5.1 神经元模型

5.1.1 M-P神经元模型

神经网络中最基本的成分是神经元(neuron)模型.

"M-P"神经元模型: 神经元接收到来自 n 个其他神经元传递过来的输信号. 这些输入信号通过带权重的连接(connection)进行传递, 神经元接收到的总输入值将与神经元 的阈值进行比较, 然后通过"激活函数"(activation function)处理以产生神经元的输出.

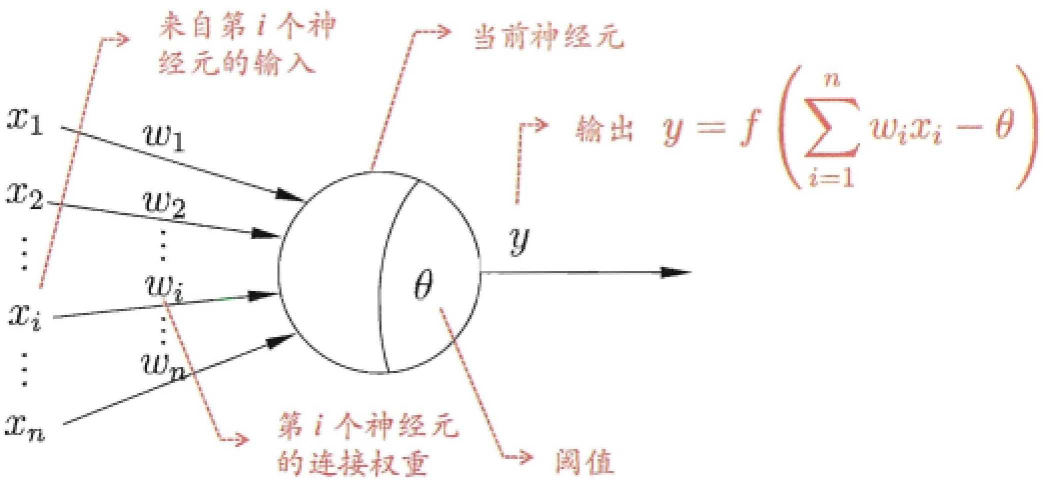

图 5.1　M-P 神经元模型

5.1.2 激活函数-阶跃函数

理想中的激活函数是图 5 .2(a) 所示的**阶跃函数**, 它将输入值映射为输出值"0"或"1", 显然 "1" 对应于神经元兴奋 , "0"对应于神经元抑制 . 然而, 阶跃函数具有不连 续 、不光滑等不太好的 性质, 因此实际常用**Sigmoid函数**作为激活函数.

典型的 Sigmoid 函数如 图 5.2(b) 所示, 它把可能在较大范围内变化的输入值挤压到 (0, 1) 输出值范围内, 因此有 时也称为 "挤压函数".

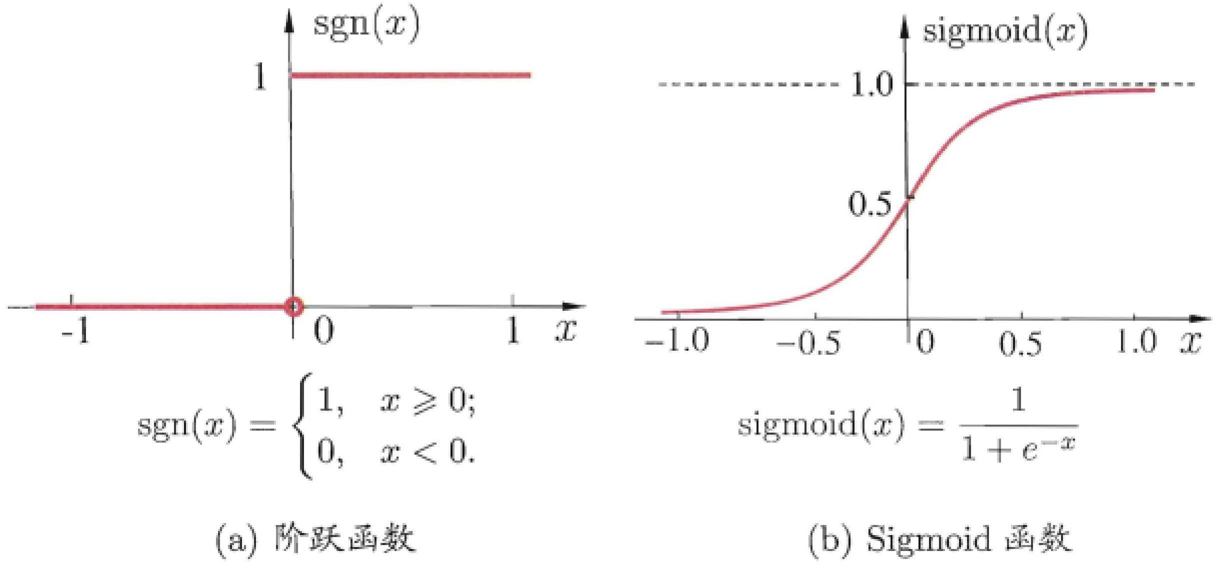

图 5.2 典型的神经元激活函数

把许多个这样的神经元按-定的层次结构连接起来, 就得到了神经网络.

5.2 感知机与多层网络

5.2.1 感知机的基本概念

感知 机(Perceptron) 由两层神经元组成, 如图5.3所示, 输入层接收外界输入信号后传递给输出层, 输出层是M-P 神经元, 也称"阈值逻辑单元".

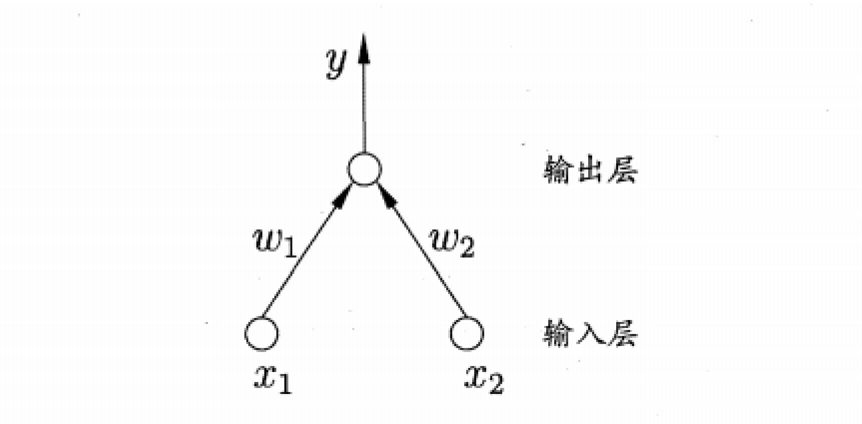

图 5.3 两个输入神经元的感知机网络结构示意图

更一般地, 给定训练数据集, 权重叫 $w_i(i=1,2,\ldots,n)$ 以及阈值 θ . 可通过学习得到. 阈值 θ 可看作一个固定输入为 -1 的"哑结点" (dummy node) 所对应的连接权重 w_{n+1} , 这样, 权重和阈值的学习就可统一为权重的学习.

感知机学习规则非常简单, 对训练样例 $(\boldsymbol{x}, y)$, 若当前感知机的输出为 $\hat{y}$, 则感知机权重将这样调整:

$$w_i \leftarrow w_i + \Delta w_i \tag{5.1}$$

$$\Delta w_i = \eta(y - \hat{y})x_i \tag{5.2}$$

其中 $\eta \in (0,1)$ 称为学习率(learning rate). 若感知机对训练样例 $(\boldsymbol{x}, y)$ 预测正确, 即 $\hat{y} = y$, 则感知机不发生变化, 否则将根据错误的程度进行权重调整.

若两类模式是线性可分的, 即存在一个线性超平面能将它们分开, 感知在的学习过程一定会收敛(converge) 而求得适当的权向量 $\boldsymbol{w} = (w_1; w_2; \ldots; w_{n+1})$; 否则感知机学习过程将会发生振荡 (fluctuation) , $\boldsymbol{w}$ 难以稳定下来, 不能求得合适解.

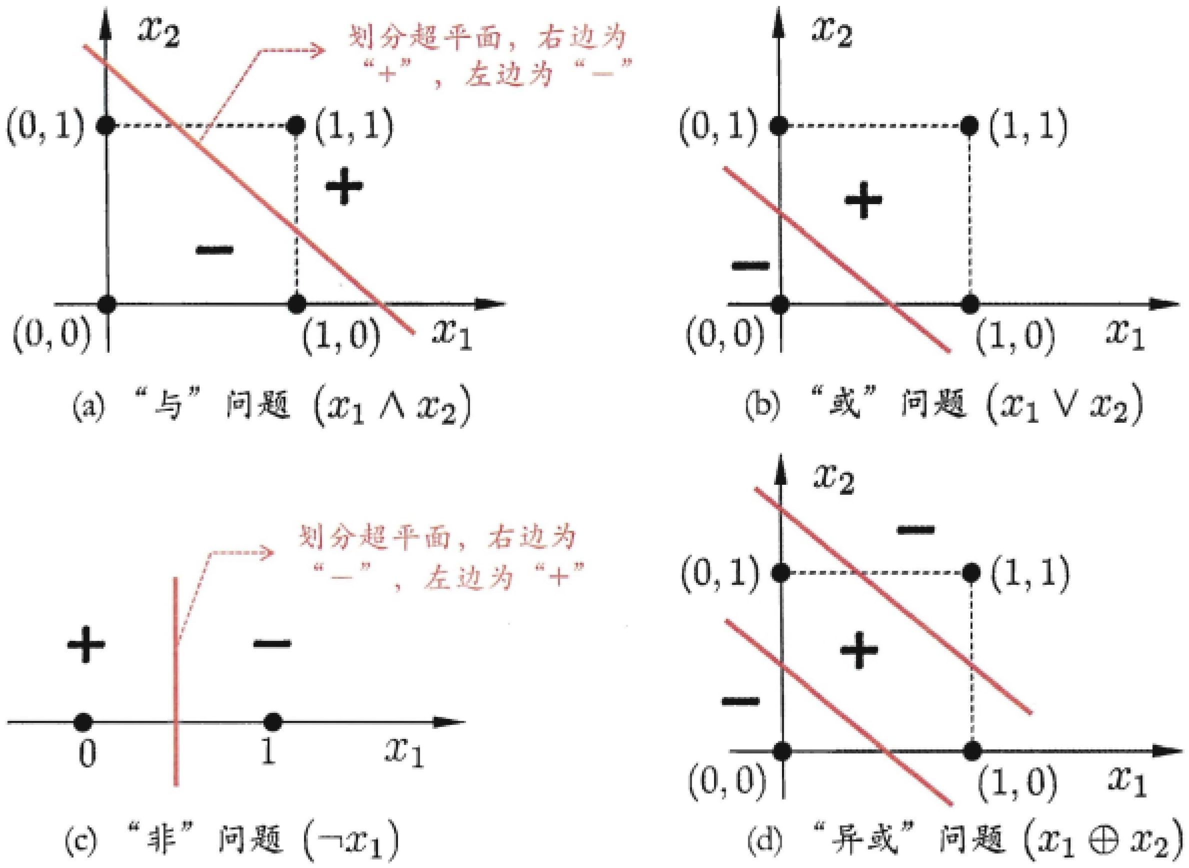

图 5.4 线性可分的“与”“或”“非”问题与非线性可分的“异或”问题

5.2.2 更一般的感知机

要解决非线性可分问题, 需考虑使用多层功能神经元. 例如图5.5中这个简单的两层感知机就能解决异或问题.

在图5.5(a)中, 输出层与输入居之间的一层神经元, 被称为**隐层**或**隐含层** (hidden layer), **隐含层**和**输出层**神经元**都是拥有激活函数**的功能神经元.

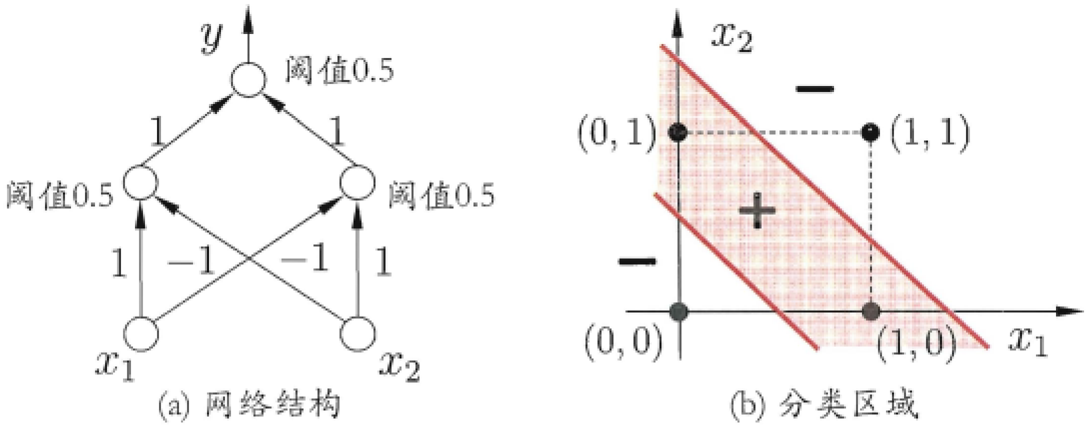

图 5.5 能解决异或问题的两层感知机

多层前馈神经网络: 更一般的, 常见的神经网络是形如图5.6所示的层级结构, 每层神经元与下层神经元全互连, 神经元之间不存在同层连接, 也不存在跨层连接. 这样的神经网络结构通常称为**"多层前馈神经网络"** .

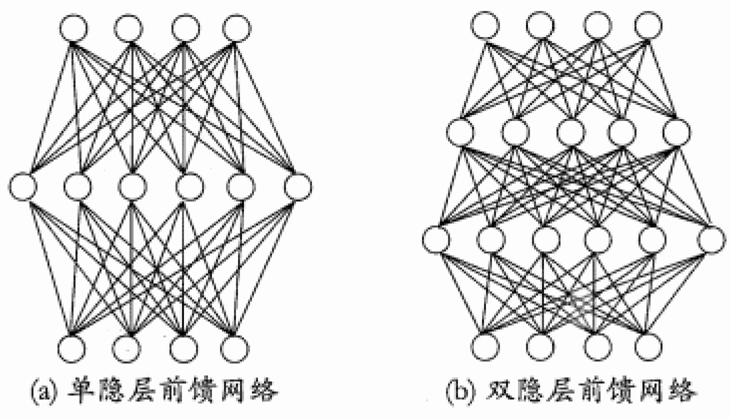

图 5.6 多层前馈神经网络结构示意图

5.3 误差逆传播算法(BP算法)

5.3.1 误差逆传播算法的基本概念

多层网络的学习能力比单层感知机强得多. 误差逆传播 (error Back Propagation, 简称BP)算法就是其中最杰出的代表, 它是迄今最成功的神经网络学习算法.

给定训练集 $D=\{(\boldsymbol{x}_1,\boldsymbol{y}_1),(\boldsymbol{x}_2,\boldsymbol{y}_2),\ldots,(\boldsymbol{x}_m,\boldsymbol{y}_m)\}$, $\boldsymbol{x}_i\in\mathbb{R}^d,\boldsymbol{y}_i\in\mathbb{R}^l$, 即输入示例由 d 个属性描述, 输出 l 维实值向量. 如图5.7给出了一个拥有 d 个输入神经元、l 个输出神经元、q 个隐层神经元的多层前馈网络结构.其中, 输出层第 j 个神经元的阈值用 θ_j 表示, 隐层第 h 个神经元的阈值用 γ_h 表示. 输入层第 i 个神经元与隐层第 h 个神经元之间的连接权为 v_{ih} , 隐层第 h 个神经元与输出层第 j 个神经元之间的连接权为 w_{hj} . 记隐层第 h 个神经元接收到的输入为 $\alpha_h=\sum_{i=1}^d v_{ih}x_i$, 输出层第 j 个神经元接收倒的输入为 $\beta_j=\sum_{h=1}^q w_{hj}b_h$, 其中 b_h 为隐层第 h 个神经元的输出. 且假设隐层和输出层神经元都使用图 5.2(b) 中的Sigmoid函数.

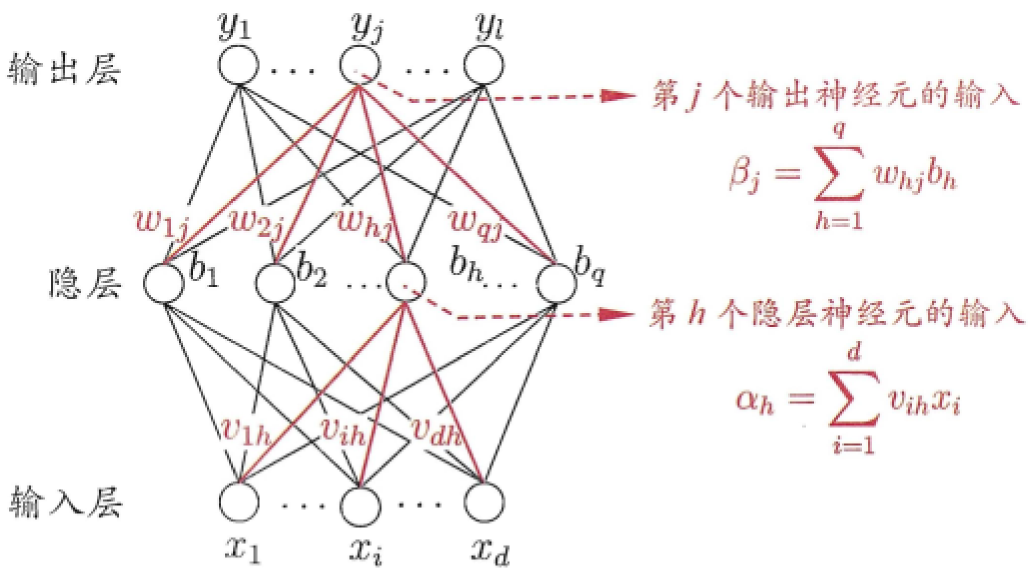

图 5.7 BP 网络及算法中的变量符号

对训练例 $(\boldsymbol{x}_k,\boldsymbol{y}_k)$, 假定神经网络的输出为 $\hat{\boldsymbol{y}}_k=\left(\hat{y}_1^k,\hat{y}_2^k,\ldots,\hat{y}_l^k\right)$, 即

$$\hat{y}_j^k=f\left(\beta_j-\theta_j\right) \tag{5.3}$$

则网络在 $(\boldsymbol{x}_k,\boldsymbol{y}_k)$ 上的均方误差为

$$E_k=\frac{1}{2}\sum_{j=1}^l\left(\hat{y}_j^k-y_j^k\right)^2 \tag{5.4}$$

参数的确定:

图5.7 的网络中有 $(d+l+1)q+l$ 个参数需确定: 输入层到隐层的 $d\times q$ 个权值、隐层到输出层的 $q\times l$ 个权值、q 个隐层神经元的阈值、l 个输出层神经元的阈值. BP 是一个迭代学习算法, 不断的对参数进行更新估计. 与式(5.1)类似, 任意参数 v 的更新估计式为:

$$v\leftarrow v+\Delta v \tag{5.5}$$

5.3.2 参数更新的推导

1 Δw_{hj} 的推导

1. BP算法基于梯度下降策略, 以目标的负梯度方向对参数进行调整. 根据式(5.4), 给定学习率 η , 有

$$\Delta w_{hj} = -\eta \frac{\partial E_k}{\partial w_{hj}} \tag{5.6}$$

2. 根据链式法则, w_{hj} 先影响到第 j 个输出层神经元的输入值 β_j , 再进而影响到其输出值 $\hat{y}_j^k$, 最后影响到 E_k , 则有:

$$\frac{\partial E_k}{\partial w_{hj}} = \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial w_{hj}} \tag{5.7}$$

3. 因为 $\beta_j = \sum_{h=1}^{q} w_{hj} b_h$, 所以有

$$\frac{\partial \beta_j}{\partial w_{hj}} = b_h \tag{5.8}$$

> 注1: 关于有求和符号的求导
>
> 参加知乎的一个回答:
>
> 几个月过去了，估计题主的问题已经解决了，答案写给后来的人看。
>
> 我刚刚也遇到了这个问题，开始没想明白，但是当我把求和符号打开的时候，一切就都明朗了。
>
> $$\frac{d(C \times \sum_{i=1}^{m} \xi_i)}{d\xi_i} = \frac{d(C \times (\xi_1 + \xi_2 + \cdots + \xi_m))}{d\xi_i} = \frac{d(C \times \xi_1 + C \times \xi_2 + \cdots + C \times \xi_m)}{d\xi_i}$$
>
> 当 $i = 1$ 时
>
> $$\frac{d(C \times \xi_1 + C \times \xi_2 + \cdots + C \times \xi_m)}{d\xi_1} = \frac{d(C \times \xi_1)}{d\xi_1} = C$$
>
> 当 $i = 2$ 时
>
> $$\frac{d(C \times \xi_1 + C \times \xi_2 + \cdots + C \times \xi_m)}{d\xi_2} = \frac{d(C \times \xi_2)}{d\xi_2} = C$$
>
> $\cdots\cdots$
>
> 当 $i = m$ 时
>
> $$\frac{d(C \times \xi_1 + C \times \xi_2 + \cdots + C \times \xi_m)}{d\xi_m} = \frac{d(C \times \xi_m)}{d\xi_m} = C$$
>
> 所以不论 i 的取值，结果都是 C
>
> 延展: 如果是 $\frac{d\left(C \times \sum_{i=1}^{m} \sum_{j=1}^{l} \beta_j \xi_i\right)}{d\xi_i}$ 求导呢, 答案**应该是** $C \times \sum_{j=1}^{l} \beta_j$

4. 同时, Sigmoid 函数有一个很好的性质:

$$f'(x)=f(x)(1-f(x)) \tag{5.9}$$

5. 根据式(5.3)和式(5.4), 定义$g_j=-\frac{\partial E_k}{\partial \hat{y}_j^k}\cdot\frac{\partial \hat{y}_j^k}{\partial \beta_j}$, 继续展开有

$$\begin{aligned} g_j &= -\frac{\partial E_k}{\partial \hat{y}_j^k}\cdot\frac{\partial \hat{y}_j^k}{\partial \beta_j} \\ &= -\left(\hat{y}_j^k-y_j^k\right) f'\left(\beta_j-\theta_j\right) \\ &= \hat{y}_j^k\left(1-\hat{y}_j^k\right)\left(y_j^k-\hat{y}_j^k\right) \end{aligned} \tag{5.10}$$

6. 将式(5.10)和(5.8)带入式(5.7), 最后带入到式(5.6), 就得到了BP算法中关于 w_{hj} 的更新公

$$\Delta w_{hj}=\eta g_j b_h \tag{5.11}$$

其中, $g_j=\hat{y}_j^k\left(1-\hat{y}_j^k\right)\left(y_j^k-\hat{y}_j^k\right)$

2. 其他类似参数的更新公式

类似的, 可以得到

$$\Delta\theta_j=-\eta g_j \tag{5.12}$$

$$\Delta v_{ih}=\eta e_h x_i \tag{5.13}$$

$$\Delta\gamma_h=-\eta e_h \tag{5.14}$$

其中, (5.13)和(5.14)中的 e_h 为:

$$\begin{aligned} e_h &= -\frac{\partial E_k}{\partial b_h}\cdot\frac{\partial b_h}{\partial \alpha_h} \\ &= -\sum_{j=1}^{l}\frac{\partial E_k}{\partial \beta_j}\cdot\frac{\partial \beta_j}{\partial b_h}f'\left(\alpha_h-\gamma_h\right) \\ &= \sum_{j=1}^{l} w_{hj} g_j f'\left(\alpha_h-\gamma_h\right) \end{aligned}$$

$$= b_h\left(1-b_h\right)\sum_{j=1}^{l} w_{hj} g_j \tag{5.15}$$

> 注2:
>
> 1. $b_h=f\left(\alpha_h-\gamma_h\right)$
> 2. 为什么会有求和符号 因为: e_h 只是关于参数 h 的, 而链式法则之后,引入了 j , 因此需要对 j 进行求和.而 j 为什么取值从1到 l , 因为, $v_i h$ 先流入b_h , 再分别流入y_l , 因此是从1到 l.

注3: 关于(5.12)、(5.13)、和(5.14)的推导, 参考**<南瓜书>**

(5.12)

$\Delta\theta_j = -\eta g_j$ [推导]: 因为 $\Delta\theta_j = -\eta\frac{\partial E_k}{\partial\theta_j}$ 又

$$\begin{aligned}\frac{\partial E_k}{\partial\theta_j} &= \frac{\partial E_k}{\partial\hat{y}_j^k}\cdot\frac{\partial\hat{y}_j^k}{\partial\theta_j} \\ &= (\hat{y}_j^k - y_j^k)\cdot f'(\beta_j-\theta_j)\cdot(-1) \\ &= -(\hat{y}_j^k - y_j^k)f'(\beta_j-\theta_j) \\ &= g_j\end{aligned}$$

所以 $\Delta\theta_j = -\eta\frac{\partial E_k}{\partial\theta_j} = -\eta g_j$

(5.13)

$\Delta v_{ih} = \eta e_h x_i$ [推导]: 因为 $\Delta v_{ih} = -\eta\frac{\partial E_k}{\partial v_{ih}}$ 又

$$\begin{aligned}\frac{\partial E_k}{\partial v_{ih}} &= \sum_{j=1}^{l}\frac{\partial E_k}{\partial\hat{y}_j^k}\cdot\frac{\partial\hat{y}_j^k}{\partial\beta_j}\cdot\frac{\partial\beta_j}{\partial b_h}\cdot\frac{\partial b_h}{\partial\alpha_h}\cdot\frac{\partial\alpha_h}{\partial v_{ih}} \\ &= \sum_{j=1}^{l}\frac{\partial E_k}{\partial\hat{y}_j^k}\cdot\frac{\partial\hat{y}_j^k}{\partial\beta_j}\cdot\frac{\partial\beta_j}{\partial b_h}\cdot\frac{\partial b_h}{\partial\alpha_h}\cdot x_i \\ &= \sum_{j=1}^{l}\frac{\partial E_k}{\partial\hat{y}_j^k}\cdot\frac{\partial\hat{y}_j^k}{\partial\beta_j}\cdot\frac{\partial\beta_j}{\partial b_h}\cdot f'(\alpha_h-\gamma_h)\cdot x_i \\ &= \sum_{j=1}^{l}\frac{\partial E_k}{\partial\hat{y}_j^k}\cdot\frac{\partial\hat{y}_j^k}{\partial\beta_j}\cdot w_{hj}\cdot f'(\alpha_h-\gamma_h)\cdot x_i \\ &= \sum_{j=1}^{l}(-g_j)\cdot w_{hj}\cdot f'(\alpha_h-\gamma_h)\cdot x_i \\ &= -f'(\alpha_h-\gamma_h)\cdot\sum_{j=1}^{l}g_j\cdot w_{hj}\cdot x_i \\ &= -b_h(1-b_h)\cdot\sum_{j=1}^{l}g_j\cdot w_{hj}\cdot x_i \\ &= -e_h\cdot x_i\end{aligned}$$

所以 $\Delta v_{ih} = -\eta\cdot -e_h\cdot x_i = \eta e_h x_i$

> ## (5.14)
>
> $\Delta\gamma_h = -\eta e_h$ [推导]：因为 $\Delta\gamma_h = -\eta\frac{\partial E_k}{\partial \gamma_h}$ 又
>
> $$\begin{aligned}\frac{\partial E_k}{\partial \gamma_h} &= \sum_{j=1}^{l}\frac{\partial E_k}{\partial \hat{y}_j^k}\cdot\frac{\partial \hat{y}_j^k}{\partial \beta_j}\cdot\frac{\partial \beta_j}{\partial b_h}\cdot\frac{\partial b_h}{\partial \gamma_h} \\ &= \sum_{j=1}^{l}\frac{\partial E_k}{\partial \hat{y}_j^k}\cdot\frac{\partial \hat{y}_j^k}{\partial \beta_j}\cdot\frac{\partial \beta_j}{\partial b_h}\cdot f'(\alpha_h-\gamma_h)\cdot(-1) \\ &= -\sum_{j=1}^{l}\frac{\partial E_k}{\partial \hat{y}_j^k}\cdot\frac{\partial \hat{y}_j^k}{\partial \beta_j}\cdot w_{hj}\cdot f'(\alpha_h-\gamma_h) \\ &= e_h\end{aligned}$$
>
> 所以 $\Delta\gamma_h = -\eta e_h$

学习率 $\eta \in (0,1)$ 控制着算法每一轮迭代中的更新步长, 若太大则容易振荡, 太小则收敛速度又会过慢. 有时为了做精细调节, 可令式(5.11)与(5.12)使用 η_1 , 式(5.13)与(5.14)使用 η_2 , 两者未必一定要相等.

5.3.3 BP算法流程

基本流程如下: 先将输入示例提供给输入层神经元, 然后逐层将信号前传, 直到产生输出层的结果; 然后计算输出层的误差(第4-5行) , 再将误差逆向传播至隐层神经元(第6行) , 最后根据隐层神经元的误差来别连接权和阈值进行调整(第7行). 该法代过程循环进行, 直到达到某些停止条件为止.

输入: 训练集 $D=\{(\boldsymbol{x}_k,\boldsymbol{y}_k)\}_{k=1}^m$;
学习率 η.

过程:

在$(0,1)$范围内随机初始化网络中所有连接权和阈值
repeat
  for all $(\boldsymbol{x}_k,\boldsymbol{y}_k)\in D$ **do**
    根据当前参数和式(5.3) 计算当前样本的输出 $\hat{\boldsymbol{y}}_k$;
    根据式(5.10) 计算输出层神经元的梯度项 g_j;
    根据式(5.15) 计算隐层神经元的梯度项 e_h;
    根据式(5.11)-(5.14) 更新连接权 w_{hj}, v_{ih} 与阈值 θ_j, γ_h
  end for
until 达到停止条件

输出: 连接权与阈值确定的多层前馈神经网络

图 5.8 误差逆传播算法

5.3.4 累计BP算法和标准BP算法

1 累积BP算法的定义

注意一点就是, BP算法的目标是要最小化训练集 D 上的累积误差

$$E=\frac{1}{m}\sum_{k=1}^{m}E_k \tag{5.16}$$

但是, 上面的"标准BP算法"每次仅针对一个训练样例更新连接权和阈值. 换句话说, 图5.8中算法的更新规则是基于单个的 E_k 推导而得. 如果类似地推导出基于累积误差最小化的更新规整, 就得到了累积误差逆传播(accumulated error backpropagation) 算法.

2 两者的区别与联系

1. 一般来说, 标准BP算法每次更新只针对单个样例, **参数更新得非常频繁**, 而且对不同样例进行更新的效果可能出现**"抵消"现象**. 因此, 为了达到同样的累积误差极小点 , 标准BP算法往往需进行更多次数的迭代.
2. 累积BP算法直接针对累积误差最小化, 它在读取整个训练集 D 一遍后才对参数进行更新, 其参数更新的频率低得多.
3. 一般, 累积误差下降到一定程度之后, 进一步下降会非常缓慢, 这时标准 BP 往往会更快获得较好的解, 尤其是在训练集 D 非常大时更明显.

5.3.5 缓解BP神经网络过拟合的策略

正是由于其强大的表示能力, BP神经网络经常遭遇过拟合, 其训练误差持续降低, 但测试误差却可能上升. 有两种策略常用来缓解BP网络的过拟合:

1. **"早停"策略**: 将数据分成训练集和验证集, 训练集用来计算梯度、更新连接权和阈值, 验证集用来估计误差, 若**训练集误差降低但验证集误差升高**, 则停止训练, 同时返回具有最小验证集误差的**连接权**和**阈值**.
2. **"正则化":** 基本思想是在误差目标函数中增加一个用于**描述网络复杂度的部分**，例如连接权与阈值的平方和. 仍令 E_k 表示第 k 个训练样例上的误差, w_i 表示连接权和阈值, 则误差目标函数(5.16) 可以改写成:

   $$E=\lambda \frac{1}{m} \sum_{k=1}^{m} E_k+(1-\lambda) \sum_i w_i^2 \tag{5.17}$$

   其中, $\lambda \in (0,1)$ 用于对经验误差与网络复杂度这两项进行折中, 常通过交叉验证法来估计.

5.4 全局最小与局部极小

5.4.1 全局最小和局部极小

1 全局最小和局部极小的概念

用 E 表示神经网络在训练集上的误差, 则它显然是关于连接权 $\boldsymbol{w}$ 和阈值 θ 的函数. 此时, 神经网络的训练过程可看作是一个参数寻优的过程, 即在参数空间中, 寻找一组最优参数使得 E 最小.

"局部极小"(local minimum)和**"全局最小"**(global minimum). 对 $\boldsymbol{w}^*$ 和 θ^* , 若存在 $\epsilon>0$ 使得

$$\forall(\boldsymbol{w};\theta) \in \{(\boldsymbol{w};\theta) \mid \|(\boldsymbol{w};\theta)-(\boldsymbol{w}^*;\theta^*)\| \leqslant \epsilon\}$$

都有 $E(\boldsymbol{w};\theta) \geqslant E(\boldsymbol{w}^*;\theta^*)$ 成立, 则 $(\boldsymbol{w}^*;\theta^*)$ 为**局部极小解**; 若**对参数空间中的任意** $(\boldsymbol{w};\theta)$, 都有 $E(\boldsymbol{w};\theta) \geqslant E(\boldsymbol{w}^*;\theta^*)$, 则 $(\boldsymbol{w}^*;\theta^*)$ 为**全局最小解**.

局部极小解是参数空间中的某个点, 其邻域点的误差函数值均不小于该点的函数值; 全局最小解则是指参数空间中所有点的误差函数值均不小于该点的误差函数值.

2 两者的联系与区别

参数空间内**梯度为零的点**, 只要其误差函数值小于邻点的误差函数值, 就是局部极小点; **可能存在多个局部极小值，但却只会有一个全局最小值**. 也就是说**"全局最小"一定是"局部极小**, 反之则不成立.

5.4.2 梯队下降和跳出局部极小"陷阱"

1 基于梯度的搜索

基于梯度的搜索是使用最为广泛的参数寻优方法.

在此类方法中, 我们从某些初始解出发, 迭代寻找最优参数值. 每次迭代中, 我们先计算误差函数在**当前点的梯度**，然后**根据梯度确定搜索方向**. 例如, 由于**负梯度**方向是函数值**下降最快的方向**, 因此**梯度下降法就是沿着负梯度方向搜索最优解**. 若误差函数在当前点的**梯度为零**, 则已达到局部极小, 更新量将为零, 这意味着参数的迭代更新将在此停止. 显然, 如果误差函数仅有一个局部极小, 那么此时找到的局部极小就是全局最小. 但实际上, 如果误差函数具有多个局部极小, 就不能保证找到的解是全局最小.

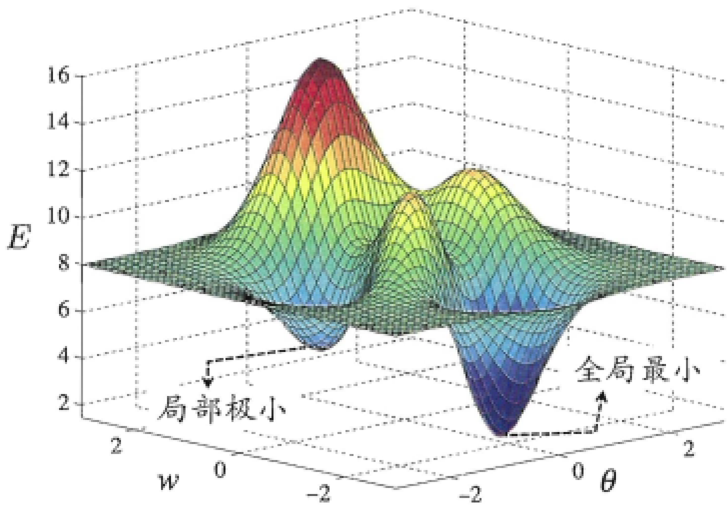

图 5.10 全局最小与局部极小

2 跳出局部极小

常用的跳出局部极小的策略有以下几种:

1. 以**多组不同参数值初始化多个神经网络**, 按标准方法训练后, 取其中误差最小的解作为最终参数 .
2. 使用 "**模拟退火**" (simulated annealing) 技术. 模拟退火在每一步都以一定的概率接受比当前解更差的结果, 从而有助于 "跳出"局部极小
3. **使用随机梯度下降**. 与标准梯度下降法精确计算梯度不同, 随机梯度下降法在计算梯度时加入了**随机因素**. 于是, 即便陷入局部极小点, 它计算出的**梯度仍可能不为零**, 这样就有机会跳出局部极小继续搜索.
4. **遗传算法** (genetic algorithms)也常用来训练神经网络以更好地逼近全局最小 .

5.5 其他常见神经网络

略

5.6 深度学习

5.6.1 深度学习的基本概念

理论上来说, 参数越多的模型复杂度越高、 "容量" (capacity)越大, 这意味着它能完成更复杂的学习任务. 但一般情形下, 复杂模型的训练效率低 , 同时易陷入过拟合.

典型的深度学习模型就是很深层的神经网络. 对神经网络模型, 提高容量的一个简单办法是**增加隐层的数目**. 隐层多了, 相应的神经元连接权、阈值等参数就会更多. 模型复杂度也可通过单纯**增加隐层神经元的数目**来实现.

从增加模型复杂度的角度来看, 增加隐层的数目显然比增加隐层神经元的数目更有效. 因为增加隐层数不仅增加了拥有激活函数的神经元数目, 还增加了激活函数嵌套的层数. 但易发散" (diverge)而不能收敛到稳定状态.

5.6.2 无监督逐层训练

无监督逐层训练 (unsupervised layer-wise training)是**多隐层网络训练的有效手段**. 其基本思想是每次训练一层隐结点, 训练时将上一层隐结点的输出作为输入, 而本层隐结点的输出作为下一层隐结点的输入, 这称为"**预训练**" (pre-training); 在顶训练全部完成后, 再对整个网络进行"**微调**" (finetuning)训练.

事实上, **"预训练+微调"**的做法可视为将大量参数分组, 对每组先找到局部看来比较好的设置, 然后再基于这些局部较优的结果联合起来进行全局寻优. 这样就在利用了模型大量参数所提供的自由度的同时, 有效地节省了训练开销.

5.6.3 权共享

另一种节省训练开销的策略是"权共享" (weight sharing), 即**让一组神经元使用相同的连接权**.这个策略在**卷积神经网络** (Convolutional Neural Network，简称 **CNN**) 中发挥了重要作用.

5.6.4 对深度学习的进一步理解

无论是DBN(深度信念网络 deep belief network, 简称 DBN) 还是 CNN , **其多隐层堆叠、每层对上一层的输出进行处理的机制**, **可看作是在对输入信号进行逐层加工**，**从而把初始的、与输出目标之间联系不太密切的输入表示**, **转化成与输出目标联系更密切的表示**, **使得原来仅基于最后一层输出映射难以完成的任务成为可能**. 换句话说, **就是通过多层处理, 逐渐将初始的"低层"特征表示转化为"高层"特征表示后, 用"简单模型" 即可完成复杂的分类等学习任务**. 由此可将**深度学习**理解为进行"**特征学习**"(feature learning)或"**表示学习**"(representation learning)

特征工程: **描述样本的特征**通常需由人类专家来设计, 这称为"特征工程"(feature engineering). **特征的好坏对 泛化性能有至关重要的影响.**