深度学习笔记

陈鸿峥

2020.03*

目录

本课程主要选用Ian Goodfellow, Yoshua Bengio, Aaron Courville的《深度学习》(Deep Learning)一书。一些众所周知的概念会在笔记中略过，而着重在一些容易忽略的知识点上面。

1 简介

图 1反映了**深度学习**与其他几个常见概念之间的关系。传统的**机器学习**（如决策树、SVM、随机森林等）常需要人工提取特征，这一步经常涉及到**特征工程**(feature engineering)，如果特征没有进行一定处理，直接丢进去让其学习，往往会产生非常糟糕的结果。在一种表示下可能可以对数据进行线性二分，而另一种表示下则没有办法。因此，为了避免对特征的强依赖性，一种方法是利用机器学习来学习**表示(representation)本身**，再将新的表示送入到后面的学习器中让它学习**表示到输出的映射**，此即**表示学习**。再到后来，深度学习则更加将这种思想发扬光大，表示学习只能学习到**浅层简单的特征**，那深度学习则尝试去学习**深层复杂的特征**。

*Build 20200324

事实上现在**图神经网络**(GNN)也是遵循这样的发展过程，最开始尝试在图上做机器学习[1, 2, 3]；然后又开始在图上以各种随机游走的方式做图表示学习-图嵌入(embedding)[4, 5]；后来发现图嵌入能够获得的特征依然太浅层了，因此现在更多则采用图神经网络[6, 7, 8, 9]的方式来做图相关的工作。

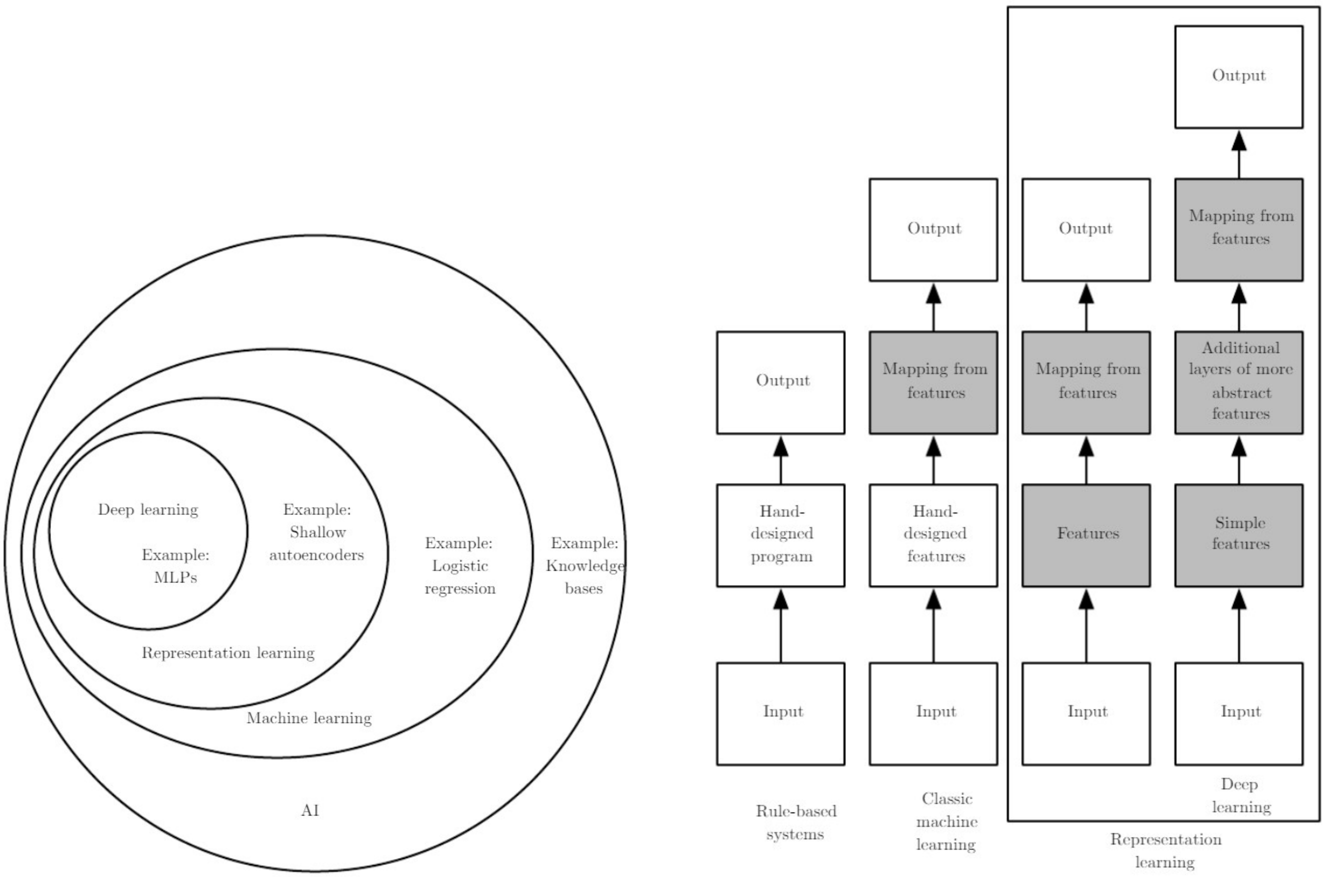


图 1: 深度学习Venn图

深度学习在发展过程中也起过几个名字：在1940年代到1960年代被称为**控制论**(cybernetics)，之后1980到1990年代则被称为**连接主义**(connectionism)，而后从2006年到现在才被称为**深度学习**。2018年的图灵奖正式颁发给深度学习三巨头—Geoffrey Hinton，Yoshua Bengio和Yann LeCun，也奠定了深度学习在学术界的历史地位。

我也一直在思考是什么造成了深度学习在2010年代的兴起，使得如今我们快速进入软件2.0时代[10]。总结来讲有以下几点：

- **数据**：我们每天产生的数据量越来越多，能够处理的数据量越来越大，名副其实地进入了大数据时代。其中的一些数据经过处理能够变得很干净，2010年拥有大量标注图片的数据集ImageNet就是这样的例子，它的提出使得有监督学习迎来了一波兴起。
- **算法**：我们有更多更优秀的模型，从AlexNet的dropout，再到后来ResNet的残差模块。ImageNet的发展历程，也是深度学习算法/模型改进和发展的历程。
- **软件系统**：早年的深度学习框架Caffe、Theano、MXNet等使得研究人员可以方便地编写神经网络模型，而2015年前后诞生的TensorFlow和PyTorch则是成为了现在深度学习框架的主流范式，程序员不需知道底层的实施细节，只用“调库”和“调参”也可以实现很高效的模型，这些软件系统的诞生极大程度推动了深度学习的遍地开花。再到现在XLA和TVM等深度学习编译器的出现，更是进一步解放程序员，使上层模型可以方便快捷地部署到后端不同硬件平台上。
- **硬件**：GPU为深度学习做出了不可磨灭的贡献，没有GPGPU的发展，很多深度学习任务根本没有办法完成。现在的TPU及各种神经网络加速器也都是在拓宽这一层面，以进一步提升深度学习的能力。

上面提到的这几点都不可或缺，它们共同造就了整个深度学习栈，从而带来现在深度学习的繁荣。

2 人工神经网络基础

2.1 神经元与多层神经网络

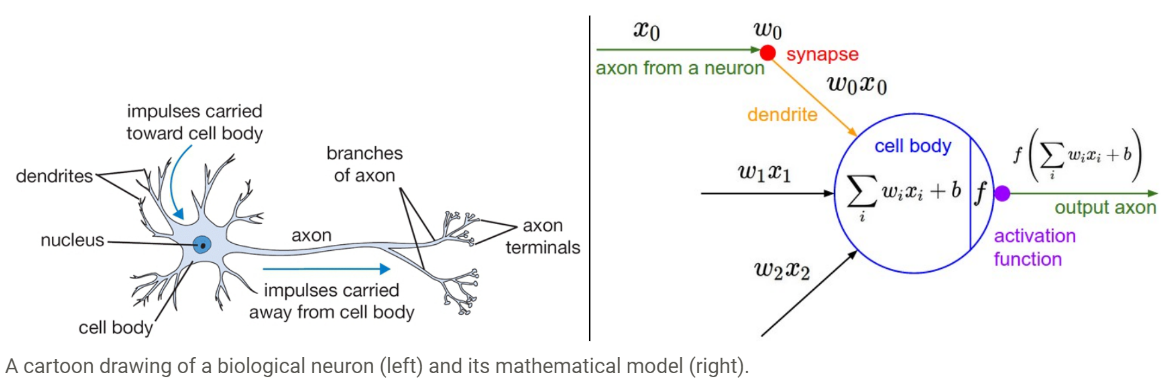


图 2: 神经元(neuron)

常见的激活函数：

- Sigmoid：$g(x) = \frac{1}{1+\mathrm{e}^{-x}}$
- Tanh：$g(x) = \tanh(x) = \frac{\mathrm{e}^{x}-\mathrm{e}^{-x}}{\mathrm{e}^{x}+\mathrm{e}^{-x}}$

- ReLU：$g(x) = \max(0, x)$

之所以要采用非线性激活函数，是因为它可以使神经网络也变成非线性的，进而捕获到更加复杂的特征。

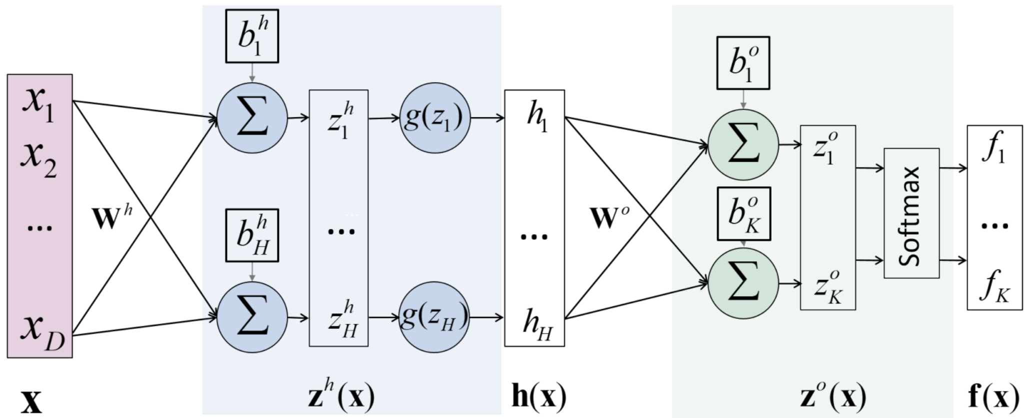


图 3: 多层神经网络(MLP)

但全连接层一个问题在于参数量巨大，当网络变大时计算量将爆炸，故要采用更加好的方法，既能提取出特征，同时计算量也能维持在一个合理的程度，因此就有了**卷积**(convolution)层。

2.2 卷积

f为原图，g为**卷积核**(kernel)或**滤波器**(filter)

$$(f * g)[i, j] = \sum_{m=-M}^{M} \sum_{n=-N}^{N} f[i-m, j-n] g[m, n]$$

即对应元素相乘后相加（但注意卷积与相关操作的不同，卷积要先取反）。卷积在边界时可用0填充(padding)。

上面的公式只是2维卷积，现实我们训练的图片通常采用4维张量表示，即NCHW格式

$$\text{n\_samples} \times \text{n\_channels} \times \text{height} \times \text{width}$$

因此对于每一张多通道的图片，卷积核也应表示为3维，且确保卷积核通道数与图片的通道数相同，这样就可以在3维空间做卷积，但出来的图片只剩1个通道了。故可以采用k个权重不同的卷积核分别对图片做卷积，那么就可以提取到k种不同的特征，出来的特征图(feature map)也会有k个通道。

在具体实施中，卷积层也是可训练的，卷积核的权重和偏置就可以通过网络学习得到。

卷积具有以下三个特征：

- **稀疏交互**(sparse interactions)：卷积层不像全连接层，是对整个图像进行权重计算，它只选择与卷积层交叠的部分进行计算。参数少了，自然计算量也少了。
- **参数共享**(parameter sharing)：由于卷积核是滑过整个图片进行计算，因此对于卷积核的参数，都是被多次运用在不同位置的；而传统的全连接层的参数在图片的每一个位置只会被使用一次，因此

捕获细节特征的能力也会相对弱一些。

- **平移等变**(equivariant representation)：哪怕图片进行一定的平移变换，卷积依然有办法将对应特征提取出来。

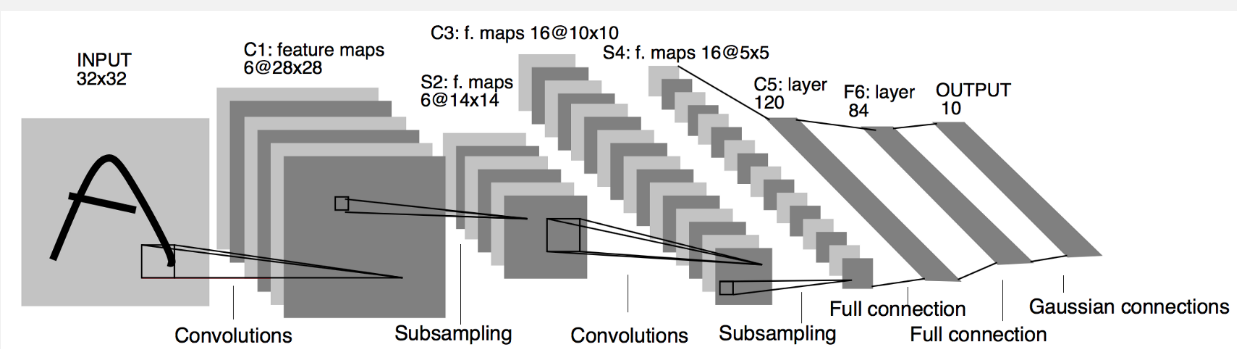
> 在图卷积神经网络(GCN)[6]中的卷积也是类似的道理，在一层中的所有图结点采用**相同**的神经网络进行聚集(aggregate)和更新(update)计算，这样子可以确保神经网络的参数共享，从而关注到图中的局部特征。

2.3 池化

普通的小卷积只能捕获到低层的特征，想要获得高层的语义特征则卷积核应该有更大的**感受野**(receptive field)，但大卷积又会使图片的细节部分被忽略。为解决这两者之间的矛盾，可以在**缩小后的特征图做高层次的卷积**。那么问题就变成了怎么缩小特征图，常见有两种方法：

- 改变步长(stride)
- **池化**(pooling)/**降采样**(subsampling)：对卷积核覆盖的范围取最值或平均

池化可以使得表示对于输入的微小变换保持近似不变(invariant)，这也是好理解的，因为池化考虑的是一片区域的整体特征，对于取最值或是取平均来说，当局部区域一并改变时，池化后的结果确实不会有太多变化。

> 看到这好像也就能明白经典的LeNet为什么这么设计了。先做卷积是为了提取图片中的小的/低层细节特征，然后做激活使得网络非线性，接着做池化，则是将图片进行缩放，方便后续的卷积提取出更大/更高层的语义特征。
>
> 
>
>
> 图 4: LeNet5

3 优化

为了方便叙述，这里采用Stanford CS229的记号，矩阵微积分的推导主要参考：

- Andrew Ng, Kian Katanforoosh, Anand Avati, *Stanford CS 229 Lecture Notes: Deep Learning*, `http://cs229.stanford.edu/notes2019fall/cs229-notes-deep_learning.pdf`

- Andrew Ng, Kian Katanforoosh, *Stanford CS 229 Lecture Notes: Backpropagation*, `http://cs229.stanford.edu/notes/cs229-notes-backprop.pdf`
- Kaare Brandt Petersen, Michael Syskind Pedersen, *Matrix Cookbook*, `https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf`
- 矩阵求导术 - 长躯鬼侠的文章 - 知乎，`https://zhuanlan.zhihu.com/p/24709748`
- Daiwk，机器学习中的矩阵、向量求导，`https://daiwk.github.io/assets/matrix+vector+derivatives+for+machine+learning.pdf`

3.1 反向传播算法

设输入特征为$x_1, x_2, \ldots$，即输入层。$x^{(i)}$代表第i个训练样本，$z_j^{[l]}$代表第l层第j个神经元的输出，$a_j^{[l]}$代表激活后的输出，有

$$\begin{aligned} x_1 &= a_1^{[0]} \\ x_2 &= a_2^{[0]} \\ z_1^{[1]} &= {W_1^{[1]}}^{\mathrm{T}}\mathbf{x} + b_1^{[1]} & a_1^{[1]} &= g(z_1^{[1]}) \\ z_1^{[2]} &= {W_1^{[2]}}^{\mathrm{T}}\mathbf{a}^{[1]} + b_1^{[2]} & a_1^{[2]} &= g(z_1^{[2]}) \end{aligned}$$

其中W是参数矩阵，W_1代表其中的第1行，激活后的输出为

$$\mathbf{a}^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ \vdots \end{bmatrix}$$

用矩阵的形式写，有

$$\underbrace{\begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ \vdots \\ z_m^{[1]} \end{bmatrix}}_{\mathbf{z}^{[1]} \in \mathbb{R}^{m\times 1}} = \underbrace{\begin{bmatrix} - & {W_1^{[1]}}^{\mathrm{T}} & - \\ & \vdots & \\ & \vdots & \\ - & {W_m^{[1]}}^{\mathrm{T}} & - \end{bmatrix}}_{W^{[1]} \in \mathbb{R}^{m\times n}} \underbrace{\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}}_{\mathbf{x} \in \mathbb{R}^{n\times 1}} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_m^{[1]} \end{bmatrix}}_{\mathbf{b}^{[1]} \in \mathbb{R}^{m\times 1}}$$

因此有三层神经网络（单隐含层）的前向传播

$$\begin{aligned} \mathbf{z}^{[1]} &= W^{[1]}\mathbf{x}^{(i)} + \mathbf{b}^{[1]} \\ \mathbf{a}^{[1]} &= h(\mathbf{z}^{[1]}) \\ \mathbf{z}^{[2]} &= W^{[2]}\mathbf{a}^{[1]} + \mathbf{b}^{[2]} \\ \hat{\mathbf{y}}^{(i)} &= \mathbf{a}^{[2]} = g(\mathbf{z}^{[2]}) \end{aligned}$$

其中真实结果$\mathbf{y}^{(i)}$为独热码(one-hot encoding)。

考虑损失函数(loss)为交叉熵

$$\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) = -\mathbf{y}^{\mathrm{T}} \ln \hat{\mathbf{y}}^{(i)}$$

且激活函数$h(\cdot)$为sigmoid函数，$g(\cdot)$为softmax函数。

接下来推导反向传播(backpropagation, BP)算法，利用链式法则求导数。比如想得到隐含层的权重，则计算

$$\frac{\partial \mathcal{L}}{\partial W^{[2]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial W^{[2]}}$$

注意上式并非良定，其中涉及到实值函数对向量求导，也涉及到向量对向量求导（雅可比矩阵），还有向量对矩阵求导。

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[2]}} &= \frac{\partial}{\partial \mathbf{a}^{[2]}} (-\mathbf{y}^{\mathrm{T}} \ln \hat{\mathbf{a}}^{[2]}) \qquad \text{实数对向量求导} \\
&= -\frac{\mathbf{y}}{\mathbf{a}^{[2]}} \qquad \text{逐元素相除}
\end{aligned}$$

设$\mathbf{u} = \exp(\mathbf{z}^{[2]})$为逐元素指数，分两步计算

$$\begin{aligned}
\frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{u}} &= \frac{\partial}{\partial \mathbf{u}} \operatorname{softmax}(\mathbf{z}^{[2]}) \\
&= \frac{\partial}{\partial \mathbf{u}} \frac{\exp(\mathbf{z}^{[2]})}{\mathbf{1}^{\mathrm{T}} \exp(\mathbf{z}^{[2]})} \\
&= \frac{\partial}{\partial \mathbf{u}} \frac{\mathbf{u}}{\mathbf{1}^{\mathrm{T}} \mathbf{u}} \\
&= \mathbf{u} \frac{\partial (1/\mathbf{1}^{\mathrm{T}} \mathbf{u})}{\partial \mathbf{u}^{\mathrm{T}}} + \frac{1}{\mathbf{1}^{\mathrm{T}} \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{u}} \qquad \text{乘法法则} \\
&= -\frac{1}{(\mathbf{1}^{\mathrm{T}} \mathbf{u})^2} \mathbf{u} \mathbf{1}^{\mathrm{T}} + \frac{1}{\mathbf{1}^{\mathrm{T}} \mathbf{u}} I \\
\frac{\partial \mathbf{u}}{\partial \mathbf{z}^{[2]}} &= \frac{\partial \exp(\mathbf{z}^{[2]})}{\partial \mathbf{z}^{[2]}} \\
&= \operatorname{diag}(\exp(\mathbf{z}^{[2]})) \\
&= \operatorname{diag}(\mathbf{u})
\end{aligned}$$

由雅可比矩阵的链式法则

$$\begin{aligned}
\frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} &= \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{z}^{[2]}} \\
&= \left(-\frac{1}{(\mathbf{1}^{\mathrm{T}} \mathbf{u})^2} \mathbf{u} \mathbf{1}^{\mathrm{T}} + \frac{1}{\mathbf{1}^{\mathrm{T}} \mathbf{u}} I \right) \operatorname{diag}(\mathbf{u}) \\
&= -\frac{1}{(\mathbf{1}^{\mathrm{T}} \mathbf{u})^2} \mathbf{u} \mathbf{u}^{\mathrm{T}} + \frac{1}{\mathbf{1}^{\mathrm{T}} \mathbf{u}} \operatorname{diag}(\mathbf{u}) \\
&= -\frac{\mathbf{u}}{\mathbf{1}^{\mathrm{T}} \mathbf{u}} \left(\frac{\mathbf{u}}{\mathbf{1}^{\mathrm{T}} \mathbf{u}} \right)^{\mathrm{T}} + \frac{1}{\mathbf{1}^{\mathrm{T}} \mathbf{u}} \operatorname{diag}(\mathbf{u}) \\
&= -\mathbf{a}^{[2]} {\mathbf{a}^{[2]}}^{\mathrm{T}} + \operatorname{diag}(\mathbf{a}^{[2]})
\end{aligned}$$

进而

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[2]}} &:= \delta^{[2]} \qquad \text{损失函数对未激活前}\mathbf{z}\text{的导数} \\
&= \left(\frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}}\right)^{\mathrm{T}} \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[2]}} \\
&= -\left(-\mathbf{a}^{[2]}\mathbf{a}^{[2]\mathrm{T}} + \mathrm{diag}(\mathbf{a}^{[2]})\right)^{\mathrm{T}} \left(\frac{\mathbf{y}}{\mathbf{a}^{[2]}}\right) \\
&= \mathbf{a}^{[2]}\left(\mathbf{a}^{[2]\mathrm{T}}\frac{\mathbf{y}}{\mathbf{a}^{[2]}}\right) - \mathrm{diag}(\mathbf{a}^{[2]})\left(\frac{\mathbf{y}}{\mathbf{a}^{[2]}}\right) \\
&= \mathbf{a}^{[2]}\mathbf{1}^{\mathrm{T}}\mathbf{y} - \mathbf{y} \\
&= \mathbf{a}^{[2]} - \mathbf{y} \qquad \text{独热码满足}\mathbf{1}^{\mathrm{T}}\mathbf{y} = 1
\end{aligned}$$

之后可计算

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial W^{[2]}} &= \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[2]}}\frac{\partial \mathbf{z}^{[2]}}{\partial W^{[2]}} &&= \delta^{[2]}\mathbf{a}^{[1]\mathrm{T}} \qquad \text{线性变换求导法则} \\
\frac{\partial \mathcal{L}}{\partial b^{[2]}} &= \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[2]}}\frac{\partial \mathbf{z}^{[2]}}{\partial b^{[2]}} &&= \delta^{[2]}
\end{aligned}$$

再往前推一层

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[1]}} &= W^{[2]\mathrm{T}}\delta^{[2]} \\
\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[1]}} &:= \delta^{[1]} \\
&= \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[1]}}\frac{\partial \mathbf{a}^{[1]}}{\partial \mathbf{z}^{[1]}} \\
&= \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[1]}} \odot h'(\mathbf{z}^{[1]}) \\
&= \left(W^{[2]\mathrm{T}}\delta^{[2]}\right) \odot h(\mathbf{z}^{[1]}) \odot (\mathbf{1} - h(\mathbf{z}^{[1]})) \\
\frac{\partial \mathcal{L}}{\partial W^{[1]}} &= \delta^{[1]}\mathbf{x}^{\mathrm{T}} \\
\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[1]}} &= \delta^{[1]} \\
\frac{\partial \mathcal{L}}{\partial \mathbf{x}} &= W^{[1]\mathrm{T}}\delta^{[1]}
\end{aligned}$$

总结来说，对于$l = N-1, N-2, \ldots, 1$层，设

$$\delta^{[l]} = \left(W^{[l+1]\mathrm{T}}\delta^{[l+1]}\right) \odot h'(\mathbf{z}^{[l]})$$

则有权重和偏置的梯度

$$\begin{aligned}
\nabla_{W^{[l]}}\mathcal{L}(W, b) &= \delta^{[l]}\mathbf{a}^{[l-1]\mathrm{T}} \\
\nabla_{b^{[l]}}\mathcal{L}(W, b) &= \delta^{[l]}
\end{aligned}$$

3.2 梯度下降

- 批梯度下降

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\eta}{N} \sum_{n=1}^{N} \nabla \mathcal{L}(\mathbf{y}_n, \mathbf{f}(\mathbf{x}_n; \boldsymbol{\theta}_t))$$

- 随机梯度下降(SGD)

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla \mathcal{L}(\mathbf{y}_n, \mathbf{f}(\mathbf{x}_n; \boldsymbol{\theta}_t))$$

- 子批(minibatch)梯度下降

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\eta}{|S|} \sum_{(\mathbf{x}_n, \mathbf{y}_n) \in S} \nabla \mathcal{L}(\mathbf{y}_n, \mathbf{f}(\mathbf{x}_n; \boldsymbol{\theta}_t))$$

添加动量(momentum)项，使得梯度下降可以保持原来的方向，以减少震荡

$$\begin{cases} \boldsymbol{\nu}_{t+1} = \gamma \boldsymbol{\nu}_t + \eta \nabla \mathcal{L}(\boldsymbol{\theta}_t) \\ \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \boldsymbol{\nu}_{t+1} \end{cases}$$

通常γ取0.9。

3.3 过拟合

参考文献

[1] Limin Yao, David Mimno, and Andrew McCallum. Efficient methods for topic model inference on streaming document collections. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009.

[2] Mu Li, David G Andersen, and Alexander Smola. Distributed delayed proximal gradient methods. In *NIPS Workshop on Optimization for Machine Learning*, 2013.

[3] Rainer Gemulla, Erik Nijkamp, Peter J. Haas, and Yannis Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2011.

[4] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: Online learning of social representations. 2014.

[5] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, 2016.

[6] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the International Conference on Learning and Representation (ICLR)*, 2017.

[7] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems (NeurIPS)*, 2017.

[8] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. In *Proceedings of the International Conference on Learning and Representation (ICLR)*, 2016.

[9] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *Proceedings of the International Conference on Learning and Representation (ICLR)*, 2018.

[10] Kunle Olukotun. Designing computer systems for software 2.0. In *Keynote of the 45th International Symposium on Computer Architecture (ISCA)*, 2018.