第5章 神经网络及基本结构

5.1 神经元介绍

神经网络最早来源于生物学意义上的神经网络，生物学意义上的神经网络一般指由神经元、细胞、触点组成的网络，这种神经网络帮助生物产生意识、思考，控制生物的行动。科学家们从生物神经网络的模式结构受到启发，提出了人工神经网络。通常来说，人工智能领域的人工神经网络和生物领域的神经网络具有不同的含义。一般人工智能领域的神经网络的定义是："神经网络是由具有适应性的简单单元组成的广泛并行互联的网络，它的组织能够模拟生物神经系统对真实世界物体所做出的交互反应"。

神经元是神经网络的基础单元。在生物领域，每个神经元都是一个独立的工作处理单元，这些微小的单元以各种形式进行相互连接，构成了一个庞大的神经网络系统，这些神经之间连接通道的强弱通过神经元之间的化学信号来控制，神经元上树状的突起负责接收激励信号，神经元的动作由接收到的各个信号的综合大小来控制，随着信号的变化而变化，当这些信号达到一定阈值时呈现兴奋或抑制状态。结构如图 5.1 所示。

Mcculloch 首次将生物神经细胞的工作过程简化抽象为 M-P 神经元模型，目前许多新的神经元模型都来源于经典的 M-P 模型。M-P 模型与生物神经元的工作过程有一定的相似性。在这个模型中，神经元模型接收神经网络的输入或者来自其他神经元传递的输入信号，这些输入信号通过加权输入到模型中。模型将接收到的总输入与模型阈值（偏置）进行比较，再通过激活函数产生模型的输出。M-P 模型的结构示意图如图 5.2 所示，输出计算表达式为

$$y = f\left(\sum_{i=1}^{n} \omega_i x_i - \theta\right) \tag{5.1}$$

图 5.1 神经细胞结构图

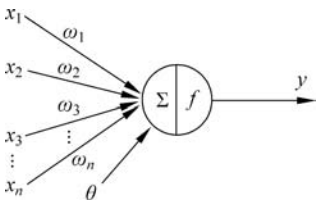


图 5.2 M-P 神经单元

神经元中的阈值（偏置）表示了神经元被激活的难易程度，激活函数在神经元模型中起着类似于"开关"的作用，控制了信息在神经网络中传播的通断，也可以把阈值看作一个开关按下的难易程度。常见的激活函数有阶跃函数、sigmoid 函数、tanh 函数、relu 函数等，值得注意的是，这些激活函数都具有非线性的特性。

例 5.1 假设存在如图 5.2 所示的 M-P 神经元，输入 $\boldsymbol{x}=[6,7,20,3]$，权重 $\boldsymbol{\omega}$ 初始化为 $[0.2,0.4,0.8,0.1]$，激活函数 f 为 sigmoid 函数（公式如下），计算输出 y 为多少。

$$\omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3 + \omega_4 x_4 = 20.3$$

$$y = f(\boldsymbol{\omega x}) \approx 1$$

阶跃函数(见图 5.3)：

$$y=\begin{cases}1, & \boldsymbol{x}\geqslant\boldsymbol{0}\\ 0, & \boldsymbol{x}<\boldsymbol{0}\end{cases} \tag{5.2}$$

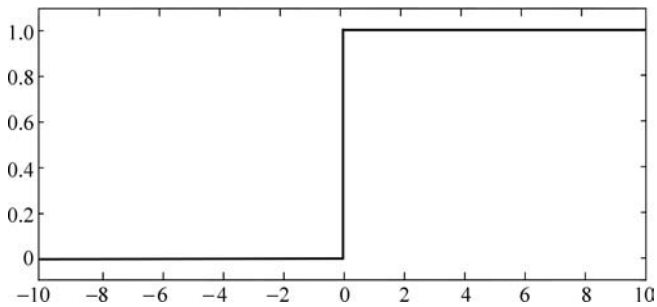


图 5.3　阶跃函数图像

sigmoid 函数(见图 5.4)：

$$y=\frac{1}{1+\mathrm{e}^{-\boldsymbol{x}}} \tag{5.3}$$

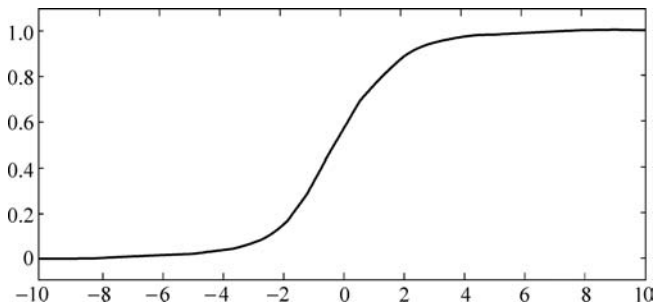


图 5.4　sigmoid 函数图像

tanh 函数(见图 5.5)：

$$y=\frac{\mathrm{e}^{\boldsymbol{x}}-\mathrm{e}^{-\boldsymbol{x}}}{\mathrm{e}^{\boldsymbol{x}}+\mathrm{e}^{-\boldsymbol{x}}} \tag{5.4}$$

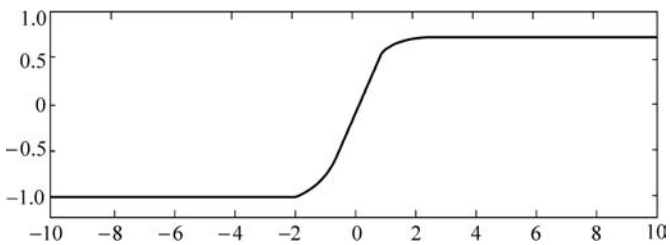


图 5.5　tanh 函数图像

relu 函数(见图 5.6)：

$$y=\max(0,\boldsymbol{x}) \tag{5.5}$$

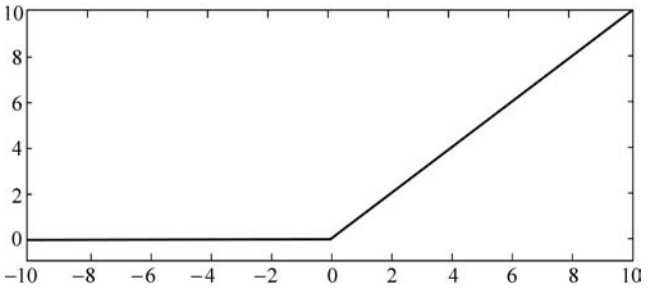


图 5.6　relu 函数图像

其中阶跃函数“01”的二值性可能更符合生物领域的神经网络的工作方式，输出值为 1 表示接收到了足够多的信号，神经元即为激活状态；输出值为 0 表示收到信号的数量不足，即为未激活状态。但是阶跃函数本身不连续、不光滑，这就给模型在反向传播（后续将介绍）时带来了极大的困难，因此现实的应用中后几种激活函数更为常见，并且需要根据实际的应用场景进行选择，并没有严格意义上的最优激活函数。作为神经元基础单元的一部分，激活函数的研究热度一直都非常高，近些年出现了 prelu、elu、glue 等激活函数，并且都取得了非常不错的效果。

假设所有的激活函数都是线性的，则无论网络叠加多少层，都是输入的线性组合，模型的拟合能力非常弱，因此除了“开关”的作用外，激活函数也为神经网络模型引入了非线性的特质，增加了模型的拟合能力，非线性也是人们在设计激活函数时首先要考虑的。

5.2　感知机

感知机是最早的神经网络的一种，它具有较为简单的结构，具有一定的表示能力。感知机由两层神经元组成，分别为输入层和输出层。输入层负责接收来自模型外部的输入，输出层由 M-P 神经单元组成。感知机通常处理线性可分问题。即针对某个数据集 M，存在一个超平面 S 可以将正负样本划分在超平面的两侧。如图 5.7 所示的数据集中的情况均为线性可分。

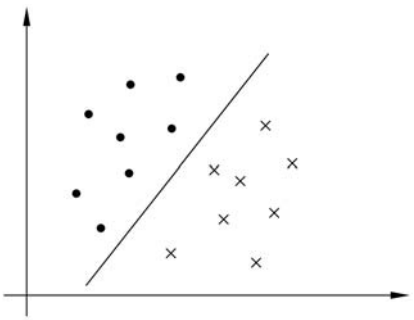


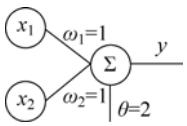
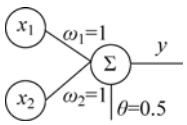
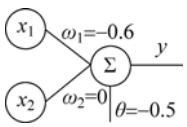
图 5.7　线性可分数据集示意图

因此也可以把感知机理解为，寻找一个可以把数据集有效划分的超平面。感知机计算表达式为

$$y = f\left(\sum_i \omega_i x_i - \theta\right) \tag{5.6}$$

其中，$\boldsymbol{x}$ 表示模型的输入；$\boldsymbol{\omega}$ 为对应输入的权重；θ 为神经元的阈值；y 为模型的输出。感知机通过调整模型中的连接权重、神经元的阈值来拟合不同的函数。其中一个很典型的例子就是感知机可以实现逻辑运算中的与、或、非运算。表 5.1 展示了当感知机模型表示不同逻辑运算时，权重、阈值的参数情况。这里假设输入、输出均为布尔值，激活函数为阶跃函数。

表 5.1　感知机实现逻辑运算

逻辑关系	感知机示意图	模型输出
与		$y=f(1\times x_1+1\times x_2-2)$ $x_1=1, x_2=1\rightarrow y=1$ $x_1=0, x_2=1\rightarrow y=0$ …
或		$y=f(1\times x_1+1\times x_2-0.5)$ $x_1=1, x_2=1\rightarrow y=1$ $x_1=0, x_2=1\rightarrow y=1$ …
非		$y=f(-0.6\times x_1+0\times x_2+0.5)$ $x_1=1\rightarrow y=0$ $x_1=0\rightarrow y=1$ …

感知机的损失函数为

$$L = \sum_{x_i \in M} y_i(\boldsymbol{\omega} \cdot x_i + \theta) \tag{5.7}$$

前面讲过，感知机可以理解为寻找一个可以把数据区分开的超平面，感知机的损失函数就是表示样本与超平面的关系。感知机的损失函数表示了误分类点与超平面的距离的关系，该函数为感知机学习的经验风险函数。从损失函数可以很直观地看出，误分类点与超平面的距离越近，损失函数就越小。感知机的优化目标就是尽可能地降低误分类点与超平面之间的距离，即损失函数 L 最小。

感知机通过权重的学习来寻找合适的超平面。在学习的过程中，感知机的权重值将不断地调整，感知机的学习规则如下：

$$\omega_i \leftarrow \omega_{i-1} + \Delta\boldsymbol{\omega} \tag{5.8}$$

$$\Delta\omega_i = \alpha(y - \hat{y})x_i \tag{5.9}$$

从 $\boldsymbol{\omega}$ 的更新规则可以看出，若预测正确或者达到了结束条件 $\boldsymbol{\omega}$ 将不再更新。针对线性可分问题，Minsky 证明若数据集是线性可分的，感知机的学习过程将是收敛的，即权

重$\boldsymbol{\omega}$的变换方向将朝着L减小的方向。

注意：上面介绍的感知机的学习能力非常有限，并不能处理线性不可分问题。这个问题也曾一度导致神经网络研究进入低谷，如逻辑关系中的异或问题。前面所述的两侧感知机的学习过程将不停地振荡，无法收敛。

为了解决线性不可分问题，需要引入更多的神经元。如图 5.8 所示的模型，在输入层和输出层之间增加了一层神经元，这一层被称为隐藏层，隐藏层与输出层一样，均为拥有激活函数的神经元。

例 5.2 设计一个两层感知机，并使其能计算异或($x_1=1, x_2=1 \rightarrow x_1 \mathrm{XOR} x_2=0$)问题。假设激活函数为阶跃函数。感知机模型结构图如图 5.9 所示。

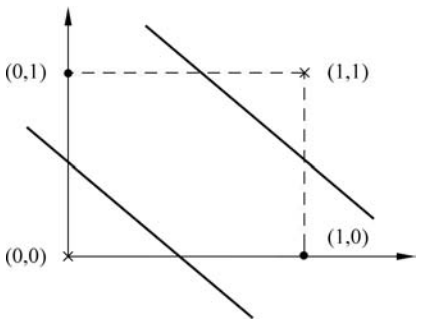


图 5.8 感知机解决异或问题

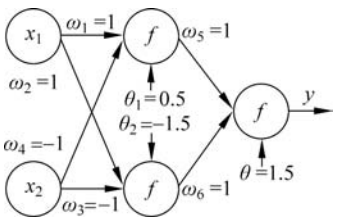


图 5.9 异或问题感知机参数示意图

当$x_1=1, x_2=1$时，

$$y_{11}=f(\omega_1 x_1+\omega_4 x_2+\theta_1)=f(1\times 1-1\times 1-0.5)=-1$$

$$y_{12}=f(\omega_2 x_1+\omega_3 x_2+\theta_2)=f(1\times 1-1\times 1+1.5)=1$$

$$y=f(\omega_5 y_{11}+\omega_6 y_{12}+\theta)=f(1\times 1-1\times 1-1.5)=-1$$

这种包含输入层、隐藏层、输出层的神经网络也称为多层前馈网络。其每一层均与前一层是全连接的结构，并且只在相邻的层与层之间存在连接关系，层内不存在神经元的连接。在前馈神经网络中，每一个神经元以上一层各个节点的输出作为输入，通过非线性的激活函数得到这个节点的输出，并传递给下一节点，信息单向向前流通，这也是“前馈”这个名字的由来。5.3 节将具体介绍这种神经网络结构。

5.3 神经网络的基本结构

前馈神经网络(feed forward neural network)又称深度前馈网络(deep feed forward network)，是一种经典的神经网络结构，前馈神经网络在自然语言处理、图像处理领域均发挥了重要的作用。

前馈神经网络这种“输入层—隐藏层—输出层”结构是神经网络的基础结构，例如，图像识别领域的卷积神经网络、针对序列化数据的循环神经网络都是基于基础的前馈神经网络衍生出来的。

从图 5.10 中可以看到前馈神经网络具有层级结构，每一层均有一定数量的神经元构成，同时也可以把这种层与层紧密相连的结构看作一个有向无环图。如果把网络的每一层看作一个函数，则这个有向无环图表示了各个函数之间的复合规则。例如，3 个函数 $f_1(x)$、$f_2(x)$和 $f_3(x)$分别代表了每一层的函数表达式，通常用链式结构来表示神经网络的运算，这个三层神经网络的计算公式为 $f(x)=f_3(f_2(f_1(x)))$。在这种链式表示中，通常约定把 $f_1(x)$称为网络的第一层，一般用 $\omega^{(1)}$表示第一层的参数，$f_2(x)$称为第二层，以此类推。这个链的全长称为神经网络模型的深度(depth)。随着神经网络层数的加深，神经网络也常常称为深度神经网络，这也正是深度学习的由来，深度学习广义上代表了具有较深层级的神经网络结构。前馈神经网络的最后一个层级是“输出层”，需要注意的是，我们需要根据任务目标来选择不同输出层，一般输出层具有全连接的结构。前馈神经网络的最终目标是寻找一个能拟合真实函数分布的模型参数，神经网络寻找拟合参数的过程通常称为神经网络的训练，即通过神经网络优化算法，让神经网络的分布 $f(x)$在一次又一次的训练过程中不断地拟合数据的真实分布 $f^*(x)$，让二者之间的差值在训练的过程中不断减小。训练数据通常为不同取值的一批数据点 x 和其对应的函数取值 y。训练样本指明了输出层在每一个点上的输出，即必须产生一个接近 y 的值。但是训练数据并不能直接影响隐藏层或者其他层，因此神经网络的学习算法必须能够改变这些隐藏层的参数，并通过这些隐藏层来影响最后的输出。后续会较为详细地介绍几种常见的神经网络优化算法。

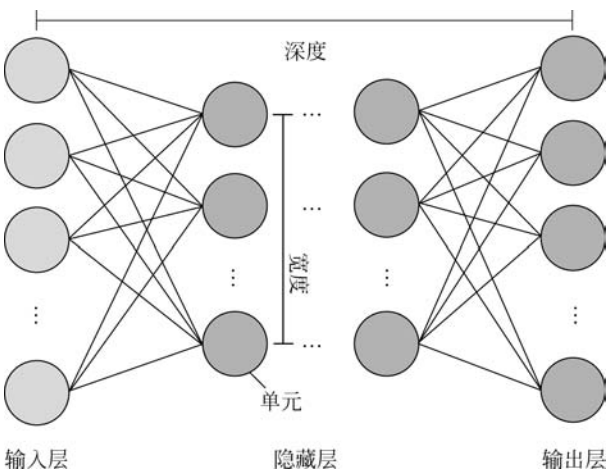


图 5.10　神经网络的一般结构

例 5.3　假设有如图 5.11 所示的神经网络，其输入向量 $\boldsymbol{X}=[2.1,0.53,1.48]$，输入层到隐藏层第 i 个神经元权重参数 $\boldsymbol{\omega}_1=[0.5,2.23,1.14]$，$\boldsymbol{\omega}_2=[2.1,0.43,1.23]$，$\boldsymbol{\omega}_3=[1.2,2.33,0.4]$，隐藏层到输出层的权重参数为 $\boldsymbol{\omega}=[0.23,1.22,3.11]$，假设隐藏层和输出层的激活函数均为 sigmoid 函数，求神经

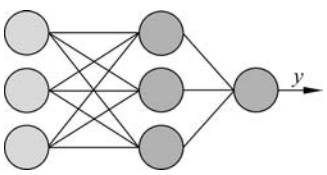


图 5.11　例 5.3 的神经网络

网络的输出 y。

$$
\begin{aligned}
x_1^{(2)} &= \text{sigmoid}(\boldsymbol{\omega}_1^{\mathrm{T}}\boldsymbol{X}^{(1)}) = \text{sigmoid}(2.1\times 0.5 + 0.53\times 2.23 + 1.48\times 1.14)\\
&= 0.9805\\
x_2^{(2)} &= \text{sigmoid}(\boldsymbol{\omega}_2^{\mathrm{T}}\boldsymbol{X}^{(1)}) = \text{sigmoid}(2.1\times 2.1 + 0.53\times 0.43 + 1.48\times 1.23)\\
&= 0.9984\\
x_3^{(2)} &= \text{sigmoid}(\boldsymbol{\omega}_3^{\mathrm{T}}\boldsymbol{X}^{(1)}) = \text{sigmoid}(2.1\times 1.2 + 0.53\times 2.33 + 1.48\times 0.4)\\
&= 0.9872\\
\boldsymbol{X}^{(2)} &= [x_1^{(2)}, x_2^{(2)}, x_3^{(2)}]\\
y &= \text{sigmoid}(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{X}^{(2)})\\
&= \text{sigmoid}(0.9805\times 0.23 + 0.9984\times 1.22 + 0.9872\times 3.11) = 0.9892
\end{aligned}
$$

神经网络具有很强的学习能力。一个前馈神经网络如果具有线性输出层和能够映射到一定范围内的非线性层，在一定情况下只要神经网络的神经元数或者层级足够多，就能够以任意的精度去拟合任何一个有限维空间的 borel 可测函数。这里仅需了解定义在$\mathbf{R}^n$ 的有界闭集上的任意连续函数均是 borel 可测的即可。这意味着无论试图学习什么函数总有一个神经网络模型能够表示这个函数，但是有可能这个模型参数量非常大，非常难以收敛。

深度学习是人工智能领域一个重要的研究方向。典型的深度学习模型就是层数较大的神经网络模型。从理论上来说，模型的参数数量越多，学习能力就越强，更能够适应更复杂的学习任务，但是从另一个角度来看，更大的参数意味着模型收敛难度的增加，计算量的增加。随着大数据、云计算时代的到来，深度学习逐渐得到越来越多人的关注。

随着人类计算能力和数据量的提升，深度学习在各个领域均发挥了巨大的作用，如随着自然语言处理技术和知识图谱技术的引入，具备了构建更智能、更高效的搜索引擎的可能；随着卷积神经网络及其他一系列神经网络的引入，在图像处理、图像识别领域的某些任务下计算机取得了超越人类的表现。深度学习技术通过模仿类比人类的思考方式，让人类在各个领域都取得了长足的进步。

除了“输入层—隐藏层—输出层”这种常见的结构，还有一些类型的神经网络也得到了应用。

1. 径向基(RBF)神经网络

径向基神经网络(见图 5.12)与前面介绍的前馈神经网络类似，但是径向基神经网络的激活函数由径向基函数构成，输出层为隐藏层输出的线性组合。常用的高斯径向基函数如下：

$$\rho(\boldsymbol{x}, \boldsymbol{c}_i) = \mathrm{e}^{-\beta_i \|\boldsymbol{x}-\boldsymbol{c}_i\|^2} \tag{5.9}$$

目前已证明 RBFNN 能够以任意精度逼近任意连续的非线性网络，被广泛用于函数逼近、语音识别、模式识别、图像处理、自动控制和故障诊断等领域。

2. 脉冲神经网络(SNN)

脉冲神经网络(见图 5.13)也被誉为下一代神经网络,旨在弥合神经科学和机器学习之间的差距,使用最拟合生物神经元机制的模型来进行计算。脉冲神经网络与目前流行的神经网络和机器学习方法有着根本上的不同,脉冲神经网络包含了时间尺度的信息。在脉冲神经网络中信息的传递是基于脉冲进行的。所以网络的输入要进行额外编码,例如,频率编码和时间编码等,将现在的数据(例如,图片的像素)转换成脉冲。同时这种基于脉冲的编码也蕴含了更多的信息。但是由于理论还不够完备,脉冲神经网络相比于前馈神经网络还没有得到大规模的应用。

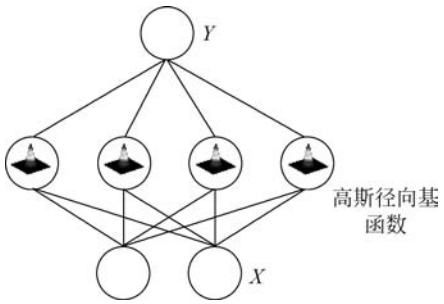


图 5.12 径向基神经网络

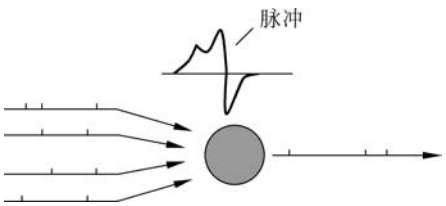


图 5.13 脉冲神经网络示意图

3. 递归神经网络

递归神经网络是一种存在环结构的网络,其中 Elman 是一种著名的递归神经网络。Elman 的结构如图 5.14 所示,它拥有与前馈神经网络很类似的结构,但不同的是其隐藏层的输出被反馈至输入部分,并与下一时刻的输入信号相结合,作为神经网络下一时刻的输入。Elman 网络可以看作是一个具有局部记忆单元和局部反馈连接的递归神经网络。Elman 神经网络在处理语音处理问题上曾经发挥了很大的作用。

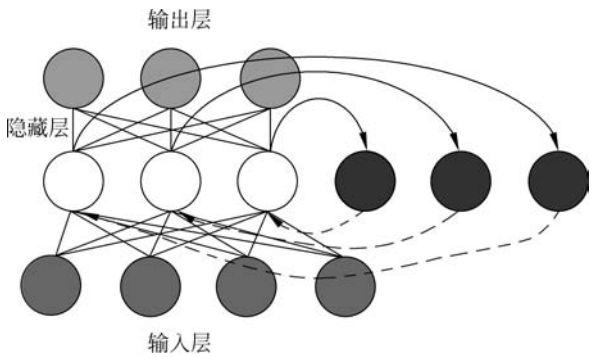


图 5.14 Elman 网络示意图

5.4 反向传播

前馈神经网络相比于感知机有着更强的学习能力，感知机的学习算法已经不再适用于多层神经网络。Werbos 在 1974 年首次提出了反向传播算法(BP)，给神经网络提供了一个高效可用的学习算法，直到今日 BP 算法仍发挥着重要的作用。在训练前馈神经网络时常采用 BP 算法，因此前馈神经网络也常被称为 BP 网络。在 BP 网络中，信息前向流通，误差反向传播。下面详细介绍 BP 算法。

假设给定数据集 $T=\{(x_1,y_1),(x_2,y_2),\cdots,(x_n,y_n)\}$，神经网络的损失值为 L。BP 算法可以分为以下两部分：

(1) 正向传播(计算模型损失函数)。在这个过程中，根据输入的样本和神经网络的初始参数计算模型输出值 $\hat{y}$ 与真实值 y 之间的损失值 L。若 L 比预期的损失值大，则执行反向传播，更新模型参数；若 L 小于预期的损失值，则停止模型参数的更新。

(2) 反向传播(误差的反向传播)将误差逆向传播至隐藏层神经元，根据隐藏层神经元的误差对模型参数进行调整。

下面给出一个具体的案例：BP 算法是如何工作的。如图 5.15 所示的一个三层神经网络，设样本数为 n，输入层由 d 个神经元组成，隐藏层由 q 个神经元组成，输出层由 l 个神经元组成，模型的激活函数设为 sigmoid 函数。

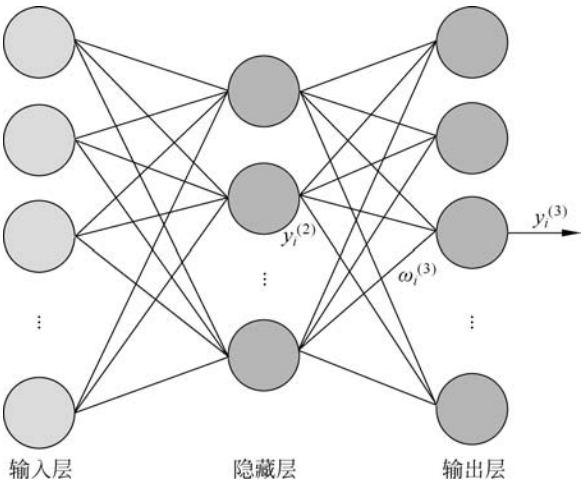


图 5.15 反向传播示意图

假设模型的损失值为

$$L=\sum L_i \tag{5.10}$$

$$L_i=\frac{1}{2}\sum_{j=1}^{l}(y_j-\hat{y}_j)^2 \tag{5.11}$$

BP 算法是通过在每一轮的迭代过程中对模型中的参数进行更新，更新规则与前述的感

知机的更新规则类似

$$\omega_i \leftarrow \omega_{i-1} + \Delta\omega \tag{5.12}$$

其中 i 表示了迭代的轮数。

BP 算法的学习策略基于梯度下降算法，根据参数负梯度的方向进行调整。这里只研究单一数据的 L_i，可以使用顺序优化的方法计算，或者使用批处理的方法在训练集上进行累加。本例仅给出权重参数 ω 的计算过程，其余参数计算过程类似，对给定的 L_i 有

$$\Delta\omega_i = -\alpha \frac{\partial L_i}{\partial \omega_{ih}}$$

$$\hat{y} = f(\beta(\omega, x))$$

$$\beta(\omega, x, \theta) = \omega x + \theta$$

根据链式求导法则

$$\Delta\omega_i = -\alpha \frac{\partial L_i}{\partial \omega_{ih}} = -\alpha \frac{\partial L_i}{\partial y^{(3)}} \frac{\partial y^{(3)}}{\partial \beta} \frac{\partial \beta}{\partial \omega_i^{(3)}}$$

其中，$y_i^{(j)}$ 表示第 j 层第 i 个神经元的输出；β 表示第三层神经元的输入。

$$\beta = \sum \omega_i^{(3)} y^{(2)}$$

$$\frac{\partial \beta}{\partial \omega_i} = y_i^{(3)}$$

对于 sigmoid 函数有以下性质

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

$$f'(x) = f(x)(a - f(x))$$

推导过程如下：

$$\begin{aligned} f'(x) &= -\frac{1}{(1 + e^{-x})^2}(-e^{-x}) \\ &= \left(\frac{1}{1 + e^{-x}}\right)\left(1 - \frac{1}{1 + e^{-x}}\right) \\ &= f(x)(1 - f(x)) \end{aligned}$$

则

$$\frac{\partial \hat{y}}{\partial \beta} = f(\beta)(1 - f(\beta))$$

根据上式可得

$$\begin{aligned} \Delta\omega_i &= -\alpha x \frac{\partial L_i}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \beta} \\ &= \alpha x (y_j - \hat{y}) f'(\beta) \\ &= \alpha x \hat{y}(1 - \hat{y})(y_j - \hat{y}) \end{aligned}$$

至此，便得到了参数 ω 在学习过程中的更新规则，其他参数的更新规则与此类似，在

神经网络的训练过程中，有一些超参数需要人为设定，如神经网络层数、模型维度、学习率等。其中学习率在神经网络的学习过程中是一个非常重要的参数，学习率过大可能导致损失值振荡无法收敛，学习率过小可能导致训练速度过慢，学习率对模型训练的影响如图 5.16 所示，在实际的神经网络训练中，对最优学习率的寻找往往要花费大量的时间，对于学习率或者其他超参数的调节，并没有统一的标准或者理论支持，更多的是通过实践经验来进行调节。此外模型的最大训练次数也是一个重要的超参数，通常在学习过程中把数据集遍历一遍称为一个 epoch。

图 5.16　学习率对训练过程的影响

现在对反向传播的整个过程做以下总结。在反向传播算法中，数据先前向流动，根据神经网络初始化的参数，计算模型的输出和损失函数，然后根据损失函数和链式求导法则分别求出每层神经元对应的梯度值，再根据梯度值计算出各个参数的更新值。在模型的损失函数不小于设定阈值或者没有达到最大迭代次数之前不断重复上述“前向流动-反向传播”的过程，直到模型达到设定条件。

综上所述，反向传播算法的伪代码如下：

给定最大迭代次数 epoch_num，训练集 $T=\{(x_1,y_1),(x_2,y_2),\cdots,(x_n,y_n)\}$

Step1. 根据参数初始化策略，随机初始化神经网络参数
Step2. for(i = 0,1,2,…,epoch_num):
Step3.　　计算模型输出 $\hat{y}_i = f(x_i,\theta)$;
Step4.　　计算模型损失函数 L;
Step5.　　计算输出层神经元的梯度;
Step6.　　计算各个隐藏层的梯度;
Step7.　　根据链式法则和梯度下降算法计算各个参数的更新值，并更新参数;
Step8.　　if($L<L_{\min}$)　　break;

例 5.4　假设有如图 5.17 所示的神经网络，其输入向量 $\boldsymbol{X}=[2.1,0.53,1.48]$，输入层到隐藏层第 i 个神经元权重参数 $\boldsymbol{\omega}_1=[0.5,2.23,1.14]$，$\boldsymbol{\omega}_2=[2.1,0.43,1.23]$，$\boldsymbol{\omega}_3=[1.2,2.33,0.4]$，隐藏层到输出层的权重参数为 $\boldsymbol{\omega}=[0.23,1.22,3.11]$，假设隐藏层和输出层的激活函数均为 sigmoid 函数，真实 y 值为 1，损失函数为平方差损失函数，求在第一次训练后 $\boldsymbol{\omega}$ 的更新值。

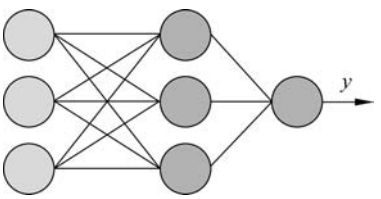


图 5.17　例 5.4 的神经网络

前向传播：

$$x_1^{(2)} = \text{sigmoid}(\boldsymbol{\omega}_1^{\mathrm{T}}\boldsymbol{X}^{(1)}) = \text{sigmoid}(2.1\times0.5+0.53\times2.23+1.48\times1.14)$$
$$= 0.9805$$
$$x_2^{(2)} = \text{sigmoid}(\boldsymbol{\omega}_2^{\mathrm{T}}\boldsymbol{X}^{(1)}) = \text{sigmoid}(2.1\times2.1+0.53\times0.43+1.48\times1.23)$$
$$= 0.9984$$
$$x_3^{(2)} = \text{sigmoid}(\boldsymbol{\omega}_3^{\mathrm{T}}\boldsymbol{X}^{(1)}) = \text{sigmoid}(2.1\times1.2+0.53\times2.33+1.48\times0.4)$$
$$= 0.9872$$
$$\boldsymbol{X}^{(2)} = [x_1^{(2)}, x_2^{(2)}, x_3^{(2)}]$$
$$\hat{y} = \text{sigmoid}(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{X}^{(2)})$$
$$= \text{sigmoid}(0.9805\times0.23+0.9984\times1.22+0.9872\times3.11) = 0.9892$$
$$L = (\hat{y}-y)^2 = 1.1664\text{e}-04$$
$$\nabla\omega_1 = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial \boldsymbol{\omega}_1} = 0.0108\times0.2468\times0.9805 = 0.0026$$
$$\nabla\omega_2 = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial \boldsymbol{\omega}_2} = 0.0108\times0.1762\times0.9984 = 0.0019$$
$$\nabla\omega_3 = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial \boldsymbol{\omega}_3} = 0.0108\times0.0424\times0.9872 = 4.5206\text{e}-04$$

反向传播算法的最终优化目标是数据集上所有样本损失函数的和，上面参数更新的依据是每个数据单独的损失函数 L，每计算一个数据的损失值更新一次参数，并通过各个数据损失函数的累加来达到整体数据集损失函数最小化的要求。还有一种累加反向传播算法，这种累加算法直接根据整体数据损失函数的和来更新参数。累加反向传播算法相比与前面介绍的反向传播算法参数更新频率要低，这种算法损失值的波动更小，但是同时更新时间会变得很慢，当数据集较大时对硬件内存也提出了更高的要求。这两种算法与 5.5 节介绍的梯度下降算法和随机梯度下降算法有着类似的区别。

5.5 梯度下降算法

在神经网络优化过程中，为了最小化损失函数，常用梯度下降算法寻找最优参数。通过第 2 章对梯度下降算法的介绍，可以知道一个函数 f 在某点的负梯度代表着函数下降最快的方向，换句话说，在负梯度上移动可以减小函数 f。这种方法称为梯度下降算法。参数更新方法如下：

$$\omega_i \leftarrow \omega_{i-1} - \alpha\nabla_\omega f \tag{5.13}$$

其中，α 表示模型的学习率；$\nabla_\omega f$ 表示函数 f 对参数 ω 的梯度。其实可以很自然地把梯度下降的过程比作蚂蚁下山的问题，为了最快到达山脚，蚂蚁总是会选择最陡峭的地方走，因为最陡峭的地方意味着更快的下降速率，这里的梯度就代表了山最陡峭的地方，即下降率最快的地方，而学习率则可以理解为每一步的步长，即沿着梯度下降的方向走的距离。梯度下降示意图如图 5.18 所示。

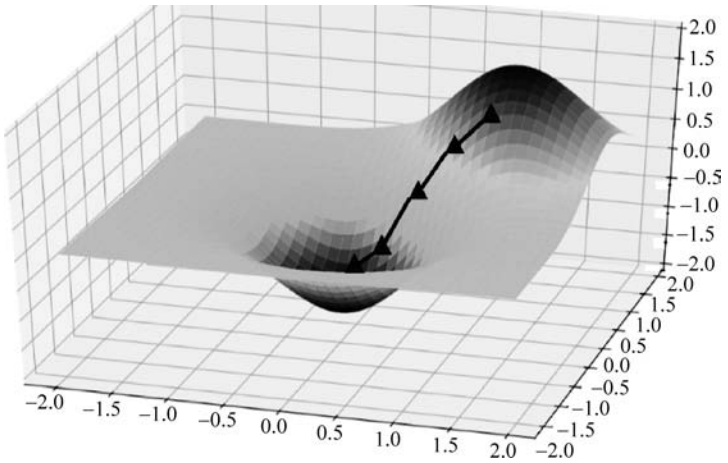


图 5.18　梯度下降示意图

梯度下降在实际使用中，产生了许多不同的方法，如标准梯度下降算法、随机梯度下降算法、mini-batch 算法等，但是无论何种梯度下降算法，最终的优化目标都是使整体数据集的损失函数之和最小，假设训练集样本数为 n，在训练过程中损失函数值为每个样本损失值之和

$$L=\sum L_i \tag{5.14}$$

本节主要介绍两种梯度下降算法，分别是标准梯度下降算法和 mini-batch 梯度下降算法，这两种算法的差异主要体现在参数的更新时机不同。

标准梯度下降算法：梯度下降算法根据所有数据的损失函数之和更新参数。注意，在基础的梯度下降算法中，利用的是全部样本的损失值之和。对于参数 ω，其梯度为

$$\nabla_\omega L=\frac{1}{n}\sum_{i=1}^{n}\nabla_\omega L \tag{5.15}$$

梯度下降算法对于寻找模型的最优解有着重要的意义，它的优点是计算效率高，产生一个稳定的误差梯度和稳定的收敛。它的缺点是，稳定的误差梯度有时可能导致收敛状态不是模型能达到的最佳状态。它还要求整个训练数据集存储在内存中并可供算法使用，但是随着样本数量的增加，计算每一步梯度的时间会大大增加，并且当数据量增加时对计算设备的内存要求增加，为了解决这些问题，梯度下降算法产生了各种变种形式。

mini-batch 梯度下降算法：mini-batch 梯度下降算法在神经网络领域起着非常重要的作用，它的提出为用于大规模数据集的更好更快的训练模型提供了解决方案。这种算法更像是上述算法的一种折中，在保证了训练效果的同时，又兼顾了训练的效率。如前所述，随机梯度下降的主要思想是将整体的数据集 n 拆分大小相等的数据集合 n'，注意，各个数据集合中的数据应满足独立同分布的原则，一般做法是进行随机划分。通过计算 n' 上的梯度的和，来对模型的参数进行更新。此时模型参数的更新规则如下：

$$\Delta\omega=\alpha\sum_{i=1}^{n'}\frac{\partial L_i}{\partial\omega}$$

$$\omega_i \leftarrow \omega_{i-1} + \Delta\omega$$

mini-batch 下降是在大规模数据上训练大型线性模型的主要方法。对于固定大小的模型，每一步梯度下降更新的计算量不是取决于训练集的大小 n，而是取决于每一个 batch 的大小。在实践中，每个 batch 的大小不一定需要随着训练集的大小变化而变化，这样相比于标准梯度下降算法，每一次更新参数的计算量就大大减少。

与学习率一样，每个 batch 的大小也是一个非常重要的超参数，如果 batch 选择得过小，可能造成模型收敛过程中的波动，甚至模型无法收敛；若 batch 选择得过大，一方面对设备的内存提出了更高的要求，另一方面也可能导致模型无法达到最小值点。注意，batch 大小的选择与硬件设备有着一定关联，在 GPU 设备上进行逆行训练时，常常采用 2 的指数幂作为 batch 的大小，在比较大的模型上这个值通常会选取得小一点。

上面简单介绍了几种梯度下降算法，在实际的应用中还有其他几种算法，需要根据具体的应用场景进行选择。

习题

1. 请解释线性不可分数据集与线性可分数据集的区别；举例说出解决线性不可分问题的几种方法。

2. 请利用神经网络表示以下逻辑运算过程

$$y = (x_1 \& x_2) \,||\, (x_3 \& x_4)$$

3. 梯度下降法得到的一定是最小值点吗？如果不是，请说明原因，并说明在什么情况下可以达到极小值。

4. 给出图 5.19 所示的神经网络，网络的激活函数为 relu 函数，请推导参数 $\omega_i^{(3)}$ 更新的过程。

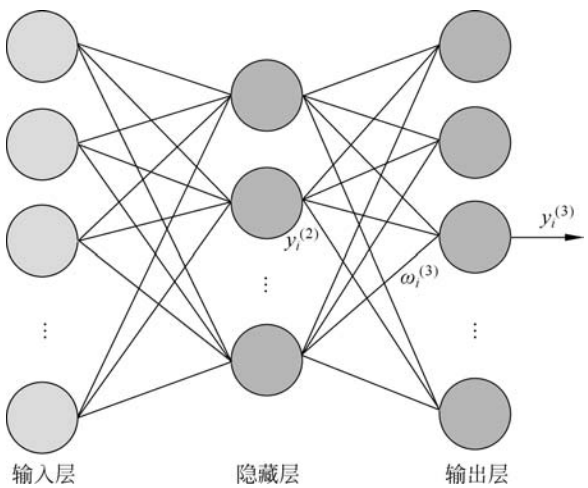


图 5.19　神经网络

5. 试描述神经网络中几种常见梯度下降方法的过程，并说明几种方法的不同。

6. 查询资料了解循环神经网络、卷积神经网络、径向基神经网络，并比较其结构上的差异。

7. 查询资料了解目前常用的损失函数计算方法，并写出 4 种。

参考文献

[1] Kohonen T. An introduction to neural computing[J]. *Neural Networks*, 1988, 1(1): 3-16.

[2] Mcculloch W S, Pitts W. A logical calculus of the ideas immanent in nervous activity[J]. *Bulletin of Mathematical Biology*, 1990, 52(4): 99-115.

[3] Rosenblatt F. The perceptron: a probabilistic model for information storage and organization in the brain. [J]. *Psychological Review*, 1958, 65(6): 386-408.

[4] Rosenblatt F. PRINCIPLES OF NEURODYNAMICS. PERCEPTRONS AND THE THEORY OF BRAIN MECHANISMS[J]. *American Journal of Psychology*, 1963, 76(4): 386-408.

[5] Marvin Minsky, Seymour Papert. *Perceptrons*. expanded edition, 1-3.

[6] Werbos P. New Tools for Prediction and Analysis in the Behavioral Sciences[D]. *Cambridge: Harvard University*, 1974: 1-3

[7] Rumelhart D E, Hinton G E, Williams R J, et al. Learning representations by back-propagating errors[J]. *Nature*, 1988, 323(6088): 696-699.

[8] Hammer B. Neural Smithing—Supervised Learning in Feedforward Artificial Neural Networks[J]. *Pattern Analysis and Applications*, 2001, 4(1): 73-74.

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

[10] Gori M, Tesi A. On the problem of local minima in backpropagation[J]. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1992, 14(1): 76-86.

[11] Haykin S. *Neural Networks: A Comprehensive Foundation*, 73-78.

[12] Montavon G, Orr G, Mller K. *Neural Networks: Tricks of the Trade*, 72-83.

[13] Lecun Y, Bottou L, Bengio Y, et al. *Gradient-based learning applied to document recognition*, 46-56.

[14] Nair V, Hinton G E. Rectified Linear Units Improve Restricted Boltzmann Machines[C]. *International conference on machine learning*, 2010: 807-814.

[15] Goodfellow I, Bengio Y, Courville A. Deep Learning[M]. MIT Press, 2016. 111-115.

[16] 周志华. 机器学习[M]. 北京：清华大学出版社，2016.

[17] Bishop C. *Pattern Recognition and Machine Learning*[M]. Springer, 2006. 233-234.