第六章　网络正则化与优化

因为深度神经网络的表达能力很强，所以很容易产生过拟合。另外，大量的参数会导致训练比较慢。在训练深度神经网络时，同时也需要掌握一定的技巧。目前，人们在大量的实践中总结了一些经验技巧，可以从以下几个方面来提高学习效率并得到一个好的网络模型：1）数据增强；2）数据预处理；3）网络参数初始化；4）正则化；5）超参数优化等。

6.1　数据增强

深层神经网络一般都需要大量的训练数据才能获得比较理想的结果。在数据量有限的情况下，可以通过**数据增强**（Data Augmentation）来增加数据量，提高模型鲁棒性，避免过拟合。目前，数据增强还主要应用在图像数据上，在文本等其它类型的数据还没有太好的方法。

图像数据的增强主要是通过算法对图像进行转变，引入噪声等方法来增加数据的多样性。增强的方法主要有几种：

- 旋转（Rotation）：将图像按顺时针或逆时针方向随机旋转一定角度；
- 翻转（Flip）：将图像沿水平或垂直方法随机翻转一定角度；
- 缩放（Zoom In/Out）：将图像放大或缩小一定比例；
- 平移（Shift）：将图像沿水平或垂直方法平移一定步长；
- 加噪声（Noise）：加入随机噪声。

6.2 数据预处理

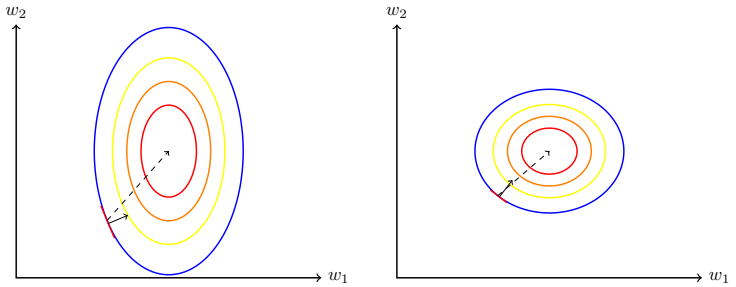
一般而言，原始的训练数据中，每一维特征的来源以及度量单位不同，会造成这些特征值的分布范围往往差异很大。当我们计算不同样本之间的欧式距离时，取值范围大的特征会起到主导作用。这样，对于基于相似度比较的机器学习方法（比如最近邻分类器），必须先对样本进行预处理，将各个维度的特征归一化到同一个取值区间，并且消除不同特征之间的相关性，才能获得比较理想的结果。虽然神经网络可以通过参数的调整来适应不同特征的取值范围，但是会导致训练效率比较低。

假设一个只有一层的网络$y = \tanh(w_1x_1 + w_2x_2 + b)$，其中$x_1 \in [0, 10]$，$x_2 \in [0, 1]$。之前我们提到$\tanh()$的导数在区间$[-2, 2]$上是敏感的，其余的导数接近于0。因此，如果$w_1x_1 + w_2x_2 + b$过大或过小，都会导致梯度过小，难以训练。为了提高训练效率，我们需要使$w_1x_1 + w_2x_2 + b$在$[-2, 2]$区间，我们需要将w_1设得小一点，比如在$[-0.1, 0.1]$之间。可以想象，如果数据维数很多时，我们很难这样精心去选择每一个参数。因此，如果每一个特征的取值范围都在相似的区间，比如$[0, 1]$或者$[-1, 1]$，我们就不太需要区别对待每一个参数，减少人工干预。

除了参数初始化之外，不同特征取值范围差异比较大时还会梯度下降法的搜索效率。图6.1给出了数据归一化对梯度的影响。其中，图6.1a为未归一化数据的等高线图。取值范围不同会造成在大多数位置上的梯度方向并不是最优的搜索方向。当使用梯度下降法寻求最优解时，会导致需要很多次迭代才能收敛。如果我们把数据归一化为取值范围相同，如图6.1b所示，大部分位置的梯度方向近似于最优搜索方向。这样，在梯度下降求解时，每一步梯度的方向都基本指向最小值，训练效率会大大提高。

归一化的方法有很多种，比如之前我们介绍的sigmoid型函数等都可以将不同取值范围的特征挤压到一个比较受限的区间。这里，我们介绍几种在神经网络中经常使用的归一化方法。

标准归一化 **标准归一化**也叫z-score归一化，来源于统计上的标准分数。将每一个维特征都处理为符合标准正态分布（均值为0，标准差为1）。假设有N个



(a) 未归一化数据的梯度 (b) 归一化数据的梯度

图 6.1: 数据归一化对梯度的影响。

样本 $\{\mathbf{x}^{(i)}\}, i = 1, \cdots, N$，对于每一维特征 x，我们先计算它的均值和标准差：

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x^{(i)}, \tag{6.1}$$

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x^{(i)} - \mu)^2. \tag{6.2}$$

然后，将特征 $x^{(i)}$ 减去均值，并除以标准差，得到新的特征值 $\hat{x}^{(i)}$。

$$\hat{x}^{(i)} = \frac{x^{(i)} - \mu}{\sigma}, \tag{6.3}$$

这里，σ 不能为0。如果标准差为0，说明这一维特征没有任务区分性，可以直接删掉。

在标准归一化之后，每一维特征都服从标准正态分布。

缩放归一化 另外一种非常简单的归一化是通过缩放将特征取值范围归一到 $[0, 1]$ 或 $[-1, 1]$ 之间：

$$\hat{x}^{(i)} = \frac{x^{(i)} - \min(x)}{\max(x) - \min(x)}, \tag{6.4}$$

其中，$\min(x)$ 和 $\max(x)$ 分别为这一维特征在所有样本上的最小值和最大值。

PCA 使用 PCA (Principal Component Analysis) 方法可以去除掉各个成分之间的相关性。

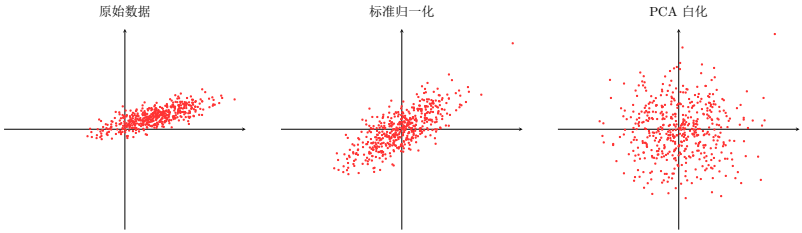


图 6.2: 数据归一化示例

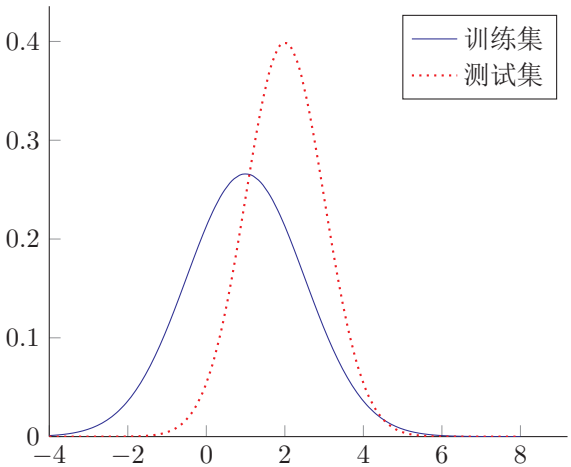


图 6.3: 协变量偏移。

6.3 批量归一化

在传统机器学习中，一个常见的问题的**协变量偏移**（Covariate Shift）。协变量是一个统计学概念，是可能影响预测结果的变量。在机器学习中，协变量可以看作是输入变量。一般的机器学习算法都要求输入变量在训练集和测试集上的分布是相似的。如果不满足这个要求，这些学习算法在测试集的表现会比较差。

在多层的神经网络中，中间某一层的输入是其前面网络的输出。因为前面网络的参数在每次迭代时也会被更新，而一旦这些参数变化会造成这一个中间层的输入也发生变化，其分布往往会和参数更新之前差异比较大。换句话说，从这一个中间层开始，之后的网络参数白学了，需要重新学习。这个中间层的深度

很大时，这种现象就越明显。这种现象叫做**内部协变量偏移**（Internal Covariate Shift）。

为了解决这个问题，通过对每一层的输入进行归一化使其分布保存稳定。这种方法称为**批量归一化**方法（Batch Normalization）[Ioffe and Szegedy, 2015]。

归一化方法可以采用第6.2节中介绍的几种归一化方法。因为每一层都要进行归一化，所以要求归一化方法的速度要快。这样，PCA 白化的归一化方法就不太合适。为了提高归一化效率，这里使用标准归一化，对每一维特征都归一到标准正态分布。相当于每一层都进行一次数据预处理，从而加速收敛速度。

因为标准归一化会使得输入的取值集中的0附近，如果使用sigmoid型激活函数时，这个取值区间刚好是接近线性变换的区间，减弱了神经网络的非线性性质。因此，为了使得归一化不对网络的表示能力造成负面影响，我们可以通过一个附加的缩放和平移变换改变取值区间。从最保守的角度考虑，可以通过来标准归一化的逆变换来使得归一化后的变量可以被还原为原来的值。

$$y_k = \gamma_k \hat{x}_k + \beta_k, \tag{6.5}$$

这里 γ_k 和 β_k 分别代表缩放和平移的参数。当 $\gamma_k = \sqrt{\sigma[x_k]}$，$\beta_k = \mu[x_k]$ 时，y_k 即为原始的 x_k。

当训练完成时，用整个数据集上的均值 μ_k 和方差 σ_k^2 来分别代替 $\mu_{\mathcal{B},k}$ 和方差 $\sigma_{\mathcal{B},k}^2$。

通过每一层的归一化，从而减少前面网络参数更新对后面网络输入带来的内部协变量偏移问题，提高训练效率。

6.4 参数初始化

神经网络的训练过程中的参数学习是基于梯度下降法进行优化的。梯度下降法需要在开始训练时给每一个参数赋一个初始值。这个初始值的选取十分关键。在感知器和logistic回归的训练中，我们一般将参数全部初始化为0。但是这在神经网络的训练中会存在一些问题。因为如果参数都为0，在第一遍前向计算时，所有的隐层神经元的激活值都相同。这样会导致深层神经元没有区分性。这种现象也称为**对称权重**现象。

为了打破这个平衡，比较好的方式是对每个参数都随机初始化，这样使得不同神经元之间的区分性更好。

算法 6.1: 批量归一化

输入: 一次 mini-batch 中的所有样本集合:

$\mathcal{B} = \{\mathbf{x}^{(i)}\}, i = 1, \cdots, m;$

参数: γ, β;

for $k = 1 \cdots K$ **do**

$$\mu_{\mathcal{B},k} = \frac{1}{m}\sum_{i=1}^{m} x_k^{(i)}, \qquad \text{// mini-batch 均值}$$

$$\sigma^2_{\mathcal{B},k} = \frac{1}{m}\sum_{i=1}^{m}(x_k^{(i)} - \mu_{\mathcal{B},k})^2. \qquad \text{//mini-batch 方差}$$

$$\hat{x}_k^{(i)} = \frac{x_k^{(i)} - \mu_{\mathcal{B},k}}{\sqrt{\sigma^2_{\mathcal{B},k} + \epsilon}}, \forall i \qquad \text{//归一化}$$

$$y_k^{(i)} = \gamma \hat{x}_k^{(i)} + \beta \equiv \mathbf{BN}_{\gamma,\beta}(\mathbf{x}^{(i)}), \forall i \qquad \text{//缩放和平移}$$

end

输出: $\{y^{(i)} = \mathbf{BN}_{\gamma,\beta}(\mathbf{x}^{(i)})\}$

但是一个问题是如何选取随机初始化的区间呢？如果参数太小，会导致神经元的输入过小。经过多层之后信号就慢慢消失了。参数过小还会使得 sigmoid 型激活函数丢失非线性的能力。以 logistic 函数为例，在 0 附近基本上是近似线性的。这样多层神经网络的优势也就不存在了。如果参数取得太大，会导致输入状态过大。对于 sigmoid 型激活函数来说，激活值变得饱和，从而导致梯度接近于 0。

因此，如果要高质量地训练一个网络，给参数选取一个合适的初始化区间是非常重要的。经常使用的初始化方法有以下几种：

Gaussian 初始化方法 Gaussian 初始化方法是最简单的初始化方法，参数从一个固定均值（比如 0）和固定方差（比如 0.01）的 Gaussian 分布进行随机初始化。

Xavier 初始化方法中，Xavier 是发明者 Glorot 的名字。

Xavier 初始化方法 Glorot and Bengio [2010] 提出一个初始化方法，参数可以在区间 $[-r, r]$ 内采用均匀分布进行初始化。

对于 logistic 函数，第 $l-1$ 到 l 层的权重，

$$r = \sqrt{\frac{6}{n^{l-1} + n^1}}, \tag{6.6}$$

这里 n^l 是第 l 层神经元个数，n^{l-1} 是第 $l-1$ 层神经元个数。

对于 tanh 函数，

$$r = 4\sqrt{\frac{6}{n^{l-1} + n^l}}. \tag{6.7}$$

假设第 l 层的一个隐藏层神经元 z^l，其接受前一层的 n^{l-1} 个神经元的输出 $a_i^{(l-1)}$，$i \in [1, n^{(l-1)}]$，

$$z^l = \sum_{i=1}^{n} w_i^l a_i^{(l-1)} \tag{6.8}$$

为了避免初始化参数使得激活值变得饱和，我们需要尽量使得 z^l 处于激活函数的线性区间，也就是其绝对值比较小的值。这时，该神经元的激活值为 $a^l = f(z^l) \approx z^l$。

假设 w_i^l 和 $a_i^{(l-1)}$ 都是相互独立，并且均值都为0，则 a 的均值为

$$E(a^l) = E(\sum_{i=1}^{n} w_i^l a_i^{(l-1)}) = \sum_{i=1}^{d} E(\mathbf{w}_i) E(a_i^{(l-1)}) = 0. \tag{6.9}$$

a^l 的方差为

$$\begin{aligned} \mathrm{Var}(a^l) &= \mathrm{Var}(\sum_{i=1}^{n^{(l-1)}} w_i^l a_i^{(l-1)}) && (6.10)\\ &= \sum_{i=1}^{n^{(l-1)}} \mathrm{Var}(w_i^l)\,\mathrm{Var}(a_i^{(l-1)}) && (6.11)\\ &= n^{(l-1)}\,\mathrm{Var}(w_i^l)\,\mathrm{Var}(a_i^{(l-1)}). && (6.12) \end{aligned}$$

也就是说，输入信号的方差在经过该神经元后被放大或缩小了 $n^{(l-1)}\,\mathrm{Var}(w_i^l)$ 倍。为了使得在经过多层网络后，信号不被过分放大或过分减弱，我们尽可能保持每个神经元的输入和输出的方差一致。这样 $n^{(l-1)}\,\mathrm{Var}(w_i^l)$ 设为1比较合理，即

$$\mathrm{Var}(w_i^l) = \frac{1}{n^{(l-1)}}. \tag{6.13}$$

同理，为了使得在反向传播中，误差信号也不被放大或缩小，需要将 w_i^l 的方差保持为

$$\mathrm{Var}(w_i^l) = \frac{1}{n^{(l)}}. \tag{6.14}$$

作为折中，同时考虑信号在前向和反向传播中都不被放大或缩小，可以设置

$$\mathrm{Var}(w_i^l) = \frac{2}{n^{(l-1)} + n^{(l)}}. \tag{6.15}$$

假设随机变量 x 在区间 $[a, b]$ 内均匀分布，则其方差为：

$$\mathrm{Var}(x) = \frac{(b-a)^2}{12}. \tag{6.16}$$

因此，若让 $w_i^l \in [-r, r]$，则 r 的取值为

$$r = \sqrt{\frac{6}{n^{l-1} + n^1}}. \tag{6.17}$$

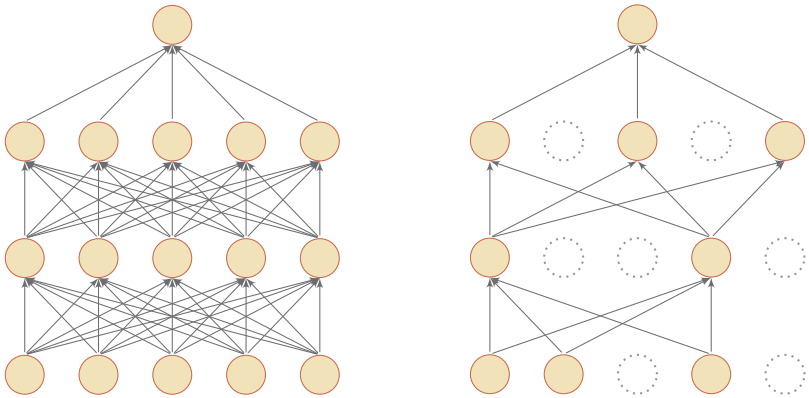
6.5 正则化

深度神经网络很容易产生过拟合现象，因为增加的抽象层使得模型能够对训练数据中较为罕见的依赖关系进行建模。对此，权重递减（ℓ_2 正规化）或者稀疏（ℓ_1 -正规化）等方法可以利用在训练过程中以减小过拟合现象 [Bengio et al., 2013]。

6.6 Dropout

训练一个大规模的神经网络经常容易过拟合。过拟合在很多机器学习中都会出现。一般解决过拟合的方法有正则化、早期终止、集成学习以及使用验证集等。Srivastava et al. [2014] 提出了适用于神经网络的避免过拟合的方法，叫**dropout**方法（丢弃法），即在训练中随机丢弃一部分神经元（同时丢弃其对应的连接边）来避免过拟合。图6.4给出了 dropout 网络的示例。

每做一次 dropout，相当于从原始的网络中采样得到一个更瘦的网络。如果一个神经网络有 n 个神经元，那么总共可以采样出 2^n 个子网络。每次迭代都相



(a) 标准网络 (b) Dropout 神经网络

图 6.4: Dropout 神经网络模型

当于训练一个不同的子网络，这些子网络都共享原始网络的参数。那么，最终的网络可以近似看作是集成了指数级个不同网络的组合模型。每次选择丢弃的神经元是随机的。最简单的方法是设置一个固定的概率p。对每一个神经元都一个概率p来判定要不要保留。p可以通过验证集来选取一个最优的值。或者，p也可以设为0.5，这对大部分的网络和任务有比较有效。

一般情况下 dropout 是针对神经元进行随机丢弃，但是也可以扩展到对每条边进行随机丢弃，或每一层进行随机丢弃。

当$p = 0.5$时，在训练时有一半的神经元被丢弃，只剩余一半的神经元是可以激活的。而在测试时，所有的神经元都是可以激活的。因此每个神经元训练时的净输入值平均比测试时小一半左右。这会造成训练和测试时网络的输出不一致。为了缓解这个问题，在测试时需要将每一个神经元的输出都折半，也相当于把不同的神经网络做了平均。

一般来讲，对于隐藏层的神经元，其 dropout 率等于 0.5 时效果最好，因为此时通过 dropout 方法，随机生成的网络结构最具多样性。对于输入层的神经元，其 dropout 率通常设为更接近 1 的数，使得输入变化不会太大。当对输入层神经元进行 dropout 时，相当于给数据增加噪声，以此来提高网络的鲁棒性。

6.7 超参数优化

在构建前馈神经网络时，有下面的超参数需要设置。

- 网络层数
- 每层的神经元数量
- 激活函数的类型
- 学习率（以及动态调整算法）
- 正则化系数
- 每次小批量梯度下降的

超参数的优化问题是一个组合优化问题，没有通用的优化方法。对于超参数的设置，一般用**网格搜索**（Grid Search）或者人工搜索的方法来进行。假设总共有 K 个超参数，第 k 个超参数的可以取 m_k 个值。如果参数是连续的，可以将参数离散化，选择几个“经验”值。比如学习率 α，我们可以设置

$$\alpha \in \{0.01, 0.1, 0.5, 1.0\}.$$

这样，这些超参数可以有 $m_1 \times m_2 \times \cdots \times m_K$ 个取值组合。所谓网格搜索就是根据这些超参数的不同组合分别训练一个模型，然后评价这些模型在检验数据集上的性能，选取一组性能最好的组合。

如果每个超参数对模型性能的影响有很大差异，有些超参数对模型性能的影响有限，而有些则非常大。在这种情况下，网格搜索可能会有些浪费。采用随机搜索会比网格搜索更加有效，而且在实践中也更容易实现 [Bergstra and Bengio, 2012]。

6.8 模型压缩

待写...

参考文献

Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. Ad-

vances in optimizing recurrent networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8624–8628. IEEE, 2013.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256, 2010.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.