*Deep Learning*读书笔记

黄帅

April 24, 2017

Contents

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

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

其中$\mathbf{V}$是$\mathbf{A}$所有特征向量组成的矩阵；$\text{diag}(\lambda)$是$\mathbf{V}$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

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

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

$\mathbf{A}$的左奇异值向量是$\mathbf{A}\mathbf{A}^T$对应的特征向量；$\mathbf{A}$的右奇异值向量是$\mathbf{A}^T\mathbf{A}$对应的特征向量；非零奇异值是$\mathbf{A}\mathbf{A}^T$或$\mathbf{A}^T\mathbf{A}$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

A的*Moore-Penrose*广义逆被定义为

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

但是通常情况下采用如下的方法进行计算

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

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

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

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

其中β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：

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

其中，$\mathbf{x}^{(i)}$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$\mathbf{x}=x$的自信息定义为

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

我们通常使用*Shannon entropy*定义一个概率分布的不确定性

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

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

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

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

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

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$\mathbf{J}\in\mathbb{R}^{m\times n}$.其中,

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

如果求二次偏导,则有**Hessian矩阵**,即

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

如果想计算某一个方向上的二阶偏导,则有$\mathbf{d}^T\mathbf{H}\mathbf{d}$其中,$\mathbf{d}$是一单位向量.特别的,如果$\mathbf{d}$是$\mathbf{H}$的特征向量,则$\mathbf{d}$方向上的二阶偏导就是$\mathbf{H}$对应于$\mathbf{d}$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$f(\mathbf{x})$做二阶Taylor展开,有

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

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

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

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$\mathbf{g}^T\mathbf{H}\mathbf{g}$等于或者小于0时,$\epsilon$可以取到很大的值;当$\mathbf{g}^T\mathbf{H}\mathbf{g}$大于0时,有

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

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

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

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

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

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

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

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$p(\mathbf{x})$,有监督学习所学到的模型是根据样本$\mathbf{x}$求其对应样本标记$\mathbf{y}$的分布$p(\mathbf{y}|\mathbf{x})$.

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$\mathbf{x} \in \mathbb{R}^n$,根据概率链式法则

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

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$p(\mathbf{x}, \mathbf{y})$,进而求得$p(\mathbf{y}|\mathbf{x})$.

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

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)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

[1]9.7和10涉及到异构特征的组织方式.

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)[2].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[3].

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$P(\mathbf{x}, y)$之间的误差被称为贝叶斯误差(Bayes error)[4]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,会导致所有的学习算法取得同样的错误率.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.

5.2.2 Regularization

通常的机器学习算法中都包含了偏好.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.

[2]被假设空间打散的最大示例集大小.
[3]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.
[4]开个脑洞:真理与认识之间的差异.

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.

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

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

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θ.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$y = f(\mathbf{x}) + \epsilon$,其中$\epsilon$是$y$不能从$\mathbf{x}$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

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

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

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

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$P(|\hat{\theta}_m - \theta| > \epsilon) \to 0$.

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$p_{\text{model}}(\mathbf{x};\theta)$是由样本估计出的整体分布,以$\theta$为参数.其目的就是将$\mathbf{x}$映射到真实的分布$p_{\text{data}}(\mathbf{x})$.极大似然度估计定义为

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

进行适当的尺度变换和归一化后,可以得到

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

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θ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:

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

令上式最小,即最小化

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

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.

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

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$p_{model}(\cdot;\theta)$中;
2. 真实分布p_{data}对应的θ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.

确定参数θ的过程首先选取一个不确定性最大即熵最大的分布$p(\theta)$,然后通过观测样本,利用贝叶斯公式求取θ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θ进行点估计,而贝叶斯估计根据θ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.

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

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

5.7.1 Probabilistic Supervised Learning

有监督学习可以等价为求取后验概率$p(y|\mathbf{x})$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.这也是解决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).

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

其中$\mathcal{N}(\mathbf{x}; \mu, \Sigma)$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,不计算算是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[5],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.

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.

[5]抽空总结一下

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θ来估计映射函数$y = f(\mathbf{x})$.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$p(\mathbf{y}|\mathbf{x};\theta)$,则使用交叉熵作为损失函数(6.2.1).如果模型是为了估计$\mathbf{y}$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

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

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$p(\mathbf{y}|\mathbf{x})$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$\mathbf{y}$的均值$\mathbf{x}$,则可以解下面的问题

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

等价于

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

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

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

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

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

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

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

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

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

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$\mathbf{z} = \mathbf{W^T h} + \mathbf{b}$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$\mathbf{z}$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$\mathbf{y}$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$f(\mathbf{x};\theta)$,与预测值并无直接关系.但它为y的分布提供参数.**假设有$f(\mathbf{x};\theta)=\mathbf{w}$,则损失函数可以被写成$-\log p(\mathbf{y};\mathbf{w}(\mathbf{x}))$.

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

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

从经验上来说,通常把$\mathbf{b}$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$h_i = \exp\left(-\frac{1}{\delta^2}\|\mathbf{W}_{:,\mathbf{i}} - \mathbf{x}\|^2\right)$
- 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

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

其中$\mathbf{v}$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$\mathbf{He}^{(i)}$,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)$\Omega(\theta)$达到限制模型capacity的目的.正则化后的目标函数为

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

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

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

对应的权重$\mathbf{w}$更新公式为

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

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

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

可以看出,与$\mathbf{H}$的第i个特征向量对应的$\mathbf{w}^*$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

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

对应的对$\mathbf{w}$导数为0的点为

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

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

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

相应的目标函数为

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

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$\mathbf{H} = diag([H_{1,1},\cdots,H_{n,n}])$,其中$H_{i,i} > 0$.有

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

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

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

相应的,解为

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

定性地分析,最优值α^*会使$\Omega(\theta)$的值不断减小,但还没有到达让$\Omega(\theta)$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$\log p(\theta)$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$J(\theta)$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$\Omega(\theta) < k$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$\log p(y|\mathbf{x})$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$P(\mathbf{x})$中采样得到的未标记样本和从$P(\mathbf{x},\mathbf{y})$中得到的有标记样本共同预测$P(\mathbf{y}|\mathbf{x})$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$P(\mathbf{x},\mathbf{y})$或$P(\mathbf{x})$和$P(\mathbf{y}|\mathbf{x})$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

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

其中,$\Omega(\mathbf{h})$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

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

其中,μ是*mask vector*,用以指定被包含进子模型的神经单元;$J(\theta; \mu)$定义了基于模型参数θ和μ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

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

其中$p(\mu)$是训练时mask vector的概率密度函数.其次在实际中,只需要对μ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.

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

进行归一化,有

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

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$\mathcal{N}(1, \mathbf{I})$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.

[3]The probability becoming part of the sample during the drawing of a single sample.
[4]TODO: figure it out.

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

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

使得$f(\mathbf{x})$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16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

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

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

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

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)

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

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

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

就有可能存在$\frac{1}{2}\epsilon^2\mathbf{g}^T\mathbf{H}\mathbf{g} > \epsilon\mathbf{g}^T\mathbf{g}$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*

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

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$\mathbf{v}$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

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

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$\mathbf{f}(t)$和速度$\mathbf{v}(t)$有关联.

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

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$-\nabla_\theta J(\theta)$;另一个力(粘滞阻力)正比于$-\mathbf{v}(t)$.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.

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

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$J(\theta)$在θ_0附近进行二阶展开,有

$$J(\theta) \approx J(\theta_0) + (\theta - \theta_0)^T \nabla_\theta J(\theta_0) + \frac{1}{2}(\theta - \theta_0)^T \mathbf{H}(\theta - \theta_0) \tag{8.12}$$

其中$\mathbf{H}$是J对θ在θ_0点附近的Hessian矩阵.如果求取critical point,有

$$\theta^* = \theta_0 - \mathbf{H}^{-1} \nabla_\theta J(\theta_0) \tag{8.13}$$

因此对于二次函数,通过缩放$\mathbf{H}^{-1}$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

$$\theta^* = \theta_0 - [H(f(\theta_0)) + \alpha \mathbf{I}]^{-1} \nabla_\theta f(\theta_0) \tag{8.14}$$

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$\mathbf{d}_t$的迭代形式为

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

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

$$\beta_t = \frac{\nabla_\theta J(\theta_t)^T \nabla_\theta J(\theta_t)}{\nabla_\theta J(\theta_{t-1})^T \nabla_\theta J(\theta_{t-1})} \tag{8.16}$$

Polak-Ribiere

$$\beta_t = \frac{(\nabla_\theta J(\theta_t) - \nabla_\theta J(\theta_{t-1}))^T \nabla_\theta J(\theta_t)}{\nabla_\theta J(\theta_{t-1})^T \nabla_\theta J(\theta_{t-1})} \tag{8.17}$$

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$\mathbf{H}^{-1}$,得到估计量$\mathbf{M}_t$.并且通过$\mathbf{M}_t$决定优化方向$\rho_t = \mathbf{M}_t \mathbf{g}_t$

$$\theta_{t+1} = \theta_t + \epsilon^* \rho_t \tag{8.18}$$

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$\mathbf{M}^{(t-1)}$初始值为单位矩阵,并且在迭代过程中仅存储用来更新$\mathbf{M}$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.

[5]TODO: 看相关论文,深化理解.

假设$\mathbf{H}$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$\mathbf{H}$,将其替换为

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

其中,μ是神经元的均值向量;σ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μ,σ.

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$\mathbf{H}'$替换为$\gamma\mathbf{H}' + \beta$.其中$\gamma$和$\beta$通过学习得到.与原始的$\mathbf{H}$表示不同,新的表示形势下,均值仅由参数$\beta$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

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

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.