* (Borel measureable function), 并且假如有足够多隐层单元, 就可以有足够低的非零错误率. 更进一步, 有限离散空间的映射以及其他类型的函数均可代入上述框架. 但是当

1. 优化算法找不到目标函数的最优参数;

2. 过拟合导致选择错误的函数类型.

时, 通用结构估计框架不能保证算法一定能学习出目标函数. 但是需要注意的是, 通用结构估计框架估计出的网络规模可能会很大, 因此前向神经网络可能不会被正确训练.

另外, **特定函数可以通过估计得出至少需要 d 层网络进行训练学习**. 如果使用 rectifier 单元, 网络层数增加, 神经网络的参数会呈指数级增长. 有 d 个输入, 深度为 l 和每层隐层有 n 个单元的深度 rectifier 网络可以划分出

$$O\left(\binom{n}{d}^{d(l-1)n^d}\right) \tag{6.17}$$

个线性区域. 但是在实际中, 并不能保证函数满足上述条件.

从上述定理可以看出, 深度神经网络暗含了一个先验知识, 我们所要学习的函数通常可以由几个稍微简单一些的子函数组合形成.

6.4.2 Other Architectural Considerations

实际使用的网络结构类型多样, 并且会根据实际任务在细节上做一些调整. 比如, 网络各层之间不一定是链式连接; 神经元之间也不一定是全连接, 会有较少的参数引入, 但是同样也有可能引入一些问题.

6.5 Back-Propagation and Other Differentiation Algorithms

逆向传播算法 (back-propagation) 前向传播过程是指训练数据在神经网络中向前传播得到损失函数的过程. 逆向传播过程是指从损失函数向后传播更新网络模型参数的过程. 传统的梯度数值计算代价很大,BP 传播算法避免了高昂的计算代价. 需要注意的是 BP 算法仅指梯度的计算方法.

6.5.1 Computational Graphs

为了更加精确地描述 BP 算法, 本节引入了计算图 (computational graph) 的概念.

6.5.2 Chain Rule of Calculus

链式法则计算是指通过计算子函数获得整个函数导数的方式.BP 算法使用链式法则可以提升效率. 链式法则可以用来计算函数对数值, 向量, 张量 (tensor) 的导数.

6.5.3 Recursively Applying the Chain Rule to Obtain Backprop

虽然梯度的计算公式很直观, 但是在编程实现的过程中, 还有其他的因素需要考虑. 比如, 许多子公式需要计算多次, 为了提升计算效率避免重复计算, 最好将这些计算结果记录下来. 本节接下来的内容主要介绍了直接计算 BP 算法的过程.BP 算法设计初衷是为了减少子式的重复计算次数. 同样的, 也存在其他的算法可以达到同样的目的, 或者为了节省内存进行重复计算.

6.5.4 Back-Propagation Computation in Fully-Connected MLP

书中的 Algorithm 6.3 和 Algorithm 6.4 是针对 MLP 问题适用的 BP 算法. 而编程实现的 BP 算法是基于计算图的一般化方法.

6.5.5 Symbol-to-Symbol Derivatives

代数公式和计算图对符号进行操作, 只有在实际运算中才会将符号替换成数字. 实际计算导数有两种方式, 一种是 Torch 和 Caffe 所使用的符号到数字 (symbol-to-number) 方式. 这种方式接收数字作为计算土的参数,

并且返回一个梯度数值; 另一种是 Theno 所使用的符号到符号 (symbol-to-symbol) 方式, 它通过在计算图中添加额外的节点以描述希望计算的导数.

其实从本质上来看,symbol-to-number 方式可以归入 symbol-to-symbol 方式之中.

6.5.6 General Back-Propagation

一般地, 在链式法则的某个环节, 梯度等于上一环节的梯度乘以当前操作所涉及到的变量的 Jacobian 矩阵. 在编程实现 BP 算法时, 通常针对同一个环节提供 operation 和 bprop 两个操作.bprop 操作实质上是计算所有输入变量的 Jacobian 矩阵.

一般情形下的梯度计算需要 $O(n^2)$ 次操作. 但是大部分神经网络的损失函数是链式结构, 因此计算梯度只需要 $O(n)$ 次操作. 并且还可以通过动态规划方法对子问题进行填表, 避免重复计算.

6.5.7 Example: Back-Propagation for MLP Training

在这一节中, 以 MLP 为例, 使用计算图计算 BP 算法. 虽然举例中涉及到的计算图很庞大, 但是可以很方便地计算出梯度.

6.5.8 Complications

前面几个章节所描述的 BP 算法仅仅是理论上的分析. 为了编程实现 BP 算法, 还需要考虑:

- 操作返回多个张量的情形;
- 内存消耗管理;
- 数据类型;
- 梯度无意义的情形.

6.5.9 Differentiation outside the Deep Learning Community

自动化微分 (automatic differentiation) 专门研究微分的算法.BP 是其研究方向之一, 被称为*反向模式累积* (reverse mode accumulation). 还有其他的方法通过调整微分次序以获得高效解法, 但获得最优解是一个 NP 难问题.

提升 BP 算法效率的途径有

1. 寻找导数计算的等价形式

2. 简化图结构

但是需要注意的是, 针对深度学习的特定的梯度算法有助于提升学习算法的效率和稳定性. 因此虽然还有其他的梯度计算方法, 但是 BP 算法在深度学习领域被广泛使用.

6.5.10 Higher-Order Derivatives

深度学习中常用到 Hessian 矩阵, 但是当参数很多时, 计算代价很高. 常用 *Krylov 方法* (Krylov methods) 简化 Hessian 矩阵的计算. 可以通过直接计算

$$\boldsymbol{H}\boldsymbol{v} = \nabla_{\boldsymbol{x}}\Big[\Big(\nabla_{\boldsymbol{x}} f(x)\Big)^T \boldsymbol{v}\Big] \tag{6.18}$$

其中 $\boldsymbol{v}$ 是一个任意向量. 为了能够计算任意 Hessian 与任意向量的乘积, 可以计算 $\boldsymbol{H}\boldsymbol{e}^{(i)}$, 其中 $i = 1, \cdots, n; e^{(i)}$ 是一个 one-hot 向量.

6.6 Historical Notes

本章介绍的前向神经网络是对非线性函数进行估计的模型, 它使用梯度下降法最小化误差. 相关的技术发展历程为:

1. 链式法则:17 世纪

2. 梯度下降法:18 世纪

3. 线性神经网络:20 世纪 40 年代

4. 非线性神经网络:20 世纪 60 到 70 年代

5. *Parallel Distributed Processing* 文献中提出 BP 算法

6. 神经网络发展高潮:20 世纪 90 年代

7. 前向神经网络:20 世纪 80 年代

从 1986 年到 2015 年, 前向神经网络得到了很大发展, 其原因有:

- 数据规模更大
- 神经网络规模更大

相应的算法上改进有:

- 交叉熵取代 MSE
- 分段线性的隐层单元取代 sigmoid 隐层神经元

在今天, 前向神经网络扭转了以往的差评局面, 但是仍然有很大的发展空间.

Chapter 7 Regularization for Deep Learning

正则化项主要用来降低测试误差, 借以提升模型的泛化能力, 是机器学习领域内的一个重要研究方向. 本章的正则化特指为了降低泛化误差而对学习算法进行的修改. 正则化项有两种作用方式: 对先验知识进行编码; 降低模型复杂度. 深度学习中使用的正则化策略绝大部分是基于正则化估计量的策略. 学习算法训练出的模型可能会出现

1. 将真实数据生成过程排除在模型外;
2. 符合真实的数据生成过程;
3. 包含了真实的数据生成过程, 但是也包含了其他可能的生成过程.

正则化的目的就是将上述情况3转化为情况2. 在深度学习的场景中, 与真实情况最匹配的模型可能是一个包含正则化项的复杂模型.

7.1 Parameter Norm Penalties

正则化出现的时间比深度学习要早, 通过在目标函数 (objective function)J 中添加范数补偿 (norm penalty)$\Omega(\boldsymbol{\theta})$ 达到限制模型 capacity 的目的. 正则化后的目标函数为

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta}) \tag{7.1}$$

其中 $\alpha \in [0, \infty)$ 是平衡正则化项权重的超参数. 在深度学习中, 可以对神经网络的每一层分别设置参数 α, 但是这样计算代价太大, 通常将所有层的参数设置成相同的.

7.1.1 L^2 Parameter Regularization

L^2 正则化又被称为岭回归 (ridge regression) 或者 *Tikhonov* 正则化 (Tikhonov regularization). 其定义为

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) + \frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w} \tag{7.2}$$

对应的权重 $\boldsymbol{w}$ 更新公式为

$$\boldsymbol{w} \leftarrow (1 - \epsilon\alpha)\boldsymbol{w} - \epsilon\nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) \tag{7.3}$$

可以看出正则化对于单步优化的影响为减小了步长.

通过对 J 在点 $\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} J(\boldsymbol{w})$ 做二次展开, 并加上正则化项求导, 设在 $\tilde{\boldsymbol{w}}$ 导数为 0, 有

$$\begin{aligned} \tilde{\boldsymbol{w}} &= (\boldsymbol{H} + \alpha\boldsymbol{I})^{-1}\boldsymbol{H}\boldsymbol{w}^* \\ &= \boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha\boldsymbol{I})^{-1}\boldsymbol{\Lambda}\boldsymbol{Q}^T\boldsymbol{w}^* \end{aligned} \tag{7.4}$$

可以看出, 与 $\boldsymbol{H}$ 的第 i 个特征向量对应的 $\boldsymbol{w}^*$ 缩小比例为 $\frac{\lambda_i}{\lambda_i+\alpha}$. 当 $\lambda_i \gg \alpha$ 时, 正则化影响很小; 当 $\lambda_i \ll \alpha$ 时, 缩小至 0 附近. 也就是说, 正则化项只有对目标函数下降比较明显的方向上的分量才会做完整的保留.

对线性回归问题, 正则化项加上均方误差为

$$(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})^{\boldsymbol{T}}(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}) + \frac{1}{2}\alpha\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{w} \tag{7.5}$$

对应的对 $\boldsymbol{w}$ 导数为 0 的点为

$$\boldsymbol{w} = (\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X} + \alpha\boldsymbol{I})^{-\boldsymbol{1}}\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{y} \tag{7.6}$$

正则化项对输入中方差较高项进行了保留.

7.1.2 L^1 Regularization

L^1 正则化项被定义为

$$\Omega(\boldsymbol{\theta}) = \|\boldsymbol{w}\|_1 = \sum_i |w_i| \tag{7.7}$$

相应的目标函数为

$$\tilde{J}(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}) = \alpha\|\boldsymbol{w}\|_1 + J(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}) \tag{7.8}$$

正则化项对于梯度的贡献只有符号. 为了简化对 1-范数求导, 假设 Hessian 矩阵是对角矩阵 $\boldsymbol{H} = diag([H_{1,1},\cdots,H_{n,n}])$, 其中 $H_{i,i} > 0$. 有

$$\hat{J}(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}) = J(\boldsymbol{w}^*;\boldsymbol{X},\boldsymbol{y}) + \sum_i \left[\frac{1}{2}H_{i,i}(\boldsymbol{w_i} - \boldsymbol{w_i^*})^2 + \alpha|w_i|\right] \tag{7.9}$$

极小值为

$$w_i = \text{sign}(w_i^*)\max\{|w_i^*| - \frac{\alpha}{H_{i,i}}, 0\} \tag{7.10}$$

可以看出,L^1 正则化让解更稀疏. 所以它常被用来作特征选择.

此外,L^2 正则化等价于高斯先验的 MAP 贝叶斯推断;L^1 正则化等价于各向同性 Laplace 分布先验的 MAP 贝叶斯推断[1].

7.2 Norm Penalties as Constrained Optimization

在前一节中提到的式7.1中的优化问题可以通过 Lagrangian 函数转化为有限制条件的优化问题

$$\mathcal{L}(\boldsymbol{\theta},\alpha;\boldsymbol{X},\boldsymbol{y}) = J(\boldsymbol{\theta};\boldsymbol{X},\boldsymbol{y}) + \alpha(\Omega(\boldsymbol{\theta}) - k) \tag{7.11}$$

相应的, 解为

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}}\max_{\alpha,\alpha\geq 0}\mathcal{L}(\boldsymbol{\theta},\alpha) \tag{7.12}$$

定性地分析, 最优值 α^* 会使 $\Omega(\boldsymbol{\theta})$ 的值不断减小, 但还没有到达让 $\Omega(\boldsymbol{\theta})$ 小于 k 的程度. 因此, 我们可以将范数补偿视为在目标函数上增加限制条件. 如果不知道限制区域的大小, 可以通过调节 α 对限制区域的大小进行调节.

上述显式限制条件有两个优点

[1]正则化项等价于式5.11中的 $\log p(\boldsymbol{\theta})$ 项, 即将参数的先验分布整合进目标函数中进行优化. 但并不是所有的正则化项都等价于 MAP 贝叶斯推断.

- 如果知道限制条件, 即式7.11中 k 的数值, 那么可以直接对 $J(\boldsymbol{\theta})$ 使用梯度下降法求下降方向, 然后在最小值点附近寻找满足 $\Omega(\boldsymbol{\theta}) < k$ 的点. 这种 reprojection 过程就避免了搜索与 k 相匹配的 α 的过程;

- 显式限制条件可以避免隐式限制条件因为引入补偿项而导致的非凸优化过程陷入局部极小点. 而 reprojection 方法可以避免这种现象;

- 增加优化过程的稳定性, 避免因为学习率参数较高导致的权重参数溢出.

在实际中, 常将列范数限制 (column norm limitation) 与显式限制条件相结合使用.

7.3 Regularization and Under-Constrained Problems

在机器学习中, 正则化项需要根据实际问题进行妥善定义, 因为正则化项需要保证未定问题迭代优化过程的收敛性.

在机器学习以外的领域, 也有通过添加正则化项来解决优化问题的方法, 如 Moore-Penrose 广义逆.

7.4 Dataset Augmentation

在机器学习中, 可以通过创造 “假” 数据的方法来增大数据集. 这种方法适用于分类问题, 但是在其他领域内不太适用, 因为可能会改变数据的原始分布.

Dataset augmentation 对物体识别问题特别有效. 需要注意的是,dataset augmentation 不能改变样本的真实标记.

Dataset augmentation 在语音识别领域也被广泛应用.

对不同算法进行性能评估比较时, 必须将 dataset augmentation 也考虑进去, 在同样的条件下进行比较.

7.5 Noise Robustness

在某些神经网络模型上, 给输入添加合适的噪声等价于对权重的范数补偿. 这种方法相比于限制权重的大小更加有效, 特别是将噪声添加到隐层的输入上.

另外, 在网络连接的权重上增加噪声也等价于对模型进行正则化.

增加噪声间接导致了模型学习出稳定的函数.

7.5.1 Injection Noise at the Output Targets

由于某些数据集中部分样本的标注不是完全正确, 所以当 y 错误时, 直接最大化 $\log p(y|\boldsymbol{x})$ 可能会出现问题. 因此 *label smoothing* 被广泛应用, 它使用 softmax 对模型的 0-1 硬标记进行替换.

7.6 Semi-Supervised Learning

半监督学习是指使用 $P(\boldsymbol{x})$ 中采样得到的未标记样本和从 $P(\boldsymbol{x}, \boldsymbol{y})$ 中得到的有标记样本共同预测 $P(\boldsymbol{y}|\boldsymbol{x})$ 的过程.

深度学习中半监督学习的目的是学习特征表示, 这样相同类别的样本就会有相似的特征表示.

与传统的有监督学习和无监督学习分离的方法相比, 半监督学习通过 $P(\boldsymbol{x}, \boldsymbol{y})$ 或 $P(\boldsymbol{x})$ 和 $P(\boldsymbol{y}|\boldsymbol{x})$ 共享参数的方法将两个过程进行融合.

7.7 Multi-Task Learning

多任务学习通过合并不同任务中的样本提升泛化性能. 训练出的模型参数分为两类

- 任务特定参数
- 通用参数

由于通用参数的存在, 模型的泛化能力得到提升, 泛化误差下降. 但是多任务学习也有一项前提假设: 不同的任务受到同一因素影响, 并且这个因素可以通过数据观察得到.

7.8 Early Stopping

早停是指泛化误差达到最小值后, 参数更新到达指定迭代次数后终止优化算法, 并将参数恢复至泛化误差最小点所对应的参数. 通过早停策略可以减少验证集上的误差, 从而提升泛化性能.

早停是一种高效的超参数选择方法, 它可以选择合适的优化算法迭代次数 (超参数), 并且不会对原始的算法产生影响. 但是同时它也会增加计算代价和存储代价.

计算代价 在优化过程中, 每隔一定迭代轮数, 需要计算当前模型在验证集上的误差;

存储代价 在优化过程中需要时刻保存一份最优参数.

早停策略也可以与其他正则化策略一起使用.

为了充分利用所有的训练数据, 可以进行二次训练:

1. 重新初始化模型, 按照第一轮的超参数进行训练;
2. 保持第一轮的最优参数, 加入新的数据继续进行训练.

从本质上来说, 早停将超参数限制在初始值附近的空间中. 在一定条件下, 早停与 L^2 正则化等价. 但是从优化过程上来看, 早停自动决定了正则化程度, 而 L^2 正则化需要调整超参数.

7.9 Parameter Tying and Parameter Sharing

有时我们需要将正则化之外的先验条件整合到模型中去. 此外, 我们可以知道模型的结构和相关领域的知识, 这样模型之间的参数可能会有一些从属关系 (dependency). 最常见的模型间的参数从属关系是共享参数.

共享参数的一个重要优势是其存储优势, 从而借助于共享参数, 可以在不增加样本的情况下提升网络的层数.[2]

7.10 Sparse Representation

另一种正则化方法是对激活函数添加补偿项使其输出稀疏化. 表示方法正则化与参数正则化的机制一致, 即

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{h}) \tag{7.13}$$

其中,$\Omega(\boldsymbol{h})$ 不仅仅可以使用7.1.2节中提到的 L^1 正则补偿项的形式, 还可以借助于其他的一些形式实现稀疏化表达.

除了上述添加补偿项实现表示稀疏, 还可以通过在激活值上添加硬限制条件实现稀疏表达. 典型的方法有 *orthogonal matching pursuit*.

从理论上来说, 任何有隐层的模型都可以进行稀疏化表示.

7.11 Bagging and Other Ensemble Methods

Bagging 是一种集成学习方法, 它在不同的训练集上分别训练出不同的模型, 对测试样本进行投票得到最终结果, 以降低泛化误差. 从整体上来看, 集成学习器至少和它当中的任何一个学习器表现相当, 如果学习器间的误差相互独立, 则它的表现将会比其中任何一个学习器都要好. 通常学习器会通过不同的学习方法生成, 即不同的算法, 不同的目标函数以及不同的模型. 特别地, 通过有放回采样获得 k 个不同的数据子集, 在数据子集上分别训练出不同的模型.

由于神经网络参数经过训练后点分布范围比较广泛, 所以通常在同一个数据集上进行训练, 并将训练完成的模型进行集成.

集成学习是一种极其有效且可靠地降低泛化错误率的方法. 但是它的计算和存储开销很大. 需要注意的是, 并不是所有的集成学习都是为了提升正则化能力, 例如 Boosting 就是为了提升模型的 capacity.

[2]CNN 网络中, 不同位置的出现的特征共享参数, 可以在物体检测任务中得到与空间无关的结果.

7.12 Dropout

Dropout 为各种模型提供便于计算的正则化方法. 特别地, 它为神经网络提供了一种低计算开销的集成方法. 具体来说,dropout 依次去除原始神经网络中的非输出单元, 形成模型子集. 子集中模型的数量与神经单元的数量呈指数级关系. 为了将模型与 dropout 进行结合, 需要使用基于 minibatch 的训练方法, 以提取出指数级数量的训练子集.

Dropout 模型的损失函数定义为

$$\mathbb{E}_{\boldsymbol{\mu}} J(\boldsymbol{\theta}; \boldsymbol{\mu}) \tag{7.14}$$

其中,$\boldsymbol{\mu}$ 是 *mask vector*, 用以指定被包含进子模型的神经单元;$J(\boldsymbol{\theta}; \boldsymbol{\mu})$ 定义了基于模型参数 $\boldsymbol{\theta}$ 和 $\boldsymbol{\mu}$ 的损失函数.

Dropout 与 bagging 的不同之处在于

- Bagging 中学习器间相互独立; 而 dropout 各个子模型之间的参数是共享的;
- Bagging 中各个学习器经过训练都会达到收敛状态; 而 dropout 大部分子模型都没有被充分地训练 (由于参数共享, 可以保证模型有较高的泛化能力).

集成学习中的 Bagging 模型需要将所有子模型对新样本的预测结果收集起来进行综合判断, 这一个过程被称为推断 (inference). 特别地, 针对 dropout 算法, 需要首先建立起计算模型

$$\sum_{\boldsymbol{\mu}} p(\boldsymbol{\mu}) p(y|\boldsymbol{x}, \boldsymbol{\mu}) \tag{7.15}$$

其中 $p(\boldsymbol{\mu})$ 是训练时 mask vector 的概率密度函数. 其次在实际中, 只需要对 $\boldsymbol{\mu}$ 进行采样就可以近似估计出推断结果.

尽管采样可以近似估计出推断结果, 但是还有更好的推断方式: 使用几何平均数代替算术平均数.

$$\tilde{p}_{ensemble}(y|\boldsymbol{x}) = \sqrt[2^d]{\prod_{\boldsymbol{\mu}} p(y|\boldsymbol{x}, \boldsymbol{\mu})} \tag{7.16}$$

进行归一化, 有

$$p_{ensemble}(y|\boldsymbol{x}) = \frac{\tilde{p}_{ensemble}(y|\boldsymbol{x})}{\sum_{y'} \tilde{p}_{ensemble}(y'|\boldsymbol{x})} \tag{7.17}$$

这种推断方式被称为 *weighted scaling inference rule*.

需要注意的是, 我们通常将包含概率[3](inclusion probability) 设为 $\frac{1}{2}$, 所以所以需要在训练过程结束后将神经元的连接权重除以 2. 也可以在训练过程中将每个神经元的状态 (states) 乘 2[4].

实验证明, weighted scaling inference rule 与蒙特卡洛估计相比, 各有好坏, 需要根据实际情况选择最优的策略. 但是总体来说, weighted scaling inference rule 的优势有

- 计算效率高, 可以与其他形式的正则化方法结合使用;
- 计算代价低;
- 对模型和训练过程几乎没有影响.

同时, 也需要认识到它的劣势

- effective capacity 受限, 为了解决这个问题, 需要增大模型的规模;
- dropout 在小规模数据集上的表现不好.

前面提到的 dropout 的随机性并不是 dropout 方法有效的关键, 随机性仅仅是一种估计的手段.*fast drop* 方法就是通过减少梯度计算时的随机性来加速算法收敛的. 另外, dropout 的随机性对其正则化的作用也不充分.

Dropout 催生出了许多正则化方法, 它们都借助于指数级别的子模型来共享参数. 因此,dropout 是迄今为止应用最为广泛的集成学习方法.

Dropout 本质上是一种共享参数的 bagging 方法. 所以, dropout 的 mask 并不一定是严格的 0-1 分布, 它也可以是诸如 $\mathcal{N}(1, \boldsymbol{I})$ 的正态分布.

dropout 还是隐层单元共享参数的方法. 它通过随机屏蔽隐层单元, 使得学习到的特征更具有鲁棒性. 本质上来说, dropout 可视为神经元输入信

[3]The probability becoming part of the sample during the drawing of a single sample.

[4]TODO: figure it out.

息的损失, 而非原始信息的损失. 相比之下, 传统的噪声注入并不能有效抹除上下文相关信息. Dropout 充分利用分布表达的优势, 充分利用上下文信息.

同时从另外一方面来说, dropout 会使噪声加倍, 如果噪声过大的话, 就没必要使用 dropout 进行正则化了.

7.13 Adversarial Training

如果想要评估高精度模型对任务的理解程度, 就需要特别关注误分样本和*对立样本* (adversarial example). 同时, 还可以借助于对立样本对训练集的扰动对模型进行*对立训练* (adversarial training), 以降低测试误差.

对立样本产生的原因是因为模型过度线性化. 对立训练使模型在训练样本周围达到局部平坦, 从而避免模型因局部线性而对扰动特别敏感的特性.

对立训练有助于提升函数族强大的正则化能力.

对立样本还有助于完成半监督学习. 假设不同类别的样本分布于不连通的流形上, 一个很小的扰动不会使得样本从其所在的流形跳到其他流形上.

7.14 Tangent Distance, Tangent Prop, and Manifold Tangent Classifier

切面距离算法 (tangent distance algorithm) 是一种非参数最近邻算法. 与最近邻算法不同的是, 它使用的是流形距离. 即假设不同类别的样本分别位于不同的流形上, 两个不同类别的样本间的距离需要通过优化问题进行求解. 为了简化距离计算, 可以将两个样本在流形上的切面间的距离作为样本间的距离.

在此基础上, *切面传播算法* (tangent prop algorithm) 训练出一个神经网络分类器, 并在其损失函数上添加补偿项

$$\Omega(f) = \sum_i \left((\nabla_{\boldsymbol{x}} f(\boldsymbol{x}))^T \boldsymbol{v}^{(i)} \right)^2 \tag{7.18}$$

使得 $f(\boldsymbol{x})$ 对已知的变量不敏感. 补偿项就包含了先验知识, 使得输出函数在训练样本附近平坦. Tangent prop 算法不仅被用于有监督学习, 还被用于强化学习.

Tangent prop 与 data augmentation 相比, 它们的相同之处为: 编码先验知识, 使得输入有扰动时, 输出不变. 不同之处在于

- data augmentation 中, 对输入样本做多种转换, 保持输出正确; 而 tangent prop 只做输出局部平坦的先验, 并不要求输出达到新的点;
- tangent prop 的较少种类的扰动对 rectified 单元求导造成了困难; 而 data augmentation 由于扰动种类较多, 所以斌不会对 rectified 单元求导造成太大苦难.

Tangent prop 与 double backprop[5]以及对立学习很相似.

- Double backprop 通过正则化将 Jacobian[6]变得很小, 对立训练通过对输入增加扰动, 使模型输出不变;
- Tangent prop 与 data augmentation 都使得输入有扰动时, 输出不变;
- Double backprop 和对立训练都要求输入样本在所有方向上有扰动时, 输出的变化很小;
- Data augmentation 相比于 tangent prop, 输入变换的种类较多;
- 对立训练相比于 double backprop, 变换的方式有很多.

流形切面分类器 (manifold tangent classifier) 避免了使用流形的切向量作为先验知识的麻烦. 它的算法很简单

1. 使用 autoencoder 模型, 无监督学习流行的结构;
2. 使用切面正则化神经网络分类器 (参考 tangent prop 算法).

[5]A new training algorithm termed double backpropagation improves generalization by simultaneously minimizing the normal energy term found in backpropagation and an additional energy term that is related to the sum of the squares of the input derivatives (gradients).

[6]见12页式4.1.

Chapter 8 Optimization for Training Deep Models

深度学习过程涉及到的优化过程耗时长且有着重要的地位. 本章重点介绍深度神经网络训练过程中的优化算法. 一般情况下, 训练过程中的优化目标是包含正则化项的损失函数. 本章将从以下几个方面介绍优化算法

1. 机器学习中的优化问题与传统优化问题的区别;

2. 深度学习中优化问题面临的挑战;

3. 自适应学习率或损失函数二阶求导;

4. 通过简单优化组合出复杂优化的策略.

8.1 How Learning Differs from Pure Optimization

对于机器学习算法中的损失函数 J, 其目的是用来提升机器学习效果 P 的, 但是对于传统优化问题来说,J 仅仅只是一个优化目标. 通常情况下, 损失函数是在所有训练样本上取平均

$$J(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},y)\sim\hat{p}_{data}} L(f(\boldsymbol{x};\boldsymbol{\theta}), y) \tag{8.1}$$

但是我们希望得到的是在数据生成模型分布上的期望泛化误差

$$J^*(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},y)\sim p_{data}} L(f(\boldsymbol{x};\boldsymbol{\theta}), y) \tag{8.2}$$

8.1.1 Empirical Risk Minimization

机器学习的目标是降低期望泛化误差, 又被称为风险 (risk), 但是通过观察式8.2可以看出, 真实数据分布 p_{data} 未知, 因此期望泛化误差不能通过计算直接得到. 因此一个替代方案就是最小化经验风险 (empirical risk)

$$\mathbb{E}_{(\boldsymbol{x},y)\sim\hat{p}_{data}}\left[L(f(\boldsymbol{x};\boldsymbol{\theta}),y)\right]=\frac{1}{m}\sum_{i=1}^{m}L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}),y^{(i)}) \tag{8.3}$$

其中 m 是训练样本的数目.

但是直接使用经验风险最小化策略容易产生过拟合, 另外由于 0-1 损失函数不能直接求导的原因, 深度学习中不使用经验风险最小化策略.

8.1.2 Surrogate Loss Functions and Early Stopping

一般从优化问题效率角度出发使用代理损失函数 (surrogate loss function) 对经验风险函数进行替代. 代理损失函数可以延长学习的过程, 增强训练出的模型的鲁棒性. 同时为了避免过拟合, 通常使用早停策略, 不会将代理损失函数优化到局部极小值点.

8.1.3 Batch and Minibatch Algorithms

机器学习中的优化问题使用部分样本的损失函数加和去估计整体损失函数, 进而对参数进行迭代更新. 在实际中, 从训练集中采样部分样本评估整体风险的做法很常见. 使用采样策略对风险进行评估相比于计算准确的风险, 会使得算法收敛速度更快. 同时, 再重复样本上计算梯度会有大量的冗余.

在全部样本上计算梯度的方法被称为 *batch/deterministic gradient method*, 如果使用单个样本计算梯度的方法被称为 *stochastic/online method*. 深度学习算法中的优化问题介于两者之间, 被称为 *minibatch stochastic method*.

minibatch 尺寸的选择因素有

- 从尺寸增大与回报中做平衡;
- 从处理器核的数目做考虑;

- 从内存角度做考虑;
- 2 的倍数;
- 小尺寸有正则化的作用.

不同的算法会以不同的形式使用 minibatch, 并使用其中的不同信息.

需要注意的是,minibatch 的随机性很重要. 实际中常对样本进行随机打乱来保证随机性. 如果样本分解到位, 可以使用异步并行的方式更新模型参数.

在样本没有重复的条件下,minibatch 方法与梯度下降方法的泛化误差同步下降, 但是当样本重复 (即开始使用样本进行第二轮迭代) 后, 偏差会上升. 在线学习由于会有数据源源不断地输入进来, 所以在泛化误差下降的方面更具有优势. 在非在线学习的情形下, 第一轮是用样本是无偏梯度估计, 之后的迭代是为了减小训练误差与测试误差之间的差异. 当训练数据集很大时, 每个样本只被使用一次, 主要关注的是欠拟合和计算效率问题.

8.2 Challenges and Neural Network Optimization

8.2.1 Ill-Conditioning

在优化问题是凸优化的条件下, 也会存在挑战.Hessian 矩阵病态条件就是其中之一. 如式4.4中的

$$-\epsilon \boldsymbol{g}^T \boldsymbol{g} + \frac{1}{2}\epsilon^2 \boldsymbol{g}^T \boldsymbol{H} \boldsymbol{g} \tag{8.4}$$

就有可能存在 $\frac{1}{2}\epsilon^2 \boldsymbol{g}^T \boldsymbol{H} \boldsymbol{g} > \epsilon \boldsymbol{g}^T \boldsymbol{g}$ 的病态情况.

其他领域内解决病态的问题的方法不能直接用来解决深度学习中的病态问题, 需要做一些调整.

8.2.2 Local Minima

在凸优化问题中, 局部极小值等价于全局最小值. 匪徒问题局部极小并非全局最小, 但这并不是主要问题.

如果一个足够大的训练数据集可以训练出唯一一组模型参数, 那么就可以说这个模型是可*辨识模型* (model identifiability). 神经网络和其他具有隐变量的模型都是不可辨识的. 神经网络具有不可辨识性的原因有

weight space symmetry 对隐层调神经元不影响最终结果;

等价缩放 对某一层的神经元做输入输出的等价缩放, 不影响最终结果.

不可辨识性导致局部极小值点很多, 但是这些局部极小值点有很多具有等价性.

如果局部极小值点的 cost function 比全局最小值点的 cost function 数值高, 就会有问题. 但是如何定量衡量这种差异还没有一个明确的标准. 但是现在, 研究者们表示, 大多数局部极小值点都会一个比较低的 cost function 数值.

8.2.3 Plateaus, Saddle Points and Other Flat Regions

鞍点 (saddle point) 可以视为代价函数在某个横截面上的极小值点, 同时也是另外一个横截面的极大值点.

对于许多随机函数来说, 高维空间中的鞍点数目很多. 同时, 许多随机函数的 Hessian 矩阵正特征值越多, 越有可能达到代价函数的取值越小. 这个性质对于神经网络来说同样适用.

鞍点对基于梯度的优化算法影响并不显著, 这些算法可以迅速逃离鞍点位置. 相比之下, 牛顿法容易陷入鞍点. 因此, 需要使用二阶优化的 saddle-free 牛顿法.

牛顿法同样容易陷入极大值点以及平坦区域.

8.2.4 Cliffs and Exploding Gradients

梯度优化算法遇到 cliff 时, 步长会变得很长. 为了避免这种 *gradient clipping* 现象, 一个简单的解决方案就是使用固定的步长, 除非到了极小值附近的区域.

8.2.5 Long-Term Dependencies

当计算图结构较深时, 优化算法就遇到了挑战. 如在 RNN 中, 相同的操作被重复多次, 就出现了梯度消失和爆炸问题 (vanishing and exploding gradient problem). 但是前向神经网络不存在相同操作被重复多次的结构, 因此就不会有梯度消失和爆炸问题.

8.2.6 Inexact Gradients

在实际中, 通常只能获取到不精确的 Hessian 矩阵的估计值. 当目标函数很难处理时, 梯度也很难被精确计算. 因此只能通过使用代理函数来避免上述问题.

8.2.7 Poor Correspondence between Local and Global Structure

当梯度的优化方向不正确, 前面所描述的问题即使都被妥善解决, 也不能提升优化效果. 在深度神经网络的训练过程中, 大部分时间都被用来选择合适的步长. 在实际中, 神经网络并不会恰好达到极大值点, 极小值点或者鞍点.

大部分研究关注的是如何选取初始值点, 以获的较好的优化结果, 而不是在优化算法中使用非局部移动策略.

梯度下降和所有的神经网络训练算法都是基于局部移动策略, 如果选取正确的初始值点, 就可以得到较好的优化效果.

8.2.8 Theoretical Limits of Optimization

针对神经网络设计的优化算法一般都有效果上限. 这些优化算法对神经网络的实际应用影响很小[1].

[1]TODO: 看一下中文版, 搞清楚以这一节是什么意思.

8.3 Basic Algorithms

8.3.1 Stochastic Gradient Descent

在随机梯度下降 (stochastic gradient descent) 中, 很重要的一个参数就是学习率 ϵ. 实际使用中, 这个参数是动态调整的.

因为 SGD 通过对样本的采样引入了梯度的噪声, 即使损失函数达到了最小值点, 噪声也不会消失, 为了保证 SGD 能够收敛, 学习率 ϵ 需要满足

$$\begin{aligned} \sum_{k=1}^{\infty} \epsilon_k &= \infty \\ \sum_{k=1}^{\infty} \epsilon_k^2 &< \infty \end{aligned} \tag{8.5}$$

在迭代过程中,ϵ 的调整策略为

$$\epsilon_k = (1-\alpha)\epsilon_0 + \alpha\epsilon_\tau \tag{8.6}$$

其中 $\alpha = \frac{k}{\tau}$. 当迭代轮数达到 τ 之后,ϵ 保持常量.

一般通过代价函数的时间曲线选取合适的 ϵ 值.

SGD 的计算时间不会随着样本数量的增加而增加, 同时还可以保证算法的收敛性. 为了研究优化算法的收敛性, 引入度量指标 *excess error*

$$J(\boldsymbol{\theta}) - \min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{8.7}$$

对于 SGD 算法, 在第 k 轮迭代后, 其 excess error 为 $O(\frac{1}{\sqrt{k}})$, 当优化问题是凸优化时,excess error 为 $O(\frac{1}{k})$. 除非引入新的前提假设, 这个上限是不会被提升的. 但是如果收敛速度超过 $O(\frac{1}{k})$ 时, 就会导致过拟合.

8.3.2 Momentum

SGD 算法的学习速率较慢, *动量算法* (Momentum Algorithm) 是用来加速学习速率的. 在动量算法中, 引入了速度 $\boldsymbol{v}$ 的概念, 用来指示在参数空间中搜索最优参数的速率和方向. 为了简便, 在这里将动量定义为速度和单

位质量质点的乘机. 因此有迭代更新公式

$$\begin{aligned} \boldsymbol{v} &\leftarrow \alpha\boldsymbol{v} - \epsilon\nabla_{\boldsymbol{\theta}}\Big(\frac{1}{m}\sum_{i=1}^{m} L(\boldsymbol{f}(\boldsymbol{x}^{(i)};\boldsymbol{\theta}),\boldsymbol{y}^{(i)})\Big) \\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \boldsymbol{v} \end{aligned} \tag{8.8}$$

其中,$\alpha \in [0,1)$ 用来指示历史梯度均值对当前的影响程度.α 也可以通过学习算法进行自适应调整.

与物理领域内的动量定义一样, 这里的动量也与力 $\boldsymbol{f}(t)$ 和速度 $\boldsymbol{v}(t)$ 有关联.

$$\begin{aligned} \boldsymbol{f}(t) &= \frac{\partial^2}{\partial t^2}\boldsymbol{\theta}(t) \\ \boldsymbol{v}(t) &= \frac{\partial}{\partial t}\boldsymbol{\theta}(t) \\ \boldsymbol{f}(t) &= \frac{\partial}{\partial t}\boldsymbol{v}(t) \end{aligned} \tag{8.9}$$

从上式可以看出, 动量算法需要解微分方程, 可以借助 *Euler 方法* (Euler's method)[2]实现.

动量算法中的力可以视为两个力的合力, 一个力正比于损失函数负梯度方向 $-\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta})$; 另一个力 (粘滞阻力) 正比于 $-\boldsymbol{v}(t)$. 当然还有其他形式的阻力, 但是最好还是用粘滞阻力的形式.

8.3.3 Nesterov Momentum

Nesterov Momentum 与标准动量不同之处在于梯度的计算方式.

$$\begin{aligned} \boldsymbol{v} &\leftarrow \alpha\boldsymbol{v} - \epsilon\nabla_{\boldsymbol{\theta}}\Big(\frac{1}{m}\sum_{i=1}^{m} L(\boldsymbol{f}(\boldsymbol{x}^{(i)};\boldsymbol{\theta}+\alpha\boldsymbol{v}),\boldsymbol{y}^{(i)})\Big) \\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \boldsymbol{v} \end{aligned} \tag{8.10}$$

Nesterov Momentum 将凸优化的 exxcess error 从 $O(\frac{1}{k})$ 提升至 $O(\frac{1}{k^2})$. 但是对于非凸问题, 则没有提升.

[2]TODO: 查一下

8.4 Parameter Initialization Strategies

有些优化算法直接将目标函数优化到最优点; 另外有些优化算法通过迭代的方式将目标函数优化到最优点, 并且不需要特别选取优化起始点. 深度学习中的优化并没有上述的优势, 其优化过程是一个迭代过程, 并且优化结果很容易受到起始点的初始值影响.

选取起始点是一个很艰难的任务, 因为

1. 由于对网络结构没有一个深刻的认识, 改进初始化策略很难;
2. 不知道哪些性质对于深度学习优化是真正有效果的;
3. 起始点对于泛化能力也有影响, 但是并没有一个严格的选取准则.

现在比较明确的一个规则是打破隐层神经元之间的对称性, 使得隐层神经元可以学习到更多的上下文信息. 显式地去搜索隐层神经元所代表的函数代价太高, 因此转而去搜索隐层神经元的权重向量, 使其正交.

权重初始分布范围对优化过程和模型的泛化能力都会产生影响. 对于优化过程来说, 如果出事权重值较大

- 有助于打破对称性;
- 避免冗余神经元;
- 避免前向或者后向传播时信号丢失.

但是权重也不能过大, 比如在 RNN 中, 权重过大将会使得模型对于输入扰动异常敏感 (chaos). 但是从泛化能力的角度来看, 大权重将会使得迭代过程中权重改变量较小, 最终优化结果离起始点更近. 一般情况下, 带有早停策略的梯度下降法并不等同于权重下降, 但是它提供了一种对于初始值选取的简单模拟.

选取权重范围的一个简单策略是从分布 $U(-\frac{1}{\sqrt{m}}, \frac{1}{\sqrt{m}})$ 随机采样, 其中 m 是网络输入个数,n 是网络输出的个数. 此外还推荐使用 *normalized initialization*

$$W_{i,j} \sim U\Big(-\sqrt{\frac{6}{m+m}}, \sqrt{\frac{6}{m+m}}\Big) \tag{8.11}$$

上述初始化方法在每层初始激活函数方差相同和所有层初始梯度方差相同之间做了折中.[3]

为神经网络中每一层的非线性函数选择合适的增益因子 (scaling or gain factor g), 可以保证收敛性与层数独立. 在前向神经网络中, 激活函数或者梯度的增大或者缩小是一个随机行走行为. 如果保持每一层权重的范数, 就可以避免梯度消失或者爆炸. 但是这些最优的权重初始化准则并不能保证最优的优化效果, 原因有

- 我们可能使用了错误的准则, 即所选取的准则于神经网络的本质并不匹配;
- 选取的准则中所使用的特性在优化过程中并没有得到保持;
- 选取的准则可能对于优化问题适用, 但是对于降低泛化误差并不适用.

所有初始权重拥有相同标准差也存在缺陷, 当网络层数较多时, 每一个权重会比较小. 适用 *sparse initialization* 可以避免这一问题.

在计算资源允许时, 将每一层权重的范围设成一个超参数进行搜索, 是一个很好的方案. 此外, 根据单个 minibatch 数据选择权重范围, 也是一个很好地方案, 它不会影响泛化误差.

除了权重参数外, 还需要关注 bias 的初始化. 通常 biase 的初始化策略与权重的初始化相匹配. 大多数情况下将其初始化为 0, 除此之外, 还有

- 对于输出层的神经元, 将 biase 初始化为统计量的边缘分布;
- 选取 bias 的初始值时, 还需要避免产生 saturation;
- 有时某个神经元会控制其他神经元是否参与到方程中进行计算, 因此将大部分 bias 设置为 $h \approx 1$.

此外还有方差或者精度的初始化. 一般可以将其设置为 1.

除了上述方法, 还可以使用机器学习的方法初始化参数; 使用相关领域内的任务初始化参数 (不相关领域的也行).

[3]TODO: 没明白, 看论文怎么说.

8.5 Algorithms with Adaptive Learning Rates

学习率 (learning rate) 是一个重要的超参数, 因为它对于模型的表现有显著影响. 动量算法避免了学习率超参数设置问题, 但是它引入了另外一个超参数. 如果我们认为不同坐标轴方向上的敏感程度不同, 因此就有必要在每个坐标轴方向上设置一个不同的学习率超参数.

delta-bar-delta 算法 (delta-bar-delta algorithm) 是一种早期的学习率选取策略, 它根据损失函数的偏导符号对学习率进行选取. 但它是 full batched 优化算法.

8.5.1 AdaGrad

AdaGrad 对学习率参数做缩放, 正比于历史值平方和平方根的倒数.AdaGrad 对于某些深度学习模型比较有效, 但不是全部模型都有效.

8.5.2 RMSProp

RMSProp 算法对 AdaGrad 算法在非凸问题情形下进行了改进. 它在累积平方和中加入了权重. 目前 RMSProp 算法是深度学习算法中一个标准选项.

8.5.3 Adam

Adam 算法可以视为 RMSProp 算法和动量算法的结合. 最简单的结合方法是在 RMSProp 中梯度缩放环节中加入动量算法.Adam 还在一阶动量估计和二阶动量初始化中加入了 biase 修正.

8.5.4 Choosing the Right Optimization Algorithm

选择哪种类型的学习率优化算法并没有共论, 这个与使用者的偏好有关.

8.6 Approximate Second-Order Methods

本章的目标函数是经验风险, 实际中还有其他形式的目标函数.

8.6.1 Newton's Method

二阶方法使用二阶导数改善优化效果[4], 最广泛使用的是牛顿法. 对 $J(\boldsymbol{\theta})$ 在 $\boldsymbol{\theta}_0$ 附近进行二阶展开, 有

$$J(\boldsymbol{\theta}) \approx J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \boldsymbol{H}(\boldsymbol{\theta} - \boldsymbol{\theta}_0) \tag{8.12}$$

其中 $\boldsymbol{H}$ 是 J 对 $\boldsymbol{\theta}$ 在 $\boldsymbol{\theta}_0$ 点附近的 Hessian 矩阵. 如果求取 critical point, 有

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) \tag{8.13}$$

因此对于二次函数, 通过缩放 $\boldsymbol{H}^{-1}$ 可以直接到达最小值点. 对于凸优化非二次问题, 在 Hessian 矩阵正定的情况下, 需要通过迭代算法获取最优解.

深度学习中的 Hessian 矩阵并不保证正定, 牛顿法的优化方向可能会产生错误, 因此需要添加正则化项

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - [H(f(\boldsymbol{\theta}_0)) + \alpha \boldsymbol{I}]^{-1} \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_0) \tag{8.14}$$

正则化后的牛顿法有两个缺陷:

- 为保证正定,α 可能会很大, 这样步长就会减小;
- 需要计算 Hessian 矩阵的逆矩阵, 计算代价大.

8.6.2 Conjugate Gradients

共轭梯度 (conjugate gradient) 通过迭代选取共轭梯度方向来避免对 Hessian 矩阵求逆. 在共轭梯度算法中, 我们每一次选取的搜索方向都是与上一次搜索方向共轭的方向. 搜索方向 $\boldsymbol{d}_t$ 的迭代形式为

$$\boldsymbol{d}_t = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) + \beta_{t-1} \boldsymbol{d}_{t-1} \tag{8.15}$$

β_t 的选取方法有

[4]TODO: 建立起优化方法的简单知识体系.

Fletcher-Reeves

$$\beta_t = \frac{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)^T \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)}{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1})^T \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1})} \tag{8.16}$$

Polak-Ribiere

$$\beta_t = \frac{(\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t) - \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1}))^T \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)}{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1})^T \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1})} \tag{8.17}$$

共轭梯度法经过修改可用于非线性的情形.*nonlinear conjugate gradients* 算法就是应用于非线性情形的算法.

8.6.3 BFGS

Broyden-Fletcher-Goldfarb-Shanno (BGFS) 算法的目的是为了减少牛顿法中的计算负担. 它通过迭代法估计式8.13中的 $\boldsymbol{H}^{-1}$, 得到估计量 $\boldsymbol{M}_t$. 并且通过 $\boldsymbol{M}_t$ 决定优化方向 $\boldsymbol{\rho}_t = \boldsymbol{M}_t \boldsymbol{g}_t$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \epsilon^* \boldsymbol{\rho}_t \tag{8.18}$$

BFGS 算法的优势在于通过线性搜索减少线性搜索的时间. 缺点在于 Hessian 逆矩阵的存储, 需要 $O(n^2)$ 的存储空间.

Limited Memory BFGS 或 L-BFGS 改进了 BFGS 的存储限制. 它添加了一个假设为 $\boldsymbol{M}^{(t-1)}$ 初始值为单位矩阵, 并且在迭代过程中仅存储用来更新 $\boldsymbol{M}$ 的向量.

8.7 Optimization Strategies and Meta-Algorithms

8.7.1 Batch Normalization

批归一化 (batch normalizai)[5]是一种自适应再参数化 (reparamterization) 方法, 主要用来解决深度模型中遇到的困难.

梯度算法有一个前提就是除了当前层, 其他所有层的参数都不会改变. 但是实际上, 所有层的参数都是同步更新的. 某一层参数的更新会影响其它

[5]TODO: 看相关论文, 深化理解.

所有层的更新. 二阶优化将二阶交互考虑进优化算法中, 但是计算代价太大. 批归一化提供了一种再参数化深度神经网络的方法.

假设 $\boldsymbol{H}$ 是将要被归一化的网络层的最小批激活函数 (以设计矩阵形式排列), 为了归一化 $\boldsymbol{H}$, 将其替换为

$$\boldsymbol{H}' = \frac{\boldsymbol{H} - \boldsymbol{\mu}}{\boldsymbol{\sigma}} \tag{8.19}$$

其中,$\boldsymbol{\mu}$ 是神经元的均值向量;$\boldsymbol{\sigma}$ 是每一个神经元对应的标准差. 训练过程中, 当进行逆向传播时, 进行归一化, 避免梯度增大标准差或者均值. 在测试时, 使用训练得到的参数 $\boldsymbol{\mu}$,$\boldsymbol{\sigma}$.

归一化去除了一阶和二阶统计量的影响, 但是允许神经元间和非线性统计量的更新.

归一化会降低神经网络的表示能力, 因此通常将 $\boldsymbol{H}'$ 替换为 $\gamma\boldsymbol{H}' + \beta$. 其中 γ 和 β 通过学习得到. 与原始的 $\boldsymbol{H}$ 表示不同, 新的表示形势下, 均值仅由参数 β 决定.

8.7.2 Coordinate Descent

coordinate descent 和 *block coordinate descent* 方法分别是指一次优化一个或者一组变量, 使目标函数收敛. 这两种方法在不同变量相互孤立或者某些变量优化效率较高的情况下才会特别有效. 在变量间有相互影响的情况下, 并不是一个特别好的解决方案.

8.7.3 Polyak Averaging

Polyak Averaging 是指对优化路径上的点取平均保证算法的收敛性. 非凸问题的优化路径很复杂,Polyak 平均的形式为

$$\hat{\boldsymbol{\theta}}^{(t)} = \alpha\hat{\boldsymbol{\theta}}^{(t-1)} + (1-\alpha)\boldsymbol{\theta}^{(t)} \tag{8.20}$$

8.7.4 Supervised Pretraining

预训练 (pretraining) 是指在用以解决复杂任务的模型之前, 首先训练简化的模型用以解决简单的任务.

贪心 (greedy) 算法不能保证组合问题的最优解, 但是它的计算效率很高. 通过贪心算法解决了组合优化问题后, 还可以进行 fine-tunning.

将贪心算法和预训练结合起来就得到了贪心有监督预训练 (greedy supervised pretraining) 方法.

原始的贪心有监督预训练对神经网络进行逐层预训练. 它之所以有效的原因是因为它为中间层提供了一种有效的指导方式. 一般来说, 预训练可以帮助优化问题收敛和提升泛化能力. 此外, 迁移学习也与与训练有关联.

FitNets 方法是另一种预训练方法. 它使用小型的表现良好的网络指导更加复杂的深层神经网络进行训练. 不仅预测最终结果, 还对中间结果进行预测.

8.7.5 Designing Models to Aid Optimization

本节主要从模型设计的角度去简化优化算法. 在实际中, 选择函数族比使用优化算法更重要.

当前的神经网络普遍使用线性变换和激活函数进行组合, 组合后的函数几乎处处可导并且有较大的斜率.

另外神经元之间的跳跃连接也可以简化优化算法.

8.7.6 Continuation Methods and Curriculum Learning

通常我们希望将初始值选在最优解附近.*continuation methods* 是一种策略族, 可以确保参数值初始值选取位于目标点附近, 以简化优化过程.

传统的 continuation methods 算法基于平滑目标函数的方法, 在保留全局最小值的情况下, 尽可能去除局部极小值. 这样, 一些非凸优化问题可以近似为凸优化问题. 尽管在深度学习中, 局部极小值点不是优化的主要困难, 但是 continuation methods 仍然是一个帮助提升深度神经网络的主要方法.

curriculum learning 或者 *shaping* 方法可以视为一种 continuation method. 它的基本思路是在学习简单概念的基础上不断迭代, 进而学习更加复杂的概念, 最终达到所期望的目标概念. 在 RNN 中使用的 *stochastic curriculum* 将简单的概念和复杂概念进行随机混合用来训练模型, 但是在每一轮迭代中, 简单概念所占比例逐步降低. 与 *deterministic curriculum* 策略相

比,stochastic curriculum 的表现要好得多.

Chapter 9 Convolutional Networks

卷积神经网络 (convolutional networks) 是一种专门用来处理 grid-like 拓扑关系数据的神经网络. 神经网络中至少有一层神经元使用卷积代替矩阵乘法操作.

9.1 The Convolution Operation

最一般形式的卷积的形式为

$$s(t) = \int x(a)w(t-a)da \tag{9.1}$$

其中,$s(t)$ 被称为 *feature map*;$x(t)$ 被称为 *input*;$w(t)$ 被称为 *kernel*.

在深度神经网络中, 常使用离散形式

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{9.2}$$

在实际中大多时候是在有限区间上求和.

对于二维空间上的卷积, 有

$$\begin{aligned} S(i,j) = (I * K)(i,j) &= \sum_m \sum_n I(m,n)K(i-m, j-n) \\ &= \sum_m \sum_n I(i-m, j-n)K(m,n) \end{aligned} \tag{9.3}$$

可见其具有可交换性 (commutative), 这也是为什么进行卷积运算时 kernel 进行翻转的原因.kernel 翻转在数学证明上有很大的便利, 但是对于机器学习来说并没有太多的含义. 但是在实际中, 常根据可交换性, 对输入的数据进行翻转而保持 kernel 不变.

同时需要注意与 *cross-correlation* 的区别

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{9.4}$$

卷积运算也可以看做是与某个特定矩阵的乘法运算, 但是需要注意的是这个特定矩阵是有限制的

- 矩阵中的某些项与其他项相等 (如 *Toeplitz matrix*, *doubly block circulant matrix*);
- 矩阵是稀疏的, 因为与输入图像相比,kernel 尺寸是非常小的.

9.2 Motivation

在神经网络中引入卷积就引入了*稀疏交互* (sparse interations), *参数共享* (parameter sharing) 和*同变性* (equivariant representations). 同时, 卷积也给神经网络提供了处理可变长输入数据的能力.

稀疏交互 稀疏交互的原因是因为 kernel 的尺寸小于输入的尺寸. 在神经网络中, 稀疏交互相当于**间接地**与更多的输入进行交互, 这样有助于对概念进行抽象.

参数共享 参数共享是指在模型中有多于一个的函数使用相同的参数. 在卷积神经网络中,kernel 在输入的各个位置上共享. 参数共享虽然不能减少计算代价, 但是可以降低存储代价.

同变性 参数共享直接导致了对于变换的同变性. 如果输入发生变化, 那么输出就会以相同的形式进行变化. 需要注意的是, 卷积并非对于所有的变换都具有同变性, 它仍然需要借助于其他的机制去处理不具有同变性的变换.

9.3 Pooling

一层典型的卷积神经网络可以分为三个处理阶段:

- 卷积计算
- detector stage

- pooling function

其中 pooling 用以统计特定区间内的统计量.Pooling 有不同的类型[1]

Pooling 对于输入中的微小变换具有近似不变性. 本质上来说,Pooling 相当于在学习的函数中加入非常强的先验. 在实际中, 可以在同一个位置 pooling 不同的卷积结果, 这样就可以学习到对于变换具有不变性的特征.

Pooling 相当于降维以提高计算效率, 同时降低存储开销.

Pooling 也是用于处理可变长输入的手段.

Pooling 的选取策略有

- 不同区域使用不同类型的 pooling
- 通过学习的方式, 在所有区域使用相同类型的 pooling

Pooling 使用 top-down 信息使得网络结构趋于复杂化.

9.4 Convolution and Pooling as Infinitely Strong Prior

先验可以分为强先验和弱先验两类. 弱先验具有较高的熵, 强先验具有较低的熵.

卷积神经网络相当于给全连接神经网络加入了极强先验. 如果把卷积神经网络当做具有极强先验的全连接神经网络, 并在此基础上进行编程实现, 将会带来巨大的计算量. 但是可以借助于这种观点对卷积神经网络进行数学分析

- 卷积神经网络和 pooling 容易导致欠拟合;
- 从学习效果统计量的角度来看, 只能用卷积神经网络模型去与其他的卷积神经网络模型作对比.

[1]如何选取不同的类型? 各种不同类型 pooling 的适用范围是什么?

9.5 Variants of the Basic Convolution Function

数学领域内的卷积与神经网络领域内的卷积有一些不同.

神经网络中单层神经网络会使用不同类型的 kernel 提取出不同类型的特征, 神经网络的输入有可能是多维的 (有多个 channel). 多维卷积只有在输入和输出 channel 相等的情况下才是可交换的 (因为 channel 之间的元素也可能存在线性操作).

可以略过 kernel 中某些位置计算卷积, 以达到降采样的目的.

为了保证输出的尺寸, 还需要在输入数据的边缘进行补零操作:

- valid convolution 不使用 zero-padding;
- same convolution 加入 zero-padding 保证输入和输出的尺寸相同;
- full convolution 加入 zero-padding 保证输入数据的每个位置都被访问 k 次 (k 是 kernel 中的元素个数).

对于 full convolution, 不能只用一种类型的 kernel 学习特征. 最优的卷积通常介于 valid convolution 与 same convolution 之间.

局部连接层 (locally connected layer) 在结构上 (邻接矩阵 adjacency matrix) 上与卷及神经网络结构相同, 但是它没有参数共享, 有时它也被称为 *unshared convolution*. 也可以对卷积层或者局部连接层加上严格的限制, 以降低存储以及统计的消耗.*tiled convolution* 相当于在在这两者之间做了一个折中, 卷积的类型在空间上有周期性的规律.

$$Z_{i,j,k} = \sum_{l,m,n} V_{i,j+m-1,k+n-1} K_{i,l,m,n,j\%t+1,k\%t+1} \tag{9.5}$$

如果 detector 单元有不同类型的输入, 并且 max-pooled 学习到了不同类型的输入并且学习到了不变性特征, 那么整个卷积层也就有相应的不变性性质.

在前文中提到了, 可以使用输入与单个矩阵相乘的观点看待卷积操作. 在误差逆传播过程中, 相当于乘以上述单个矩阵的转置的操作[2]. 卷积, 从输

[2]TODO: 补充一下

出到权重的逆传播, 从输入到输入的逆传播这三种操作满足任何类型的前向卷积神经网络的训练过程.

通常还会在非线性变换之前加入 bias,bias 共享参数的策略有

- 局部连接矩阵不共享参数;
- tiled 卷积神经网络相同的 tiling pattern 共享参数;
- 卷及神经网络共享相同的参数;
- 如果输入是已知的并且尺寸大小固定, 则可以通过学习得到不同的参数. 虽然这样降低了统计效率, 但是可以修正图像在不同位置的统计偏差.

9.6 Structured Outputs

卷积神经网络可以输出结构化的目标数据 (张量). 但是其中会存在一个问题, 输出平面的尺寸会比输入平面的尺寸小. 有三种方法可以解决这一问题

- 避免 pooling 操作;
- 输出低分辨率标签网格;
- 在单位 stride 上执行 pooling 操作.

此外还有一种方法就是通过迭代的方法输出 pixel-wise 标记图像.

需要注意, 结构化输出的前提是, 大量连续的像素族总是倾向于有相同的标签.

9.7 Data Types

卷积神经网络的一大优势是它们也可以处理输入在空间范围内变化的数据. 卷及神经网络会根据输入图像的尺寸做卷积, 并对卷积结果 (feature map) 所尺寸缩放. 需要注意, 只有可变尺寸的输入数据中包含有同样的 pattern 时, 上述处理方式才会生效.

9.8 Efficient Convolution Algorithms

一般可以通过选择合适的卷计算法来加速训练过程.

卷积等价于将信号和卷积核使用傅里叶变换转换到频域中做点对点相乘, 然后再经过傅里叶逆变换转换回原始空时域.

一种方法是利用可分离核 (seperable kernel) 减少计算量, 但是这种方法也存在局限性. 因为不是每次都会使用可分离核.

因此, 精确计算或者估计卷积是一个热门的研究方向. 快速计算卷积不仅有利于训练过程, 对于提升预测过程的效率也有和大帮助. 部署一个卷积神经网络所需要的资源也是值得关注的.

9.9 Random or Unsupervised Feature

特征学习环节的计算代价较高. 一个解决方案就是不使用有监督学习方法去进行训练. 在卷积神经网络中,kernel 生成方法有

- 随机初始化
- 手工设计
- 无监督学习方法

无监督学习得到的特征与最终的分类层相互独立. 可以通过改变最终的分类层, 将模型用于不同的任务.

随机 filter 在卷积神经网络上效果很好, 可以借助这个方法进行模型选择, 以较低的计算开销计算模型选择.

另一种间接的特征学习方法是不适用全部的前向与逆向传播 (类似于前向神经网络中的贪心逐层初始化). 进一步地, 在卷积神经网络中使用局部初始化进行特征学习. 但是这种方法使用较少.

9.10 The Neuroscientific Basis for Convolutional Networks

神经网络本质的设计规则是从神经科学中得到的. 其中卷积神经网络与人脑 V1 区域相对应, 卷积神经网络符合 V1 的三种特性

1. V1 神经元按照空间映射关系进行排列;
2. V1 中包含*简单神经元* (simple cells), 与卷积神经网络中的 detector units 对应;
3. V1 中包含*复杂神经元* (complex cells), 与卷积神经网络中的 pooling units 以及 cross-channel pooling strategies 相对应.

高层的脑神经对概念有反馈, 并且对于各种几何变换具有不变性. 与卷积神经网络中的最后一层特征曾比较相似的脑区域是 inferotemporal cortex(IT).

同时卷积神经网络与人类视觉系统有以下区别

1. 人眼除了被称为 *fovea* 的 patch 外的分辨率很低;
2. 人类视觉系统常常与其他感官系统结合;
3. 人类视觉系统的任务不仅仅只是物体识别;
4. 即使是简单的 V1 区域仍然深深地受到高层神经元的影响. 在神经网络模型中, 反馈被广泛应用, 但是并没有有实质性的发展;
5. 前向 IT 捕捉到的信息与卷积神经网络捕捉到的特征很相似, 但是它们的中间过程的相似程度仍然未知.

脑神经科学并没有给出如何训练卷积神经网络的方法.

通常通过 *reverse correlation* 理解单个神经元对应的函数. 对于大部分 V1 神经元细胞,reverse correlation 展示出它们与 Gabor 函数很类似.

$$w(x, y; \alpha, \beta_x, \beta_y, f, \phi, x_0, y_0, \tau) = \alpha \exp(-\beta_x x'^2 - \beta_y y'^2) \cos(f x' + y) \quad (9.6)$$

其中

$$\begin{aligned} x' &= (x - x_0)\cos(\tau) + (y - y_0)\sin(\tau) \\ y' &= -(x - x_0)\sin(\tau) + (y - y_0)\cos(\tau) \end{aligned} \tag{9.7}$$

$\alpha, \beta_x, \beta_y, f, \phi, x_0, y_0, \tau$ 是 Gabor 函数的控制参数. 式9.6中 Gaussian 乘子可以被视为门禁因子; 余弦乘子 $\cos(fx' + \phi)$ 用来控制简单细胞对 x' 方向上的亮度变化的反应.

值得注意的是, 通过机器学习得到的特征与 V1 得到的特征很接近.

9.11 Convolutional Networks and the History of Deep Learning

卷积神经网络在深度神经发展历史中具有重要意义

- 卷积神经网络具有坚实的理论基础;
- 卷积神经网络是首先被训练出的深度学习模型之一;
- 卷积神经网络具有广阔的商业应用前景.

卷积神经网络在各种图像相关竞赛中表现出众.

卷积神经网络是 BP 算法首先被成功应用的模型之一.

总体来看, 卷积神经网络被广泛用来处理具有清晰网络拓扑结构的数据 (一般是二维数据). 下一章介绍的 RNN 专门用来处理一维数据.

Chapter 10 Sequence Modeling: Recurrent and Recursive Nets

Recurrent neural networks(RNN) 是专门用来处理序列数据的神经网络. 它使用共享参数使得模型可以对不同格式 (长度) 的数据进行处理.RNN 还在一维时域做卷积.

10.1 Unfolding Computational Graphs

计算图中的 unfolding 操作是将迭代操作转换为重复结构非递归操作的过程.RNN 有多种形式, 任意一种包含递归的函数均可使用 RNN 来表示.RNN 的典型形式为

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}) \tag{10.1}$$

其中 $\boldsymbol{h}$ 是状态变量. 由此可见,RNN 将 t 时刻任务相关的历史信息经过有损压缩整合成 $\boldsymbol{h}^{(t)}$.

RNN 模型有两种图的表现形式: 第一种是使用一个包含所有成分的结点; 第二种是将前一种形式的结点进行展开.

结点展开有两个优点:

1. 不管序列的长度有多长, 学习到的模型始终有相同的输入长度;

2. 每一步都可以使用相同的转移函数 f 以及相同的参数.

其中, 共享参数可以提升模型的泛化能力.

两种形式的计算图各有用处, 并无绝对的优劣之分.

10.2 Recurrent Neural Networks

RNN 有三种形式

- 隐层单元间有 recurrent 连接, 每一步都会输出结果;
- 输出层与隐层单元间有 recurrent 连接, 每一步都会输出结果;
- 隐层单元间有 recurrent 连接, 但是只有接受了全部的输入后才会输出结果.

在 RNN 反向传播过程中, 损失函数的梯度计算代价很高, 不能并行化. 此外, 正向传播时的状态必须一直记录直到反向传播, 因此存储代价也很高. 因此, 需要考虑替代方案. 在此, 将对 unrolled graph 上进行并且有 $O(\tau)$ 复杂度的逆向传播算法称为 *back-propagation through time*(BPTT).

10.2.1 Teacher Forcing and Networks with Output Recurrence

仅仅在从前一个时间状态的输出单元到后一个时间状态的隐层单元存在连接的 RNN 网络的学习能力是非常弱的. 因为这种类型的网络缺少从隐层到隐层的周期性连接. 为了取得较好的效果, 就要求输出必须能够捕捉全面的历史信息. 由于损失函数中与时间相关的步骤没有耦合, 可以对训练过程进行并行化操作.

Teacher forcing 操作在每一步都是用上一步的正确输出标记作为当前步骤的输入. 引入 teacher forcing 的目的是为了避免 BPTT 操作. 但是, 一旦隐层单元所表示的函数是历史时间状态的函数, 就必须进行 BPTT 计算.

Teacher forcing 的劣势也很明显, 在 *open-loop* 模式下, 当网络的输出作为输入的一部分时, 输出反馈的输入在训练环节的分布可能和测试环节时的分布有很大差异. 对于这个问题, 有两种解决方案:

- 模型训练时的输入包含 teacher forcing 输入和 free-running 输入;
- 随机选择使用生成的数值或者实际数据的数值作为输入, 以尽量弥补训练时的输入和测试时的输入.

10.2.2 Computing the Gradient in a Recurrent Neural Network

RNN 中梯度的计算很直接. 需要注意的是, 由于参数共享, 所以共享参数的梯度是将所有时刻的梯度值进行相加求取.

10.2.3 Recurrent Networks as Directed Graphical Models

对于 RNN 理论上可以使用任何一种损失函数, 最常用的是交叉熵损失函数.

一般地,RNN 对应的有向图是一个完备图, 任何一对输出节点之间都会有有向依赖关系. 通常会对图中交互关系较弱的两个节点之间的边进行简化处理. 将隐层单元视为随机变量, 这样就可以与历史状态进行解耦合. 但是, 某些操作仍然会有计算代价, 比如缺失值填充和参数优化操作.

RNN 中的参数共享意味着 t 与 $t+1$ 时刻的条件分布的参数是固定的.

如果从图中进行采样,RNN 必须确定序列的长度. 具体方法有

- 在原始序列中插入特殊分隔符;
- 通过伯努利分布判定序列是否结束;
- 将长度 τ 作为预测目标进行学习预测.

10.2.4 Modeling Sequences Conditioned on Context with RNNs

RNN 对于条件概率分布有着不同的实现方式. 当随机变量 $\boldsymbol{x}$ 是一个固定长度的向量时, 有以下方式将其作为附加信息加入 RNN 模型用以产生 $\boldsymbol{y}$ 序列:

- 作为每一个时间步骤的额外输入;
- 作为一个初始状态 $\boldsymbol{h}^{(0)}$;

- 以上两种方法的结合.

添加的额外信息 $\boldsymbol{x}$ 可以视为前向神经网络中的 bias.

当有多个输入 $\boldsymbol{x}^{(t)}$ 时, 可以将 t 时刻的输出作为 $t+1$ 时刻的隐层单元的输入. 但是限制条件是, 这些序列的长度必须相同.

10.3 Bidirectional RNNs

除了上文提到的当前状态与输入序列中历史值相关的情形, 还存在着当前状态与整个输入序列有依赖关系的情形. 因此需要引入双向 RNN 来解决这一问题. 在双向 RNN 中, 当前状态对历史状态和未来的状态都存在着依赖关系.

同样地, 可以将双向 RNN 的输入拓展到二维空间, 这样就需要在四个方向上分别训练一个 RNN.

10.4 Encoder-Decoder Sequence-to-Sequence Architectures

在许多应用情形中, 输入序列和输出序列的长度不一定相同.*encoder-decoder* 专门用来处理输入输出序列长度不同的情形. 其中,encoder 用于生成 context,decoder 用于生成输出序列.context 的长度不一定和 decoder 隐层长度相同. 同时 encoder-decoder 结构还存在一个缺陷, 如果 context 太短的话就不能完整总结较长的输入序列的信息.

10.5 Deep Recurrent Networks

RNN 网络中的计算包含 3 中类型的变换

- 从输入到隐状态的变换;
- 从前一个隐状态到下一个隐状态的变换;
- 从隐状态到输入的变换.

RNN 中三种操作是一种 “浅” 变换. 为训练出任务所需要的变换, 就需要增加深度.

10.6 Recursive Neural Networks

Recursive neural networks 与 RNN 的链式结构不同, 它是一种深度树结构. 主要被用来处理数据结构.Recursive 神经网络中树的深度大大减少, 并且可以用来处理长期依赖. 数的结构可以独立于数据进行选取. 还有外部方法可以用来选择树的结构. 除此之外,recursive 还有其他的变化类型.

10.7 The Challenge of Long-Term Dependencies

长期交互相比于短期交互, 权重水平明显要低. 由于 RNN 中存在着参数共享, 所以在状态传递的过程中, 特征值会消失或者爆炸. 这种情况只存在于 RNN 网络中, 同时也是不可避免的. 由此可见, 长期依赖是难以学习的.

10.8 Echo State Networks

在 RNN 中, 从输入单元到隐层单元的权重以及隐层单元到隐层单元的权重很难学习.*Echo state networks*(ESN) 或者 *liquid state machines* 通过使用 *reservoir computing* 机制设定上述上重权重, 进而只要学习输出层的权重即可.reservoir computing 在思路上类似于核方法, 将长度不定的输入序列转换为定长向量, 然后借助于线性判别器对目标任务进行解决.

设置权重的策略为通过设置合适的权重, 将状态到状态转移函数的 Jacobian 矩阵的特征值接近于 1. 但是如果引入了非线性单元, 许多环节中的梯度会接近于 0. 这种收缩性映射会导致网络遗忘历史状态.

固定权重使得信息可以向前传播但不会爆炸.

ESN 设置权重的方法可以用来初始化 RNN 网络的参数.

Leaky Units and Other Strategies for Multiple Time Scales

为处理长期依赖, 模型要在不同时间尺度上处理状态. 常用的方法有

- 添加跳跃连接 *skip connection*
- 通过 *leaky unit* 以不同权重整合信号
- 去除精细时间尺度上的某些连接

10.8.1 Adding Skip Connections through Time

为了获取粗略时间尺度上的信息, 可以直接从过去的变量上直接添加连接到当前的变量上. 但是这种方式并不能很好地处理所有长期的依赖关系.

10.8.2 Leaky Units and a Specturm of Different Time Scales

另一种方式是在节点上增加线性自连接. 增加了线性自连接的隐层单元类似于 *running average*,

$$\mu^{(t)} \leftarrow \alpha\mu^{(t-1)} + (1-\alpha)v^{(t)} \tag{10.2}$$

其中,$v^{(t)}$ 是历史状态的累积量,$\mu^{(t-1)}$ 是当前的状态,α 是时间常数. 时间常数有两种设置方法: 第一, 固定为常量; 第二, 设置为参数, 通过学习算法进行学习.

这种处理方式方式可以使得信号更加平滑灵活.

10.8.3 Removing Connections

这种方式主要是通过去除短期的连接增加长期的连接, 进而使得信息可以流畅地进行传播.

10.9 The Long Short-Term Memory and Other Gated RNNs

在实际中使用频率比较高的 RNN 网络是门限 *RNN*(gated RNNs). 门限 RNN 会构建出导数不会消失或者爆炸的路径. 因此连接权重在每一步都会产生变化. 同时, 门限 RNN 将会选择合适对历史状态进行清零.

10.9.1 LSTM

LSTM 模型的核心在于

- 引入自循环构建梯度传播路径, 保证梯度可以维持较长时间;
- 自适应调节自循环的权重;
- 时间范围可以动态调整.

在 LSTM 单元中, 有不同的门限

- *forget gate* 单元

$$f_i^{(t)} = \sigma\left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum)_j W_{i,j}^f h_j^{(t-1)}\right) \tag{10.3}$$

- *external input gate* 单元

$$g_i^{(t)} = \sigma\left(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum)_j W_{i,j}^g h_j^{(t-1)}\right) \tag{10.4}$$

- *output gate* 单元

$$q_i^{(t)} = \sigma\left(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum)_j W_{i,j}^o h_j^{(t-1)}\right) \tag{10.5}$$

其中,$\boldsymbol{x}^{(t)}$ 是当前输入向量;$\boldsymbol{h}^{(t)}$ 是当前隐层神经元向量;$\boldsymbol{b}, \boldsymbol{U}, \boldsymbol{W}$ 分别是偏置, 输入权重, 门限的循环权重.

LSTM 更容易捕捉到长期依赖.

10.9.2 Other Gated RNNs

与 LSTM 不同的是,*gated recurrent units*(GRU) 用一个门禁同时控制遗忘单元和更新单元.LSTM 和 GRU 在多种任务上都有很好的表现, 并且超越其他变型.

10.10 Optimization for Long-Term Dependencies

在优化问题中, 二阶导数和一阶导数可能会同时消失. 但使用二阶导数优化方法会有诸多缺陷.Mesterov 动量法在精心选取初始化参数后, 通过一阶方法也有可能达到同样的效果.[1]

10.10.1 Clipping Gradients

梯度下降过快会导致步长很长, 进而可能会导致目标函数值上升. 因此引入梯度截断 (clipping the gradient).

- 一种方法是在最小批上对参数梯度进行截断 (element-wise);
- 另一种方法是在参数更新前对梯度范数进行截断 (jointly).

  当梯度的范数超过阈值时, 随机选择一步, 效果也会很好.
  但是不同最小批上的平均范数阶段梯度并不等价于真实梯度的截断.

10.10.2 Regularizing to Encourage Information Flow

解决梯度消失的方法有

- 使用 LSTM 或者其他自循环以及门限机制;
- 另一种方法是正则化或者限制参数, 以维持信息传递.

[1]机器学习中, 设计一个更容易优化的模型比设计一个性能更好的优化方法更常见.

通常我们希望,

$$(\nabla_{\boldsymbol{h}^{(t)}} L)\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} \tag{10.6}$$

与

$$\nabla_{\boldsymbol{h}^{(t)}} L \tag{10.7}$$

的大小相同. 因此, 引入正则化项

$$\Omega = \sum_t \left(\frac{\|(\nabla_{\boldsymbol{h}^{(t)}} L)\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}\|}{\|\nabla_{\boldsymbol{h}^{(t)}} L\|} - 1 \right)^2 \tag{10.8}$$

上式中的正则化项比较复杂, 还有近似的估计形式.

这种方法的缺点很明显, 在数据比较丰富的情况下, 有效性不如 LSTM.

10.11 Explicit Memory

知识可以分为显式知识和隐式知识两种类型. 人工神经网络在记忆显式知识的时候很吃力. 愿意在于其缺少工作记忆 (working memory) 系统用来存储样本. 为了解决这一问题, 记忆网络引入记忆单元, 可通过寻址机制对记忆单元进行访问. 神经网络图灵机 (neural Turning machine) 使用无监督的方式对记忆单元进行访问, 它可以同时读或者写多个记忆单元, 并且在每个记忆单元中存储的是向量. 这与人类的记忆方式比较类似.

在记忆内容被复制后, 信息可以被传递而不用关心梯度是否消失或者爆炸.

Chapter 11 Practical Methodology

在机器学习模型训练过程中, 涉及到的环节有收集更多的信息, 增大或者减少模型的 capacity, 提升模型的估计推断以及调试软件. 在不同的阶段需要做出正确的选择而非盲目猜测. 除此之外, 还应正确使用常见的算法. 实际过程可以归纳为

1. 确定任务目标;
2. 建立端到端流程;
3. 调试模型确定结果瓶颈;
4. 对模型做出增量改变.

11.1 Performance Metrics

错误度量指标决定了模型训练的后续动作. 首先需要明确的是不可能达到 0 错误率, 错误率的下限是贝叶斯错误率.

训练数据集的规模通常也受到限制.

那么基于上述两点, 如何能够预估出模型的合理表现? 与之相关联的是度量指标的选择. 最常见的度量指标是错误率, 但是许多应用要求使用更加复杂的度量指标, 如*精准率*, *召回率*, 以及基于精准率和召回率的 *F-score*.

在一些应用中, 需要模型对某些样本做出拒绝判定. 这是就要引入*覆盖率* (coverage) 的度量指标. 即模型可以做出判别的样本占全部样本的比例.

11.2 Default Baseline Models

接下来的步骤是建立端到端的系统. 在根据任务内容判定是否使用 AI 模型后, 首先根据数据集的结构选择模型类别; 其次选择合适的优化算法,

通常使用基于动量的 SGD 算法或者 Adam 算法; 最后选择正则化方式.

除了使用上述步骤, 还可以从其他类似任务建立的模型中进行借鉴.

对于是否使用无监督学习的方式, 需要根据问题领域进行细分.

11.3 Determining Whether to Gather More Data

通常会优先考虑收集更多的数据而非改进学习算法.

首先需要根据训练集错误率判断是否需要增加样本; 接着从测试集表现判断是否需要收集更多的样本; 在确定了需要收集更多样本的结论后, 最后需要确定需要收集多少样本.

如果收集样本的代价很高, 就需要改善学习算法来降低泛化错误率.

11.4 Selecting Hyperparamters

超参数有两种选择方式: 手工选择与自动选择.

11.4.1 Manual Hyperparameter Tuning

手动设定超参数需要对超参数与训练错误率, 泛化误差, 计算代价的关系有深刻的了解.

手动设定超参数的目的就在运行时间以及内存允许的前提下达到最低泛化误差. 除此之外, 手动设定超参数还有一个目的是为了使得模型的 capacity 与任务的复杂度相匹配.

一般来说, 泛化错误率高有两种原因: 第一种是因为模型欠拟合; 第二种是由于训练误差和测试误差之间的差距较大.

按照超参数调节的优先级来说, 学习率调节的优先级最高. 调节学习率以外的参数要同时监控训练集和测试集.

其他的大多数参数是根据它们增大或者减少模型的 capacity 来设定的.

正则化只是降低测试误差的一种方式, 还有其他的方式可以降低测试误差.

11.4.2 Automatic Hyperparameter Optimization Algorithm

人工设定超参数的前提是人工设定的其实范围比较接近最优解, 但是这个前提并不是一直都成立. 因此从理论上来说, 可以设计一个优化算法自动选择超参数. 但是这样就会引入新的超参数. 但是新的超参数相比于原有的超参数更加容易设定.

11.4.3 Grid Search

当超参数的数量小于或等于 3 个时, 通常使用网格搜索方法设定超参数.

超参数搜索的范围需要根据先验知识进行设定.

网格搜索有一个很明显的缺点就是其计算代价随着参数数目的增长呈指数级增长.

11.4.4 Random Search

相比于网格搜索, 随机搜索更加易于实现, 方便部署, 收敛速度快.

不同于网格搜索, 随机搜索中的超参数不用进行离散化或者二值化操作.

随机搜索效率较高的原因是其不用进行实验性的搜索测试.

11.4.5 Model-Based Hyperparameter Optimization

自动超参数优化算法需要计算梯度, 为了避免进行梯度计算, 建立验证集错误率模型, 通过这个模型评估超参数的泛化误差.

Bayesian 超参数优化方法很常见, 但并不是一种具有普遍适用性的方法.

基于模型的超参数优化方法需要实验性测试来提取出实验中的有效信息. 因此效率并不是特别高.

11.5 Debugging Strategies

机器学习模型很难进行调试, 同时也很难确定出现的问题是算法的原因还是软件实现的原因.

由于算法的行为很难进行预估, 这一点加剧了调试的难度. 如果模型各个子模块具有自适应性, 调试就会更加困难.

大多数调试策略避开了上述两项困难.

一些主要的调试策略包括:

- 模型动作可视化;
- 最差结果可视化;
- 通过训练错误率和测试错误率对软件进行推断;(如果训练错误率低但是测试错误率高, 那么就很有可能训练算法正常但是模型过拟合; 如果训练误差和测试误差都很高, 那么就很难判断软件有问题还是欠拟合.)
- 在小规模数据集上进行测试;
- 将逆传播导数与数值导数进行比对, 例如将

$$f'(x) = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{\epsilon} \tag{11.1}$$

与

finite difference

$$f'(x) \approx \frac{f(x+\epsilon) - f(x)}{\epsilon} \tag{11.2}$$

centered difference

$$f'(x) \approx \frac{f(x+\frac{1}{2}\epsilon) - f(x-\frac{1}{2}\epsilon)}{\epsilon} \tag{11.3}$$

进行对比. 此外还有对向量以及对复数的比对.

- 监控激活函数和梯度的直方图.

深度学习的算法还提供了一些对结果的保证.

11.6 Example: Multi-Digit Number Recognition

项目开始的第一步就是选择度量指标, 并设定指标的期望值. 通常情况下是根据商业目标设定度量准则的. 在本例中, 为了提高精度牺牲了一部分覆盖率.

第二步是建立算法基线.

第三步是对模型进行迭代完善. 包括: 根据覆盖率度量指标和数据的结构对模型进行完善; 根据模型在训练集和测试集上的表现判断模型处于过拟合还是欠拟合状态; 可视化置信度最高的错误模型, 对模型进行调试.

最后一步是调整超参数提升最终的效果.

Chapter 12 Applications

12.1 Large-Scale Deep Learning

但个神经元并不能体现出智能的特性, 只有大量神经元组合在一起才能体现出智能的特性.

12.1.1 Fast CPU Implementations

精心调试的 CPU 会有较高的性能, 并且有多种策略可供选择, 对运行在 CPU 上的程序进行优化.

12.1.2 GPU Implementations

绝大部分现代神经网络是在 GPU 上实现的.GPU 具有高度并行性和较大内存带宽, 但是相比于 CPU, 它的时钟频率较低, 并且分支能力较弱. 人工神经网络有同样的特性, 因此适合运行在 GPU 上. 基于此,GPU 厂商开发出了*通用 GPU*(general purpose GPUs) 用以处理 GPU 编程代码, 而不仅仅是图形渲染.

但是 GPU 编程比较困难, 大部分工程人员使用现成的代码进行组合以实现自己的模型, 尽量避免编写底层代码.

12.1.3 Large-Scale Distributed Implementations

由于单个机器的计算能力有限, 因此我们希望可以将训练过程和推断过程转化成分布式计算.

推断过程的分布式计算很简单, 因为每个机器可以分别运行模型的不同部分, 处理同一个样本, 这又被称为*数据并行* (data parallelism).

多个机器在单个数据点上进行协同计算的过程被称为模型并行 (model parallelism).

异步 *SGD*(asynchronous stochastic gradient descent) 可以解决数据并行问题.

12.1.4 Model Compression

模型推断时对内存资源的要求更高. 通常使用模型压缩 (model compression) 来获取一个对内存需求更少, 运行时间更短的小模型来进行存储和执行.

模型压缩可行的前提是原始模型为防止过拟合将不同的模型进行集成, 组成了大模型.

12.1.5 Dynamic Structure

对数据处理系统进行加速的一项通用策略是设计带有在图上具有动态结构 (dynamic structure) 的模型去处理输入数据. 在神经网络中根据输入使用不同的子网络进行计算的结构被称为条件计算 (conditional computation). 动态结构是从计算机理论到软件工程领域被广泛使用的一项基本原则.

一种动态结构是层级结构分类器, 用以提升计算效率. 决策树本身就是典型的动态结构. 基于同样的思路,*gater* 根据当前输入选择不同的专家网络进行计算. 当专家网络数目较少时, 使用 *hard mixture of experts* 进行计算. 但是当专家网络数目较多时, 组合数目也增多, 就需要借助额外的手段去训练专家网络的组合了.

另外一种动态结构是开关形式, 隐层神经元根据上下文从不同的输入单元获取输入.

需要注意的是, 动态结构大大降低了神经网络的并行化操作, 往往很难高效地进行编程实现.

12.1.6 Specialized Hardware Implementations of Deep Networks

神经网络有不同的硬件实现方式:

- ASICs: application-specific integrated circuit
- digital: based on binary representations of numbers
- analog: based on physical implementations of continuous values as voltages or currents
- hybird implementations: field programmable gated array

深度神经网络专用硬件得以发展的原因有两点: 第一, 可以在推断时候使用较低的精度; 第二, 通用处理器 (CPU/GPU) 计算速率增长放缓, 需要通过提升并行特性来提升整体性能.

12.2 Computer Vision

12.2.1 Preprocessing

计算机视觉往往需要很少的预处理步骤. 最常见的有像素值归一化和输入图像尺寸归一化. 对于训练数据集的预处理有 *dataset augmentation*. 同时对于训练数据集以及测试数据集进行的预处理主要是为了排除数据扰动对模型的影响. 通常需要对于输入数据扰动有明确的描述, 并且需要确认去除扰动后对模型训练不会产生影响.

Contrast Normalization

通常情况下, 绝大多数任务中可以去除的样本间差异就是对比度差异. 设图像为 $X \in \mathcal{R}^{r\times c\times 3}$, 其中 $X_{i,j,1}$ 代表红色通道中第 i 行第 j 列所对应的像素亮度. 整幅图像的对比度为

$$\sqrt{\frac{1}{3rc}\sum_{i=1}^{r}\sum_{j=1}^{c}\sum_{k=1}^{3}(X_{i,j,k}-\overline{X})^2} \tag{12.1}$$

其中,$\overline{X}$ 是整幅图像的亮度均值

$$\overline{X} = \frac{1}{3rc}\sum_{i=1}^{r}\sum_{j=1}^{c}\sum_{k=1}^{3}X_{i,j,k} \tag{12.2}$$

全局对比度归一化 (Global contrast normalization, GCN) 旨在防止图像间的对比度有较大差异. 常将图像的减去均值亮度, 并将亮度分布归一化. 即

$$X'_{i,j,k} = s\frac{X_{i,j,k} - \overline{X}}{\max\left\{\epsilon, \sqrt{\lambda + \frac{1}{3rc}\sum_{i=1}^{r}\sum_{j=1}^{c}\sum_{k=1}^{3}(X_{i,j,k} - \overline{X})^2}\right\}} \tag{12.3}$$

使用 L^2 范数而非标准差归一化的原因在于, 神经网络对于方向的响应要比对于准确位置的响应要好.

需要注意的是, *球化* (sphering) 与 GCN 并不相同. 球化是指将主成分的方差进行缩放达到一致. 球化通常又被称为*白化* (whitening).

全局对比度归一化并不能对图像中局部细节进行突出显示, 由此催生出了*局部对比度归一化* (local contrast normalization, LCN). 局部对比度归一化可以通过可分离式卷积进行快速实现. 此外,LCN 也可作为神经网络的非线性单元.

在实际使用中, 需要特别注意 LCN 的正则化策略.

Dataset Augmentation

除了之前介绍的 augmentation 方法, 数据集 augumentation 有更加高级的变换方式.

12.3 Speech Recognition

语音识别的目的是将包含语言信息的声音信号 $\boldsymbol{X}$ 转换成文字序列 $\boldsymbol{y}$. *自动语音识别* (automatic speech recognition) 的模型可以概括为

$$f^*_{ASR}(\boldsymbol{X}) = \arg\max_{\boldsymbol{x}} P^*(\boldsymbol{y}|\mathbf{X} = \boldsymbol{X}) \tag{12.4}$$

传统方法使用 HMM 与 GMM 结合的策略进行语音识别. 随着深度模型的发展, 许多方法借助于深度神经网络提取特征, 并用提取出的特征训练

HMM 和 GMM 模型. 之后深度模型继续发展, 并打破了 HMM-GMM 模型十多年没有提升的僵局.

语音识别领域内的一项突破是使用卷积神经网络进行语音识别. 它以二维的视角看待声音信号.

另外一项突破就是端到端的深度神经网络直接摒弃了 HMM 模型的使用. 其中, 声音世界的音素级别的对齐也是一个重要环节.

12.4 Natural Language Processing

自然语言处理 (natural language processing) 是使用计算机对人类语言进行处理的技术. 除了神经网络领域内的通用技术外, 还需要引入专业领域的知识.

12.4.1 n-grams

语言模型 (language model) 是指标记 (token) 序列的概率分布. 标记序列定长为 n 的模型被称为 n-gram.

$$P(x_1, x_2, \cdots, x_\tau) = P(x_1, \cdots, x_{n-1}) \prod_{t=n}^{\tau} P(x_t | x_{t-n+1}, \cdots, x_{t-1}) \tag{12.5}$$

其中, 根据 n 取值不同,n-gram 可以分为:$unigram(n = 1)$,$bigram(n = 2)$ 和 $bigram(n = 3)$.

通常使用最大似然度对 n-gram 进行直接训练. 并且训练过程中可以对 $n-1$-gram 一起进行训练.

在句子起始位置的分布通常使用边缘概率分布.

对于条件概率为 0 的情况, 需要引入平滑 (smoothing) 策略. 一种方法是对所有的概率分布加入非零常量. 另一种策略是 *back-off* 方法, 使用低阶 n-gram 模型代替高阶模型.

对于维度灾难问题, 使用最近邻查找的方法进行缓解, 但不能从根本上克服这一问题. 由此引入基于类别的语言模型 (class-based language model) 根据词汇类别共享词汇间的统计信息. 但是这种处理方式在信息表示过程中有信息损失.

12.4.2 Neural Language Models

网络语言模型 (Neural Language Models, NLM) 专门用来解决语言模型建模过程中的维度灾难问题. 它根据词汇上下文对统计信息进行共享. 在相似的句子间建立起关联.

词汇表示又被称为词汇嵌入 (word embeddings).NLM 模型更加注重词汇嵌入这种分布式表示方法.

其他模型也可以使用分布式表示提升模型的性能.

12.4.3 High-Dimensional Outputs

基于词汇的概率分布计算代价很大. 最朴素的做法是从隐单元到输入空间建立起放射变换, 再用 sotmax 进行概率估计.

Use of a Short List

为减少朴素做法的计算代价, 引入有限词汇集, 并进一步区分为高频词汇表与低频词汇表. 但是泛化能力受到高频词汇的限制.

Hierarchical Softmax

另一种减少计算代价的方法是对词汇进行逐级分类, 最终形成一棵树. 每个单词的概率可以通过到达该词汇所有路径的概率总和得到.

层级结构可以通过机器学习的方法得到.

在训练和测试过程中都会有计算效率高的优势.

同时, 这种方法不能解决在给定上下文的时候, 相似单词选取的问题.

相比于基于采样的方法 (sampling-based method), 这种方法在实际中的效果并不是很好.

Importance Sampling

一种加速 NLM 训练过程的策略就是避免计算所有词汇出现在下一个位置的概率. 但是可以通过对词汇数据集进行采样, 计算子集中的每个词汇

的概率.

$$\begin{aligned}\frac{\partial \log P(y|C)}{\partial \boldsymbol{\theta}} &= \frac{\partial \log \operatorname{softmax}_y(\boldsymbol{a})}{\partial \boldsymbol{\theta}} \\ &= \frac{\partial a_y}{\partial \boldsymbol{\theta}} - \sum_i P(y=i|C)\frac{\partial a_i}{\partial \boldsymbol{\theta}}\end{aligned} \tag{12.6}$$

上式中后一项需要从模型本身进行采样, 仍然需要计算所有单词的概率. 因此, 一般做法是 *importance sampling* 从另外一个分布中进行采样, 并进行修正. 针对上述情形, 提出了 *biased importance sampling*,

$$w_i = \frac{p_{n_i}/q_{n_i}}{\sum_{j=1}^{N} p_{n_j}/q_{n_j}} \tag{12.7}$$

因此有

$$\sum_{i=1}^{|\mathbb{V}|} P(i|C)\frac{\partial a_i}{\partial \boldsymbol{\theta}} \approx \frac{1}{m}\sum_{i=1}^{m} w_i \frac{\partial a_{n_i}}{\partial \boldsymbol{\theta}} \tag{12.8}$$

更一般地, 为了加速训练过程, 可以使用稀疏输出层.

Noise-Contrastive Estimation and Ranking Loss

Ranking loss 将 NLM 的输出看做每个单词作为一个分数, 并且正确的分数 a_y 排序比其他的分数 a_i 要靠前, 对应的损失函数为

$$L = \sum_i \max(0, 1 - a_y + a_i) \tag{12.9}$$

但是它不能提供条件概率的估计.

12.4.4 Combining Neural Language Models with n-grams

n-gram 模型的一大优势是模型 capacity 比较高, 但是计算量比较低. 同时 NLM 的计算量较高. 因此考虑将这两者进行结合.

12.4.5 Neural Machine Translation

机器翻译的任务是读入一种语言的句子, 并将其翻译成同等含义的另一种语言的句子. 从整体上看, 机器翻译系统的一个部分是产生候选翻译选项, 另外一个部分是对翻译候选项进行评价.

传统的语言模型仅仅输出自然语言句子的概率, 为使其能够满足机器翻译任务的需求, 需要将其拓展至条件概率的范畴.

基于 MLP 模型方法的一大缺陷是语句必须是定长. 对于这个问题, 可以借助于 RNN 模型, 形成 encoder-decoder 框架. 为了产生基于原始句子的条件概率, 这个模型必须能够对原始的句子有一个完整的表示策略.

Using an Attention Mechanism and Aligning Pieces of Data

机器翻译的一种高效做法是, 一次读入整个句子, 然后根据相关的上下文, 一次输出一个单词. 基于这种思想, 提出了 *attention* 模型, 主要包含以下几个主要组成部分

- 读入原始数据并转换成分布式表达方法;
- 前一步输出的特征向量列表;
- 依次对内存中数据进行处理并输出结果的模块.

12.4.6 Historical Perspective

略.

12.5 Other Applications

12.5.1 Recommender Systems

推荐系统主要分为两种应用情形: 在线广告和商品推荐. 推荐系统相关的任务可以归纳为有监督学习: 给出用户和商品的相关信息, 对用户的兴趣指标进行预测, 最终具化为一个回归问题或者分类问题.

常用的一种方法是协同过滤 (collaborative filtering). 可以通过有参数方式和无参数方式分别实现. 双线性预测 (bilinear prediction) 实有参数实现方式之一. 在具体应用中, 双线性预测知识集成学习方法的学习器之一.

除了双线性方法, 基于网络实现的协同过滤方法是 RBM 无向概率模型.

协同过滤方法有一个明显的缺陷就是其冷启动问题. 即对于新加入的用户或者商品, 算法很容易失效. 解决这个问题的方法是引入商品和用户的额外信息.

Exploration Versus Exploitation

一方面, 推荐系统存在一个问题就是获取到的用户喜好有偏差. 对于未给用户进行推荐的商品是无法预测出用户的反应的. 另一方面, 强化学习可以对任何动作和反馈进行学习, 并且强化学习在 exploration 和 exploitation 之间取得平衡. 决定 exploration 于 exploitation 偏好的因素之一是时间尺度.

另外存在一个困难就是如何对比不同的强化学习策略.

12.5.2 Knowledge Representation, Reasoning and Question Answering

Knowledge, Relations and Question Answering

在知识表达领域, 一个有意思的研究方向就是如何训练出一种表达方式, 使得分布式表达能够捕捉到不同实体间的关系.

在数学领域, 二维关系 (binary relation) 是描述有序元素对之间关系的集合.

在 AI 领域, 关系指的是由简单句法构成的并且高度结构化语言描述的句子.

为了在神经网络上表示关系并进行推理, 首先需要建立训练数据集. 通常使用 *relational database* 和 *knowledge base*; 此外, 还需要选择合适的模型族.

知识表达的一种实际应用是 *link prediction*, 但是它的结果很难量化评价. 另一种实际应用是 *word-sense disambiguation*.

Chapter 13 Linear Factor Models

许多深度学习学者对输入建立起概率模型 $p_{\text{model}}(\boldsymbol{x})$, 借助于该模型使用概率推断 (probabilistic inference)[1]可以利用环境中的变量去预测环境中的其他变量. 这其中许多模型包含隐变量 $\boldsymbol{h}$, 即 $p_{\text{model}}(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{h}} p_{\text{model}}(\boldsymbol{x}|\boldsymbol{h})$. 包含隐变量的分布式表达可以囊括深度前向网络和 RNN 的所有优势.

包含隐变量的最简单模型是线性因子模型 (linear factor model). 它可以用来组成其他更加复杂的模型. 它的数学表达式为

$$\boldsymbol{x} = \boldsymbol{W}\boldsymbol{h} + \boldsymbol{b} + \text{noise} \tag{13.1}$$

其中,$\boldsymbol{h} \sim p(\boldsymbol{h})$, 并且其各个分量间相互独立, 有 $p(\boldsymbol{h}) = \prod_i p(h_i)$; 噪声项通常服从各个分量间相互独立的 Gaussian 分布.

13.1 Probabilistic PCA and Factor Analysis

各种线性因子模型在本质上都是一样的, 都可以用式13.1进行概括. 不同之处在于, 噪声分布的选择以及在观察 $\boldsymbol{x}$ 之前对模型隐变量 $\boldsymbol{h}$ 的先验分布的选择.

Factor Analysis Factor analysis 隐变量服从单位方差 Gaussian 分布

$$\boldsymbol{h} \sim \mathcal{N}(\boldsymbol{h}; \boldsymbol{0}, \boldsymbol{I}) \tag{13.2}$$

[1]Probabilistic inference is the task of deriving the probability of one or more random variables taking a specific value or set of values. For example, a Bernoulli (Boolean) random variable may describe the event that John has cancer. Such a variable could take a value of 1 (John has cancer) or 0 (John does not have cancer). DeepDive uses probabilistic inference to estimate the probability that the random variable takes value 1: a probability of 0.78 would mean that John is 78% likely to have cancer.

输入变量在给定 $\boldsymbol{h}$ 的情况下条件独立; 噪声分布服从对角方差 Gaussian 分布,ψ.

隐变量被用来捕捉不同输入变量间的依赖关系.

$$\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{x}; \boldsymbol{b}, \boldsymbol{W}\boldsymbol{W}^{\boldsymbol{T}} + \boldsymbol{\psi}) \tag{13.3}$$

Probabilistic PCA 为了将 PCA 引入到概率框架中, 需要把 factor analysis 中的条件方差限定为全部相同, 因此有

$$\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{x}; \boldsymbol{b}, \boldsymbol{W}\boldsymbol{W}^{\boldsymbol{T}} + \boldsymbol{\sigma}^{\boldsymbol{2}}\boldsymbol{I}) \tag{13.4}$$

大部分数据变量都可以通过隐变量 $\boldsymbol{h}$ 进行捕捉, 只存在很小的重构错误 (reconstruction error) 残差 σ^2.

13.2 Independent Component Analysis (ICA)

ICA 建立线性因子模型将原始信号分解为独立成分. 它有许多不同的形式, 但都要求使用者预先设定好 $p(\boldsymbol{h})$, 接着模型就可以利用定式生成 $\boldsymbol{x} = \boldsymbol{W}\boldsymbol{h}$.

使用者所选择的 $p(\boldsymbol{h})$ 尽可能独立, 这样就可以还原出尽可能相互独立的因子. 此外还需要注意的是,$p(\boldsymbol{h})$ 不能服从 Gaussian 分布, 否则 $\boldsymbol{W}$ 就不唯一.

ICA 仅仅能够获得 $\boldsymbol{x}$ 与 $\boldsymbol{h}$ 之间的变换关系, 它并不清除 $p(\boldsymbol{h})$ 的表示方式, 因此也不能获得基于 $p(\boldsymbol{x})$ 的分布.

ICA 也有一些拓展形式

- 非线性生成式模型: 使用非线性方程 f 生成观察到的数据;
- nonlinear independent components estimation (NICE)
- 用来学习特征群 (feature groups), 寻找群内统计独立的特征但是群间不独立的特征.

13.3 Slow Feature Analysis

SFA 通过是学信号学习不变特征. 相比于单个度量指标, 环境中的重要特征变化很缓慢. 基于慢速原则 (slowness principle) 可以在损失函数中引入

$$\lambda \sum_t L(f(\boldsymbol{x}^{(t+1)}), f(\boldsymbol{x}^{(t)})) \tag{13.5}$$

其中,$lambda$ 是超参数;f 是特征提取函数;L 是损失函数;t 指示时刻.

SFA 算法可以归纳为如下的优化问题

$$\begin{aligned} &\min_\theta \mathbb{E}_t (f(\boldsymbol{x}^{(t+1)})_i - f(\boldsymbol{x}^{(t)})_i)^2 \\ &s.t. \mathbb{E}_t f(\boldsymbol{x}^{(t)})_i = 0 \\ &\quad \mathbb{E}_t [f(\boldsymbol{x}^{(t)})_i^2] = 1 \end{aligned} \tag{13.6}$$

其中, 第一个限制条件确保该优化问题有唯一解; 第二个限制条件防止病态问题将所有特征坍缩至 0. 对于多特征的情形, 还需要添加限制条件

$$\forall i < j, \mathbb{E}_t [f(\boldsymbol{x}^{(t+1)})_i, f(\boldsymbol{x}^{(t)})_j] = 0 \tag{13.7}$$

这条限制条件对不同特征进行解耦合.

SFA 常被拓展至非线性领域用来学习非线性特征.

SFA 的一大优势是可以很容易预测出它所学习到的特征形式, 但同时它表现受限的原因也很不清晰, 推测可能是因为速度施加了过强的影响.

13.4 Sparse Coding

稀疏编码是一种线性因子模型, 在无监督特征学习和特征提取机制中被广泛研究. 稀疏编码假设噪声服从精度为 β 的各向同性 Gaussian 分布.

$$p(\boldsymbol{x}|\boldsymbol{h}) = \mathbb{N}(\boldsymbol{x}; \boldsymbol{W}\boldsymbol{h} + \boldsymbol{b}, \frac{1}{\beta}\boldsymbol{I}) \tag{13.8}$$

$p(\boldsymbol{h})$ 通常选定为在 0 附近有尖锐峰值的分布, 如 Laplace 分布和 Sutdent-t 分布.

由于模型不能通过最大似然度进行训练, 通常使用交替训练策略进行 decoder 和 encoder 的训练.

并不是所有的稀疏编码方式都显式构建 $p(\boldsymbol{h})$ 和 $p(\boldsymbol{x}|\boldsymbol{h})$.

稀疏编码与非参数 encoder 紧密结合, 相比于其他参数 encoder, 可以更好地最小化重构误差和对数先验. 另一项优势是它没有泛化误差.

稀疏编码的一项劣势是给定 $\boldsymbol{x}$ 求取 $\boldsymbol{h}$ 的过程是一个迭代过程, 因此需要耗费较长训练时间. 另一项劣势是非参数 encoder 进行反向传播操作并不直观.

13.5 Manifold Interpretation of PCA

线性因子模型中的 PCA 和 factor analysis 方法可以被视为学习一个流形.

设 encoder 为

$$\boldsymbol{h} = f(\boldsymbol{x}) = \boldsymbol{W}^T(\boldsymbol{x} - \boldsymbol{\mu}) \tag{13.9}$$

重构过程为

$$\hat{\boldsymbol{x}} = g(\boldsymbol{x}) = \boldsymbol{b} + \boldsymbol{V}\boldsymbol{h} \tag{13.10}$$

线性 encoder 和 decoder 的选择可以最小化重构误差

$$\mathbb{E}[\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2] \tag{13.11}$$

本章介绍的线性因子模型可以被拓展至 autoencoder 网络和深度概率模型, 在同样的任务中可以有更强大的功能和模型族的自由度.

Chapter 14 Autoencoders

Autoencoder 是一个试图将输入复制到输出的神经网络. 通常只是将输入和输出限制为大致相似, 并且只对输入中有用的部分进行描述. 传统上 autoencoder 是降维和特征学习的方式之一, 但是近来它也被视为生成式模型的学习方法. 尽管 autoencoder 与前向神经网络有不同之处, 它仍旧可以使用 *recirculation*[1]进行训练.

14.1 Undercomplete Autoencoders

Undercomplete autoencoders 的目的在于获取隐变量 $\boldsymbol{h}$ 中有用的特性. 因此在模型中对其施加了 $\boldsymbol{h}$ 维度要比 $\boldsymbol{x}$ 维度低的限制.

学习过程可以被视为最小化损失函数

$$L(\boldsymbol{x}, g(f(\boldsymbol{x}))) \tag{14.1}$$

的过程. 当 f, g 都是线性函数时,autoencoder 可以被视为张开一个和 PCA 相同的子空间.

但是当 encoder 和 decoder 的 capacity 过高时,autoencoder 可能提取不到有用的信息.

14.2 Regularized Autoencoders

在隐变量维度和输入变量维度相同, 以及过完备的情况下,autoencoder 学不到输入数据的分布. 但是借助于正则化,encoder 和 decoder 可以根据模型数据分布训练 autoencoder. 正则化项可以刺激模型训练出直接复制这种形式以外的表示方法.

[1]TODO: 是什么?

除了上述的 antoencoder, 几乎所有的包含隐变量以及推断过程的生成式模型都可以被视为一种特殊形式的 autoencoder.

Sparse Autoencoders

Sparse autoencoders 是指将 autoencoder 的训练准则中加入稀疏补偿项 $\Omega(\boldsymbol{h})$

$$L(\boldsymbol{x}, g(f(\boldsymbol{x}))) + \Omega(\boldsymbol{h}) \tag{14.2}$$

sparse autoencoders 的目的是学习对其他任务有用的特征. 但是这种正则化项并没有直观的贝叶斯表示方法.

稀疏补偿项使模型隐变量的分布, 它提供了近似训练生成式模型的方法.

14.2.1 Denoising Autoencoders

Denoising autoencoders 通过改变重构错误项来训练 autoencoder, 它最小化

$$L(\boldsymbol{x}, g(f(\tilde{\boldsymbol{x}}))) \tag{14.3}$$

其中,$\tilde{\boldsymbol{x}}$ 是被噪声干扰的输入 $\boldsymbol{x}$.

14.2.2 Regularizing by Penalizing Derivatives

这种情况下的补偿项为

$$\Omega(\boldsymbol{h}, \boldsymbol{x}) = \lambda \sum_i \|\nabla_{\boldsymbol{x}} h_i\|^2 \tag{14.4}$$

它迫使模型在输入 $\boldsymbol{x}$ 有微小变化时, 模型输出变动不会太大.

14.3 Representation Power, Layer Size and Depth

Autoencoder 还可以有深层结构. 它本身就是一种前向网络, 继承了前向网络的所有优势. 理论上, 包含隐层的深度 autoencoder 可以近似拟合任

何一种函数.

14.4 Stochastic Encoders and Decoders

前向网络的损失函数和输出单元同样可以用于 autoencoder.

以激进的观点抛开前向神经网络来看,*encoding functionf*($\boldsymbol{x}$) 可以被一般化为 *encoding distribution* $p_{encoder}(\boldsymbol{h}|\boldsymbol{x})$.

任何一个包含隐变量的模型 $p_{encoder}(\boldsymbol{h}|\boldsymbol{x}) = p_{model}(\boldsymbol{h}|\boldsymbol{x})$ 可以定义一个 stochastic encoder

$$p_{encoder}(\boldsymbol{h}|\boldsymbol{x}) = p_{model}(\boldsymbol{h}|\boldsymbol{x}) \tag{14.5}$$

和一个 stochastic decoder

$$p_{decoder}(\boldsymbol{x}|\boldsymbol{h}) = p_{model}(\boldsymbol{x}|\boldsymbol{h}) \tag{14.6}$$

14.5 Denoising Autoencoders

Denoising autoencoder (DAE) 是一种将受到噪声干扰的数据还原为原始数据的 autoencoder. 它可以归结为前向神经网络, 使用与前向神经网络相同的方法进行训练.

14.5.1 Estimating the Score

Score mathcing 是极大似然度的替代方法. 它是一种对概率分布的一致性估计. 它利用的方法是促使模型与数据分布在每个训练数据点上拥有相同的 score.DAE 一个重要的特性是其训练准则使得 autoencoder 可以学习出一个向量场 $(g(f(\boldsymbol{x})) - \boldsymbol{x})$ 用以估计数据分布的 score.

训练过程中使用 Gaussian 噪声和均方误差的 autoencoder 等价于训练一个 RBM 模型, 该模型拥有 Gaussian 可见单元.

14.6 Learning Manifolds with Autoencoders

和其他的机器学习模型一样,autoencoder 模型假设数据集中在低维度的流形或者流行子集上. 但是 autoencoder 不仅学习函数来拟合流形, 还需要去学习流行的结构.

流形学习有两个因素

- 学习 $\boldsymbol{h}$ 和 $\boldsymbol{x}$, 建立从输入到输出的映射;
- 满足限制条件或者正则化补偿项.

流行还有一个重要特性就是其切平面. 结合上述两点因素可以看出, 流形只对重要变量进行表示, 此时只需要对流形切平面上的变化做出反映即可.

最原始的流行学习方法是基于最近邻的非参数化方法. 它的缺点是当流行不是很平滑时, 需要大量的数据对新数据点进行插值拟合, 很难对未知的数据进行泛化. 但是常见的 AI 问题在流形上都是不平滑的, 不能通过插值法进行泛化.

14.7 Contractive Autoencoders

Contractive autoencoder 引入正则项

$$\Omega(\boldsymbol{h}) = \lambda \left\| \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} \right\|_F^2 \tag{14.7}$$

促使 f 的导数尽可能小.

Contractive autoencoder 和 denoising autoencoder 都能在一定程度上抵抗扰动, 但是前者是在特征提取环节实现的, 后者是在重构环节实现的.

需要注意的是,contractive autoencoder 只是局部收缩, 全局来看, 可能会存在两个点相互远离的情况.

CAE 的目的是为了学习到数据所表示的流行的结构.

深度 autoencoder 的计算代价较高, 并且可能会捕捉到无用的结果.

14.8 Predictive Sparse Decomposition

Predictive sparse decomposition (SPD) 是稀疏编码和参数 autoencoder 的结合体. 其中,encoder $f(\boldsymbol{x})$ 和 decoder $h(\boldsymbol{x})$ 都是参数化模型.

训练过程可以归结为最小化

$$\|\boldsymbol{x} - g(\boldsymbol{x})\|^2 + \lambda|\boldsymbol{h}|_1 + \gamma\|\boldsymbol{h} - f(\boldsymbol{x})\|^2 \tag{14.8}$$

14.9 Application of Autoencoders

Autoencoder 被广泛用于降维和信息重获取任务中.

低维度特征可以在许多任务上运用, 提升性能. 同时, 降维也可以提升泛化能力.

在信息重获取任务中,autoencoder 可以使得信息检索在某些低维度空间上效率更加高效. 具体实现可以参考语义哈希 (semantic hashing).

Chapter 15 Representation Learning

特征表示的方式直接决定信息处理任务的难易程度. 一般情况下, 一个好的特征表示方式可以使得后续学习任务更容易完成.

有监督学习方式训练出的前向神经网络是一种特征学习方法. 但是它并没有对中间特征显式地施加影响. 对于学习出的特征表示方式, 也可以供不同的任务使用.

大多数特征学习任务必须要在保持尽可能多的信息与获取更优异的特性这两者之间进行权衡.

特征学习提供了一种实现无监督学习和半监督学习的实现方式, 可以帮助解决无监督学习和半监督学习问题.

15.1 Greedy Layer-Wise Unsupervised Pretraining

贪婪逐层无监督预训练 (Greedy Layer-Wise Unsupervised Pretraining) 是基于单层特征学习算法 (如 RBM) 进行实现的. 它使用无监督学习方式规避有监督学习任务中深度网络各层联合训练的困难. 贪婪逐层训练是第一种成功应用于全连接深度结构的训练方法. 贪婪是指它对结果进行分块优化, 而不是进行整体优化. 预训练是指其仅仅只是训练过程的第一步, 之后还要进行参数的精细化调整. 因此, 在有监督学习的框架下, 它仅仅只是一种正则化方法或者是一种参数初始化策略. 通常我们使用的预训练不仅仅是指预训练阶段, 还包含之后的有监督学习阶段.

贪婪逐层无监督预训练不仅仅用于深度神经网络模型上, 还可以用于其他无监督学习算法的初始化.

15.1.1 When and Why Does Unsupervised Pretraining Work?

由于无监督预训练有时会对任务有帮助, 但是在大多数情况下反而会恶化学习任务. 因此需要搞清楚它为什么会有效, 因此就可以针对特定任务运用它.

无监督预训练包含两个核心思想

1. 深度神经网络的初始参数对算法有较强的正则作用;

2. 学习输入的分布有助于建立从输入到输出的映射.

其中, 核心思想中的第一点并没有被完全理解. 我们并没有准确认识到初始化参数所反映的特性在有监督学习阶段被保留多少. 第二点的认识稍微深入一些, 有些对于无监督学习比较有效果的特征可能同样对有监督学习有效果. 但是并不清楚哪种任务可以从无监督学习中取得收益.

从特征学习的角度来看, 无监督预训练在原始特征表示比较弱的情况下, 无监督预训练会更加有效. 从正则化角度来看, 无监督预训练在有标记样本数量较少的情况下, 对提升模型的效果更有帮助.[1]

无监督预训练还可以提升分类任务以外其他任务的结果, 还可以用来提升优化环节而不仅仅只是正则化环节.

无监督预训练可以帮助深度神经网络初始参数保持在特定区域中, 从而使得结果可以更好地收敛.

一个重要问题是无监督预训练如何能够起到正则化作用. 其中一个假设就是, 预训练过程可以使得学习算法发现与数据生成过程中的暗含因素相关的特征.

无监督预训练的劣势有

- 它将显式地将学习过程分为两个阶段;
- 正则化强度不能控制;
- 两个阶段有各自的超参数, 超参数过多, 事前不能调节.

[1]其他无监督预处理对提升模型效果有正向作用的例子可以参见 P532.

目前除了自然语言处理领域以外的其他领域很少使用无监督预训练. 但是预训练的核心思想已经渗透到有监督预训练中.

15.2 Transfer Learning and Domain Adaption

迁移学习 (transfer learning) 和领域自适应 (domain adaptation) 是指在一种分布 P_1 中学习到的知识被用于提升另外一种分布 P_2 的泛化能力.

在迁移学习中, 我们假定分布 P_1 中的部分变量可以被用于学习 P_2 的分布. 通常情况下, 迁移学习, 多任务学习和领域自适应可以通过特征学习的方式进行实现, 其中的特征对于不同的分布或者任务均有效, 这些特征对应于暗含的因子. 然而在某些情况下, 不同的任务间共享的不是输入的语义信息而是输出的语义信息. 因此需要在较高的逻辑层次进行概念迁移.

领域自适应是指任务相同但是输入的分布有轻微的不同.

另外还有一个相关的问题是概念漂移 (concept drift), 可以将其视为一种随着时间发展对数据分布进行逐渐演进的迁移学习. 概念漂移和迁移学习都可以被视为多任务学习的特例.

特征学习的核心思想在于相同的表示方式对于不同的分布均有效.

如果使用更深的模型进行特征表示, 那么在新的分布上, 其效果会更好.

迁移学习的极端情况是 *one-shot learning* 和 *zero-shot / zero-data learning*.one-shot learning 学习有效的原因可能是其在第一阶段就学习到了数据分布中暗含的因素.

zero-shot learning 和 zero-data learning 仅当附加信息在训练过程中被运用时才会生效.zero-shot learning 要求任务的描述方式具有一定的泛化能力才会生效.

15.3 Semi-Supervised Distangling of Causal Factors

对于如何评判一种特征表示方式优劣的问题, 一个假设是, 理想的特征表示方式和底层暗含的数据生成因素相对应, 即特征之间不存在耦合关系.

除此之外, 特征表示还要比较容易进行塑造. 对于许多 AI 任务, 上述两项特性是一致的.

如果 $\boldsymbol{y}$ 与 $\boldsymbol{x}$ 的起因 (causal factors) 关系密切, 那么 $p(\boldsymbol{x})$ 与 $p(\boldsymbol{y}|\boldsymbol{x})$ 就会有紧密联系. 非监督特征表示试图对变量中暗含的因素进行解耦合, 这种策略与半监督学习策略很类似.$p(\boldsymbol{x})$ 与 $p(\boldsymbol{y}|\boldsymbol{x})$ 就会有紧密联系体现在

$$\begin{aligned} p(\boldsymbol{h}, \boldsymbol{x}) &= p(\boldsymbol{x}|\boldsymbol{h})p(\boldsymbol{h}) \\ p(\boldsymbol{x}) &= \mathbb{E}_{\boldsymbol{h}} p(\boldsymbol{x}|\boldsymbol{h}) \end{aligned} \tag{15.1}$$

最佳的关于 $\boldsymbol{x}$ 的分布是 $\boldsymbol{h}$ 揭示出了 $\boldsymbol{x}$ 的真实分布.

在现实中, 输入变量通常是多种因素共同作用的结果. 一个直接的方式就是暴力搜索. 但是在绝大多数情况下, 暴力搜索方式并不可行, 替代策略有

- 同时使用有监督学习与无监督学习去学习特征;
- 如果使用无监督特征学习策略, 就是用更大的特征集.

对于无监督学习策略, 可以去改变显著性的定义. 具体来说, 可以使用生成式对抗网络 (generative adversarial networks) 定义显著性.

学习暗含的起因的收益是, 如果生成式过程的果为 $\boldsymbol{x}$, 因为 $\boldsymbol{y}$, 那么 $p(\boldsymbol{x}|\boldsymbol{y})$ 对 $p(\boldsymbol{y})$ 的变化具有鲁棒性.

15.4 Distributed Representation

分布式特征表示 (Distributed Representation) 的定义为, 特征表示的元素间相互可分离. 与之相对应的概念为符号特征表示 (symbolic representation), 每一个单个的符号代表一个类别.

对于非分布式特征表示, 参数的个数和特征空间中定义的区域的数目是线性关系.

分布式特征表示与符号特征表示的区别在于由于在不同概念间共享属性因而具有泛化能力[2]. 分布式特征表示包含有丰富的相似度空间, 在这个空间中, 语义上相似的概念有较近的距离, 这是符号特征表示所不具备的特性.

[2]generalization arises due to shared attributes between different concepts.

当使用很复杂的结构可以使用少量参数进行表示时, 分布式特征表示具有统计上的优势. 传统的符号特征表示泛化能力的前提假设是平滑假设. 但是容易受到维度灾难的影响, 并且很难根究已有的符号泛化到新的符号.

对于分布式表示, 如果有 $O(nd)$ 个参数, 那么可以将输入空间划分为 $O(n^d)$ 个区域. 分布式表示的 capacity 仍然受限, 因此泛化能力比较强.

15.5 Exponential Gains from Depth

上节提到, 深度结构可以提升统计效率. 在实际中, 深度特征表示可以对低层次特征进行抽象, 并进行非线性的加工.

15.6 Providing Clues to Discover Underlying Causes

大多数特征学习的策略是引入有助于发现底层因素的线索. 具体有

- Smoothness
- Linearity
- Multiple explanatory factors
- Causal factors
- Depth, or a hierarchical organization of explanatory factors
- Shared factors across tasks
- Manifolds
- Natural clustering
- Temporal and spatial coherences
- Sparsity
- Simplicity of Factor Dependencies

Chapter 16 Structured Probabilistic Models for Deep Learning

结构化概率模型 (Structured Probabilistic Models) 是一种使用图的形式描述随机变量间交互关系的方法.

16.1 The Challenge of Unstructured Modeling

深度学习的目标是对机器学习算法进行拓展, 以应对为了解决人工智能问题所要面对的挑战.

大部分使用结构化概率模型的任务对输入的结构做一个完整地理解和构建, 这些任务包括

- 密度估计
- 去噪
- 缺失值填充
- 采样

直接使用表格法存储各个随机变量间的交互情况并不可行, 原因有

- 内存限制
- 统计效率

- 推断代价
- 采样代价

但是在实际上, 各个变量子集间的影响是间接的. 结构化概率模型只关注随机变量间的**直接关系**.

16.2 Using Graphs to Describe Model Structure

图模型可以分为*有向无环图*与*无向图*两类.

16.2.1 Directed Models

有向图的边是有向的, 体现在概率分布上就是条件概率分布.

只要每个变量在图中只有一小部分父节点, 那么联合分布就可以使用少量的参数进行表示.

需要注意的是, 有向图仅仅对相互间条件独立的变量有简化作用.

16.2.2 Undirected Models

无向图的边是没有指向的, 图模型中的边与条件概率无关.

无向图模型中的因子 (factor) 或者说团势 (clique potential)$\phi(\mathcal{C})$ 用以度量团团 (clique)$\mathcal{C}$ 中每个变量间的紧密关系. 但是因子并不能保证产生一个有效的概率分布.

16.2.3 The Partition Function

为保证得到有效的概率分布, 需要使用规范化因子 Z.ϕ 必须可以让 Z 易于计算. 在深度学习中, 常对 Z 进行近似估计.

有向图和无向图的一个重要区别是有向图直接基于概率分布构建模型, 而无向图使用 ϕ 构建模型, 进而转化为概率分布.

需要特别注意的是, 每个变量对于相应的势函数都会有影响.

16.2.4 Energy-Based Models

许多无向图模型都是基于假设 $\forall \boldsymbol{x}, \tilde{p}(\boldsymbol{x}) > 0$ 构建的. 一种简便的方式是使用基于能量的模型 (energy-based model)

$$\tilde{p}(\boldsymbol{x}) = \exp(-E(\boldsymbol{x})) \tag{16.1}$$

定义模型. 能量函数可以完全自由地选择, 没有团势函数的限制.

能量函数中的每一项对应图中的一个团. 这也被称为 *product of experts*. 也就是说, 每一个专家仅仅关注低维度内随机变量的映射, 但是多个专家相乘则对更加复杂的高维度空间内的变量进行了限制.

一般情况下, 更常用的是自由能量 (free energy)

$$\mathcal{F}(\boldsymbol{x}) = -\log \sum_{\boldsymbol{h}} \exp(-E(\boldsymbol{x}, \boldsymbol{h})) \tag{16.2}$$

16.2.5 Separation and D-Separation

由于图模型中的边对应的是变量间的直接交互, 那么研究变量间间接交互 (即变量间条件独立) 情况就对应于图的分离. 其中, 有向图的分离被定义为 *d-separation*.

但是需要强调的是, **图并不能展现所有的独立性关系**. 图不会展现出不存在的独立性关系, 但是可能会无法展现出某些独立性关系.

16.2.6 Converting between Undirected and Directed Graph

有向图模型和无向图模型各有优劣, 需要根据具体任务选择使用. 每一个概率分布既可以使用有向图表示也可以使用无向图表示.

为了使得图模型可以尽可能完整地表示出独立性关系, 可以使用有向图模型. 有向图模型中有无向图不能进行完美表示的子结构, 被称为 *immorality* 结构 (子结点两个父节点间没有直接关联的结构.) 有向图转换为无向图的方法为在图中建立起有向图中原有的连接, 并且将 immorality 结构转换为 *moralized graph*.

无向图中也有有向图无法完美表示的子结构 (最大长度超过 4 的环, 并且不相邻的节点间没有弦). 在无向图转换为有向图之前, 首先需要进行弦化或者三角化转换. 然后给结点进行排序, 加上边的指向, 避免产生环结构.

16.2.7 Factor Graph

因子图 (Factor Graph) 是另一种构造无向图的方法, 它可以避免标准无向图变量在图表示上的二义性. 因子图显式规定了势函数的作用范围. 圆形结点是随机变量, 方形结点是势函数.

16.3 Sampling from Graphical Models

图模型也可用于采样任务. 图模型的一项优势是借助于 *ancestral sampling* 过程从模型所标示的联合分布中进行高效率采样. 但是需要按照拓扑顺序进行采样.

但是 ancestral sampling 的劣势有

- 它仅仅只针对有向图模型进行采样;
- 它并不能对任意的有向图模型进行采样.

无向图可以使用 *Gibbs sampling* 方法进行采样.

16.4 Advantages of Structured Modeling

结构化概率模型的首要优势在于它可以减少特征表示的代价以及学习和推断过程的代价. 结构化概率模型通过忽略某些变量间的交互节省运行时间也内存消耗.

另外一项优势是它可以显式地分离知识表示的学习过程和推断现有的知识的过程.

16.5 Learning about Dependencies

好的生成式模型必须尽可能准确捕捉可见变量 (visible variants) 的分布情况. 在深度学习的情境中, 最常见的使用方式是借助于隐变量定义可见变量间的依赖关系.

当模型通过建立直接连接来捕捉所有变量间的独立关系, 需要在所有图中变量间建立起两两连接. 学习模型结构的过程叫做 *structure learning*. 如果引入隐变量, 学习过程就只需要学习相应的参数, 而不用对模型的结构进行变动.

16.6 Inference and Approximate Inference

通过概率模型我们可以获得变量间的相互关联关系.

$$\log p(\boldsymbol{v}) = \mathbb{E}_{\boldsymbol{h}\sim p(\boldsymbol{h}|\boldsymbol{v})}[\log p(\boldsymbol{h}, \boldsymbol{v}) - \log p(\boldsymbol{h}|\boldsymbol{v})] \tag{16.3}$$

我们经常需要计算 $p(\boldsymbol{h}|\boldsymbol{v})$ 以实现上述学习规则. 但是对于大部分深度模型, 这些推断问题通常不易解决.

可以很直观地看出, 一般的图模型计算边缘概率分布是一个 #P 难问题. 这样就需要通过近似推断来避免这一问题.

16.7 The Deep Learning Approach to Structured Probabilistic Models

在深度学习情境中, 我们通常使用不同的设计决策来对传统的计算工具进行组合. 这样整体的算法和模型相比于传统算法有不同的特点.

深度学习中的隐变量和结构化概率模型的隐变量设计方式不同.

深度学习中的连接方式和结构化概率模型的连接方式不同.

相比于结构化概率模型, 深度学习对于未知情况有一定的容忍度.

16.7.1 Example: The Restricted Boltzmann Machine

有限制 *Boltzmann* 机 (Restricted Boltzmann Machine, RBM, 又称 harmonium) 是图模型用于深度学习的精髓部分.

从整体上来说,RBM 向概率图模型演示了典型的深度学习方法: 使用隐层变量进行特征表示学习, 以及不同层的神经元参数之间的高效交互.

Chapter 17 Monte Carlo Methods

随机算法可以分为两类:*Las Vegas 算法*和 *Monte Carlo 算法*.Las Vegas 算法返回一个精确结果, 但是耗费的时间不确定;Monte Carlo 算法返回一个近似正确的结果, 但是耗费时间比较固定.

17.1 Sampling and Monte Carlo Methods

在机器学习中, 通常会从某个概率分布中采样一定的样本, 并且使用这些样本去估计某些变量.

17.1.1 Why Sampling?

采样提供了一种灵活的方法, 它可以以较低的代价去估计求和以及积分结果.

17.1.2 Basics of Monte Carlo Sampling

核心思想 Monte Carlo 采样的核心是将求和或者积分运算视为在某个分布下的期望值运算, 并且通过求取平均值计算期望值.

$$s = \sum_{\boldsymbol{x}} p(\boldsymbol{x}) f(\boldsymbol{x}) = E_P \left[f(\boldsymbol{x}) \right] \tag{17.1}$$

或者

$$s = \int p(\boldsymbol{x}) f(\boldsymbol{x}) d\boldsymbol{x} = E_P \left[f(\boldsymbol{x}) \right] \tag{17.2}$$

一般而言, 直接求取分布是比较困难的, 因此通常采用的方法是从分布中采样来代替直接求取分布. 最后问题可以归结为如何对分布进行采样, 从而使得通过采样计算出的期望值近似等于目标和或者目标积分值.

17.2 Importance Sampling

将求和过程或者求积分过程进行分解, 其中一个重要环节是确定哪一部分为 $p(\boldsymbol{x})$, 哪一部分为 $f(\boldsymbol{x})$. 但是这种分解并没有统一标准, 因为分解形式并不唯一.

$$p(\boldsymbol{x})f(\boldsymbol{x}) = q(\boldsymbol{x})\frac{p(\boldsymbol{x})f(\boldsymbol{x})}{q(\boldsymbol{x})} \tag{17.3}$$

optimal importance sampling $p(\boldsymbol{x})f(\boldsymbol{x})$ 并不一定是最优的分解方式. 最优选择 q^* 被称为 *optimal importance sampling*.

biased importance sampling 另外一种重要的分解方式是 *biased importance sampling*, 它不要求 p 和 q 必须进行归一化.

q 的选择直接对 Monte Carlo 估计的效率产生影响, 因此进行选择时必须加以注意[1].

尽管 importance sampling 有许多危险之处, 但是它包括深度学习在内的机器学习算法中起到了重要作用.

17.3 Markov Chain Monte Carlo Methods

在许多情况下, 我们希望使用 Monte Carlo 方法, 但是很难获取到可用的方法去从 p_{model} 中精确采样或者从一个足够好的分布 q 中进行 importance sampling. 在无向图中, 通常使用无向图表示分布 p_{model}, 因此就可以借助于 *Markov Chain* 去近似采样. 这种方法被称为 *Markov Chain Monte Carlo 方法* (MCMC 方法).

MCMC 方法的限制是模型的每一个状态都没有 0 概率. 因此该方法适用于基于能量的方法 (energy-based method, EBM).

在 EBM 中进行 *ancestral sampling* 会遇到先有鸡还是先有蛋的问题, 为避免这一问题需要采用 Markov 链方法. 核心思想是状态 $\boldsymbol{x}$ 以任意值作为起始状态, 不断迭代更新, 最终逼近真实的分布[2]. 这一过程最终收敛到

[1] 见 P594 页中间部分.

[2] TODO: 再看一遍参数化的过程.

stationary distribution 或者 *equilibrium distribution*. 如果状态转移 T 选取正确, 那么稳态分布 q 将会与分布 p 相同.

难点 所有的 Markov Chain 都包含随机更新过程直至收敛到 equilibrium distribution 的过程, 并从中采样. 这一过程被称为 *burning in*. 这一过程产生的相邻两个样本相互间是耦合的, 为了产生不耦合的样本, 需要间隔一定的次数进行采样, 时间代价会很高. 可以使用并行化进行操作.

另外一个难点是事先并不知道经过多少步可以达到稳定状态. 其中迭代的步数被称为 *mixing time*.$\boldsymbol{A}$ 中比 1 小的特征值决定了 mixing time, 但是实际中 Markov Chain 并不能使用矩阵进行标示, 因此只能通过估测判定是否 mix.

17.4 Gibbs Sampling

确保 $q(\boldsymbol{x})$ 成为可用分布的方法有两种:

- 从学习得到的 p_{model} 中获取到 T;
- 直接参数化 T 并对其进行学习.

一个从 p_{model} 中进行采样并建立起 Markov Chain 的方法是使用 Gibbs sampling[3].

17.5 The Challenge of Mixing between Separated Modes

MCMC 方法的一个难点是其很有可能 mix 不充分. 这样会导致在高维空间中,MCMC 采样会非常耦合. 耦合的原因是其在低能量内进行转移, 不能顺利逃离低能量区域. 这个问题在一定条件下可以通过找到独立成分并进行同步更新加以解决, 但是当独立关系很复杂时, 这个方法就失效了.

[3]具体算法过程可见周志华 < 机器学习 > 中 7.5 节相关内容.

17.5.1 Tempering to Mix between Modes

当一个分布有明显尖峰并且其周围相对平坦时, 它很难与其他 mode 进行 mix. 基于能量的模型可以通过引入参数 β 控制尖峰的分布

$$p_\beta \propto \exp(-\beta E(\boldsymbol{x})) \tag{17.4}$$

其中 β 常被称为 *temperature* 的倒数, 反映员是基于能量模型的统计物理量.Tempering 是一种常用的 mixing 策略. 但是其仍有局限性[4].

17.5.2 Depth May Help Mixing

当从包含隐变量的模型 $p(\boldsymbol{h}, \boldsymbol{x})$ 中进行采样时, 如果 $p(\boldsymbol{h}|\boldsymbol{x})$ 对 $\boldsymbol{x}$ 编码过于完善的话, 那么 $p(\boldsymbol{x}|\boldsymbol{h})$ 并不会使 $\boldsymbol{x}$ 变化太大, 进而使得 mixing 效果欠佳. 一种解决方案是使 $\boldsymbol{h}$ 称为深度特征表示.

[4]TODO: 再读一遍本节.

Chapter 18 Confronting the Partition Function

在一些情况下, 我们需要将 $\tilde{p}$ 除以 *partition function* $Z(\theta)$ 进行归一化, 以获得有效概率分布

$$p(\boldsymbol{x};\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})}\tilde{p}(\boldsymbol{x};\boldsymbol{\theta}) \tag{18.1}$$

其中 partition function 是一个对所有未归一化概率的积分 (连续变量) 或者求和 (离散变量) 操作.

$$\begin{aligned} &\int \tilde{p}(\boldsymbol{x})d\boldsymbol{x} \\ &\sum_{\boldsymbol{x}} \tilde{p}(\boldsymbol{x}) \end{aligned} \tag{18.2}$$

18.1 The Log-Likelihood Gradient

无向图模型学习在最大化对数似然特别困难是因为它的 partition function 依赖于参数.

$$\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x};\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}}\tilde{p}(\boldsymbol{x};\boldsymbol{\theta}) - \nabla_{\boldsymbol{\theta}} \log Z(\boldsymbol{\theta}) \tag{18.3}$$

这就是我们所熟知的 *positive phase* 和 *negative phase* 分解[1].

$$\nabla_{\boldsymbol{\theta}} \log Z = \mathbb{E}_{\boldsymbol{x}\sim p(\boldsymbol{x})} \nabla_{\boldsymbol{\theta}} \log \tilde{p}(\boldsymbol{x}) \tag{18.4}$$

上述等式是 Monte Carolo 方法估计极大似然的基础.

[1]The term positive and negative do not refer to the sign of each term in equation, but rather reflect to their effect on the probability density defined by the model.(http://deeplearning.net)

在 positive phase 项中, 我们增大从**数据**中采样的 $\boldsymbol{x}$ 所对应的 $\log \tilde{p}(\boldsymbol{x})$ 的值. 在 negative phase 项中, 我们通过降低从**模型**中采样的 $\log \tilde{p}(\boldsymbol{x})$ 的值而降低 partition function 的值[2].

18.2 Stochastic Maximum Likelihood and Contrastive Divergence

naive approach 式18.4的一种直观计算方法为, 每次需要计算梯度时都从随机状态 burning in 一个 Markov 链. 可以看出, 这样的方法计算代价较高并且非常低效.

MCMC 方法最大化似然度可视为在两种力量间进行平衡: 一种力量是提升数据所在的模型分布位置的概率; 另外一种力量是压低模型样本出现位置的模型分布概率.

contrastive divergence contrastive divergence(CD, 或者 CD-K 用以指示 k 步 Gibbs 的 CD 算法) 在初始化 Markov 链的每一步都使用从数据分布中采样到的样本.

CD 算法容易在 supurious modes 的情况下失效. 并且对于直接训练深度模型没有太大的帮助.

CD 算法可以视为对模型进行补偿, 当 Markov 链的输入是从数据分布中采样得到时, 其结果变化剧烈.

stochastic maximum likelihood(SML) 另外一种方法是在计算梯度而初始化 Markov 链的时候, 都将前一步的梯度方向考虑进来, 用以初始化 Markov 链. 由于可以储存观测变量和隐变量, 所以 SML 可以同时初始化观测变量和隐变量.SML 可以高效训练深度模型.

相比于精确采样,CD 算法的方差较低, 但是 SML 的方差偏高.

[2]There are many ways that a learner can capture knowledge about the data generating distribution p_{data} from which training examples were obtained. Some of this encapsulate a model p_{model} of that distribution (or of a derived distribution such as a distribution of output variables given input variables).

基于 MCMC 采样的方法适用于几乎所有的 MCMC 变体.

Fast PCD 一种加速学习过程 mixing 的方法不是改变 Monte Carlo 方法, 而是改变模型和代价函数的参数化过程. 即将参数 θ 替换为

$$\boldsymbol{\theta} = \boldsymbol{\theta}^{(slow)} + \boldsymbol{\theta}^{(fast)} \tag{18.5}$$

尽管当快速权重可以自由改变时效果才会体现, 但是这样就可以使得 Markov 链进行快速 mix.

18.3 Pseudolikelihood

Monte Carlo 直接面对 partition function 的计算过程, 与之对应的其他方法则绕考计算 partition function 的过程, 也就是本节所提到的 *pseudo-likelihood* 方法.

Pseudolikelihood 方法通过概率的比例部分消除掉 partition function 的影响. 假设将变量 $\boldsymbol{x}$ 分割为 $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$, 那么有

$$p(\boldsymbol{a}|\boldsymbol{b}) = \frac{p(\boldsymbol{a}, \boldsymbol{b})}{p(\boldsymbol{b})} = \frac{p(\boldsymbol{a}, \boldsymbol{b})}{\sum_{\boldsymbol{a},\boldsymbol{c}} p(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})} = \frac{\tilde{p}(\boldsymbol{a}, \boldsymbol{b})}{\sum_{\boldsymbol{a},\boldsymbol{c}} \tilde{p}(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})} \tag{18.6}$$

在实际中,$\boldsymbol{c}$ 的数目会很多, 如果将 $\boldsymbol{c}$ 并入 $\boldsymbol{b}$ 会减少计算量, 这样就产生了 pseudolikelihood 目标函数

$$\sum_{i=1}^{n} \log p(x_i | \boldsymbol{x}_{-i}) \tag{18.7}$$

将计算复杂度向偏差让渡, 可得 *generalized pseudolikelihood* 估计

$$\sum_{i=1}^{m} \log p(x_{\mathbb{S}^{(i)}} | \boldsymbol{x}_{-\mathbb{S}^{(i)}}) \tag{18.8}$$

基于 pseudolikelihood 的方法表现很大程度上取决于模型如何使用.generalized pseudolikelihood 适用的情形为 $\mathbb{S}$ 捕捉到了变量间的重要耦合关系. 但是它的缺点为, 当与其他近似估计方法一起使用时, 仅能提供 $\tilde{p}(\boldsymbol{x})$ 的下限.

它也不能用于深度模型. 但是它可以用于深度网络的单层神经元训练以及不基于下限的近似推断方法.

同时需要注意的是它的计算代价比较大.

18.4 Score Matching and Ratio Matching

Score Matching 在训练模型时可以不用对 Z 及其导数进行估计.$\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x})$ 被称为 *score.score matching* 是指

$$\begin{aligned} L(\boldsymbol{x}, \boldsymbol{\theta}) &= \frac{1}{2} \| \nabla_{\boldsymbol{x}} \log p_{model}(\boldsymbol{x}; \boldsymbol{\theta}) - \nabla_{\boldsymbol{x}} \log p_{data}(\boldsymbol{x}; \boldsymbol{\theta}) \|_2^2 \\ J(\boldsymbol{\theta}) &= \frac{1}{2} \mathbb{E}_{p_{data}(\boldsymbol{x})} L(\boldsymbol{x}, \boldsymbol{\theta}) \\ \boldsymbol{\theta}^* &= \min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \end{aligned} \tag{18.9}$$

同时可以看出,score matching 需要知道真实的数据分布 p_{data}. 但是在一定条件下[3],$L(\boldsymbol{x}, \theta)$ 等价于

$$\tilde{L}(\boldsymbol{x}, \boldsymbol{\theta}) = \sum_{j=1}^{n} \left(\frac{\partial^2}{\partial x_j^2} \log p_{model}(\boldsymbol{x}; \boldsymbol{\theta}) + \frac{1}{2} (\frac{\partial}{\partial x_j} \log p_{model}(\boldsymbol{x}; \boldsymbol{\theta}))^2 \right) \tag{18.10}$$

上式不适用于离散观察变量, 但是可以用于离散隐变量.

Generalized Score Matching(GSM) GSM 适用于观察变量是离散的情形, 但是不适用于高维离散空间中某些事件概率为 0 的情形.

Ratio Matching Ratio Matching 是特别针对二值数据提出的.

$$L^{(RM)}(\boldsymbol{x}; \boldsymbol{\theta}) = \sum_{j=1}^{n} \left(\frac{1}{1 + \frac{p_{model}(\boldsymbol{x}; \boldsymbol{\theta})}{p_{model}(f(\boldsymbol{x}), j; \boldsymbol{\theta})}} \right) \tag{18.11}$$

但是 ratio matching 的计算代价较高.

Ratio Matching 适用于高维离散的情形.

18.5 Denoising Score Matching

在一些情况下, 我们希望正则化 score matching, 就需要将真实数据分布 p_{data} 替换为

$$p_{smoothed}(\boldsymbol{x}) = \int p_{data}(\boldsymbol{y}) q(\boldsymbol{x} | \boldsymbol{y}) d\boldsymbol{y} \tag{18.12}$$

[3]见 P618

如果给定足够的 capacity, 任何一致性估计都会使得 p_{model} 在训练集上样本上形成 Dirac 分布. 对 q 进行平滑有助于减轻这个问题, 但同时也损失了一部分渐进一致性.

18.6 Noise-Contrastive Estimation

Noise-contrastive estimation(NCE) 在 NCE 方法中, 概率分布估计被显式定义为

$$\log p_{model}(\boldsymbol{x}) = \log \tilde{p}_{model}(\boldsymbol{x}; \boldsymbol{\theta}) + c \tag{18.13}$$

其中, 引入的 c 是 $-\log Z(\boldsymbol{\theta})$ 的近似估计.

如果采用最大似然度对式18.13进行估计, 那么 c 将会被置为一个非常高的值, 而不是产生一个有效分布.

Noise Distribution 因此引入噪声分布 $p_{model}(\boldsymbol{x})$,

$$\begin{aligned} p_{joint}(y=1) &= \frac{1}{2} \\ p_{joint}(\boldsymbol{x}|y=1) &= p_{model}(\boldsymbol{x}) \\ p_{joint}(\boldsymbol{x}|y=0) &= p_{noise}(\boldsymbol{x}) \end{aligned} \tag{18.14}$$

其中 y 是一个开关变量, 决定是从模型分布还是噪声分布中产生 $\boldsymbol{x}$. 这样就可以使用最大似然度解决上述问题

$$\boldsymbol{\theta}, c = \arg\max_{\boldsymbol{\theta}, c} \mathbb{E}_{\boldsymbol{x}, \boldsymbol{y} \sim p_{train}} \log p_{joint}(y|\boldsymbol{x}) \tag{18.15}$$

NCE 使用最为成功的是没有太多随机变量的情形, 但是这些随机变量可以有很数目的数值. 同时它的缺陷是 p_{noise} 必须容易估计, 并且很容易从过度限制中进行采样.

NCE 的核心思想是一个好的生成式模型应该可以区分开数据和噪声.

18.7 Estimating the Partition Function

在模型评估, 监控训练效果, 以及对比模型的过程中, 必须计算 partition function.

如果我们需要计算 $\mathcal{M}_A$ 或者 $\mathcal{M}_B$ 的真实分布, 并且知道两个分布的 partition function 的比值, 那么只需要计算其中一个 partition function 就可以了.

$$Z(\boldsymbol{\theta}_B) = rZ(\boldsymbol{\theta}) = \frac{Z(\boldsymbol{\theta}_B)}{Z(\boldsymbol{\theta}_A)} Z(\boldsymbol{\theta}_A) \tag{18.16}$$

一个简单的办法就是使用 Monte Carlo 方法估计 partition function.

$$\begin{aligned} Z_1 &= \int \tilde{p}(\boldsymbol{x}) d\boldsymbol{x} \\ &= \int \frac{p_0(\boldsymbol{x})}{p_0(\boldsymbol{x})} \tilde{p}_1(\boldsymbol{x}) d\boldsymbol{x} \\ &= Z_0 \int p_0(\boldsymbol{x}) \frac{\tilde{p}_1(\boldsymbol{x})}{\tilde{p}_0(\boldsymbol{x})} d\boldsymbol{x} \\ \hat{Z}_1 &= \frac{Z_0}{K} \sum_{k=1}^{K} \frac{\tilde{p}_1(\boldsymbol{x}^{(k)})}{\tilde{p}_0(\boldsymbol{x}^{(k)})} \quad \text{s.t. } \boldsymbol{x}^{(k)} \sim p_0 \end{aligned} \tag{18.17}$$

如果 p_0 和 p_1 的分布情况比较接近, 那么上式是一种比较有效的 partition function 的估计方式. 但是在大部分情况下,p_1 分布很复杂并且维度很高. 那么这就需要借助于一定的方法去弥补 p_0 和 p_1 之间的差异.

18.7.1 Annealed Importance Sampling

在 $DL(p_0 \| p_1)$ 很大时, 考虑使用 *annealed importance sampling*(AIS). 考虑分布序列 $p_{\eta_0}, p_{\eta_1}, \cdots, p_{\eta_n}$, 其中,$0 = \eta_0 < \eta_1 < \cdots < \eta_{n-1} < \eta_n = 1$.

中间分布需要针对具体问题进行特别设计, 通常的做法是使用权重几何平均

$$p_{\eta_j} \propto p_1^{n_j} p_0^{1-n_j} \tag{18.18}$$

18.7.2 Bridge Sampling

Bridge sampling 的原理是, 借助于单个分布 p_*, 去对一个 partition function 的分布和一个未知 partition function 的分布做桥接.

$$\frac{Z_1}{Z_0} \approx \sum_{k=1}^{K} \frac{\tilde{p}_*(\boldsymbol{x}_0^{(k)})}{\tilde{p}_0(\boldsymbol{x}_0^{(k)})} \Big/ \frac{\tilde{p}_*(\boldsymbol{x}_0^{(k)})}{\tilde{p}_1(\boldsymbol{x}_0^{(k)})} \tag{18.19}$$

p_* 选择策略是尽可能覆盖 p_0 和 p_1. 式18.19通常采用迭代方法最终获取精确值 r.

Linked importance sampling 当 p_0 和 p_1 的差距过大时考虑使用 AIS.

Estimating the partition function while training 另外一种策略是在训练过程中对 partition function 进行估计.

Chapter 19 Approximate Inference

概率模型训练的难点在于推断. 推断的目的是为了求得**目标变量的边际分布**或者**以某些可观测变量为条件的条件分布**. 深度学习中推断问题比较困难是由于隐变量间存在交互关系.

19.1 Inference as Optimization

精确推断可以归结为一个优化问题, 而近似推断则是精确推断的估计.

假设概率模型包含可观测变量 $\boldsymbol{v}$ 和隐变量 $\boldsymbol{h}$, 我们的目的是计算可观测变量的对数概率 $\log p(\boldsymbol{v};\boldsymbol{\theta})$. 在有些情况下边际化 $\boldsymbol{h}$ 很困难, 可以计算 $\log p(\boldsymbol{v};\boldsymbol{\theta})$ 的下限 $\mathcal{L}(\boldsymbol{v},\boldsymbol{\theta},q)$, 即 *evidence lower bound*(ELBO).ELBO 的另一个更常见的名称是 *negative variation free energy*.

$$\mathcal{L}(\boldsymbol{v},\boldsymbol{\theta},q)=\log p(\boldsymbol{v};\boldsymbol{\theta})-D_{KL}\Big(q(\boldsymbol{h}|\boldsymbol{v})\Big\|p(\boldsymbol{h}|\boldsymbol{v};\boldsymbol{\theta})\Big) \tag{19.1}$$

其中 q 是关于 h 的任意分布.

因此我们可以通过最大化 $\mathcal{L}$ 逼近真实分布进行推断. 也可以通过控制 $\mathcal{L}$ 的松紧程度灵活控制计算量.

19.2 Expectation Maximization

Expectation maximization(EM) 算法是一种基于最大化 $\mathcal{L}$ 的推断算法. 它适用于训练包含隐变量的模型. 但它并不是近似推断的方法, 而是学习近似后验概率的方法.

EM 算法可以分两步, 第一步通过选择分布 q 来最大化 $\mathcal{L}$; 第二步通过选择 $\boldsymbol{\theta}$ 来最大化 $\mathcal{L}$[1].

[1]具体算法步骤可见原书 P634 页下方.

EM 算法有两点本质:

1. 学习过程有一个基本的结构, 那就是通过更新模型参数来提升完备数据集的似然度, 其中所有的缺失变量可以通过后验概率进行估计填充;

2. 在 EM 算法中, 我们在更新了 $\boldsymbol{\theta}$ 值之后, 仍然可以沿用同一个 q 的值进行训练.

19.3 MAP Inference and Sparse Coding

推断也可以指计算条件概率的过程. 另外一种推断是计算出缺失变量的一个最可能的取值, 而不是推断出在所有可能取值上的概率分布, 即

$$\boldsymbol{h}^* = \arg\max_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v}) \tag{19.2}$$

这又被称为*最大后验概率推断* (maximum a posteriori inference, MAP inference).

通常我们并不将 MAP 推断视为一种近似推断, 因为它并不能准确地计算 $\boldsymbol{h}^*$. 如果基于最大化 $\mathcal{L}$ 的策略进行学习, 那么可以视为 MAP 提出了一个 q 值. 从这种程度上来看,MAP 推断是一种近似推断, 因为它并不能提供最优分布 q.

MAP 推断在深度学习领域既被视为一种特征提取器, 又被视为一种学习机制. 它主要被用于稀疏编码模型中.

19.4 Variational Inference and Learning

变分学习 (variational learning) 的核心思想是通过限制 q 的分布族来最大化 $\mathcal{L}$. 通常 q 分布相对简单或者具有良好的结构.

最常见的 q 的限制是

$$q(\boldsymbol{h}|\boldsymbol{v}) = \prod_i q(h_i|\boldsymbol{v}) \tag{19.3}$$

即 q 为乘积分布. 这也就是常说的平均场 (mean field) 方法. 更一般的, 可以使用任何图模型结构来代表 q, 这种通用图模型方法被称为结构化变分推断 (structured variational inference)[2]

变分方法的一大优势是我们不需要特别指定 q 的参数化形式. 即使当隐变量是离散情形时, 并不需要变分法, 但是变分法仍然是变分学习或者变分推断的名称来源.

19.4.1 Discrete Latent Variables

隐变量是离散情形的变分推断方法相对直观. 基于平均场的方法定义分布 q, 建立查找表, 通过查表法确定 $q(\boldsymbol{h}|\boldsymbol{v})$ 分布.

在确定 q 的形式后就需要优化其参数. 由于优化过程出现在学习过程的内循环中, 所以要求优化算法的收敛速度要快.[3]

19.4.2 Calculus of Variations

变分法 (calculus of variations) 寻求极值函数, 使泛函取得极值, 是处理函数的领域.

在机器学习领域中, 我们通常会求取使得 $J(\boldsymbol{\theta})$ 取得极小值的 $\boldsymbol{\theta}$. 同样的, 在某些情况下, 我们的目标是求取使得 $J(f(\boldsymbol{x}))$ 取得极小值的函数 $f(\boldsymbol{f})$, 这就是变分法的应用场景.

一个方程的方程被称为泛函 (functional)$J[f]$. 泛函对应于任意一个取值 $\boldsymbol{x}$ 的 $f(\boldsymbol{x})$ 的导数被称为泛函导数 (functional derivatives) 或者变分导数 (variational derivatives), 记为 $\frac{\delta}{\delta f(x)}J$.

对于可微函数 $f(\boldsymbol{x})$ 和具有连续导数的可微函数 $g(y, \boldsymbol{x})$, 有

$$\frac{\delta}{\delta f(x)} \int g(f(\boldsymbol{x}), \boldsymbol{x}) d\boldsymbol{x} = \frac{\delta}{\delta y} g(f(\boldsymbol{x}), \boldsymbol{x}) \tag{19.4}$$

我们可以在泛函导数为 0 的点的集合内取得泛函优化问题的最优值[4].

[2]什么是结构化变分推断? 怎样从学习到推断?

[3]变分推断具体可见周志华所著的机器学习相关章节, 这里略过.

[4]优化过程举例见原书 P646-648.

19.4.3 Continuous Latent Variables

当图模型中包含有连续隐变量时, 我们希望仍然可以使用变分推断以及变分学习的方式来最大化 $\mathcal{L}$. 但是, 由于隐变量是连续的, 所以不能像离散情形那样对每个离散值分别进行求导.

在大部分情形下, 我们并不需要解任何变分法. 设平均场近似估计为

$$q(\boldsymbol{h}|\boldsymbol{v}) = \prod_i q(h_i|\boldsymbol{v}) \tag{19.5}$$

对于所有 $j \neq i$ 的 $q(h_j|\boldsymbol{v})$ 视为定值, 那么 $q(h_i|\boldsymbol{v})$ 的最优值可通过归一化如下分布得到

$$\tilde{q}(h_i|\boldsymbol{v}) = \exp(\mathbb{E}_{\boldsymbol{h}_{-i}\sim q(\boldsymbol{h}_{-\mathbf{1}}|\boldsymbol{v})} \log \tilde{p}(\boldsymbol{v}, \boldsymbol{h})) \tag{19.6}$$

需要注意的是,p 在任何变量的联合分布中取值都不能为 0.

只有在使用新形式的变分学习时, 才会使用变分法.

19.4.4 Interactions between Learning and Inference

在学习算法中引入近似推断 (使用 q 去计算 $\mathcal{L}$) 将会影响学习过程, 然后学习过程又会影响推断算法的精度 (影响近似推断 q 的精度).

特别地, 学习算法倾向于将模型进行调整, 以使得近似推断算法暗含的近似假设更加贴近于现实分布.

泛函近似对于模型的真实影响很难量化.

19.5 Learned Approximate Inference

在之前的章节我们可以看到, 推断可以被视为增大 $\mathcal{L}$ 数值的优化过程. 这一过程的计算代价会很大而且会很耗时. 许多推断方法则会避免进行近似推断. 特别地, 我们可以将优化过程视为一个函数 f, 它将输入 $\boldsymbol{v}$ 映射到一个近似分布 $q^* = \arg\max_q \mathcal{L}(\boldsymbol{v}, q)$.

19.5.1 Wake-Sleep

训练一个模型从 $\boldsymbol{v}$ 到 $\boldsymbol{h}$ 进行推断的主要困难在于, 没有一个有监督训练集去训练这个模型.*wake-sleep* 算法则解决了这一难题. 同时它的缺点也很

明显, 它只能对那些已很高概率出现的样本进行训练.

19.5.2 Other Forms of Learned Inference

上述使用学习策略进行的近似推断还被应用于许多其他的模型中. 见原书 P652.

Chapter 20 Deep Generative Models

20.1 Boltzmann Machines

Boltzmann 机是一种基于能量的模型, 这也就意味着我们需要使用能量函数来定义联合概率分布

$$P(\boldsymbol{x}) = \frac{\exp(-E(\boldsymbol{x}))}{Z} \tag{20.1}$$

其中 Z 是 partition 函数,E 是能量函数

$$E(\boldsymbol{x}) = -\boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{U}\boldsymbol{x} - \boldsymbol{b}^{\boldsymbol{T}}\boldsymbol{x} \tag{20.2}$$

其中,$\boldsymbol{U}$ 是模型参数的权重矩阵,$\boldsymbol{b}$ 是偏置参数的向量.

从上式可以看出, 某个单元的概率可能由其它单元的概率线性组合而成, 这一个局限性. 而引入隐变量可以打破这种局限性. 如果引入隐变量, 可以实现对离散变量的概率分布进行全估计.

引入了隐变量之后, 能量函数定义为

$$E(\boldsymbol{x}) = -\boldsymbol{v}^{\boldsymbol{T}}\boldsymbol{R}\boldsymbol{v} - \boldsymbol{v}^{\boldsymbol{T}}\boldsymbol{W}\boldsymbol{h} - \boldsymbol{h}^{\boldsymbol{T}}\boldsymbol{S}\boldsymbol{h} - \boldsymbol{b}^{\boldsymbol{T}}\boldsymbol{v} - \boldsymbol{c}^{\boldsymbol{T}}\boldsymbol{h} \tag{20.3}$$

Boltamann Machine Learning Boltamann 机学习算法通常基于最大似然度, 需要对 partition 函数进行估计. 在学习过程中, 连接两个单元的权重更新仅仅与这两个单元的统计量 (如 $P_{model}(\boldsymbol{v})$,$\hat{P}_{data}(\boldsymbol{v})P_{model}(\boldsymbol{h}|\boldsymbol{v})$) 有关联. 也就是说, 学习时局部性的, 这在生物学上也是可以接受的.

20.2 Restricted Boltzmann Machines

与 Boltzmann 机一样,RBM 也是基于能量的模型

$$P(\mathbf{v} = \boldsymbol{v}, \mathbf{h} = \boldsymbol{h}) = \frac{1}{Z}\exp(-E(\boldsymbol{v}, \boldsymbol{h})) \tag{20.4}$$

其中能量 E 定义为

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\boldsymbol{b}^T\boldsymbol{v} - \boldsymbol{c}^T\boldsymbol{h} - \boldsymbol{v}^T\boldsymbol{W}\boldsymbol{h} \tag{20.5}$$

没有了 $\boldsymbol{v}$ 与 $\boldsymbol{v}$,$\boldsymbol{h}$ 与 $\boldsymbol{h}$ 的交互项;Z 是 partition 函数

$$Z = \sum_{\boldsymbol{v}} \sum_{\boldsymbol{h}} \exp\{-E(\boldsymbol{v}, \boldsymbol{h})\} \tag{20.6}$$

20.2.1 Conditional Distributions

虽然 $P(\boldsymbol{v})$ 很难计算, 但是 RBM 的二部图结构特点使得 $P(\mathbf{h}|\mathbf{v})$ 与 $P(\mathbf{v}|\mathbf{h})$ 可以相对容易地进行计算以及采样.

原书 P658 推理过程可得,

$$P(\boldsymbol{h}|\boldsymbol{v}) = \prod_{j=1}^{n_h} \sigma\Big((2\boldsymbol{h} - 1) \odot (\boldsymbol{c} + \boldsymbol{W}^T\boldsymbol{v})\Big)_j \tag{20.7}$$

$$P(\boldsymbol{v}|\boldsymbol{h}) = \prod_{i=1}^{n_v} \sigma\Big((2\boldsymbol{h} - 1) \odot (\boldsymbol{b} + \boldsymbol{W}^T\boldsymbol{h})\Big)_i \tag{20.8}$$

20.2.2 Training Restricted Boltzmann Machines

RBM 可以使用第18章中任意一种具有难以计算的 partition 函数的训练方式 (如 CD,SML,PCD,ratio matching 等) 进行训练.

20.3 Deep Belief Networks

深度信念网络 (deep belief network, DBN) 是具有多层隐变量的生成式模型. 隐变量是二值离散的, 可见单元既可以是离散的, 也可以是连续的. 图的最上两层是无向的, 其余各层是有向的, 箭头均指向离数据最近的层. 因此 DBN 的图模型是有向图和无向图的混合. 特别地, 仅由一层隐层的模型是 RBM.

深度信念网络的采样方法是, 最上两层隐层使用 Gibbs 采样, 后续层上使用 ancestral 采样.

深度信念网络使用逐层训练的方法进行训练

1. 使用 constractive divergence 或者 stochastic maximum likelihood 方法最大化 $\mathbb{E}_{\mathbf{v}\sim p_{data}} \log p(\boldsymbol{v})$ 来训练一个 RBM

2. 第二个 RBM 使用近似地最大化

$$\mathbb{E}_{\mathbf{v}\sim p_{data}} \mathbb{E}_{\mathbf{h}^{(1)}\sim p^{(1)}(\boldsymbol{h}^{(1)}|\boldsymbol{v})} \log p^{(2)}(\boldsymbol{h}(1)) \tag{20.9}$$

   其中 $p^{(1)}$ 是第一个 RBM 所代表的概率分布,$p^{(2)}$ 是第二个 RBM 所代表的概率分布.

3. 重复以上过程, 直到深度信念网络达到预期的层数.

在大多数应用中, 不对深度信念网络进行整体训练, 但可以通过 wake-sleep 算法进行参数的精细调节.

虽然深度信念网络是生成式模型, 但可以用来提升分类模型的效果. 我们可以使用深度信念网络的权重定义一个多层感知器网络 (MLP)

$$\begin{aligned} \boldsymbol{h}^{(1)} &= \sigma(b^{(1)} + \boldsymbol{v}^T \boldsymbol{W}^{(1)}) \\ \boldsymbol{h}^{(l)} &= \sigma(b_i^{(l)} + \boldsymbol{h}^{(l-1)T} \boldsymbol{W}^{(l)}) \end{aligned} \tag{20.10}$$

但是这样定义的多层感知器网络忽略了许多深度信念网络中的重要交互.

深度信念网络特制在最深层使用无向连接而在其它层使用指向数据的有向连接的模型. 需要特别注意与信念网络进行区分, 因为信念网络有时特指有向图模型.

20.4 Deep Boltzmann Machines

深度 *Boltzmann* 机 (deep Boltzmann machine, DBM) 是一种多隐层全无向图.

DBM 是一种基于能量的模型, 假设一个 Boltzmann 有一个可视层 $\boldsymbol{v}$, 三个隐层 $\boldsymbol{h}^{(1)}$,$\boldsymbol{h}^{(2)}$ 和 $\boldsymbol{h}^{(3)}$, 那么联合概率为

$$P(\boldsymbol{v}, \boldsymbol{h^{(1)}}, \boldsymbol{h^{(2)}}, \boldsymbol{h^{(3)}}) = \frac{1}{Z(\boldsymbol{\theta})} \exp\Big(- E(\boldsymbol{v}, \boldsymbol{h^{(1)}}, \boldsymbol{h^{(2)}}, \boldsymbol{h^{(3)}}; \boldsymbol{\theta})\Big) \tag{20.11}$$

其中能量定义为

$$E(\boldsymbol{v}, \boldsymbol{h^{(1)}}, \boldsymbol{h^{(2)}}, \boldsymbol{h^{(3)}}; \boldsymbol{\theta}) = -\boldsymbol{v^T W^{(1)} h^{(1)}} - \boldsymbol{h^{(1)T} W^{(2)} h^{(2)}} - \boldsymbol{h^{(2)} W^{(3)} h^{(3)}} \tag{20.12}$$

与 RBM 的能量定义式20.5相比,DBM 的能量包含了隐层之间的交互. 在后续章节我们可以看到, 这些交互一方面会影响模型所表现出的行为, 另外一方面也会影响模型的推断. 另外,DBM 奇偶层之间具有条件独立性. 这种二部图结构意味着我们可以使用与 RBM 相同的条件分布去决定 DBM 的条件分布. 同样由于其条件独立性,Gibbs 采样可以在 2 轮迭代内完成. 高效率的采样对训练中的随机量最大似然优化算法尤为重要.

20.4.1 Interesting Properties

与 DBN 相比,DBM 的后验概率更为简单, 但是 DBM 的后验概率有更为丰富的估计形式.DBN 下估计的下限不一定准确.

DBM 的一项缺点是其采样相对困难.

20.4.2 DBM Mean Field Inference

DBM 的后验概率使用变分推断很容易实现, 尤其是平均场近似.

在变分近似推断中, 我们使用一些简单的分布族去逼近一个特定的目标分布.

假设 $Q(\boldsymbol{h^{(1)}}, \boldsymbol{h^{(2)}}|\boldsymbol{v})$ 是 $P(\boldsymbol{h^{(1)}}, \boldsymbol{h^{(2)}}|\boldsymbol{v})$ 的近似估计, 根据平均场假设有

$$Q(\boldsymbol{h^{(1)}}, \boldsymbol{h^{(2)}}|\boldsymbol{v}) = \prod_j Q(h_j^{(1)}|\boldsymbol{v}) \prod_k Q(h_k^{(2)}|\boldsymbol{v}) \tag{20.13}$$

但是其缺点也很明显, 每次使用一个新的 $\boldsymbol{v}$ 的数值都要重新进行推断以确定 Q 的分布.

20.4.3 DBM Parameter Learning

DBM 学习过程面临两大挑战:partition 函数难以计算; 后验概率难以计算. 由于 DBM 包含 RBM, 所以 DBM 所遇到的困难同样是 RBM 所面临的困难.

20.4.4 Layer-Wise Pretraining

从一个随机初始值开始, 使用随机最大似然方法训练 DBM 通常会以失败告终. 最朴素同时也是最流行的方法是使用贪心逐层预训练方法. 贪心逐层训练方法不仅仅是坐标上升法[1], 它与坐标上升法有不同之处.

DBM 中间每一层都会有自顶向下和自底向上的输入, 因此需要对权重做相应的调整.

20.4.5 Jointly Training Deep Boltzmann Machines

DBM 在训练过程中的训练效果不容易追踪, 因为完整的 DBM 特性难以评估.Boltzmann 机顶部的 MLP 会让概率模型损失许多优势.

解决 DBM 训练难题有两种途径:*centered deep Boltzmann machine* 和 *multi-prediction deep Boltzmann machine*.

Centered Deep Boltzmann Machine 这种方法会产生一个不使用贪心逐层训练策略的模型, 但是它并不能像 MLP 那样经过正则化形成一个分类器.

Centered deep Boltzmann machine 引入向量 $\boldsymbol{\mu}$ 减去所有的状态

$$E'(\boldsymbol{x};\boldsymbol{U},\boldsymbol{b}) = -(\boldsymbol{x}-\boldsymbol{\mu})^{\boldsymbol{T}}\boldsymbol{U}(\boldsymbol{x}-\boldsymbol{\mu}) - (\boldsymbol{x}-\boldsymbol{\mu})^{\boldsymbol{T}}\boldsymbol{b} \tag{20.14}$$

其中,$\boldsymbol{\mu}$ 是一个超参数, 在训练开始时就固定下来.

Multi-Prediction Deep Boltzmann Machine 这种方法使用一种替代训练策略, 引入逆传播算法, 从而避免对梯度的 MCMC 估计.

MP-DBM 将平均场等式视为 recurrent 网络族用以近似解决所有可能的推断问题.

图推断引入逆传播算法有两项优势

- 训练过程和近似推断过程统一;
- 逆传播计算损失函数的精确梯度.

[1]一种优化问题求解方法.

20.5 Boltzmann Machines for Real-Valued Data

Boltzmann 机在最初是用于处理二值数据的, 但是随着诸如图像和音频数据的建模要求, 逐渐拓展至实值数据的范围. 本节笔记重点关注能量方程是如何定义的.

20.5.1 Gaussian-Bernoulli RBMs

实值数据最常见的形式是二值隐层, 实值可见层的 RBM, 其中可见单元的条件分布是高斯分布, 平均值是隐单元的函数. 因此能量函数定义为

$$E(\boldsymbol{v}, \boldsymbol{h}) = \frac{1}{2}\boldsymbol{v}^{\boldsymbol{T}}(\boldsymbol{\beta} \odot \boldsymbol{v}) - (\boldsymbol{v} \odot \boldsymbol{\beta})^{\boldsymbol{T}}\boldsymbol{W}\boldsymbol{h} - \boldsymbol{b}^{\boldsymbol{T}}\boldsymbol{h} \tag{20.15}$$

能量函数是由 RBM 的结构, 隐变量, 可见变量的分布共同推导出的.

20.5.2 Undirected Models of Conditional Covariance

Gaussian RBM 不能捕捉条件协方差信息. 基于这个缺陷, 就有了以下方法:

Mean and Covariance RBM mcRBM 使用隐单元独立地对所有可见单元的条件均值和方差进行编码. 能量函数定义为

$$E_{mc}(\boldsymbol{x}, \boldsymbol{h}^{(\boldsymbol{m})}, \boldsymbol{h}^{(\boldsymbol{c})}) = E_m(\boldsymbol{x}, \boldsymbol{h}^{(\boldsymbol{m})}) + E_c(\boldsymbol{x}, \boldsymbol{h}^{(\boldsymbol{c})}) \tag{20.16}$$

其中,E_m 是标准 Gaussian-Bernoulli RBM 能量方程

$$E_m(\boldsymbol{x}, \boldsymbol{h}^{(\boldsymbol{m})}) = \frac{1}{2}\boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{x} - \sum_j \boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{W}_{:j}h_j^{(m)} - \sum_j b_j^{(m)}h_j^{(m)} \tag{20.17}$$

E_c 是 cRBM 的能量方程, 用以对条件协方差进行建模

$$E_c(\boldsymbol{x}, \boldsymbol{h}^{(c)}) = \frac{1}{2}\sum_j h_j^{(c)}(\boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{r}^{(j)})^2 - \sum_j b_j^{(c)}h_j^{(c)} \tag{20.18}$$

Mean-Product of Student's t-distributions mPoT 的能量方程定义为

$$
\begin{aligned}
&E_{mPoT}(\boldsymbol{x}, \boldsymbol{h}^{(m)}, \boldsymbol{h}^{(c)}) \\
&\quad = E_m(\boldsymbol{x}, \boldsymbol{h}^{(m)}) + \sum_j \left(h_j^{(c)} \left(1 + \frac{1}{2} (\boldsymbol{r}^{(j)T} \boldsymbol{x})^2 \right) + (1 - \gamma_j) \log h_j^{(c)} \right)
\end{aligned}
\tag{20.19}
$$

其中,$\boldsymbol{r}^{(j)T}$ 是与 $h_j^{(c)}$ 相关的协方差权重向量;$E_m(\boldsymbol{x}, \boldsymbol{h}^{(m)})$ 的定义同式20.17.

Spike and Stab Restricted Boltzmann Machines ssRBM 模型的能量函数定义为

$$
\begin{aligned}
E_{ss}(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{h}) = &- \sum_i \boldsymbol{x}^{\boldsymbol{T}} \boldsymbol{W}_{:,is_ih_i} + \frac{1}{2} \boldsymbol{x}^T (\boldsymbol{\Lambda} + \sum_i \boldsymbol{\Phi_i} h_i \boldsymbol{x}) \\
&+ \frac{1}{2} \sum_i \alpha_i s_i^2 - \sum_i \alpha_i \mu_i s_i h_i - \sum_i b_i h_i + \sum_i \alpha_i \mu_i^2 h_i
\end{aligned}
\tag{20.20}
$$

其中 b_i 是 spike h_i 的偏移量;$\boldsymbol{\Lambda}$ 是观测量 $\boldsymbol{x}$ 的对角精度矩阵.

20.6 Convolutional Boltzmann Machines

基于能量的模型需要 pooling 操作来减少计算量, 但是策略并不明确.*Probabilistic max pooling* 就解决了这一问题, 其主要目的是对检测单元进行限制, 这样在同一时刻至多只有一个单元被激活.Probabilistic max pooling 操作可能使用有效的正则化手段, 也可能是限制模型 capacity overlapping 场景.

尽管构思很巧妙, 但是它的缺陷也很明显, 在实际中的表现并不好.

- 输入尺寸不能随意改变;
- 原始方式直接计算 pooling 的计算代价太大;
- 边缘需要使用 zero-pad 填充.

20.7 Boltzmann Machine for Structured or Sequential Outputs

表示 $\boldsymbol{y}$ 间各个元素间关系的一种方式是使用概率分布 $P(\mathbf{y}|\boldsymbol{x})$.

Boltzmann 即可以以 $P(\mathbf{x}^{(t)}|\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(t-1)})$ 相乘的形式完成条件概率建模任务.

20.8 Other Boltzmann Machines

Boltzmann 机也有许多其他类型的变体. 为了最大化 $\log p(y|\boldsymbol{v})$, 可以基于判别式去训练 RBM, 但是 RBM 并不是一种理想的有监督学习器.

能量方程中的高阶交互是一个热门的研究方向.

20.9 Back-Propagation through Random Operation

对于生成式模型, 希望对输入进行随机变换, 一种最直接的方法是扩充输入, 加入随机变量. 具体举例可见原书 P687-688. 这种技术经常被称为 *reparameterization trick*, *stochastic back-propagation* 或者 *perturbation analysis*.

20.9.1 Back-Propagating through Discrete Stochastic Operations

当一个模型输出一个离散变量 $\boldsymbol{y}$,reparameterization trick 方法就不可行了. 但是在这种情况下, 有两大问题, 一个是阶跃函数的在任意一点上的导数都不可用; 另一个更大的问题是在阶跃边界间的区域, 导数处处为 0.

REINFORCE 算法 (REward Increment = Non-negative Factor × Offset Reinforcement × Characteristic Eligibility) 提供了一个强大的解决方案. 其

核心思想在于, 尽管 $J(f(\boldsymbol{z};\boldsymbol{w}))$ 是一个阶跃函数, 并且其导数并没有太大用处, 但是其 $\mathbb{E}_{z\sim p(z)}J(f(\boldsymbol{z};\boldsymbol{w}))$ 是一个光滑函数, 并且可以使用梯度下降法.

这种方法的缺点是方差较大, 但是可以使用 variace reduction 方法消除此影响, 或者使用 variance normalization 方法消除此影响.

20.10 Directed Generative Nets

20.10.1 Sigmoid Belief Nets

Sigmoid belief 网络 (Sigmoid Belief Nets) 是一种有着特定条件概率分布的简单有向图模型. 通常情况下, 我们可以将 sigmoid belief 网络是为一个有二值状态 $\boldsymbol{s}$ 响亮的网络, 每一个状态元素受其祖先元素影响

$$p(s_i) = \sigma(\sum_{j<i} W_{j,i}s_j + b_i) \tag{20.21}$$

这种网络采样效率高, 但是推断操作效率低. 有两种解决方法:

- 对 sigmoid belief 网络建立一个不同的下界, 专门用来执行推断操作;
- 使用学习推断机制.

20.10.2 Differentiable Generator Nets

许多生成式模型就是基于 *differentiable generator network*, 该模型将样本的隐变量 $\mathbf{z}$ 转换为 $\mathbf{x}$ 或者基于 $\mathbf{x}$ 的分布. 通常包含三个部分

- variational autoencoders;
- generative adversarial networks;
- techniques that train generator networks in isolation.

Generator networks 本质上是参数化计算过程, 用以生成样本.

- 提供分布族用以采样;
- 通过参数从分布族中选取合适的分布.

一种方法是使用非线性变换 g 将关于 $\mathbf{z}$ 的分布转换成关于 $\mathbf{x}$ 的分布.

$$p_x(\boldsymbol{x}) = \frac{p_z(g^{-1}(\boldsymbol{x}))}{|\det(\frac{\partial g}{\partial z})|} \tag{20.22}$$

另一种方法是将 g 定义为一个以 $\mathbf{z}$ 为条件 $\mathbf{x}$ 为变量的条件分布. 通过边际化 $\mathbf{z}$ 求取关于 $\mathbf{x}$ 的分布.

上述两种方法的优劣互补. 相比于分类模型或者回归模型, 生成式模型的优化准则并不明确, 应为输入和输出并不明确. 这就要求 differentiable generator networks 必须有足够的 capacity, 这样优化算法才能得到足够好的优化结果. 这样难点就在于对应于每一个 $\boldsymbol{x}$ 的 $\boldsymbol{z}$ 在预先并不是已知且固定的.

20.10.3 Variational Autoencoders

Variational autoencoder(VAE) 是一种有向图模型, 使用学习策略进行近似推断, 并且可以基于梯度方法进行训练.VAE 的采样步骤为

1. 从 $p_{model}(z)$ 中采样一个样本 z;
2. 执行 differentiable generator network $g(z)$;
3. 最终从 $p_{model}(\boldsymbol{x}; g(z)) = p_{model}(\boldsymbol{x}|z)$ 中采样出 $\boldsymbol{x}$.

VAE 核心思想是训练一个参数化编码器, 用以产生 q 的参数.

VAE 的优势非常简洁, 理论上可解释以及易于实现. 此外, 这个框架还可以应用到其他范围的模型架构上.

20.10.4 Generative Adversarial Networks

GAN 基于博弈论, 生成网络必须与对手竞争. 生成网络生成样本, 分辨网络区分样本是真实样本还是生成的样本.

生成器网络 (generator network) 直接产生样本 $\boldsymbol{x} = g(\boldsymbol{z}; \boldsymbol{\theta}^{(g)})$; 判别器网络 (discriminator network) 区分样本的来源, 结果用 $d(\boldsymbol{x}; \boldsymbol{\theta}^{(d)})$ 表示. 这两者间是零和博弈,$v(\boldsymbol{\theta}^{(g)}, \boldsymbol{\theta}^{(d)})$ 决定了判别器的代价; 相应的, 生成器的代价为

$-v(\boldsymbol{\theta}^{(g)}, \boldsymbol{\theta}^{(d)})$. 在学习过程中, 每一方面都确保最大化自己的代价, 最终使得

$$g^* = \arg\min_g \max_d v(g, d) \tag{20.23}$$

收敛. 通常 v 采取的形式为

$$v(\boldsymbol{\theta}^{(g)}, \boldsymbol{\theta}^{(d)}) = \mathbb{E}_{\mathbf{x}\sim p_{data}} \log d(\boldsymbol{x}) + \mathbb{E}_{\mathbf{x}\sim p_{model}} \log(1 - d(\boldsymbol{x})) \tag{20.24}$$

最终在收敛时, 生成器的样本不可区分, 判别器也分不清样本的来源.

GAN 遇到的困难在于, 当 d 和 g 是神经网络时,$\max_d v(g, d)$ 是非凸的. 同时对两者执行梯度下降算法并不能保证达到平衡状态. 因此产生了替代零和博弈的策略, 即生成器增大对数概率犯错的概率.

GAN 学习算法的稳定性是一个开放问题, 但是可以通过精心选择模型结构和超参数得到解决.GAN 学习问题可以通过拆分生成过程加以简化.

GAN 框架也可以用与其他模型.

20.10.5 Generative Moment Matching Networks

Generative moment matching networks 不需要与其他网络进行配对. 它基于*动量匹配* (moment matching) 技术进行训练. 它的通过最小化 *maximum mean discrepancy* 损失函数达到训练的目的.

20.10.6 Convolutional Generative Networks

在处理图像的场景时, 将卷积结构引入生成网络. 为了使得 pooling 过程可逆, 引入 un-pooling 操作.

20.10.7 Auto-Regressive Networks

Auto-regression 网络是不包含隐变量的有向图模型. 它们将联合概率使用链式法则按照 $P(x_d|x_{d-1}, \cdots, x_1)$ 的形式进行分解.

20.10.8 Linear Auto-Regressive Networks

形势最简单的 auto-regressive 网络是没有隐藏单元和共享参数或者特征的.

20.10.9 Neural Auto-Regressive Networks

与 auto-regressive 有相同的图结构, 但是对条件概率有不同的参数化方法.

20.10.10 NADE

neural autoregressive density estimator(NADE) 是近期一种非常成功的 auto-regressive 网络.NADE 的连接与原始 auto-regressive 网络相同, 但是 NADE 引入了附加的共享参数策略. 使用共享参数策略的原因是 NADE 模型使用平均场推断方法去填充 RBM 输入中的缺失值. 如果拓展至连续情形, 则使用高斯混合模型.

auto-regressive 结构另外一个非常有趣的拓展是其避免使用任意阶数的观测变量.

20.11 Drawing Samples from Autoencoders

本节研究如何从 autoencoder 中进行采样.

20.11.1 Markov Chain Associated with any Denoising Autoencoder

P711 叙述了如何从估计分布中生成 Markov 链的步骤.

20.11.2 Clamping and Conditional Sampling

与 Boltzmann 机非常类似,denoising autoencoders 和它们的一般形式可以用来从条件概率分布 $p(\mathbf{x}_f|\mathbf{x}_o)$ 中进行采样. 其策略是截断观测变量 $\mathbf{x}_o$,

在给定 $\mathbf{x}_o$ 的条件下对 $\mathbf{x}_f$ 以及隐变量进行重采样.

20.11.3 Walk-Back Training Procedure

Walk-back 训练过程是加速生成式训练收敛的策略.

20.12 Generative Stochastic Networks

Generative stochastic networks(GSNs) 是 denoising autoencoder 的拓展, 在 Markov 链中引入了隐变量.

GSN 参数化过程可见 P715.

GSN 与传统概率模型的不同之处在于其对生成式过程本身进行参数化.

20.12.1 Discriminant GSNs

可以对 GSN 进行又该用以优化 $p(\mathbf{y}|\boldsymbol{x})$.

20.13 Other Generation Schemes

一种方法是使用热力学不等式提出 diffusion inversion 训练策略去学习生成式模型. 这种方法的核心思想在于我们希望从结构化的分布中进行采样. 因此, 对模型进行训练, 逐步还原分布的结构.

另外一种方法是使用 *approximate Baysian computation* 框架去采样.

20.14 Evaluating Generative Models

生成式模型间的对比比较困难的微妙, 只能近似对比, 但是要明确对比指标. 但是评价指标一般也很难确定. 原因有

- 输入数据的预处理方式可能会影响评价结果;

- 相同预处理方式的方法才能进行对比, 否则相似度度量空间就不是同一个了;
- 过拟合或者欠拟合的单独结果可能会非常好, 但是要从全局上进行评价;
- 样本视觉质量评价并不可靠, 常使用测试样本的对数似然度进行评价;
- 度量方式必须符合模型的使用意图.

即便如此, 所有的度量准则都有严重的缺陷.

20.15 Conclusion

生成式模型提供了解决 AI 系统问题的希望, 因为这种框架提供了一种解决不同直观概念的框架, 并且对这些概念以不确定性的方式进行推理.