或

$$w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + w_{n+1} = 0 \tag{12.2.31}$$

这是在 n 维模式空间中的一个超平面方程。在几何上,前 n 个系数确定了超平面的方位,而末尾的系数是与从原点到超平面的垂直距离成比例的。因此,如果 $w_{n+1}=0$,超平面通过模式空间的原点。同样,如果 $w_j=0$,超平面平行于 x_j 轴。

图 12.14(a)中门限元素的输出取决于 $d(\mathbf{x})$的符号。代替测试全部函数以确定它是正还是负,可以对项 w_{n+1}检验式(12.2.29)的和,此时系统的输出是:

$$O = \begin{cases} +1 & \sum_{i=1}^{n} w_i x_i > -w_{n+1} \\ -1 & \sum_{i=1}^{n} w_i x_i < -w_{n+1} \end{cases} \tag{12.2.32}$$

在图 12.14(b)中所示的实现方案同图 12.14(a)中所示的等价,仅有的区别是,门限函数表示成了量 $-w_{n+1}$,并且不存在常量 1 的输入。当在这一节的后面讨论多层神经网络的实现时,再回顾这两个表达形式的等价性。

另一个经常使用的公式是,通过增加第$(n+1)$个元素对模式矢量进行扩展,这个元素总等于 1,而与类成员无关。即,扩展的模式矢量 $\mathbf{y}$ 是从一个模式矢量 $\mathbf{x}$ 通过令 $y_i = x_i, i = 1,2,\cdots,n$,并增加元素 $y_{n+1}=1$ 得到的。式(12.2.29) 就变成:

$$\begin{aligned} d(\mathbf{y}) &= \sum_{i=1}^{n+1} w_i y_i \\ &= \mathbf{w}^T \mathbf{y} \end{aligned} \tag{12.2.33}$$

这里 $\mathbf{y} = (y_1, y_2, \cdots, y_n, 1)^T$ 现在是一个扩展模式矢量,并且 $\mathbf{w} = (w_1, w_2, \cdots, w_n, w_{n+1})^T$ 称为权矢量。这个表达式通常在表达符号方面更为简便。然而忽略使用的公式,关键问题是使用一个给定模式矢量的训练集合找到 $\mathbf{w}$,这个模式矢量训练集合来自两类中的一个。

训练算法

下面展开讨论的算法都是过去这些年来为训练感知器提出的大量方法中具有代表性的方法。

线性可分离的类

对于两个线性可分离的训练集为了得到权重矢量可遵循一个简单的迭代算法。对于两个分别属于类 ω_1 和 ω_2 的扩展了模式矢量的训练集合,令 $\mathbf{w}(1)$代表任选的一个类初始权重矢量。然后,在迭代的第 k 步,如果 $\mathbf{y}(k) \in \omega_1$ 并且 $\mathbf{w}^T(k)\mathbf{y}(k) \leqslant 0$,则用下列公式代替 $\mathbf{w}(k)$:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + c\mathbf{y}(k) \tag{12.2.34}$$

这里 c 是一个正的补偿增量。相反,如果 $\mathbf{y}(k) \leqslant \omega_2$ 并且 $\mathbf{w}^T(k)\mathbf{y}(k) \geqslant 0$,则用下列公式代替 $\mathbf{w}(k)$:

$$\mathbf{w}(k+1) = \mathbf{w}(k) - c\mathbf{y}(k) \tag{12.2.35}$$

否则,令 $\mathbf{w}(k)$不变:

$$\mathbf{w}(k+1)=\mathbf{w}(k) \tag{12.2.36}$$

这个算法只有模式在训练过程的第 k 步被错误地分类时才改变 $\mathbf{w}$。假设校正增量 c 是正的，现在来说是一个常量。有时候这个算法作为固定增量校正规则。

当两个类的整个训练集通过机器循环而没有出现任何错误时，算法就会收敛。如果模式的两个训练集合是线性可分离的，那么固定增量校正规则在有限步内收敛。这个结果的证明称做感知器训练定理，可以在 Duda, Hart 和 Stork [2001], Tou 和 Gonzalez [1974] 和 Nilsson [1965] 等人所著的书中找到。

例 12.5　*感知器算法解释*

考虑图 12.15(a)所示的两个训练集，每一个集由两个模式组成。因为两个训练集是线性可分离的，所以可以成功应用训练算法。算法应用之前，模式先被扩展。对类 ω_1 生成训练集合 $\{(0,0,1)^T,(0,1,1)^T\}$，对 ω_2 生成训练集合 $\{(1,0,1)^T,(1,1,1)^T\}$。令 $c=1$，$\mathbf{w}(1)=0$，并按下列顺序生成模式：

$$\mathbf{w}^T(1)\mathbf{y}(1)=[0,0,0]\begin{bmatrix}0\\0\\1\end{bmatrix}=0 \qquad \mathbf{w}(2)=\mathbf{w}(1)+\mathbf{y}(1)=\begin{bmatrix}0\\0\\1\end{bmatrix}$$

$$\mathbf{w}^T(2)\mathbf{y}(2)=[0,0,1]\begin{bmatrix}0\\1\\1\end{bmatrix}=1 \qquad \mathbf{w}(3)=\mathbf{w}(2)=\begin{bmatrix}0\\0\\1\end{bmatrix}$$

$$\mathbf{w}^T(3)\mathbf{y}(3)=[0,0,1]\begin{bmatrix}1\\0\\1\end{bmatrix}=1 \qquad \mathbf{w}(4)=\mathbf{w}(3)-\mathbf{y}(3)=\begin{bmatrix}-1\\0\\0\end{bmatrix}$$

$$\mathbf{w}^T(4)\mathbf{y}(4)=[-1,0,0]\begin{bmatrix}1\\1\\1\end{bmatrix}=-1 \qquad \mathbf{w}(5)=\mathbf{w}(4)=\begin{bmatrix}-1\\0\\0\end{bmatrix}$$

这里，如式(12.2.34)和式(12.2.35)所示，因为分类错误，权重矢量中的校正在第 1 步和第 3 步中做出。因为只有当通过所有训练模式，算法产生一个完全无错误的迭代时，才可能得到一个解，所以训练集必须被多次提供。机器的学习过程是不断以同样的方式令 $\mathbf{y}(5)=\mathbf{y}(1)$，$\mathbf{y}(6)=\mathbf{y}(2)$，$\mathbf{y}(7)=\mathbf{y}(3)$，$\mathbf{y}(8)=\mathbf{y}(4)$。$k=14$ 时收敛，生成权值矢量解 $\mathbf{w}(14)=(-2,0,1)^T$。相应的判别函数是 $d(\mathbf{y})=-2y_1+1$。通过令 $x_i=y_i$ 返回初始的模式空间，生成 $d(\mathbf{x})=-2x_1+1$。当集合等于 0 时，判别函数变成图 12.15(b)中所示的判别边界的方程式。

不可分离的类

实际上，线性可分离模式类是个例外，不常见到。因此，在 20 世纪 60 年代和 20 世纪 70 年代，大量有意义的研究工作转向处理不可分离的模式类的技术开发领域。伴随着最近在神经网络训练方面取得的进展，许多处理不可分离模式的方法成为只具有历史意义的项目。然而，早期的方法之一与这一讨论有直接的关系，这就是原始的德尔塔规则。用于训练感知器的众所周知的 Widrow-Hoff[或叫做最小均方(LMS)]德尔塔规则，在训练的每一步都可以在真实的和期望

的响应间将错误降至最低。

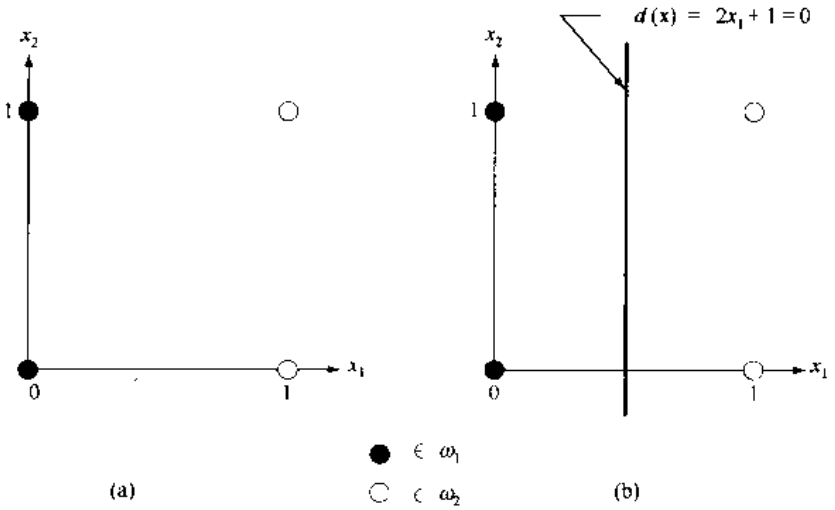

图 12.15 (a)属于两个类的模式,(b)通过训练确定的判别边界

考虑准则函数:

$$J(\mathbf{w}) = \frac{1}{2}(r - \mathbf{w}^T\mathbf{y})^2 \tag{12.2.37}$$

这里 r 是期望的响应(即,当扩展训练模式矢量 y 属于类 ω_1 时,$r = +1$,当 y 属于类 ω_2 时,$r = -1$)。目的是当 $r = \mathbf{w}^T\mathbf{y}$ 时,在 $J(\mathbf{w})$ 梯度的相反方向逐渐调整 $\mathbf{w}$,以便找到函数的最小值,即,对应正确分类的最小值。如果 $\mathbf{w}(k)$ 表示迭代的第 k 步的权重矢量,则梯度下降算法可写成下式:

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \alpha\left[\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}}\right]_{\mathbf{w}=\mathbf{w}(k)} \tag{12.2.38}$$

这里 $\mathbf{w}(k+1)$ 是 $\mathbf{w}$ 的新值,并且 $\alpha > 0$ 给出校正量。由式(12.2.37)得到:

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = -(r - \mathbf{w}^T\mathbf{y})\mathbf{y} \tag{12.2.39}$$

把这一结果代入式(12.2.38)得到:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \alpha[r(k) - \mathbf{W}^T(k)\mathbf{y}(k)]\mathbf{y}(k) \tag{12.2.40}$$

所用的初始权重矢量 $\mathbf{w}(1)$ 是任意的。

在权重矢量中,定义增量为:

$$\Delta\mathbf{w} = \mathbf{w}(k+1) - \mathbf{w}(k) \tag{12.2.41}$$

以德尔塔校正算法的形式写出式(12.2.40):

$$\Delta\mathbf{w} = \alpha e(k)\mathbf{y}(k) \tag{12.2.42}$$

这里

$$e(k) = r(k) - \mathbf{w}^T(k)\mathbf{y}(k) \tag{12.2.43}$$

当模式 $\mathbf{y}(k)$ 存在时,$e(k)$ 是涉及权重矢量 $\mathbf{w}(k)$ 的误差。

式(12.2.43)给出了权重矢量 $\mathbf{w}(k)$ 的误差。如果把它变成 $\mathbf{w}(k+1)$,但保留模式不变,

误差就变成：

$$e(k) = r(k) - \mathbf{w}^T(k+1)\mathbf{y}(k) \tag{12.2.44}$$

误差的变化量是：

$$\begin{aligned}\Delta e(k) &= [r(k) - \mathbf{w}^T(k+1)\mathbf{y}(k)] - [r(k) - \mathbf{w}^T(k)\mathbf{y}(k)] \\ &= -[\mathbf{w}^T(k+1) - \mathbf{w}^T(k)]\mathbf{y}(k) \\ &= -\Delta\mathbf{w}^T\mathbf{y}(k)\end{aligned} \tag{12.2.45}$$

但，$\Delta\mathbf{w} = \alpha e(k)\mathbf{y}(k)$，所以：

$$\Delta e = -\alpha e(k)\mathbf{y}^t(k)\mathbf{y}(k) = -\alpha e(k)\|\mathbf{y}(k)\|^2 \tag{12.2.46}$$

因此，权重的变化由于有因子 $\alpha\|\mathbf{y}(k)\|^2$ 而减少了误差。下一个输入模式开始新的自适应循环过程，使用因子 $\alpha\|\mathbf{y}(k+1)\|^2$ 减少下一个误差，如此继续下去。

α 的选择控制着稳定性和收敛速度(Widrow 和 Stearns [1985])。稳定性要求 $0 < \alpha < 2$。α 的实际范围在 $0.1 < \alpha < 1.0$ 之间。尽管这里没有给出证明，但实际上式(12.2.40)或式(12.2.42)和式(12.2.43)收敛于模式训练集合上一个最小均方误差解。当模式类可分离的时候，使用刚才讨论的算法得出的解也许会产生一个分离超平面。就是说，从感知器训练理论的角度看，一个均方误差解并不意味着会真有一个解。这种不确定性是在这个特殊的公式中在可分离和不可分离的情况下使用收敛算法所付出的代价。

到目前为止讨论过的这两个感知器训练算法可以推广到两类以上和非线性判别函数领域。基于先前做出的历史评价，寻找多类训练算法没有太大价值。相反，将在神经网络的内容中讨论多类训练问题。

多层前馈神经网络

在这一节中，主要关注多类模式识别问题的判别函数，包括由感知器计算节点层构成的体系结构，而不考虑类是否是可分离的。

基本的体系结构

图 12.16 显示了考虑的神经网络模型的体系结构。体系由排列结构相同的计算节点(神经元)层构成。这样，一层上每个神经元的输出都输入到下一层的每个神经元。称为 A 的第一层中的神经元个数为 N_A。一般 $N_A = n$，是输入的模式矢量维度。称为 Q 层的输出层的神经元数目由 N_Q 代表。N_Q 等于 W，是已经训练好用来识别的神经网络模式类数目。正如下面进行讨论的那样，如果网络的第 i 个输出为“高”，而其他输出为“低”，则网络识别模式矢量 $\mathbf{x}$ 属于类 ω_i。

如图 12.16 中的放大显示图所示，每个神经元都有与先前讨论的感知模型相同的形式(见图 12.14)，除了硬限激活函数被软限“S”函数代替。沿着神经网络所有路径的可微分性也是推导训练规则所需要的。下列的“S”激活函数具有需要的微分性：

$$h_j(I_j) = \frac{1}{1 + e^{-(I_j + \theta_j)/\theta_0}} \tag{12.2.47}$$

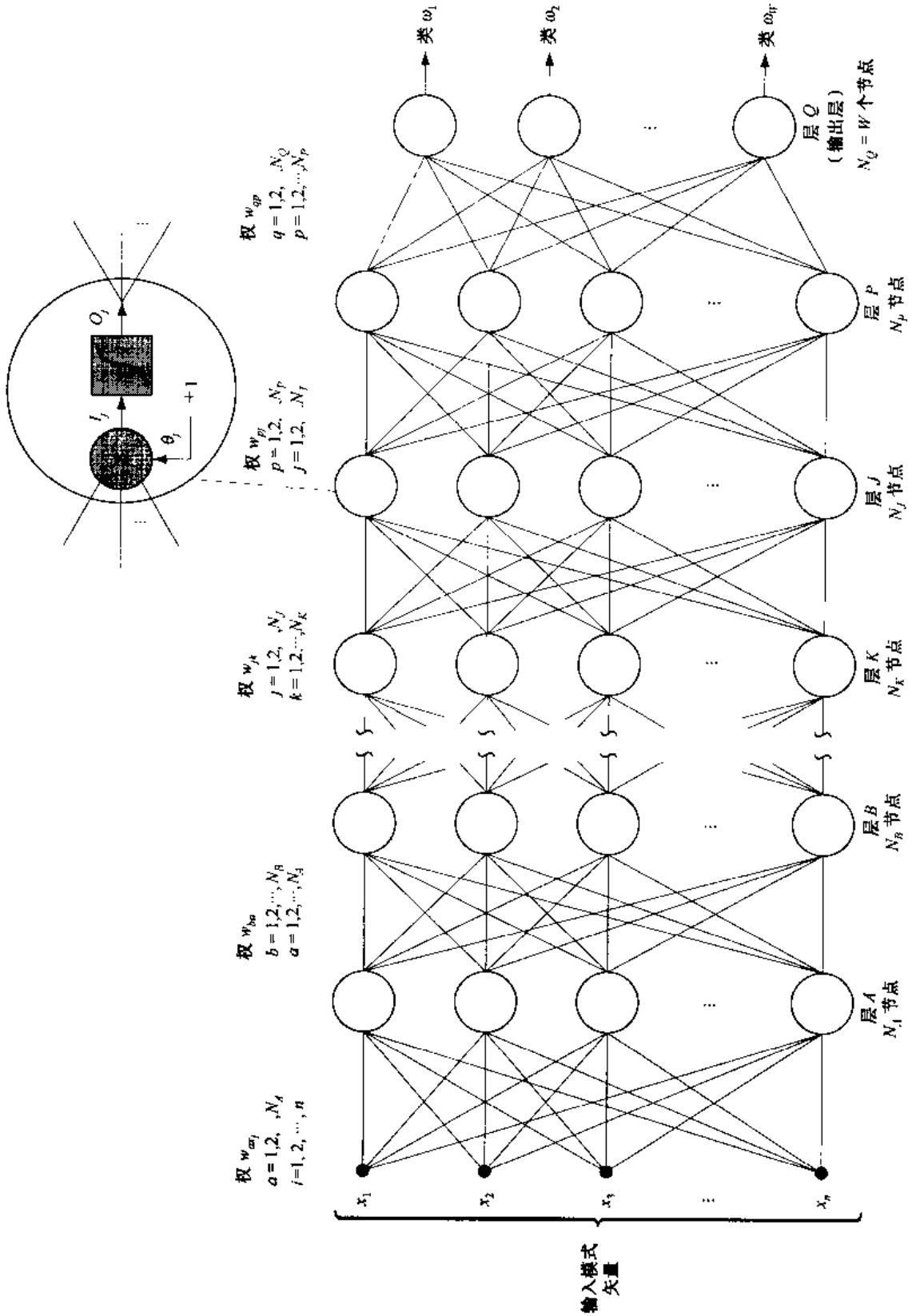

图 12.16 多层反馈神经网络模型。放大部分显示了整个网络中每个神经元的基本结构,偏移量θ_j当做不同的权值处理

这里 $I_j, j=1,2,\cdots,N_j$，是输入到网络第 J 层每一个节点的激活元素，θ_j 是偏移量，θ_0 控制“S”函数的形状。

图 12.17 中沿着每个节点响应的“高”和“低”输出描绘了式(12.2.47)。因此，当使用这个特殊函数时，系统对所有大于 θ_j 的 I_j 值输出一个“高”读数。同样，系统对所有小于 θ_j 的 I_j 值输出一个“低”读数。如图 12.17 所示，“S”激活函数总是正的，而且只有在激活元素分别为负无穷或正无穷时，才趋近于它的极限值 0 和 1。由于这个原因，在图 12.16 中，用 0 和 1 的近似值(比如 0.05 和 0.95)定义神经元输出的“低”和“高”值。原则上，不同类型的激活函数用于不同的层次，甚至用于神经网络同一层中的不同节点。实际上，通常的方法是对整个网络使用同样形式的激活函数。

参考图 12.14(a)，图 12.17 中显示的偏移量与以前讨论感知器时谈到的权值系数 w_n+1 类似。要实现这种门限函数的替换可以如图 12.14(a)中所示的形式，吸收偏移量 θ_j 作为修正输入网络节点的单位常量的附加系数。为了遵循在文献中普遍使用的符号，不把 +1 这个单独的常量输入显示在图 12.16 的所有节点中。代之而来的，这个输入及它的修正权值 θ_j 是网络节点的整数对。正如在图 12.16 中放大显示的，J 层中的 N_j 个节点的每一个都有一个这样的系数。

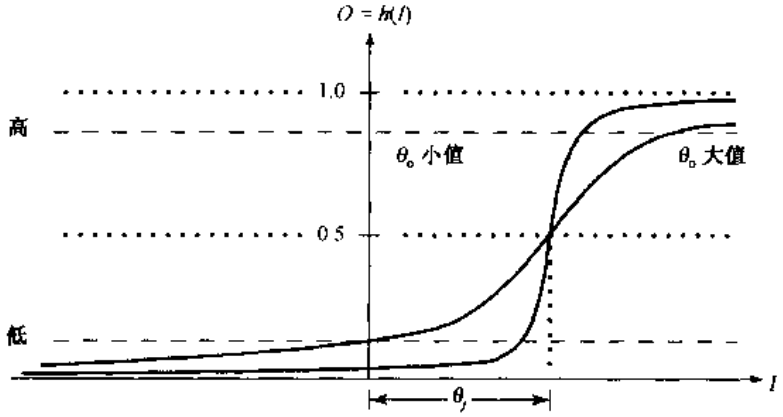

图 12.17　式(12.2.47)中的激活函数“S”

在图 12.16 中，任何层中的节点输入都是前一层输出的加权和。令层 K 代表层 J 的前一层(在图 12.16 中，层的划分并不按字母顺序)。由层 K 生成层 J 的每一个节点的激活元素，用 I_j 代表：

$$I_j = \sum_{k=1}^{N_k} w_{jk} O_k \tag{12.2.48}$$

$j=1,2,\cdots,N_j$，这里 N_j 是层 J 中节点的数目，N_K 是层 K 中节点的数目，w_{jk} 是层 K 中节点的输出 O_k 馈给层 J 中节点之前，在层 K 中修正节点输出 O_k 的权值。层 K 的输出为：

$$O_k = h_k(I_k) \tag{12.2.49}$$

$k = 1,2,\cdots,N_k$。

对式(12.2.48)所用的下标符号有一个清晰的理解是很重要的。因为在这一节的余下部

分都将用到它。首先,注意 $I_j, j = 1,2,\cdots,N_J$,表示层 J 第 j 个节点的激活元素输入。因此,I_1 表示层 J 第 1 个(顶端)节点的激活元素输入,I_2 表示层 J 第 2 个节点的激活元素输入,以此类推。对层 J 的每一个节点都有 N_K 个输入,但每个单个输入的加权都不同。因此,层 J 中第 1 个节点的输入 N_k 由系数 $w_{1k}(k = 1,2,\cdots,N_K)$进行加权;层 J 第 2 个节点的输入用系数 $w_{2k}(k = 1,2,\cdots,N_K)$进行加权;以此类推。所以,当层 K 的输出输入层 J 时,$N_J \times N_K$ 个系数作为层 K 输出的权值是必要的。一个附加的 N_J 偏移量系数 θ_j 需要用来完全确定层 J 中的节点。

将式(12.2.48)代入式(12.2.47)得到:

$$h_j(I_j) = \frac{1}{1 + e^{-\left(\sum_{k=1}^{N_K} w_{jk} O_k + \theta_j\right)/\theta_0}} \tag{12.2.50}$$

这是在本节余下部分中要用到的激活函数形式。

训练过程中,在输出层中修改神经元是很简单的,因为希望得到的每个节点的输出是已知的。训练一个多层网络过程中的主要问题在于,要调整称为隐藏层的权值。即,除了输出层以外的那些层中的权值。

反向传播的训练

我们先来关注输出层。输出层 Q 各节点的期望响应 r_q 和真实响应 O_q 之间总误差的平方为:

$$E_Q = \frac{1}{2}\sum_{q=1}^{N_Q}(r_q - O_q)^2 \tag{12.2.51}$$

这里 N_Q 是输出层 Q 的节点数目,系数 $\frac{1}{2}$ 使后面求导时在表示上比较便利。

目的是推导类似于德尔塔规则的训练规则。使用这种训练规则可以调整每一层的权值,以便找到式(12.2.51)所示形式的误差函数最小值。像前面一样,与误差相对于权值的偏导数成比例地调整权值可以达到这个目的。换句话说,

$$\Delta w_{qp} = -\alpha \frac{\partial E_Q}{\partial w_{qp}} \tag{12.2.52}$$

这里层 P 先于层 Q,Δw_{qp} 如式(12.2.42)所定义的,α 是正的校正增量。

误差 E_Q 是输出 O_q 的函数,O_q 则是输入 I_q 的函数。使用连锁规则,求出 E_Q 的偏导数,如下所示:

$$\frac{\partial E_Q}{\partial w_{qp}} = \frac{\partial E_Q}{\partial I_q}\frac{\partial I_q}{\partial w_{qp}} \tag{12.2.53}$$

由式 (12.2.48),得到:

$$\frac{\partial I_q}{\partial w_{qp}} = \frac{\partial}{\partial w_{qp}}\sum_{p=1}^{N_P} w_{qp} O_p = O_p \tag{12.2.54}$$

将式(12.2.53)和式(12.2.54)代入式(12.2.52)得到:

$$\Delta w_{qp} = -\alpha \frac{\partial E_Q}{\partial I_q} O_p$$
$$= \alpha \delta_q O_p \tag{12.2.55}$$

这里

$$\delta_q = -\frac{\partial E_Q}{\partial I_q} \tag{12.2.56}$$

为了计算偏导数$\partial E_Q / \partial I_q$，用连锁规则使用 E_Q 相对于 O_q 的变化率和 O_q 相对于 I_q 的变化率表示偏导数：

$$\delta_q = -\frac{\partial E_Q}{\partial I_q} = -\frac{\partial E_Q}{\partial O_q}\frac{\partial O_q}{\partial I_q} \tag{12.2.57}$$

由式 (12.2.51)得到：

$$\frac{\partial E_Q}{\partial O_q} = -(r_q - O_q) \tag{12.2.58}$$

从式 (12.2.49)得到：

$$\frac{\partial O_Q}{\partial I_q} = \frac{\partial}{\partial I_q} h_q(I_q) = h_q'(I_q) \tag{12.2.59}$$

将式(12.2.58)和式(12.2.59)代入式(12.2.57)给出：

$$\delta_q = (r_q - O_q) h_q'(I_q) \tag{12.2.60}$$

它与误差量值($r_q - O_q$)成比例。将式(12.2.56)到式(12.2.58)代入式(12.2.55)最终得到：

$$\Delta w_{qp} = \alpha (r_q - O_q) h_q'(I_q) O_p = \alpha \delta_q O_p \tag{12.2.61}$$

确定了函数 $h_q(I_q)$后，式(12.2.61)中的所有项都是已知的或者可以在网络中见到。换句话说，对于网络输入的任何训练模式的表达，我们知道每一个输出节点 r_q 的期望响应应该是什么。每个输出节点的值 O_q 可以被看做 I_q(输入到层 Q 的激活元素)和 O_p(层 P 的节点输出)。因此，知道如何调整修改网络最末一层和次末层间链接的权值。

继续回到输出层的处理上来，现在分析层 P 发生的情况。以上述同样的方式进行处理，产生：

$$\Delta w_{pj} = \alpha (r_p - O_p) h_p'(I_p) O_j = \alpha \delta_p O_j \tag{12.2.62}$$

误差项是：

$$\delta_p = (r_p - O_p) h_p'(I_p) \tag{12.2.63}$$

除了 r_p，式(12.2.62)和式(12.2.63)中的所有项都已知或可在网络中见到。项 r_p 在内层是无意义的，因为不知道模式成员项中的内部节点响应应该是什么。可以仅在网络最终产生模式分类输出的地方指定所希望的响应项 r。如果知道内部节点的信息，就不需要下一层的信息。因此，必须找到一种方法在量上重定义 δ_p，使其成为已知的或在网络中可见的。

回到式(12.2.57)，对层 P 写出误差项为：

$$\delta_p = -\frac{\partial E_p}{\partial I_p} = -\frac{\partial E_p}{\partial O_p}\frac{\partial O_p}{\partial I_p} \tag{12.2.64}$$

项$\partial O_p/\partial I_p$不难表示。以前,它表示成:

$$\frac{\partial O_p}{\partial I_p} = \frac{\partial h_p(I_p)}{\partial I_p} = h_p'(I_p) \tag{12.2.65}$$

我们知道,h_p是已确定的,因为I_p是可观测的。产生r_p的项是偏导数$\partial E_p/\partial O_p$,所以这一项必须以不包含$r_p$的方法表示。使用连锁规则,将偏导数写成:

$$\begin{aligned} -\frac{\partial E_p}{\partial O_p} &= -\sum_{q=1}^{N_Q}\frac{\partial E_p}{\partial I_q}\frac{\partial I_q}{\partial O_q} = \sum_{q=1}^{N_Q}\left(-\frac{\partial E_p}{\partial I_q}\right)\frac{\partial}{\partial O_p}\sum_{p=1}^{N_P} w_{qp}O_p \\ &= \sum_{q=1}^{N_Q}\left(-\frac{\partial E_p}{\partial I_q}\right)w_{qp} \\ &= \sum_{q=1}^{N_Q}\delta_q w_{qp} \end{aligned} \tag{12.2.66}$$

最后一步来自式(12.2.56)。将式(12.2.65)和式(12.2.66)代入式(12.2.64)得到对δ_p所希望的表达式:

$$\delta_p = h_p'(I_p)\sum_{q=1}^{N_Q}\delta_q w_{qp} \tag{12.2.67}$$

因为所有项已知,现在参量δ_p是可计算的。因此,式(12.2.62)和式(12.2.67)完全完成了层P的训练法则。式(12.2.67)的重要性在于从δ_q和w_{qp}计算出δ_p,这两个量是在紧接着层P的层中计算出来的项。对于层P,误差项和权值计算出来后,这些量也可以类似地用于直接计算层P之前层的误差项和权值。换句话说,已经找到一种方法,从输出层误差开始把误差反向传播到网络。

我们可以总结和归纳训练过程如下。对任何层K和J,层K直接接在层J的前边,计算权值w_{jk},w_{jk}通过使用下列公式,改进了两层的连接:

$$\Delta w_{jk} = \alpha\delta_j O_k \tag{12.2.68}$$

如果层J是输出层,δ_j为:

$$\delta_j = (r_j - O_j)h_j'(I_j) \tag{12.2.69}$$

如果层J是内部层,并且层P是它的下一层(右边),则δ_j由下式给出:

$$\delta_j = h_j'(I_j)\sum_{p=1}^{N_P}\delta_p w_{jp} \tag{12.2.70}$$

$j=1,2,\cdots,N_j$。使用式(12.2.50)中的激活函数,并令$\theta_0=1$得到:

$$h_j'(I_j) = O_j(1-O_j) \tag{12.2.71}$$

在这种情况下,式(12.2.69)和式(12.2.70)遵循如下假定时,有特别吸引人的形式:

$$\delta_j = (r_j - O_j)O_j(1-O_j) \tag{12.2.72}$$

对应于输出层，有：

$$\delta_j = O_j(1 - O_j)\sum_{p=1}^{N_p}\delta_p w_{jp} \tag{12.2.73}$$

对应于内部层。在式(12.2.72)和式(12.2.73)中，$j = 1,2,\cdots,N_j$。

式(12.2.68)到式(12.2.70)构成图 12.16 的训练多层反馈神经网络的推广德尔塔法则。这个过程由一个通过整个网络的任意(但不全相等)的权重集合开始。然后，推广的德尔塔法则用于任何涉及两个基本阶段的迭代步骤。第一个阶段，对于网络存在一个训练矢量，并且容许层传播对每个节点计算其输出。输出层节点的输出 O_q 与对它们期望的响应值 r_p 进行比较，生成误差项 δ_q。第二阶段包括一条遍历网络的反向路径，在此阶段，合适的误差信号通过每个节点并且相应的权重做出改变。这个过程同样应用于权值偏离量 θ_j。在以前讨论的细节中，这些权值偏离量仅被看做附加权值，作为输入到网络中每个节点总连接的单位量修正值。

通常的做法是对网络误差和与单个模式相联系的误差进行跟踪。在成功的训练中，网络误差随迭代步骤次数的增加而减少并且这一过程收敛于一个稳定的权值集合，它对附加训练仅呈现很小的影响。接下来要完成的方法需要确定一个模式是否已被正确分类，训练时确定与获得模式的模式类有关的输出层节点响应是否是高的，如以前定义的那样，所有其他节点的输出是低的。

系统进行训练之后，使用在训练阶段中设定的参量对模式进行分类。在标准操作中，所有反馈路径是不连通的。任何输入模式允许通过不同层进行传播，并且模式被划归输出为高的节点所属的类。此时，其他所有节点输出为低。如果被标记为高的节点不止一个，或没有节点输出被标记为高，则可选的做法是，声明进行了错误的分类或简单地将模式划归输出节点的类并赋予最大值。

例 12.6　使用神经网络进行图形分类

现在说明如何训练一个如图 12.16 所示形式的神经网络去识别图 12.18(a)所示的 4 种形状及这些形状带有噪声的类型，样本如图 12.18(b)所示。

通过计算形状的归一化信号(见 11.1.3 节)得到每个信号的 48 个隔开的均匀分布样本来产生模式矢量。得到的 48 维矢量输入到图 12.19 所示的三层前馈神经网络。第一层神经元节点的数目定为 48，与输入模式矢量的维数一致。第三层(输出层)的四个节点与模式类的数目一致，并且中间层的神经元数目确定为 26(输入和输出层神经元的平均数)。由于不知道确定神经网络内部层次中节点数目的规则，因此这个数目一般是基于以前的经验或任意指定并通过检验来完善。在输出层，从上到下的四个节点此时各代表类 $\omega_j, j = 1,2,3,4$。在设定网络结构后，必须为每个单元和层选定激活函数。所有选定的激活函数要满足当 $\theta_o = 1$ 时的式(12.2.50)，根据以前的讨论，要使用式(12.2.72)和式(12.2.73)。

训练过程分为两个部分。在第一部分中，权值被初始化为带有零均值的小随机数，然后，使用与类似于图 12.18(a)所示的相应无噪声样本的模式矢量对网络进行训练。外形图如图 12.18(a)中所示。输出节点在训练期间是受到监控的。对类 w_i 的所有训练模式，当输