第3章 深度学习基础

人工智能技术经历了三起两落，目前处于第三次人工智能浪潮期。这一浪潮中，既有计算机算力增长、互联网大数据剧增等外部因素影响，也有人工智能技术本身迅猛发展的因素，而其中的核心就是深度学习技术。因此，了解深度学习的基本概念与原理，了解深度学习网络的关键问题至关重要。

本章首先介绍神经元概念，在此基础上给出深度学习的基本定义和特点；然后给出深度学习网络设计的三个核心问题，即网络结构的定义、深度学习网络的目标函数以及网络的优化算法；最后针对网络优化算法中的关键技术——后向传播算法进行重点阐述与推导。

3.1 深度学习的基本定义和特点

3.1.1 神经元与生物神经网络

在了解人工神经元之前，先要了解人脑学习的机制。人脑在学习、思考和认知事物时，原理比较简单，首先通过感觉器官接收信息，如通过视觉、听觉、触觉、嗅觉和味觉等感知外面世界；其次将接收的信息在脑内进行信息加工，然后将输出信息以某种反应的形式输出，这个过程不断交叉重复，逐渐累积知识或能力，从而形成创造力。而信息加工的主体就是人脑中的生物神经网，生物神经网络是由很多神经元相互连接而构成的极为庞大而又错综复杂的系统，神经元是生物神经系统结构和功能的基本单位[1]。生物神经元和生物神经网络分别如图 3-1 和图 3-2 所示。神经元的形状和大小差异比较大，但具有一些共同结构。神经元主要包括细胞体和突起两部分，细胞体中央有细胞核，细胞核是细胞的能量中心。细胞体向外伸出两种胞体：呈树枝状的称为树突，它接收其他神经元的信息并传至本细胞体；一根细长的突起称为轴突，它把神经冲动由胞体传至远处，传给与之连接的其他神经元的树突或者肌肉与腺体。

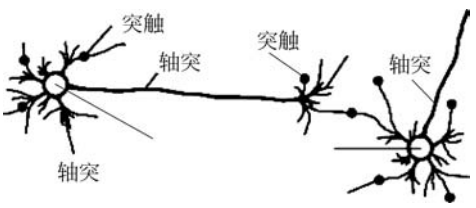


图 3-1 生物神经元的结构

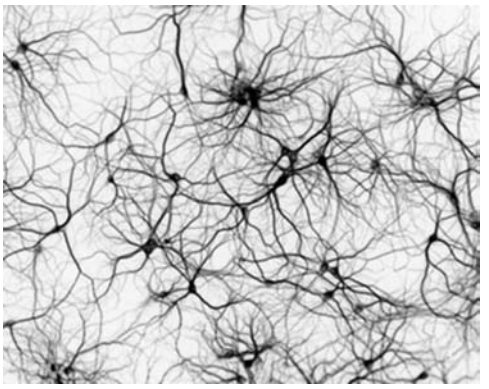


图 3-2 生物神经网络

神经元之间的连接不是固定不变的，在人类成长过程中，一些新连接会逐渐被建立，

一些连接可能会消失。研究表明，人类在 8 岁左右神经元达到相对比较稠密的峰值状态，随后神经网络会变得稀疏。

3.1.2 人工神经元及其分类能力

人工神经元是对生物神经元的功能模拟模型，前面讲神经元模型时，重点强调了树突、轴突和细胞体。因此一个简单的人工神经元构造如图 3-3 所示，同时接收其他神经元传递过来的 D 个信号，分别为 $x_1, x_2, \cdots, x_D$，由于各个信号的强度不同，反应刺激也不同，因此将其赋予了不同的权值 $w_1, w_2, \cdots, w_D$ 后进行求和叠加，然后利用一个非线性函数 g 来模拟输出的反应。神经元可以表示为两个部分，一是神经元的线性输入 $a(\boldsymbol{x})$，二是神经元的非线性输出 $h(\boldsymbol{x})$，其中：

$$a(\boldsymbol{x}) = b + \sum_{i=1}^{D} w_i x_i = \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x} + b \tag{3.1}$$

$$h(\boldsymbol{x}) = g(a(\boldsymbol{x})) = g\left(b + \sum_{i=1}^{D} w_i x_i\right) = g(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x} + b) \tag{3.2}$$

式中：b 为神经元偏置；g 为激活函数。

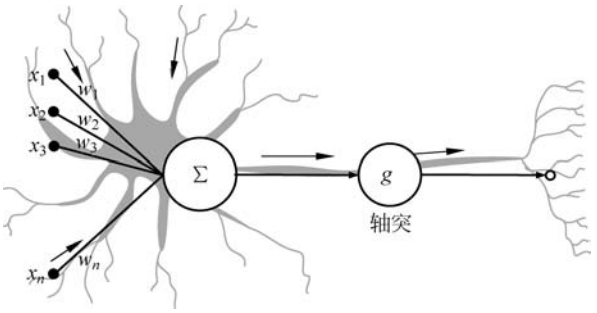
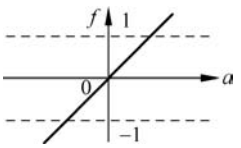
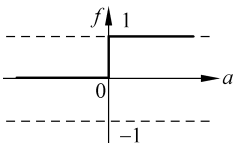


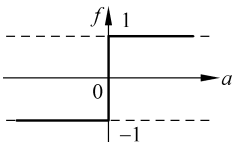
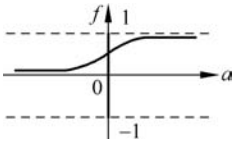
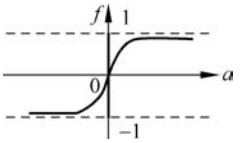
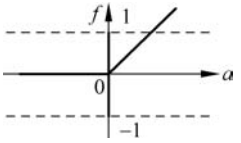
图 3-3 神经元的结构

常用的非线性激活函数有六种，如表 3-1 所示。

表 3-1 激活函数

函数名称	映射关系	表示图	说明
线性函数	$f=g(a)=a$		输入即输出
step 函数	$f=g(a)=\begin{cases}1, & a\geqslant 0\\ 0, & a<0\end{cases}$		输入信号大于或等于 0 时，输出为 1；否则，为 0

续表

函数名称	映射关系	表示图	说明
sign 函数	$f=g(a)=\begin{cases}1, & a\geqslant 0\\ -1, & a<0\end{cases}$		输入信号大于或等于 0 时，输出为 1；否则，为 −1
Sigmoid 函数	$f=g(a)=\dfrac{1}{1+\exp(-a)}$		有界函数，将输入信号变换到(0,1)区间内
tanh 函数	$f=g(a)=\dfrac{\exp(a)-\exp(-a)}{\exp(a)+\exp(-a)}$		有界函数，将输入信号变换到(−1,1)区间内
ReLU 函数	$f=g(a)=\begin{cases}a, & a\geqslant 0\\ 0, & a<0\end{cases}$		半波整流函数，当输入信号大于或等于 0 时，信号直接输出；否则，为 0

下面以 g 函数为 sign 函数时来了解下神经元的基本能力[2]。sign 函数将空间分成两类，一是+1，另一类是−1；分类面如图 3-4 中所示悬崖立面，具体表示为 $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b=0$，而权重向量 $\boldsymbol{w}$ 是分类面的法向量。对于分类面上的任意两点 $\boldsymbol{x}_1$ 和 $\boldsymbol{x}_2$，有

$$\begin{cases}\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_1+b=0\\ \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_2+b=0\end{cases}\Rightarrow \boldsymbol{w}^{\mathrm{T}}(\boldsymbol{x}_1-\boldsymbol{x}_2)=0 \tag{3.3}$$

由于 $\boldsymbol{w}$ 与分类面上的任意两点 $\boldsymbol{x}_1$ 和 $\boldsymbol{x}_2$ 都垂直，因此 $\boldsymbol{w}$ 是分类面的法向量。原点到平面的距离为 $b/\|\boldsymbol{w}\|$。

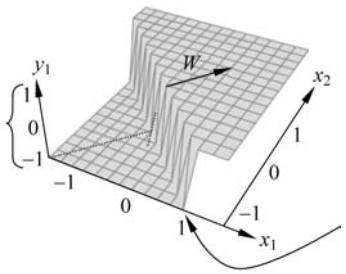


图 3-4　神经元的分类面

利用单个神经元进行二分类问题，当采用 Sigmoid 函数时，可以用单个神经元模拟二分类的概率 $P(y=1|\boldsymbol{x})$，这个分类器通常也称为逻辑回归分类器。若 $P(y=1|\boldsymbol{x})\geqslant 0.5$，则判决为类别 1；否则，为类别 0。因此，单个神经元的分类能力是线性的。单个神经元构成的分类器也称为感知机，该分类器可以解决线性可分问题。如图 3-5 所示的三个例子，图 3-5(a)是 x_1 或 x_2 问题，图 3-5(b)是 x_1 非与 x_2 问题，图 3-5(c)是 x_1 与 x_2 非问题，其中 x_1 和 x_2 的取值都只有 0 和 1 两种。由于都是线性可分问题，单个神经元构成的感知机分类器是可以进行分类的。

1969 年，著名的人工智能学家马文·明斯基提出异或问题，如图 3-6(a)所示，即对角的两个点类别标签相同[3]。由前面所述可知，单个神经元的分类能力是线性的，无法实

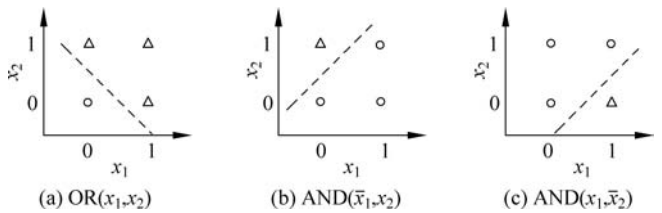


图 3-5　线性可分问题

现线性不可分问题。1974 年，加拿大著名学者辛顿提出感知机网络，他的思想非常简单，“多则不同”，即单个神经元无法解决，通过多个神经元构成的网络来共同完成。由于 $\mathrm{XOR}(x_1,x_2)=\mathrm{OR}(\mathrm{AND}(\bar{x}_1,x_2),\mathrm{AND}(x_1,\bar{x}_2))$，因此，根据图 3-5，单个神经元可以完成 $\mathrm{AND}(\bar{x}_1,x_2)$，也可以完成 $\mathrm{AND}(x_1,\bar{x}_2)$，再通过一个线性分类器 OR 就可以对 $\mathrm{AND}(\bar{x}_1,x_2)$和 $\mathrm{AND}(x_1,\bar{x}_2)$的分类，从而通过 2 层分类器实现对 XOR 问题的分类。

图 3-7 给出了感知机网络的分类面，一个神经元无法完成异或问题的分类，但是两个神经元组合后就可以完成异或问题的分类，即对角的两点就可以分成一类。需要说明的是，图 3-7 中的神经元都采用 sign 激活函数。更多的神经元组合成的神经网络可对更加复杂的非线性问题进行描述。图 3-8 给出了 4 个神经元映射后的分类面，可以看出，神经元个数越多，可以描述的情形就越复杂。

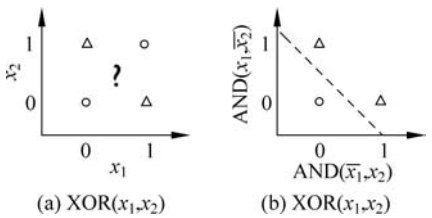


图 3-6　异或问题的实现

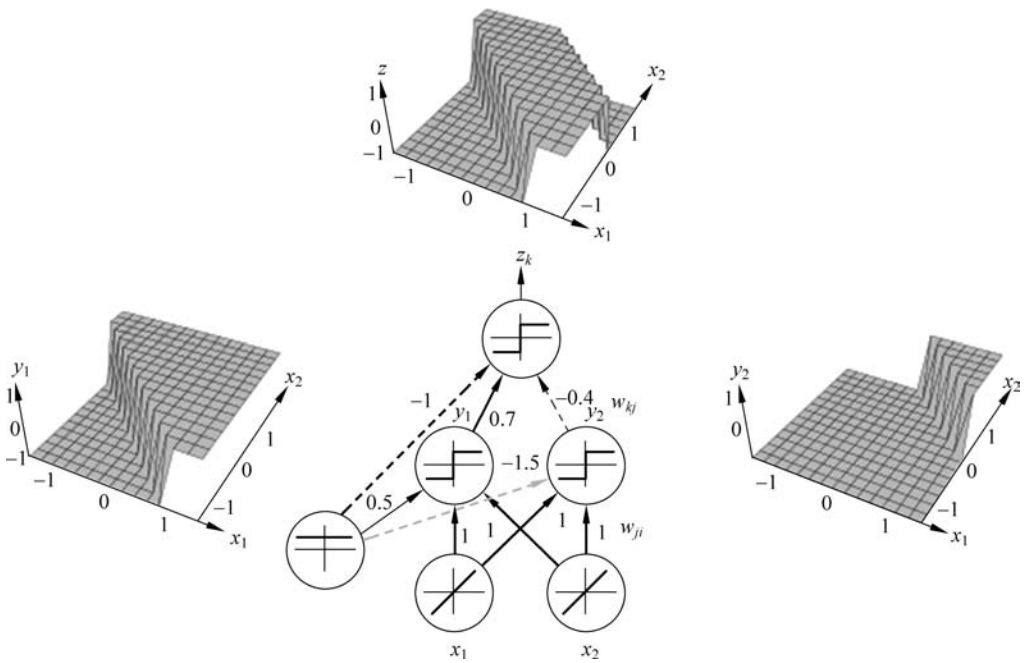


图 3-7　感知机网络的分类面

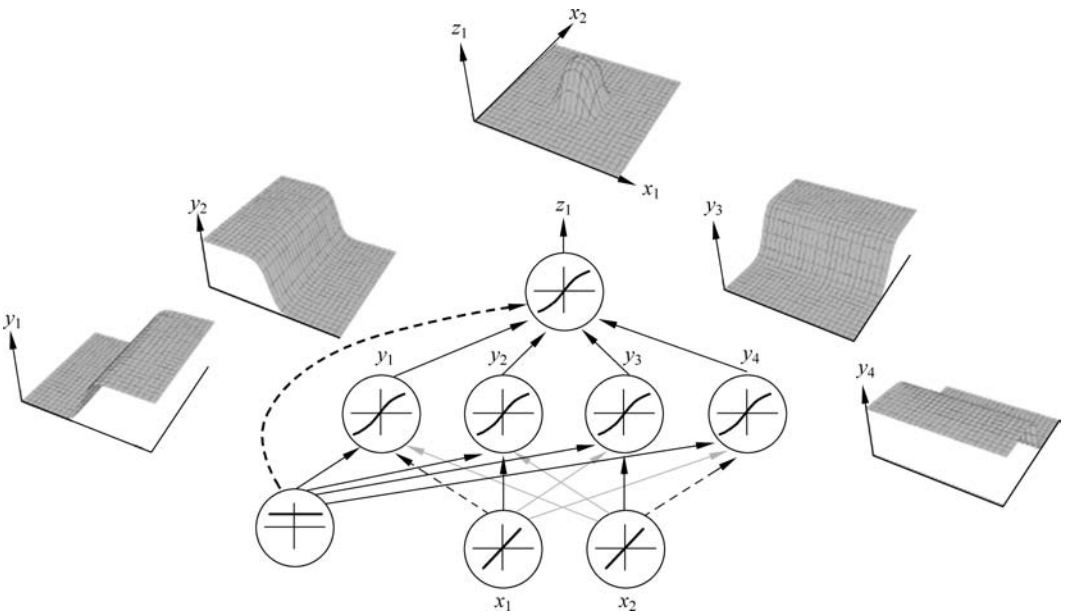


图 3-8　感知机网络的分类面

人工神经网络就是将多个人工神经元连接起来构成的神经网络，图 3-9 是典型神经网络。其中第一层是输入层，直接跟输入信号或特征相连，最后一层是网络输出层，中间内部的节点是隐含层，图 3-9 所示的隐含层数为 2 层。

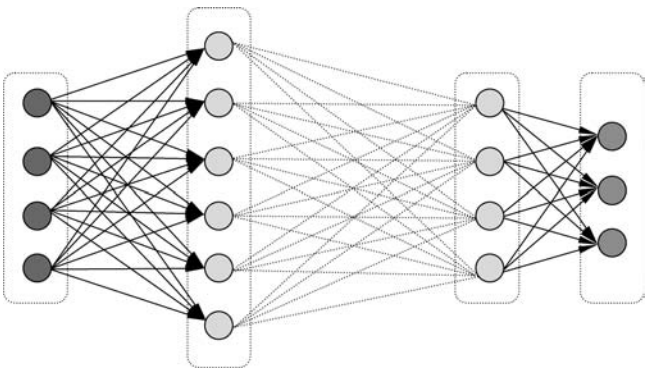


图 3-9　人工神经元示意图

3.1.3　单隐含层神经网络的能力

下面讨论单隐含层的神经网络的能力。根据 3.1.2 节的讨论，当隐含层节点数增多时，能够实现更加复杂的非线性分类能力，如图 3-10 所示。单隐含层神经网络可对任意复杂的非线性问题进行描述。

人工神经网络中存在一个非常有价值的定理——通用近似定理(也称为万能逼近定理)[4],可以在理论上证明“一个包含线性输出层和至少一层隐含层的前馈神经网络,只要给予足够数量的神经元,就能以任意精度逼近任意预定的连续函数”。

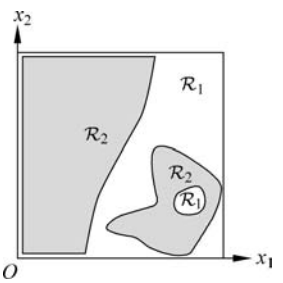


图 3-10 复杂的非线性分类能力

由通用近似定理可知,不管函数 $f(\boldsymbol{x})$ 在形式上有多复杂,总能确保找到一个神经网络,对任何可能的输入 $\boldsymbol{x}$,以任意高的精度近似输出 $f(\boldsymbol{x})$,即使函数有多个输入和输出,即 $f(\boldsymbol{x}_1,\boldsymbol{x}_2,\cdots,\boldsymbol{x}_n)$,通用近似定理的结论仍然成立[5]。应用该定理时需要注意三点,一是可以通过神经网络的结构和参数设计尽可能逼近某个特定函数,而非“准确”描述该函数,其实质是通过增加隐含层神元的个数来提升近似的精度;二是被近似函数必须是连续函数或平方可积函数,否则,神经网络就不再适用;三是虽然神经网络能够近似,但找到这样的近似函数并不容易,定理是给出了一个理论界。生成对抗网络(Generative Adversarial Network,GAN)的提出者伊恩·古德费洛(Ian Goodfellow)曾指出:“仅含有一层的前馈网络的确足以有效地表示任何函数,但是,这样的网络结构可能会格外庞大,进而无法正确地学习和泛化。”这也进一步说明了寻找理论界的困难性。

3.1.4 深度学习

前面提到了人工智能、机器学习等概念,这里阐述人工智能、机器学习和深度学习的关系。简而言之,人工智能在于打造智能机器的学科,而机器学习就是让机器变得更智能的方法和手段,而深度学习是人工智能连接学派的典型代表,也是机器学习中的一种重要方法。

深度学习作为机器学习的一个分支,其目的是在一整套算法的基础上,通过构建非线性和线性变换组成的多个处理层所构成的深度图,实现数据的高层抽象表示与建模。简而言之,当图 3-9 中隐含层超过两层时,就可以认为是深度学习网络。深度学习网络不是一个新概念,在人工神经网络诞生之后,很多学者开始用单个隐含层的神经网络来解决问题,在 20 世纪 80 年代,很多学者就想到扩展隐含层的层数,可以采用两层、三层或者是任意层,但网络隐含层扩展后,神经网络的参数增多,导致需要的数据量增大,且由于层数扩展后模型的优化越来越困难,在实际中得不到想要的结果,因此深度学习网络就相对沉寂。而基于统计学习理论的支持向量机(Support Vector Machine,SVM)非常适合解决小样本高维实际问题,获得了巨大的发展。因此,深度学习网络不是人工智能第三次浪潮才提出的新概念。

通用近似定理已经告诉我们,只要神经网络中神经元的数目足够多,即使只有一层隐含层,也能够对任意复杂实际问题进行模拟。既然单隐含层能够解决这些问题,为何还要把浅层网络往神经网络拓展呢?其实,深度学习的发展、壮大主要基于两点[6]:一

是仅含有一层的前馈网络的确足以有效地表示任何函数，但是隐含层神经元数目的过于庞大，导致难以正确学习和优化。这意味着，对于复杂的问题，虽然理论上存在这样的网络能够对该问题进行近似，但实际中优化得到这样的网络参数非常困难。二是最重要，也是最本质的原因，即深度学习网络相对于浅层网络而言，是更紧致的模型。紧致模型在相同的参数规模下具有更强的表现力，或者在同等参数规模下对复杂模式表现得更充分。

下面以语音识别为例给出具体说明[7]，如图 3-11 所示。表 3-2 给出了浅层网络和深层网络的实验对比图。表中给出了当隐含层数为 1、2、3、4、5、7 时连续语音的词错误率结果。可以看出，当隐含层数增加时，连续语音识别系统的性能不断提升。这是因为在充足的训练语料下，隐含层数目增加，意味着模型的参数增加，模型的表现力更强。但是，这种对比不能说明深度学习网络的优势。为此，研究人员进行了相同参数规模下的同等对比，即为了匹配同等参数规模下的深度网络，单隐含层的浅层网络的节点数需要大量增加。表 3-2 中给出了隐含层数为 5 和 7 时的深度网络性能，以及相同参数规模下的浅层网络的性能。从表中可以看出，当隐含层为 5 时，深层网络的词错误率为 17.2%，而浅层网络的词错误率为 22.5%；7 层网络时，深度网络的词错误率为 17.1%，而浅层

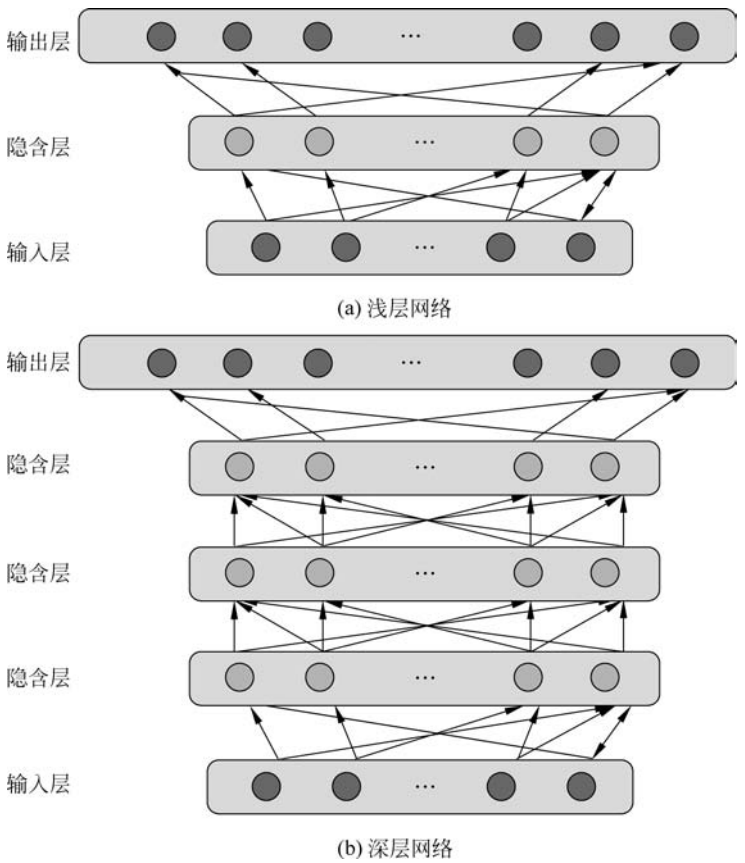


图 3-11　浅层网络和深层网络对比

网络的性能为22.6%。从表中进一步说明，当深层网络从单层扩展到5层时，系统词错误率从24.2%降低为17.2%，而浅层网络达到同样参数规模，其错误率从24.2%降低为22.5%，可以看出，深度网络在同等参数规模下，相较于浅层网络具有更好的模型表现力，因此深度网络是一种更紧致的模型。

表3-2 浅层网络与深层网络语音识别对比

深层网络		浅层网络	
层数×层节点数	词错误率/%	层数×层节点数	词错误率/%
1×2k	24.2	—	—
2×2k	20.4	—	—
3×2k	18.4	—	—
4×2k	17.8	—	—
5×2k	17.2	1×3772	22.5
7×2k	17.1	1×4634	22.6
—	—	1×16k	22.1

3.2 深度学习网络设计的三个核心问题

设计一个深度学习网络的三个核心问题如图3-12所示[8]，主要包括三部分：一是定义网络结构，即决定采用什么样的网络结构，从数学角度而言，定义网络结构相当于定义了一套函数，设定了问题求解的范围。二是网络目标函数选择，目标函数的选择决定了前一步设定的函数集合内，什么样的函数是最终需要寻找的好函数。目标函数选择与任务密切相关，但同一类型的任务也可以从不同角度来进行表征。三是优化算法，就是根据第二步确定的准则，如何找到最优的函数，其中包括优化算法选择和数据训练两个子部分。

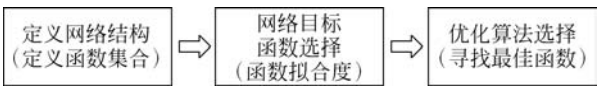


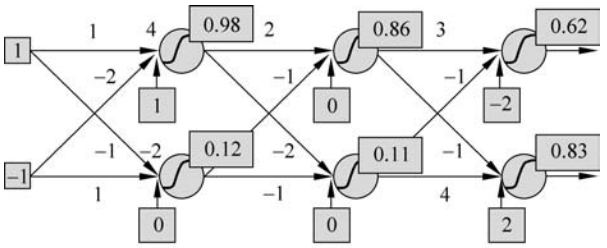
图3-12 DNN设计的三个核心问题[8]

下面以Keras框架下的设计为例，简单展示三个核心问题的基本思想和设计步骤[8]。

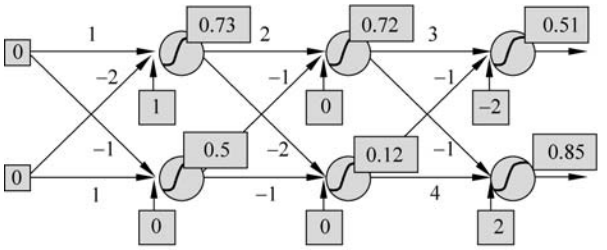
3.2.1 定义网络结构

图3-13给出了一个简单的神经网络示例，图3-13(a)所示两个输入分别为1和−1，假设神经网络的权重和偏置也给定，激活函数选择Sigmoid函数，因此可以计算第一个隐含层的两个输出分别为0.98和0.12，以此类推，同样可以计算第二个隐含层的输出为0.86和0.11，以及第三个隐含层的输出0.62和0.83。图3-13(b)所示两个输入分别为0

和 0，最终输出为 0.51 和 0.85。



(a) 输入为1和−1



(b) 输入为0和0

图 3-13　神经网络示例

由前面可知，神经网络本质上就是一个函数映射，因此图 3-13(a)、(b)可以表示为

$$f\left(\begin{bmatrix}1\\-1\end{bmatrix}\right)=\begin{pmatrix}0.62\\0.83\end{pmatrix}\quad f\left(\begin{bmatrix}0\\0\end{bmatrix}\right)=\begin{pmatrix}0.51\\0.85\end{pmatrix}$$

因此，给定网络结构和参数，神经网络就是一个具体的函数；只给定网络结构，而没有给定参数，相当于是一个函数集合。

图 3-14 给出含有 L 个隐含层的前向全连接神经网络，输入层有 D 个节点，x_1，x_2，…，x_D，每一隐含层线性输入分别为 $\boldsymbol{a}^{(1)}$，$\boldsymbol{a}^{(2)}$，…，$\boldsymbol{a}^{(L)}$，非线性输入表示为 $\boldsymbol{h}^{(1)}$，$\boldsymbol{h}^{(2)}$，…，$\boldsymbol{h}^{(L)}$；网络的输出为 $y^{(1)}$，$y^{(2)}$，…，$y^{(M)}$。下面分析该网络的具体表示形式。

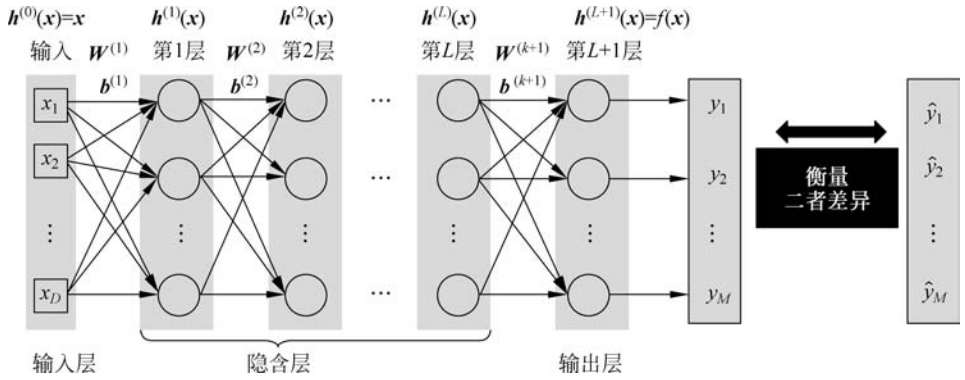


图 3-14　L 个隐含层的神经网络

为了更加通用表示，令 $\boldsymbol{h}^{(0)}(\boldsymbol{x})=\boldsymbol{x}=[x_1,x_2,\cdots,x_D]^{\mathrm{T}}$，那么第 k 层的线性输入 $\boldsymbol{a}^{(k)}(\boldsymbol{x})$和非线性输出 $\boldsymbol{h}^{(k)}(\boldsymbol{x})$分别表示为

$$\boldsymbol{a}^{(k)}(\boldsymbol{x})=\boldsymbol{W}^{(k)}\boldsymbol{h}^{(k-1)}(\boldsymbol{x})+\boldsymbol{b}^{(k)} \tag{3.4}$$

$$\boldsymbol{h}^{(k)}(\boldsymbol{x})=g(\boldsymbol{a}^{(k)}(\boldsymbol{x})) \tag{3.5}$$

输出层表示为

$$\boldsymbol{h}^{(L+1)}(\boldsymbol{x})=o(\boldsymbol{a}^{(L+1)}(\boldsymbol{x}))=f(\boldsymbol{x}) \tag{3.6}$$

式中：o 为输出激活函数。

整个网络可以表示为

$$\begin{aligned}f(\boldsymbol{x})&=o(\boldsymbol{a}^{(L+1)}(\boldsymbol{x}))=o(\boldsymbol{W}^{(L+1)}\boldsymbol{h}^{(k)}(\boldsymbol{x})+\boldsymbol{b}^{(k)})\\&=o(\boldsymbol{W}^{(L+1)}\underline{g(\boldsymbol{W}^{(L)}\boldsymbol{h}^{(L-1)}(\boldsymbol{x})+\boldsymbol{b}^{(L)})}+\boldsymbol{b}^{(L+1)})\\&=o(\boldsymbol{W}^{(L+1)}g(\boldsymbol{W}^{(L)}\underline{g(\boldsymbol{W}^{(L-1)}\boldsymbol{h}^{(L-2)}(\boldsymbol{x})+\boldsymbol{b}^{(L-1)})}+\boldsymbol{b}^{(L)})+\boldsymbol{b}^{(L+1)})\\&=o(\boldsymbol{W}^{(L+1)}g(\boldsymbol{W}^{(L)}\cdots(\boldsymbol{W}^{(2)}\underline{g(\boldsymbol{W}^{(1)}\boldsymbol{h}^{(0)}(\boldsymbol{x})+\boldsymbol{b}^{(1)})}+\boldsymbol{b}^{(2)})\cdots+\boldsymbol{b}^{(L)})+\boldsymbol{b}^{(L+1)})\end{aligned} \tag{3.7}$$

因此，进一步说明整个神经网络就是一个关于 $\boldsymbol{W}^{(1)},\boldsymbol{W}^{(2)},\cdots,\boldsymbol{W}^{(L+1)}$ 以及 $\boldsymbol{b}^{(1)},\boldsymbol{b}^{(2)},\cdots,\boldsymbol{b}^{(L+1)}$ 的复杂非线性函数。当给定网络结构和所有 $\boldsymbol{W}^{(1)},\boldsymbol{W}^{(2)},\cdots,\boldsymbol{W}^{(L+1)}$ 和 $\boldsymbol{b}^{(1)},\boldsymbol{b}^{(2)},\cdots,\boldsymbol{b}^{(L+1)}$ 参数时，该网络就是一个具体函数；而当只给定网络结构未给定参数时，它是一个函数集合。

3.2.2 目标函数选择

如图 3-14 所示，如何衡量一个神经网络模型参数的好坏？需要设定一个网络的目标函数[9]。由机器学习知识可知，通常选择两种目标函数：

(1) 拟合问题：

$$l=(\boldsymbol{y}^{(n)}-\hat{\boldsymbol{y}}^{(n)})^2 \tag{3.8}$$

式中：l 为单样本损失函数；$\boldsymbol{y}^{(n)}=[y_1^{(n)},y_2^{(n)},\cdots,y_M^{(n)}]^{\mathrm{T}}$ 为神经网络得到的样本值，M 为输出样本维度；$\hat{\boldsymbol{y}}^{(n)}=[\hat{y}_1^{(n)},\hat{y}_2^{(n)},\cdots,\hat{y}_M^{(n)}]^{\mathrm{T}}$ 为相应样本真实值。

(2) 多分类问题：

$$l^{(n)}=-\sum_{m=1}^{M}\hat{y}_m^{(n)}\log y_m^{(n)} \tag{3.9}$$

式中：l 为单样本损失函数，该损失函数是样本类别标签 $\boldsymbol{y}^{(n)}=[y_1^{(n)},y_2^{(n)},\cdots,y_M^{(n)}]^{\mathrm{T}}$ 和相应样本标签 $\hat{\boldsymbol{y}}^{(n)}=[\hat{y}_1^{(n)},\hat{y}_2^{(n)},\cdots,\hat{y}_M^{(n)}]^{\mathrm{T}}$ 之间的交叉熵，$\hat{\boldsymbol{y}}^{(n)}=[\hat{y}_1^{(n)},\hat{y}_2^{(n)},\cdots,\hat{y}_M^{(n)}]^{\mathrm{T}}$ 为独热向量，真实类别标签位置为 1，其他为 0。

神经网络优化时采用总损失函数，总损失函数定义为

$$J=\sum_{n=1}^{N}l^{(n)} \tag{3.10}$$

式中：$l^{(n)}$ 为第 n 个样本的损失函数；J 为总损失函数。其具体过程如图 3-15 所示。

也可以根据其他的任务设置相对类型的类别函数。对于多分类问题，除了采用经典的

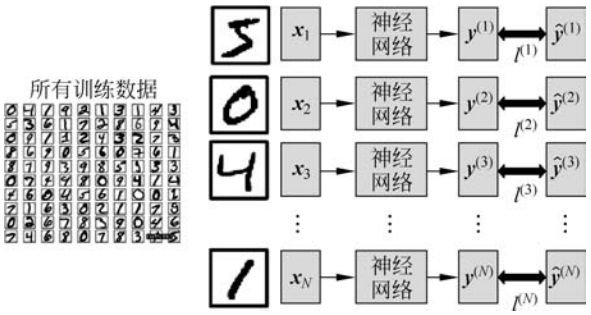


图 3-15　神经网络的总损失函数

交叉熵函数外，还可以采用很多的区分性函数来进一步提升分类性能，具体介绍见第 7 章。

3.2.3　优化算法选择

为了简单起见，假设网络只有参数 θ_0、θ_1，并假设损失函数是非常理想的，如图 3-16 所示，损失函数是凸函数，具有唯一极小值。图中 A 点和 B 点为不同方法得到的初始化权重参数。那么如何获得损失函数极小值点，以及极小值所对应的参数呢？

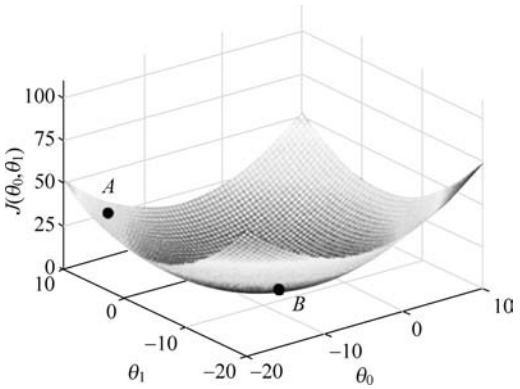


图 3-16　神经网络的总损失函数随参数变化

第 2 章机器学习理论中已经介绍过，当从损失误差很大的初始点寻找最优极小值点时，沿梯度反方向是到达极小值最快的方向，因此这种方法也称为梯度下降法。反之，如果寻找极大值点，则采用梯度上升法。一般采用迭代方法逐渐逼近极小值，具体公式为

$$\theta_{t+1}=\theta_t-\alpha\nabla_{\theta_t}J(\theta) \tag{3.11}$$

式中：θ_{t+1} 为第 $t+1$ 步更新后的参数；θ_t 为更新前的参数；$J(\theta)$ 为目标函数；α 为步长。

对于图 3-14 所示的任意神经网络，网络参数 $\boldsymbol{\theta}=\{\boldsymbol{W}^{(1)},\boldsymbol{W}^{(2)},\cdots,\boldsymbol{W}^{(L+1)};\ \boldsymbol{b}^{(1)},\boldsymbol{b}^{(2)},\cdots,\boldsymbol{b}^{(L+1)}\}$，因此需要计算目标函数对于参数的梯度，即

$$\nabla_{\theta_t} J = \begin{bmatrix} \dfrac{\partial J}{\partial \boldsymbol{W}_1} \\ \dfrac{\partial J}{\partial \boldsymbol{W}_2} \\ \vdots \\ \dfrac{\partial J}{\partial \boldsymbol{b}_1} \\ \dfrac{\partial J}{\partial \boldsymbol{b}_2} \\ \vdots \end{bmatrix} \tag{3.12}$$

将式(3.12)代入式(3.11)可以进行迭代更新,迭代到指定次数,或者是当损失函数达到指定阈值,则停止参数更新。具体过程如图 3-17 所示[10]。

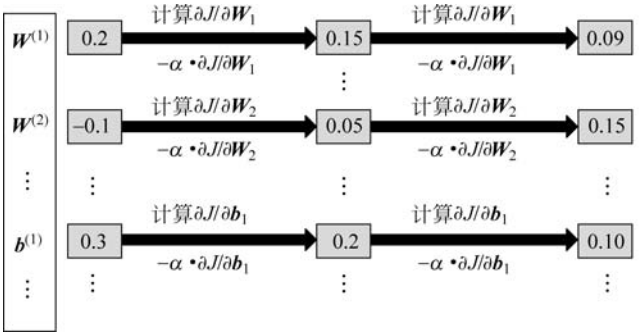


图 3-17 梯度下降法示意图

由此看来,深度学习网络优化的基本思想并不像我们想象的那么复杂,采用梯度下降法就可以得到。然而,还有一个问题没有解决,如何获取式(3.12)的参数梯度?后向传播(Back Propagation,BP)是获取参数梯度的一种有效算法[11]。

3.3 后向传播算法

下面以一个多分类问题为例,对后向传播算法进行具体分析[12]。由于是多分类问题,神经网络的目标函数采用式(3.9)所示的交叉熵,则有

$$J = \sum_{n=1}^{N} l(f(\boldsymbol{x}^{(n)}), \hat{\boldsymbol{y}}^{(n)}) = \sum_{n=1}^{N} l(\boldsymbol{y}^{(n)}, \hat{\boldsymbol{y}}^{(n)}) = -\sum_{n=1}^{N}\sum_{m=1}^{M} \hat{y}_m^{(n)} \log y_m^{(n)} \tag{3.13}$$

因为真实类别标签向量是一个独热向量,只有真实类别标签维度为 1,其他为 0,当样本 $\boldsymbol{x}^{(n)}$ 属于第 c 类样本时,则有

$$J = \sum_{n=1}^{N} l(\boldsymbol{y}^{(n)}, \hat{\boldsymbol{y}}^{(n)}) = -\sum_{n=1}^{N}\sum_{m=1}^{M} \hat{y}_m^{(n)} \log y_m^{(n)} = -\sum_{n=1}^{N} \log y_c^{(n)} \tag{3.14}$$

为了清晰了解 BP 算法的原理,以图 3-18 所示的神经网络为例进行推导,其中隐含

层为两层。

可以从不同角度对 BP 算法进行推导，下面给出一种常用的方法——计算流图法。图 3-19～图 3-22 给出了一条路径、两条路径、n 条路径以及复杂流程图的链式准则。图 3-18 所示的神经网络就是一个复杂流程图的实例。

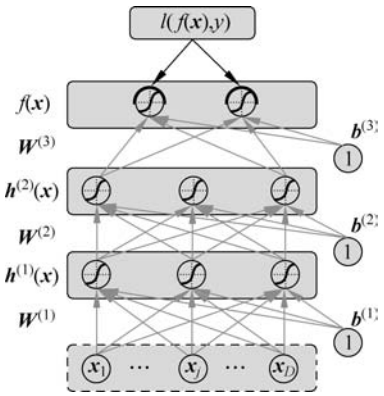


图 3-18　BP 算法推导网络结构图

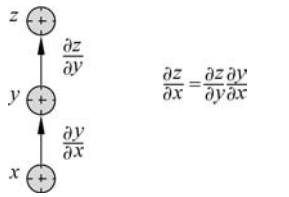


图 3-19　一条路径链式关系

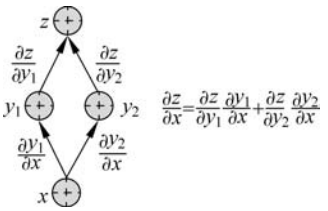


图 3-20　两条路径链式关系

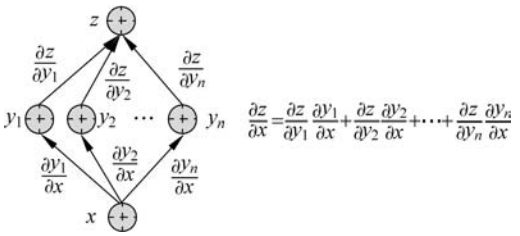


图 3-21　n 条路径链式关系

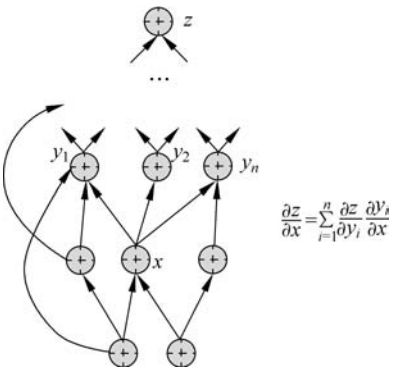


图 3-22　更加复杂的流程图

在计算流图中包括如下两种运算：

(1) 前向传播：给定前续节点计算每个节点的值。

(2) 后向传播：初始化输出梯度为 1，然后逆序访问每个节点，最后计算每个节点的梯度。

3.3.1　输出端的损失梯度

下面计算输出层的损失梯度值。首先计算第 n 个样本 $\boldsymbol{x}^{(n)}$ 的损失函数 $l(\boldsymbol{y}^{(n)}, \hat{\boldsymbol{y}}^{(n)})$

对输出 $y_c^{(n)}$ 的偏导数，即

$$\frac{\partial}{\partial y_m^{(n)}}[-\log y_c^{(n)}]=-\frac{1_{(c=m)}}{y_c^{(n)}}=\begin{cases}-\dfrac{1}{y_c^{(n)}}, & m=c\\ 0, & m\neq c\end{cases} \tag{3.15}$$

所以损失函数 $l(\boldsymbol{y}^{(n)},\hat{\boldsymbol{y}}^{(n)})$ 对输出层激活输出 $\boldsymbol{y}^{(n)}$ 的梯度为

$$\frac{\partial}{\partial \boldsymbol{y}^{(n)}}[-\log y_c^{(n)}]=-\frac{1}{y_c^{(n)}}\begin{bmatrix}1_{(c=1)}\\ \vdots\\ 1_{(c=M)}\end{bmatrix} \tag{3.16}$$

式中：$1_{(c=m)}$ 代表当 $c=m$ 成立时，该值为 1；否则，为 0。

假定输出端激活函数选择 Sigmoid 函数，下面的推导中会用到 Sigmoid 函数求导的便捷形式，即

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

$$g'(x)=g(x)(1-g(x))$$

读者可按照求导准则自行验证。

因此，损失函数 $l(\boldsymbol{y}^{(n)},\hat{\boldsymbol{y}}^{(n)})$ 对输出层线性输入端第 m 个元素的偏导数为

$$\frac{\partial}{\partial a^{(L+1)}(\boldsymbol{x})_m}[-\log y_c^{(n)}]=-(1_{(c=m)}-y_m^{(n)}) \tag{3.17}$$

输出层线性输入端的损失梯度为

$$\nabla_{\boldsymbol{a}^{(L+1)}(\boldsymbol{x})}[-\log y_c^{(n)}]=-(\boldsymbol{e}(c)-\boldsymbol{y}^{(n)})$$

$$=-\left(\begin{bmatrix}1_{(c=1)}\\ 1_{(c=2)}\\ \vdots\\ 1_{(c=M)}\end{bmatrix}-\begin{bmatrix}y_1^{(n)}\\ y_2^{(n)}\\ \vdots\\ y_M^{(n)}\end{bmatrix}\right) \tag{3.18}$$

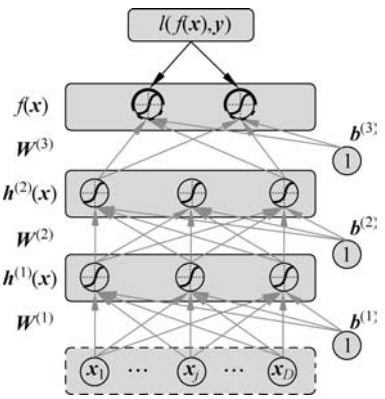


图 3-23 BP 算法推导输出层梯度计算示意图

其具体梯度传导的过程如图 3-23 所示。需要重点说明的是，在理解 BP 算法过程中，梯度值沿反向传递，可以把每个神经元看成两个部分：一个是上半部分的非线性输出，如实线半圈所示；另一个是下半部分的线性输入，如虚线半圈所示。

3.3.2 隐含层的损失梯度

下面获取隐含层的梯度，即图 3-24 中所示的实线路径到达节点的梯度。这里梯度的计算更加复杂，因此需要简化问题。根据图 3-22 所示流程图的链式准则，如果函数 $p(a)$ 是中间变量 $q_i(a)$ 的函数，则可得

$$\frac{\partial p(a)}{\partial a}=\sum_i \frac{\partial p(a)}{\partial q_i(a)}\frac{\partial q_i(a)}{\partial a} \tag{3.19}$$

那么可以进行对应假设，a 是某个隐含层中的某个节点，$q_i(a)$代表上一层的线性输入部分(虚线半圈)，如图 3-25 所示，图中所示 $q_i(a)$个数为 2，即上一层节点个数，也是输出层节点个数。

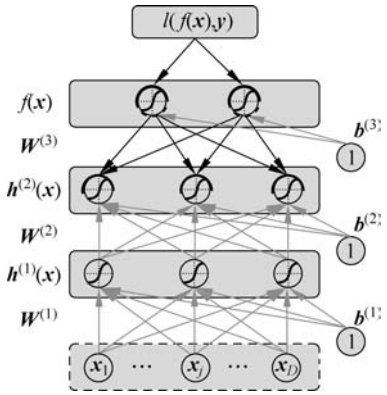


图 3-24　BP 算法推导隐含层非线性输出端梯度计算示意图

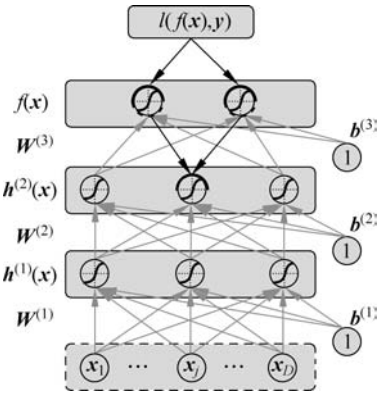


图 3-25　BP 算法推导隐含层非线性输出端梯度计算示意图(特定节点)

1. 损失函数对隐含层非线性输出端的梯度

第 n 个样本 $\boldsymbol{x}^{(n)}$ 的损失函数 $l(\boldsymbol{y}^{(n)},\hat{\boldsymbol{y}}^{(n)})$对第 k 个隐含层第 j 个节点的非线性输出 $h^{(k)}(\boldsymbol{x})_j$ 端的偏导数为

$$\begin{aligned}\frac{\partial}{\partial h^{(k)}(\boldsymbol{x})_j}[-\log y_c^{(n)}]&=\sum_i \frac{\partial[-\log y_c^{(n)}]}{\partial a^{(k+1)}(\boldsymbol{x})_i}\frac{\partial a^{(k+1)}(\boldsymbol{x})_i}{\partial h^{(k)}(\boldsymbol{x})_j}\\&=\sum_i \frac{\partial[-\log y_c^{(n)}]}{\partial a^{(k+1)}(\boldsymbol{x})_i}W_{i,j}^{(k+1)}\\&=(\boldsymbol{W}_{\cdot,j}^{(k+1)})^{\mathrm{T}}(\nabla_{\boldsymbol{a}^{(L+1)}(\boldsymbol{x})}[-\log y_c^{(n)}])\\&\begin{pmatrix}\text{前向递推公式：}a^{(k+1)}(\boldsymbol{x})_i=b_i^{(k)}+\sum_j W_{i,j}^{(k+1)}h^{(k)}(\boldsymbol{x})_j\\\boldsymbol{a}^{(k+1)}(\boldsymbol{x})=\boldsymbol{b}^{(k+1)}+\boldsymbol{W}^{(k+1)}\boldsymbol{h}^{(k)}(\boldsymbol{x})\end{pmatrix}\end{aligned} \tag{3.20}$$

则第 k 个隐含层非线性输出端的梯度为

$$\nabla_{\boldsymbol{h}^{(k)}(\boldsymbol{x})}[-\log y_c^{(n)}]=(\boldsymbol{W}^{(k+1)})^{\mathrm{T}}\nabla_{\boldsymbol{a}^{(L+1)}(\boldsymbol{x})}[-\log y_c^{(n)}] \tag{3.21}$$

2. 损失函数对隐含层线性输入端的梯度

同样根据链式准则，损失函数对第 k 隐含层第 j 个节点线性输入端的偏导数可以变成两个连乘项的积，一项是损失函数对相应节点非线性输出端的偏导数，另一项是该节

点非线性输出对线性输入的偏导数，即

$$\begin{aligned}\frac{\partial}{\partial a^{(k)}(\boldsymbol{x})_j}[-\log y_c^{(n)}] &= \frac{\partial[-\log y_c^{(n)}]}{\partial h^{(k)}(\boldsymbol{x})_j}\frac{\partial h^{(k)}(\boldsymbol{x})_j}{\partial a^{(k)}(\boldsymbol{x})_j} \\ &= \frac{\partial[-\log y_c^{(n)}]}{\partial h^{(k)}(\boldsymbol{x})_j} g'(a^{(k)}(\boldsymbol{x})_j) \\ &\begin{pmatrix}\text{前向递推公式：} h^{(k)}(\boldsymbol{x})_j = g(a^{(k)}(\boldsymbol{x})_j) \\ \boldsymbol{h}^{(k)}(\boldsymbol{x}) = g(\boldsymbol{a}^{(k)}(\boldsymbol{x}))\end{pmatrix}\end{aligned} \tag{3.22}$$

其具体示意图如图 3-26 所示。

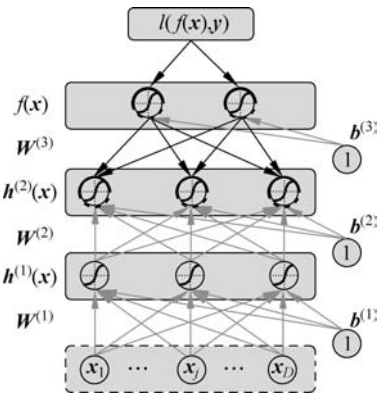


图 3-26　BP 算法推导隐含层线性输入端梯度计算示意图

则损失函数对隐含层线性输入端的梯度表示为

$$\begin{aligned}\nabla_{\boldsymbol{a}^{(k)}(\boldsymbol{x})}[-\log y_c^{(n)}] &= (\nabla_{\boldsymbol{h}^{(k)}(\boldsymbol{x})}[-\log y_c^{(n)}])^{\mathrm{T}}\,\nabla_{\boldsymbol{a}^{(k)}(\boldsymbol{x})}\boldsymbol{h}^{(k)}(\boldsymbol{x}) \\ &= (\nabla_{\boldsymbol{h}^{(k)}(\boldsymbol{x})}[-\log y_c^{(n)}]) \odot [\cdots, g'(a^{(k)}(\boldsymbol{x})_j), \cdots]\end{aligned} \tag{3.23}$$

3.3.3　神经网络参数的损失梯度

利用上面的结果可以得到权重参数 $W_{i,j}^{(k)}$ 的偏导数，即

$$\begin{aligned}\frac{\partial}{\partial W_{i,j}^{(k)}}[-\log y_c^{(n)}] &= \frac{\partial[-\log y_c^{(n)}]}{\partial a^{(k)}(\boldsymbol{x})_i}\frac{\partial a^{(k)}(\boldsymbol{x})_i}{\partial W_{i,j}^{(k)}} \\ &= \frac{\partial[-\log y_c^{(n)}]}{\partial a^{(k)}(\boldsymbol{x})_i} h_j^{(k-1)}(\boldsymbol{x})\end{aligned} \tag{3.24}$$

则损失函数对权重参数的梯度为

$$\nabla_{\boldsymbol{w}^{(k)}}[-\log y_c^{(n)}] = (\nabla_{\boldsymbol{a}^{(k)}(\boldsymbol{x})}[-\log y_c^{(n)}])\boldsymbol{h}^{(k-1)}(\boldsymbol{x})^{\mathrm{T}} \tag{3.25}$$

同理，可得损失函数对参数 $b_i^{(k)}$ 的偏导数，即

$$\frac{\partial}{\partial b_i^{(k)}}[-\log y_c^{(n)}] = \frac{\partial[-\log y_c^{(n)}]}{\partial a^{(k)}(\boldsymbol{x})_i}\frac{\partial a^{(k)}(\boldsymbol{x})_i}{\partial b_i^{(k)}}$$

$$=\frac{\partial[-\log y_c^{(n)}]}{\partial a^{(k)}(\boldsymbol{x})_i} \tag{3.26}$$

则损失函数对偏置向量的梯度为

$$\nabla_{\boldsymbol{b}^{(k)}}[-\log y_c^{(n)}]=(\nabla_{\boldsymbol{a}^{(k)}(\boldsymbol{x})}[-\log y_c^{(n)}]) \tag{3.27}$$

得到式(3.25)和式(3.27),就可以按照梯度下降法进行参数优化更新。

3.3.4 算法整理流图

图 3-27 给出了 BP 算法完整的流程图,可以清晰地看到梯度计算的依赖关系。为了获取目标函数 $l(\boldsymbol{y}^{(n)},\hat{\boldsymbol{y}}^{(n)})$对参数的梯度,首先前向计算出各个节点的取值,再按照上节所示求 $l(\boldsymbol{y}^{(n)},\hat{\boldsymbol{y}}^{(n)})$对参数 $a^{(3)}(\boldsymbol{x})$、$a^{(2)}(\boldsymbol{x})$和 $a^{(1)}(\boldsymbol{x})$的梯度,获得 $a^{(3)}(\boldsymbol{x})$的梯度,就能得到参数 $\boldsymbol{W}^{(3)}$ 和 $\boldsymbol{b}^{(3)}$ 的梯度,获得 $a^{(2)}(\boldsymbol{x})$的梯度,就能得到 $\boldsymbol{W}^{(2)}$ 和 $\boldsymbol{b}^{(2)}$ 的梯度,同理得到 $\boldsymbol{W}^{(1)}$ 和 $\boldsymbol{b}^{(1)}$ 的梯度。可以看出,整个过程中梯度的计算正好与前向计算的路径相反。

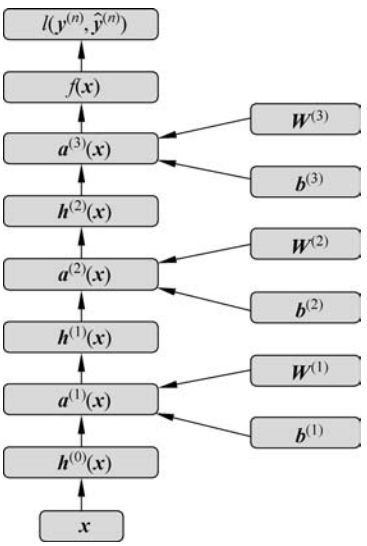


图 3-27　BP 算法推导隐含层线性输入端梯度计算示意图

3.4　本章小结

本章在介绍学习与机器学习概念的基础上,阐述深度学习和机器学习的区别与联系;然后系统介绍深度学习的基本定义和特点;指出深度学习网络设计的三个核心问题,即定义网络结构、设定网络目标函数以及优化算法,并且详细分析三个基本问题的内涵;最后针对网络优化算法中的核心关键技术——后向传播算法进行重点阐述与推导。

参考文献

[1] Ian Goodfellow, Yoshua Bengio. Deep Learning[M]. MIT Press, 2016.

[2] Lrochelle H. 蒙特利尔大学神经网络课件(前向神经网络部分)[EB/OL]. http://info.usherbrooke.ca/hlarochelle/ift725/1_03_capacity_of_single_neuron.pdf.

[3] Minsky M L, Papert S A. Perceptrons[M], MIT Press, 1969.

[4] Hornik K, Stinchcombe M, White H. Multilayer feedforward networks are universal approximators [J]. Neural Networks, 1989, 2(5): 359-366.

[5] Nielsen M A. A visual proof that neural nets can compute any function: Neural networks and deep learning[EB/OL]. http://neuralnetworksanddeeplearning.com/chap4.html.

[6] Boney R. Theoretical motivations for deep learning[EB/OL]. http://rinuboney.github.io/2015/10/18/theoretical-motivations-deep-learning.html, 2015-10-18.

[7] Seide, Frank, Li G, et al. Conversational speech transcription using context-dependent deep neural networks[C]. Proceedings of The 12th Annual Conference of the International Speech Communication Association(INTERSPEECH), 2011: 437-440.

[8] 李宏毅. 台湾大学机器学习课程课件(深度学习介绍)[EB/OL]. http://speech.ee.ntu.edu.tw/~tlkagk/courses/ML_2016/Lecture/DL%20(v2).pdf.

[9] 李宏毅. 台湾大学机器学习课程课件(深度学习部分)[EB/OL]. https://speech.ee.ntu.edu.tw/~hylee/ml/ml2021-course-data/classification_v2.pdf.

[10] Lrochelle H. 蒙特利尔大学神经网络课件(神经网络训练部分)[EB/OL]. http://info.usherbrooke.ca/hlarochelle/ift725/2_11_optimization.pdf.

[11] Werbos P. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences [D]. Harvard University, Cambridge, MA, 1974.

[12] Rumelhart D E, Geoffrey E H, Ronald J W. Learning representations by back-propagating errors [J]. Nature, 1986, 323: 533-536.