第3章 神经网络

3.1 引言

本书其他地方将讨论机器学习(*learn*)的方法。本章将讨论S-R机器执行动作的选择计算是如何通过与每个输入的适当的动作配对的一组输入实例来学习的。虽然存在许多不同的可用的计算结构，但在这儿将集中讨论具有可调节权值的TLU网络。网络系统通过不断调节权值，直至其动作计算表现令人满意来完成学习。如上一章所述，TLU网络称为神经网络(*neural network*)是因为它模仿了生物神经元的一些特性。这里不讨论神经网络与大脑或部分大脑功能的关系，而是把这些网络严格地视为有趣且有用的工程设备。

考虑以下学习问题：给定一个由n维向量**X**组成的集合Ξ，分量为x_i，i=1,...,n。这些向量将是由一个响应agent的感知处理单元计算出的特征向量。这些分量的值可以是数值，也可以是布尔值（如前所述）。也已知集合Ξ中每个**X**所对应的恰当的动作a。这些动作也许是学习者所观察到的一个教师对一组输入的响应。这些相关的动作有时称为向量的“标号（*label*）”或“类别(*class*)”。集合Ξ与相应的标号组成了“训练集合(*training set*)”。机器学习的问题就是寻找一个函数，如f(**X**),“令人满意地”与训练集合的成员相对应。通常，我们希望，由f计算出的动作尽可能与Ξ中向量的标号一致。因为同时给出了标号和输入向量，所以我们认为这一学习过程“受到监控”。

即使能找到一个可对训练集合做出正确响应的函数，我们又有何理由相信它能对训练过程中未遇到过的输入做出正确的响应呢？一些实验证据证明它可以，另外，还有一种理论证明：（在一定条件下）若此训练集合对于其他可能会遇到的输入种类来说具备“代表性”，那么，这些输入“可能”会“引发”“几乎正确”的输出。有关此论题的其他论述，请参阅有关“可能接近正确（*probably approximately correct*, PAC）”的机器学习理论[Kearns & Vazirani 1994，Haussler 1998，Haussler 1990]。实际上，许多方法可用来评测一个已学习过的函数对相似（但尚未觉察）输入的反应的准确程度。我会在本章的后面部分介绍其中一部分。

3.2 训练单个TLU

3.2.1 TLU几何学

首先介绍如何调节或“训练”单个TLU的权值，从而使其对某一训练集合产生正确的输出。若用一个TLU来计算一个动作，则其输入应为数值（这样才能计算加权总和）；若感知处理过程产生类别特征，那么应用某种方法将其转为数字。当然，用单个TLU来计算动作的响应机器只能根据TLU的两种可能输出做出两种动作。单TLU也称为*Perceptron*和*Adaline*（即可调节线性元素）。Rosenblatt和Widrow[Rosenblatt 1962, Widrow 1962]对它们的使用进行了广泛的探讨。对一个TLU的训练是通过调节其可变权值来实现的。用几何学解释TLU如何对其输入作出反应，能帮助我们直观地理解TLU的训练方法。

一个TLU由其权值和阈值来定义。权值（$w_1,\cdots,w_i,\cdots,w_n$）可由一个权向量W来表示。我用θ来表示TLU的阈值。这里，我们假设输入向量X，具有数字分量（这样才能计算这些数字分量的加权总和）。若向量点积 $s = \mathbf{X}\cdot\mathbf{W}$，比θ大，则TLU输出为1；否则为0。如图3-1所示，TLU用一个线性边界把输入向量的空间分开。在二维空间里，此边界为一条线，而在三维空间里为一个平面。在多维空间里此线性边界称为“超平面”(*hyperplane*)。此超平面把$\mathbf{X}\cdot\mathbf{W}-\theta > 0$的向量与$\mathbf{X}\cdot\mathbf{W}-\theta < 0$的向量分开。超平面方程为$\mathbf{X}\cdot\mathbf{W}-\theta = 0$。我们可以通过调节阈值来改变超平面（相对原点）的位置，而通过调节权值可以改变其方向。

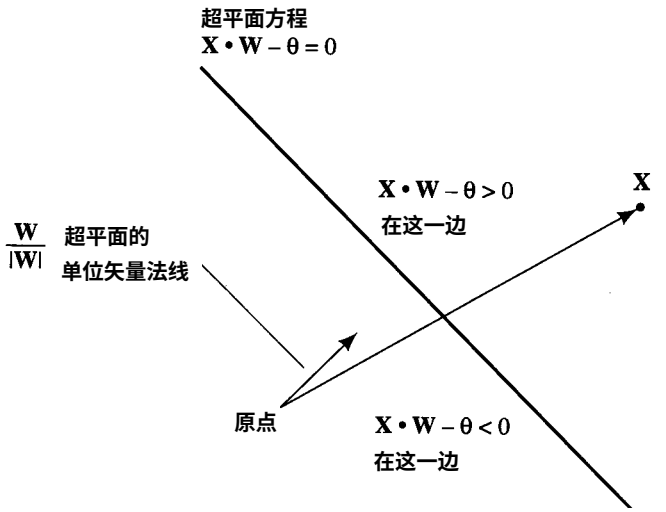


图3-1 TLU几何学

3.2.2 扩充向量

实际上，调节一个TLU的权值有好几个方法。如果我采用TLU的阈值总是等于0这一常规约定，就会简化对这些方法的解释。与此不同，我们运用 n+1维“扩充”向量来实现任意阈值。扩充输入向量的第 n+1个分量的值总为1；扩充权向量的第 n+1个分量，w_{n+1}，设定为所希望的阈值θ的负数。今后，当我们运用 $\mathbf{X}\cdot\mathbf{W}$这一符号表示时，采用的就是 n+1维扩充向量。那么，当$\mathbf{X}\cdot\mathbf{W} \geq 0$时，TLU的输出为1；否则为0。

3.2.3 梯度下降方法

研究训练一个TLU使其能对训练向量作出恰当响应的方法之一，就是定义一个误差函数，使其能通过调节权值达到最小值。通常使用的误差函数是平方差：

$$\varepsilon = \sum_{\mathbf{X}_i \in \Xi} (d_i - f_i)^2$$

式中，f_i是TLU对输入X_i的实际响应，d_i是所希望的响应。然后我们对训练集合中的所有向量求和。对于固定的Ξ来说，ε（通过f_i）依权值而定。我们可通过下降梯度求到ε的最小值。为了计算下降梯度，先计算“权空间”中ε的梯度；然后把权向量沿此梯度的反方向（下山）移动。计算ε的梯度的困难之一，就是ε依Ξ中所有输入向量而定。通常，我们倾向于先用Ξ中的一个成员，对其权值进行调节后，再用Ξ中的另一个成员——一个运用由已作标号的输入向量所组

成的序列Σ的递增训练过程。当然，递增训练的效果只能接近所谓的批处理方式的效果，然而这一近似值通常十分有效。这里，对递增方式作一个描述。

一个单输入向量X（当所希望的输出为d时引发输出f）的平方差为：

$$\varepsilon = (d-f)^2$$

ε对权值的梯度为：

$$\frac{\partial \varepsilon}{\partial \mathbf{W}} = \left[\frac{\partial \varepsilon}{\partial w_1}, \cdots, \frac{\partial \varepsilon}{\partial w_i}, \cdots, \frac{\partial \varepsilon}{\partial w_{n+1}}\right]$$

（标量φ对向量W的梯度有时表示为 $w\phi$）

由于ε对W的依赖完全通过点积 $s = \mathbf{X} \cdot \mathbf{W}$产生，故可用链式规则（*chain rule*）书写如下：

$$\frac{\partial \varepsilon}{\partial \mathbf{W}} = \frac{\partial \varepsilon}{\partial s}\frac{\partial s}{\partial \mathbf{W}}$$

然后，因为 $\frac{\partial \varepsilon}{\partial \mathbf{W}} = \mathbf{X}$，故

$$\frac{\partial \varepsilon}{\partial \mathbf{W}} = \frac{\partial \varepsilon}{\partial s}\mathbf{X}$$

注意到：$\frac{\partial \varepsilon}{\partial s} = -2(d-f)\frac{\partial f}{\partial s}$。这样，

$$\frac{\partial \varepsilon}{\partial \mathbf{W}} = -2(d-f)\frac{\partial f}{\partial s}\mathbf{X}$$

在求f对s的偏导数时我们遇到一个问题。由于阈值函数的存在，TLU的输出f对s而言不是连续可导的。点积中许多微小的变化根本无法改变f，而且f一发生变化就从1变到0或从0变到1。我将介绍解决此问题的两种程序：一种是忽略阈值函数，且令 $f = s$；另一种用一个可求导的非线性函数替代此阈值函数。

3.2.4 Widrow-Hoff程序

假设我们试图通过调节权值从而使每个标号为1的训练向量产生的点积精确地等于1，使每个标号为0的向量产生的点积正好等于- 1。这里，$f = s$，则增量平方差 $\varepsilon=(d - f)^2=(d-s)^2$，且$\frac{\partial f}{\partial s} = 1$。这样，梯度为：

$$\frac{\partial \varepsilon}{\partial \mathbf{W}} = -2(d-f)\mathbf{X}$$

把权向量沿梯度的反方向移动，并把因数2融入学习率（*learning rate*）参数c，则权向量的新值为：

$$\mathbf{W} \longleftarrow \mathbf{W} + c(d-f)\mathbf{X}$$

当$(d-f)$为正时，把一部分输入向量加到权向量中去，这令点积变大而（$d-f$）变小；当（$d-f$）为负时，从权向量中减去一部分输入向量——则产生相反效果。这一程序就是著名的*Widrow-Hoff*或*Delta*规则[Widrow & Hoff 1960]㊀。当然，在找出一组可求出平方差的最小值的

㊀ 熟悉数字方法的人均会发现Widrow-Hoff程序是解决线性方程的relaxation method之一。

权值后（设$f = s$），就可重新考虑阈值函数来求出f的值（0或1）。

3.2.5 一般化Delta程序

Werbos[Werbos 1974]提出了另一种处理不可求导的阈值函数的程序，其他几位研究者对此也分别作了探讨，如 [Rumelhart,et al. 1986]。这一方法涉及用 S型可求导函数、即所谓的“ sigmoid ”⊖来替换阈值函数。常用的阈值函数为$f(s) = \frac{1}{1+e^{-s}}$，这里$s$为输入，$f$为输出。图3-2显示了一个sigmoid 函数及相应的阈值函数。

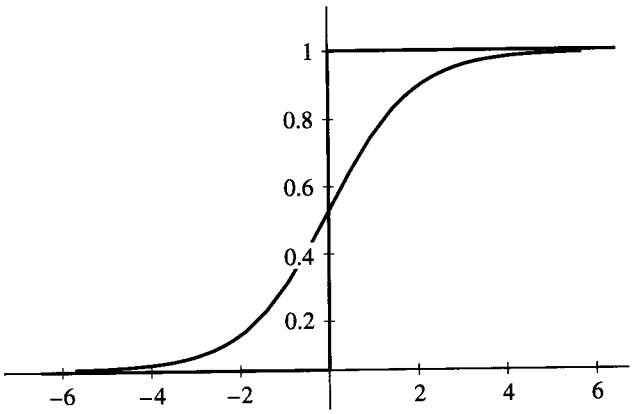


图3-2 一个sigmoid函数

选择这一sigmoid函数可得出如下偏导数：

$$\frac{\partial f}{\partial s} = f(1-f)$$

把此表达式代入 $\frac{\partial \varepsilon}{\partial \mathbf{W}} = -2(d-f)\frac{\partial f}{\partial s}\mathbf{X}$，得

$$\frac{\partial \varepsilon}{\partial \mathbf{W}} = -2(d-f)f(1-f)\mathbf{X}$$

于是得到了下面的权变化规则，即“一般化Delta程序”：

$$\mathbf{W} \leftarrow \mathbf{W} + c(d-f)f(1-f)\mathbf{X}$$

以下是Widrow-Hoff与一般化Delta的不同之处：

1) 前者所希望的输出d为1或-1，而后者为1或0；

2) 前者的实际输出f等于点积s，而后者的f为sigmoid的输出；

3) 由于sigmoid函数的存在，后者的表达式中多出了$f(1-f)$这一项。$f(1-f)$的值随sigmoid函数的变化而在0到1之间变化。当f为0时，$f(1-f)$也为0；当f为1时，$f(1-f)$也为0；当f为1/2时，$f(1-f)$为最大值1/4（即当sigmoid的输入为0时）。sigmoid函数可视为一个“模糊的”超平面。对于远离此模糊超平面的输入向量，其$f(1-f)$的值接近0，而且无论所希望的输出如何，一般化Delta程序对权值的改变甚微，甚至无法改变。在围绕模糊超平面的区域之内（这是惟一一个权变化对f颇有影响的地方）才有权变化，而且这些变化不断地纠正错误。

在一般化Delta程序中找到一组权值后，如有需要还可再用阈值函数替换 sigmoid函数。

⊖ [Russell & Norving, P.595]把此观点的运用归功于[Bryson & Ho 1969]。

3.2.6 纠错程序

在下面这种方法中，我们保留阈值不变（并不替换成sigmoid函数），而且仅当TLU的响应出错时（即当（$d-f$）的值为1或-1时）才调节权向量。这一程序称为“纠错程序”。其改变权值的规则如下：

$$\mathbf{W} \longleftarrow \mathbf{W} + c(d-f)\mathbf{X}$$

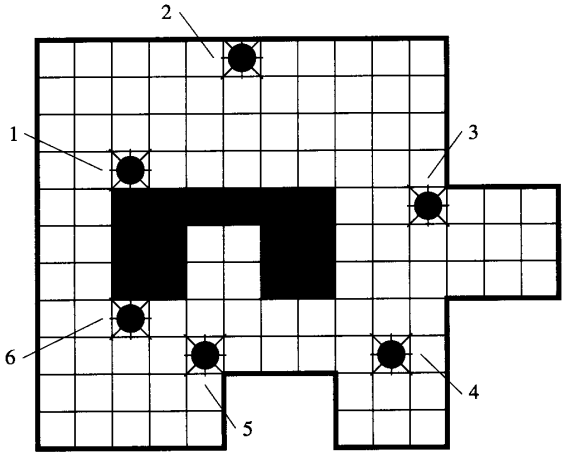
当然，这一改变是趋向于纠正错误的（也许能彻底纠正错误，这依赖于学习率参数的值）。这一规则与Widrow-Hoff的区别是：前者的d和f的值为0或1，而后者的d为1或-1，且$f=s$为点积的值。

可以证明，若存在某一个为Ξ中所有输入向量产生正确输出的权向量W，那么，在经过有限的输入向量的表达式之后，纠错程序最终能得到此权向量，从而不再做权值的调节（尽管这一程序确保在表达完有限个Σ中的输入向量之后便终止，但这一证明要求每个输入向量均在训练序列Σ中出现无限次）。对于非线性可分的输入向量集合来说，纠错程序会永不停止，从而不能用来寻找一个“最佳”权向量来作为判定错误的标准。然而，Widrow-Hoff和一般化Delta程序都可找到最小平方差解，尽管这时最小误差不为0。

在第2章中，我们碰到过线性可分函数。回想一下为沿边界运动的机器人设定的产生式规则。下面是其中的一个规则：

$x_1\overline{x_2} \longrightarrow$ east

将其改写为与传感器输入有关的表达式，得到 $x_1\overline{x_2} = (s_2 + s_3)\overline{s_4}\,\overline{s_5}$。这一函数是线性可分的，并且可以由如图2-6所示的TLU来执行。这样，我们希望，对一大批作好标号的传感器的向量



输入数量	传感器矢量	$x_1\overline{x_2}$ 向东移动
1	00001100	0
2	11100000	1
3	00100000	1
4	00000000	0
5	00001000	0
6	01100000	1

图3-3 学习何时向东移动的一个训练集合

的纠错训练，最终可使TLU正确地区分让机器人东移的输入和不让其东移的输入。如图 3-3 所示，我们可把训练集合排列到一起。你也许想用这些向量以及其他已作标号的输入向量来试试纠错程序（千万别忘了输入$s_9 \equiv 1$和加权值w_9！）可是，经过训练的TLU如何处理那些未训练过的输入呢？训练集合应包括多少输入向量才能使TLU的表现令人满意呢？

至于纠错程序的背景、引用、证明和实例，请参阅［Nilsson 1965］。

3.3 神经网络

3.3.1 动机

实际存在许多单TLU学不会的刺激—响应集合（这些训练集合可能是线性不可分的）。这种情况下，TLU网络却可以给出正确的响应。由TLU网络执行的函数不仅依赖于其拓扑，而且依赖于单TLU的权值。“前向网络(*feedforward networks*)”没有循环：在此网络中，任一TLU的输入并不（通过0个或多个中间TLU）依赖于这一TLU的输出(不具前向性的网络称为“递归网络(*recurrent network*)”。在后面的章节中，我们将学习一例递归网络)。若用层来组织前向网络中的TLU，且层j的组件只接受来自层$j-1$的TLU的输入，那么，我们称这样的网络为“分层前向网络(*layered feedforward network*)”。图3-4中的网络执行函数$f = x_1x_2 + \bar{x}_1\bar{x}_2$（称为“偶校验（*even-parity*）”函数），它是一个有两个层的（具有权值）分层前向网络（有些人在计算TLU层个数时，把输入也算作一层，因此，他们会称此图中的网络为三层网络）。

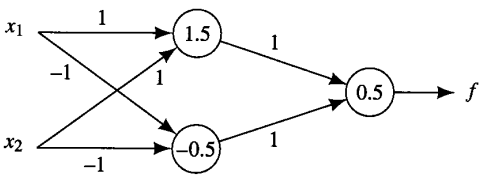


图3-4 执行偶校验函数的TLU网络

下一节将介绍训练多层前向网络的通用程序，即运用梯度下降的“反向传播程序(*backpropagation*)”。既然将求出误差函数对权向量的偏导，我们就把网络中所有阈值函数换成sigmoid函数（训练完毕，再把sigmoid函数换成阈值函数）。

3.3.2 表示符号

在图3-5中，我给出了一个由sigmoid单元组成的通用k层前向网络。除了在最后一层中的，其他所有的sigmoid单元称为“隐藏单元(*hidden unit*)”，因为它们的输出仅间接影响最终输出。图3-5中的网络只有一个输出单元；当存在两种以上的动作和输入类型时，应该使用几个输出单元（这时，应用一个译码表（*decoding scheme*）来把输出向量转换成类型[Brain, et al. 1962]采用的是基于最大移位寄存器序列的代码，［Dietterich & Bakiri 1991, Dietterich & Bakiri 1995］也对相似的纠错代码进行了研究）。

用图3-5来介绍一些有用的表示符号。正如同输入特征是一个输入向量的分量一样，我们认为每一个sigmoid单元层的输出也是向量的分量。sigmoid的第j ($1 \le j < k$)层将把向量$\mathbf{X}^{(j)}$作为其输出。然后，这一向量成为$j+1$层的输入向量。输入向量用$\mathbf{X}^{(0)}$来表示，（第k层TLU的）最终输出为f。每一层中的每一个sigmoid均有一个权向量（与其输入相关）；第j层的第i个sigmoid单元的一个权向量用$\mathbf{W}_i^{(j)}$来表示。如前所述，我们假设“阈值权”是相关权向量的最后一个分量。我们用$s_i^{(j)}$来表示j层的第i个sigmoid单元的加权总和输入。一个sigmoid单元的这一输入可称为

这一个单元的“激励（*activation*）”，由以下公式给出：

$$s_i^{(j)} = \mathbf{X}^{(j-1)} \cdot \mathbf{W}_i^{(j)}$$

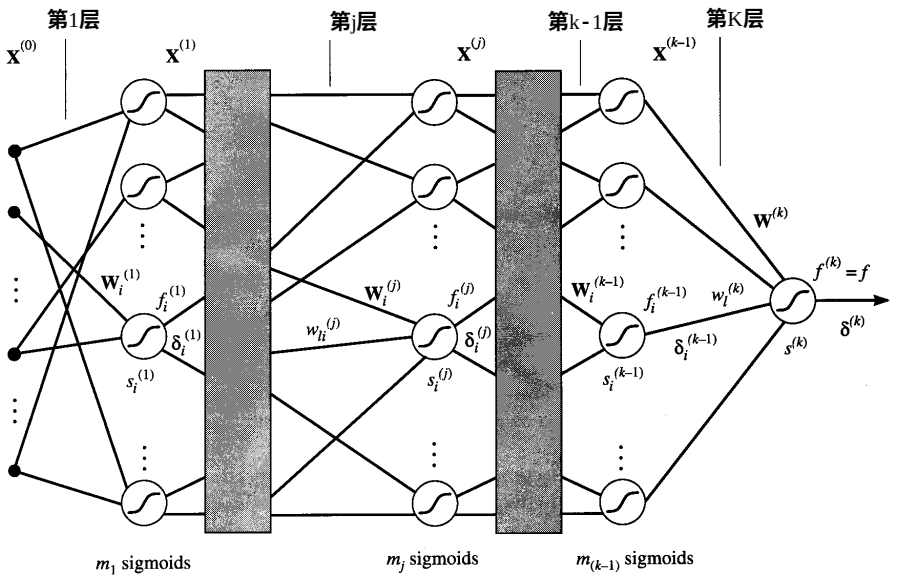


图3-5 *k*层sigmoid单元网络

第*j*层中sigmoid单元的数目用m_j来表示。向量$\mathbf{W}_i^{(j)}$的组件为$W_{l,i}^{(j)}$，这里$l = 1$，$m_{(j-1)}+1$。

3.3.3 反向传播方法

我们将计算平方差函数$\varepsilon = (d-f)^2$的梯度。此处用来计算梯度的权向量应包含网络中所有权值，然而，推导ε相对于单sigmoid的权向量相应的各组权值的偏导比较容易。ε相对$\mathbf{W}_i^{(j)}$的偏导为：

$$\frac{\partial \varepsilon}{\partial \mathbf{W}_i^{(j)}} \stackrel{\text{def}}{=} \left[\frac{\partial \varepsilon}{\partial w_{1i}^{(j)}}, \cdots, \frac{\partial \varepsilon}{\partial w_{li}^{(j)}}, \cdots, \frac{\partial \varepsilon}{\partial w_{m_{j-1}+1,i}^{(j)}} \right]$$

其中$W_{lj}^{(j)}$是$\mathbf{W}_l^{(j)}$的第l个分量。

如前所述，既然ε是完全通过$s_i^{(j)}$来依赖于$W_i^{(j)}$的，我们可用链式规则书写如下：

$$\frac{\partial \varepsilon}{\partial \mathbf{W}_i^{(j)}} = \frac{\partial \varepsilon}{\partial s_i^{(j)}} \frac{\partial s_i^{(j)}}{\partial \mathbf{W}_i^{(j)}}$$

而且因为$s_i^{(j)} = \mathbf{X}^{(j-1)} \cdot \mathbf{W}_i^{(j)}$，$\frac{\partial s_i^{(j)}}{\partial \mathbf{W}_i^{(j)}} = \mathbf{X}^{(j-1)}$。将其代入上式，得

$$\frac{\partial \varepsilon}{\partial \mathbf{W}_i^{(j)}} = \frac{\partial \varepsilon}{\partial s_i^{(j)}} \mathbf{X}^{(j-1)}$$

现在，$\frac{\partial \varepsilon}{\partial s_i^{(j)}} = \frac{\partial (d-f)^2}{\partial s_i^{(j)}} = -2(d-f)\frac{\partial f}{\partial s_i^{(j)}}$，所以

$$\frac{\partial \varepsilon}{\partial \mathbf{W}_i^{(j)}} = -2(d-f)\frac{\partial f}{\partial s_i^{(j)}}\mathbf{X}^{(j-1)}$$

量$(d-f)\frac{\partial f}{\partial s_i^{(j)}} = -\frac{1}{2}\frac{\partial \varepsilon}{\partial s_i^{(j)}}$在我们的计算中十分重要，可用$\delta_i^{(j)}$来表示。每一个$\delta_i^{(j)}$可反映出网络输出的平方差对相应的sigmoid函数的输入中的变化的灵敏度。用δ书写的ε的梯度如下：

$$\frac{\partial \varepsilon}{\partial \mathbf{W}_i^{(j)}} = -2\delta_i^{(j)}\mathbf{X}^{(j-1)}$$

因为我们将沿梯度的反方向改变权向量，所以整个网络的权改变的基本规则如下：

$$\mathbf{W}_i^{(j)} \leftarrow \mathbf{W}_i^{(j)} + c_i^{(j)}\delta_i^{(j)}\mathbf{X}^{(j-1)}$$

这里$C_i^{(j)}$是这一权向量的学习率常数（通常，网络中所有权向量的学习率相同）。我们发现，这一规则与用于一个单TLU或sigmoid单元的过程中的规则十分相似。一个权向量的改变值为一个因数与其（未加权的）输入向量的乘积。

3.3.4 计算最后一层的权值变化

接下来计算$\delta_i^{(j)}$。根据定义可得到：

$$\delta_i^{(j)} = (d-f)\frac{\partial f}{\partial s_i^{(j)}}$$

为了计算最后一个sigmoid单元的权变化，我们先来计算$\delta^{(k)}$

$$\delta^{(k)} = (d-f)\frac{\partial f}{\partial s^{(k)}}$$

因为f是sigmoid函数，$s^{(k)}$是其输入，所以得（如前）$\frac{\partial f}{\partial s^{(k)}} = f(1-f)$，这样：

$$\delta^{(k)} = (d-f)f(1-f)$$

所以，最后一层单元的反向传播权值调节可写为：

$$\mathbf{W}^{(k)} \leftarrow \mathbf{W}^{(k)} + c^{(k)}(d-f)f(1-f)\mathbf{X}^{(k-1)}$$

当最后一个sigmoid单元是网络中惟一的单元且$\mathbf{X}^{(k-1)} = \mathbf{X}$时，这一规则就与一般化Delta程序中所给出的规则相同。

3.3.5 计算中间层的权值变化

运用δ的表达式，我们可用相似的方法计算出如何改变网络中的任一权向量。回想一下：

$$\delta_i^{(j)} = (d-f)\frac{\partial f}{\partial s_i^{(j)}}$$

我们再次使用链式规则，最终输出f是通过j+1层的sigmoid的每一个总和输入来依赖$s_i^{(j)}$的。所以：

$$\delta_i^{(j)} = (d-f)\left[\frac{\partial f}{\partial s_1^{(j+1)}}\frac{\partial s_1^{(j+1)}}{\partial s_i^{(j)}} + \cdots + \frac{\partial f}{\partial s_l^{(j+1)}}\frac{\partial s_l^{(j+1)}}{\partial s_i^{(j)}} + \cdots + \frac{\partial f}{\partial s_{m_{j+1}}^{(j+1)}}\frac{\partial s_{m_{j+1}}^{(j+1)}}{\partial s_i^{(j)}}\right]$$

$$= \sum_{l=1}^{m_{j+1}} (d-f) \frac{\partial f}{\partial s_l^{(j+1)}} \frac{\partial s_l^{(j+1)}}{\partial s_i^{(j)}} = \sum_{l=1}^{m_{j+1}} \delta_l^{(j+1)} \frac{\partial s_l^{(j+1)}}{\partial s_i^{(j)}}$$

现在，还应计算 $\frac{\partial s_1^{(j+1)}}{\partial s_i^{(j)}}$ ，为了便于计算，先这样写：

$$s_l^{(j+1)} = \mathbf{X}^{(j)} \cdot \mathbf{W}_l^{(j+1)}$$

$$= \sum_{v=1}^{m_j+1} f_v^{(j)} w_{vl}^{(j+1)}$$

然后，因为权值不依赖于s，所以：

$$\frac{\partial s_l^{(j+1)}}{\partial s_i^{(j)}} = \frac{\partial \left[\sum_{v=1}^{m_j+1} f_v^{(j)} w_{vl}^{(j+1)} \right]}{\partial s_i^{(j)}} = \sum_{v=1}^{m_j+1} w_{vl}^{(j+1)} \frac{\partial f_v^{(j)}}{\partial s_i^{(j)}}$$

我们注意到，除非$v=i$（这时，$\frac{\partial f_v^{(j)}}{\partial s_v^{(j)}} = f_v^{(j)}(1-f_v^{(j)})$，$\frac{\partial f_v^{(j)}}{\partial s_i^{(j)}} = 0$。）所以，

$$\frac{\partial s_l^{(j+1)}}{\partial s_i^{(j)}} = w_{il}^{(j+1)} f_i^{(j)} (1-f_i^{(j)})$$

我们把上式代入 $\delta_i^{(i)}$ 的表达式中，得

$$\delta_i^{(j)} = f_i^{(j)}(1-f_i^{(j)}) \sum_{l=1}^{m_{j+1}} \delta_l^{(j+1)} w_{il}^{(j+1)}$$

上面这一等式是δ的递归等式（有趣的是，我们发现这一等式与误差函数无关，误差函数仅直接影响$\delta^{(k)}$的计算）。计算出$j+1$层的 $\delta_i^{(j+1)}$ 后，可用此等式来计算$\delta_i^{(j)}$。我们已经计算出基数$\delta^{(k)}$：

$$\delta^{(k)} = (d-f)f(1-f)$$

在一般权变化规则中把这个表达式用于δ，即

$$\mathbf{W}_i^{(j)} \leftarrow \mathbf{W}_i^{(j)} + c_i^{(j)} \delta_i^{(j)} \mathbf{X}^{(j-1)}$$

这一规则虽略显复杂，但它有一个直观合理的解释。量$\delta^{(k)}=(d-f)f(1-f)$控制了网络中所有权值调节的整体数量和符号（权值调节随着最终输出渐渐趋向 0或1而减小，因为它们对f的影响逐渐消失。）正如δ的递归等式所示，对“进入”j层的一个sigmoid单元里的权值的调节与此调节对这个sigmoid单元的输出的影响是成比例的（此比例因数为（$f^{(j)}(1-f^{(j)})$）。这些调节也与这个sigmoid单元的输出的任意变化对最终输出的一种“平均”影响成比例。这种平均影响取决于两个因素：一个是从j层sigmoid单元“输出”的权值（权值越小，它对“下游”的影响越小）；另一个是$j+1$层sigmoid单元输出的变化对最终输出的影响（由$\delta^{(j+1)}$来衡量）。通过反向的权值反向传播可轻松地完成这些计算（这一算法名为“backprop”）。有关其他backprop及其应用的信息，请参阅［Chauvin & Rumelhart 1995］。

作为反向传播处理过程的一例，我们来看看图 3-6中训练神经网络的一个步骤——从图中

的随机权值开始（为了使运用backprop方程更加容易，这里采用标准的网络表达方式）。我们的目标函数是由两个二进制变量组成的偶校验函数。输入（包括阈值输入）和所希望的标号为：

1) $x_1^{(0)}=1$，$x_2^{(0)}=0$，$x_3^{(0)}=1$，$d=0$

2) $x_1^{(0)}=0$，$x_2^{(0)}=0$，$x_3^{(0)}=1$，$d=1$

3) $x_1^{(0)}=0$，$x_2^{(0)}=1$，$x_3^{(0)}=1$，$d=0$

4) $x_1^{(0)}=1$，$x_2^{(0)}=1$，$x_3^{(0)}=1$，$d=1$

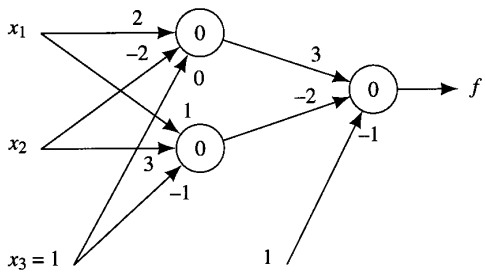


图3-6 用backprop训练的一个网络

第一个输入向量为(1, 0, 1)，它引发第一层面的输出$f_1^{(1)}=0.881$、$f_2^{(1)}=0.500$以及最终输出$f=0.665$。我们用基数方程算出$\delta^{(2)}=-0.148$。通过第二层的权值反向传播这个δ，得出$\delta_1^{(1)}=-0.047$和$\delta_2^{(1)}=0.074$。然后，运用权调节方程（和一个学习率常数c=1），得出新权值：

$$\mathbf{W}_1^{(1)}=(1.953,\ -2.000\ -0.047)$$
$$\mathbf{W}_2^{(1)}=(1.074,\ 3.000,\ -0.926)$$
$$\mathbf{W}^{(2)}=(2.870,\ -2.074,\ -1.148$$

我们注意到，因为$x_2^{(0)}=0$，$W_{2,2}^{(1)}$和$W_{1,2}^{(1)}$未被调节（即使$f_2^{(1)}=500$——这里是对调节的敏感度最高的地方）。对第一层的调节使$f_1^{(1)}$和$f_2^{(1)}$的值减小，然后它与第二层面的权值调节一起使f的值减小，同时也使这个输入向量的误差减小。试着写一个程序或用电子表格来继续这个训练过程，你定会受益匪浅。

3.4 一般化、准确度和过度拟合

上一节中介绍的神经网络的训练绝非典型。因为其维数太低，我们能训练所有四个可能的输入向量。根据图3-4，存在一组对所有输入向量进行正确分类的权值。然而，多数应用中的维数要高得多，通常为100或100以上。有100个二进制元素就有2^{100}种可能的输入向量。我们只可能训练其中的一小部分。而且，即使网络能对整个训练集合正确归类（这是不太可能的[⊖]），也无法保证它会按我们的要求对其他向量进行正确的归类。当一个网络可对不属于训练集合的向量进行正确的分类时，就称之为“一般化(*generalize*)”网络。一般化能力由其归类的准确度来衡量。后面会介绍如何估量一般化准确度。

为什么我们希望神经网络能够一般化？这与曲线拟合有些类似。当我们试图拟合一条直线或一条低次多项式曲线时，若已知大量数据且对这些数据的拟合十分贴切，那么，我们就有信心挖掘出隐含在数据中的函数关系。这一拟合曲线可用来估计（具有合理的可信度）在拟合过程中未曾使用过的新数据点。若一条直线对数据的拟合不贴切，那么我们也许会再用一个二次曲线来试试，依此类推。神经网络与此十分相似。一个神经网络计算出其输入的一个复杂的非线性函数。如果对这些输入的归类实际上是某一函数，这个函数与由网络执行的函数组中的某一个十分接近，同时，如果一个已经训练的网络对训练数据的拟合十分贴切——那么，对于大量输入来说，这一训练过程很可能挖掘出了输入与归类之间隐含的函数关系。这样的话，将能

[⊖] 执行总会受噪声限制。输入向量的有些分量可能有噪声。如某自于有噪声的传感器。或者训练集合中的有些分量被标错了。前者为属性噪声，后者为分类噪声。

对新输入很好地一般化。

训练输入向量的数量应当比网络的自由度的数量（即可变权值的数量）大，这一点十分重要。我们再次运用曲线拟合这一类比来帮助理解。即使一条直线无法完全拟合数据，但若存在m个数据点，那么就必定能找到$m-1$次多项式来完全拟合数据。然而，由于不管这m个数据点如何布局，我们都可以找到这样一种拟合方式，因而无法发现数据的特殊之处。与此相反，我们过度拟合数据。若数据有噪声（也许，一条直线可拟合没有噪声的数据），那么，额外增加的自由度本质上正好拟合了噪声。众所周知，曲线拟合中，数据点的数量比用来拟合的多项式的次数大得多。而且，有了足够的数据， Occam的Razor原则指出，应选择那个能适度拟合数据的最低次多项式[⊖]。在图3-7中，通过图解说明了曲线拟合的一些观点。在图中，与没有像高次函数那样过度拟合数据的曲线和简单的直线相比，二次函数提供了一种更恰当的拟合方式。与此相似的原则也适用于神经网络。

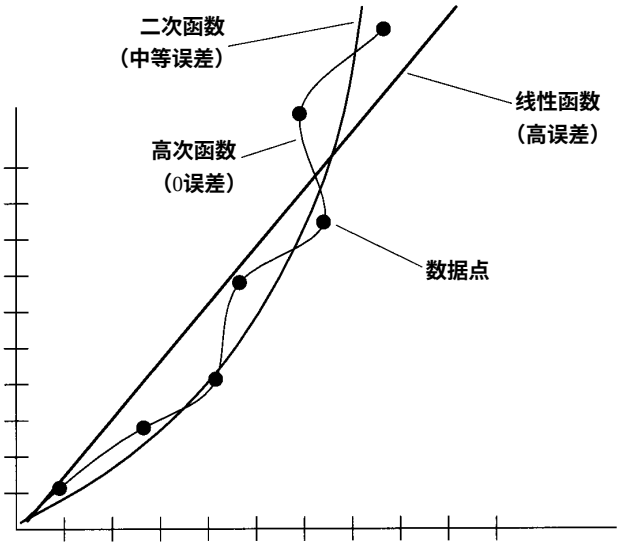


图3-7 曲线拟合

我们来考虑一下一个有n个输入、h个隐藏层单元和一个输出的二层前向神经网络。这样的网络有接近$nh+h=(n+1)h$个可变权值（不包括阈值权值）。对于固定的输入维来说，自由度实质上由隐藏单元的数量来控制。我们希望，对训练集合的归类的误差百分比随隐藏单元的数量的增加而减小——直至达到最小误差百分比（当然，若训练集合是线性可分的，那么，我们只要用一个隐藏单元就可以使训练集合达到零误差）。图3-8中，给出了一个典型的例子，其中训练集合误差是隐藏单元的数量的函数。

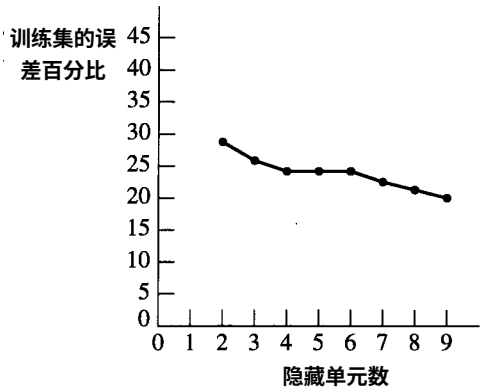


图3-8 误差与隐藏单元的数量（从[Duda, Hart & Stork 1998]中改编）

⊖ William of Occam (1285~1349年)是英国的一位哲学家，据说他曾这样说过：实体在没有需要时不应被扩大。

然而，如前所述，为了避免过度拟合，我们并不需要$(n+1)h$这么多的隐藏单元，因为它已接近训练输入向量的数量了。

尽管训练集合误差小，但其一般化并不一定好。我们可用各种方法来估计对不属于训练集合、但来自与训练集合相同的内在布局的误差率。这一误差率在统计学中称为“抽样（*out-of-sample-set*）”误差率。也许，最简单的技术就是把有待训练的输入向量分为两个互不相交的集合，然后用其中一个集合来进行训练，称之为“训练集合(*training set*)”。训练完毕，用另一个集合来估量抽样误差率，称之为“检验集合”。若两个集合中的向量的数量均很多，那么，检验集合所得出的误差率就是对一般化准确度的合理估量（当然，它常常高估了实际的抽样误差率。为什么？）。有经验的设计者会把2/3的可变向量归入训练集合，而另外1/3则归入检验集合。

另一种流行的估量一般化准确度的方法称为“交叉检验(*cross validation*)”。这种方法把有待训练的向量分成k（通常k为10）个互不相交的子集，即所谓的“fold”。然后，选其中一个fold作为检验集合而另外k-1个（组合起来）作为训练集合。这样做k次，每次选不同的一个fold作为检验集合而剩下的作为训练集合（在对相应的训练集合训练完毕后）。为每个检验集合计算误差率，并求出这些误差率的平均值作为对抽样误差的估量。在$k=m$的特例中，m为可使用的有标号的向量的数量，我们进行所谓的“排除一个（*leave-one-out*）”的交叉检验。实验结果表明，有10个fold的交叉检验所给出的一般化准确度（略显悲观）比较合理。图3-9中，通过图解说明了为解决这一典型的归类问题，以及检验集合对抽样误差率的估量是如何随隐藏单元的数量变化而变化的。注意，检验集合误差是怎样开始由于过度拟合而随隐藏单元的数量的增加而增大的。

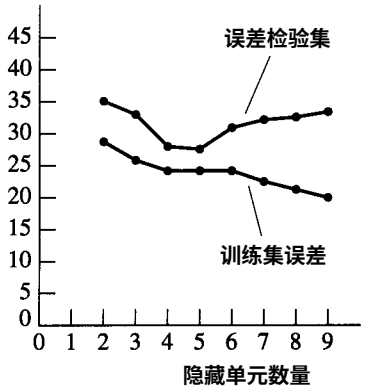


图3-9 一般化误差估计与隐藏单元数量（从[Duda,Hart, & Stork 1998]中改编）

3.5 补充读物和讨论

神经网络已经运用于解决模式识别、自动控制和大脑功能模型化的问题中，其中典型的例子有手写（ZIP译码）识别[LeCun, et al. 1989]、语音识别[Waibel, et al. 1988]和学习阅读手写文本中的语句[Sejnowski & Rosenberg 1987]。但值得注意的是，为这些应用而对神经网络的设计和训练仍是一门极需经验和实验的艺术。一些最佳的神经网络的研究成果和应用展示于年度“神经处理系统会议（NIPS）”，此会议发表的会议论文集名为“神经信息处理系统的进展(*Advances in Neural Information Processing Systems*)”。

前面提到，一个神经网络最基本的单元是TLU，它用一个超平面把输入向量的空间分开。

若两个训练输入子集的凸包（*convex hull*）不相交，那么超平面即可完成分割。可从本章所述的训练过程中或运用线性编程方法（众所周知，这种方法具有多项式复杂度）来得到一个分割超平面[Karmarkar 1984]。

我只介绍了分层前向网络。有关递归网络的行为的分析更加复杂，[Hertz, Krogh, & Palmer 1991]一书运用与物理学动态系统的类比对此作出了清晰的解释。反向传播这一算法已被([Pineda 1987, Almeida 1987, Rohwer & Forrest 1987])一般化，从而运用到递归网络中（当这样的网络汇集成稳定的状态时）([Hertz, Krogh, & Palmer 1991, pp.172-176]介绍了这种算法)。

神经网络仅是许多机器学习结构的一种，另一种为“决策树”——它之所以深得一些人的喜爱是因为由它执行的函数（如DNF布尔函数）比神经网络中的函数更易理解。在人工智能领域中，Ross Quinlan首先提出决策树学习方法ID3[Quinlan 1979]和C4.5[Quinlan 1993]（统计学家们也已经开发出了相似的技术[Breiman, et al.1984]）。

本书的其他地方继续介绍其他学习技术，但在这里，我想提一下有关这一论题的信息资源。最重要的年度会议是国际机器学习会议（ICML），该会议发行会议论文集。计算学习理论进展研讨会（COLT）发表有关计算学习理论的论文。期刊主要有《Machine Learning》(机器学习)。重要的教材是[Mitchell, T. 1997, Langley 1996, Weiss & Kulikowski 1991]。有关神经网络的书有[Fu 1994, Haykin 1994, Hertz, Krogh, & Palmer 1991]。[Shavlik & Dietterich 1990]是一本论文集[Dietterich 1990]是对机器学习领域的一次出色的综述。

习题

3.1 一个具有权向量W和阈值θ的TLU完成一个超平面边界。推导一个表示此超平面与原点之间的欧几里德距离的表达式。从一个任意点X开始（参照图3-1）。

3.2 下面这个训练集合是线性可分的。

输入	输出
1 0 0	1
0 1 1	0
1 1 0	1
1 1 1	0
0 0 1	0
1 0 1	1

（手工）训练此训练集合中的线性阈值单元。你的单元包括执行阈值的输入在内的四个输入。设所有权值的初始值为0。用固定递增纠错程序来训练你的单元直至找到一个解。在每次训练循环后标出各组权值。以前面的输入为顶点画出一个三维立方体的草图，并根据最终权集画出分割平面的草图。

3.3 用一个TLU对一组n维输入向量归类。但假设，由于技术原因只能用非负权值。你打算如何完成这一归类？

3.4 设计（无需训练）一个前向网络来执行一个由两个输入组成的“或”函数。你的网络具备：(1) 一个由接收输入x_1、x_2的线性阈值单元组成的隐藏层面。(2) 一个把隐藏层的输出作为输入（但不是从x_1、x_2来的输入）的最终输出单元。

3.5 证明：用仅由一个隐藏层面组成的阈值单元网络便可实现任一由n个输入构成的布尔函数。

3.6 考虑下面图中的非线性单元：

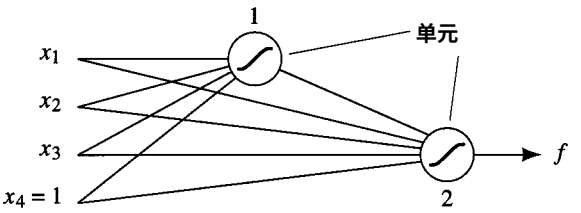
如果 X · W<-b，f(X)=0

如果 X · W<-b，f(X)=1

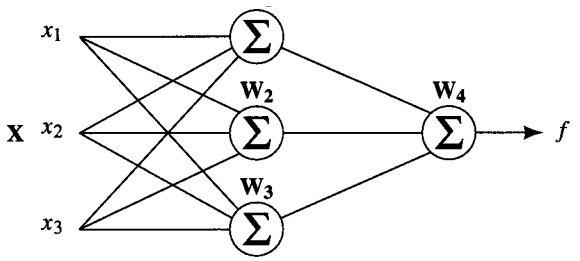
否则，f(X) =(1/2b)(X · W+b)

与通常的阈值或sigmoid的非线性不同，我们现有一个如图所定义的“斜坡（ramp）”函数。通过在每个输入向量表达中，递增地应用下降最陡峭的程序，使平方差 ε（即实际输出f与所希望的输出d之间的误差）最小，从而推导出权向量 W的权调节规则。评论一下你的结论。

3.7 考虑下图所示的“级联（cascade）”网络。它有三个输入，x_1、x_2和x_3。sigmoid单元1接收所有这些输入（包括一个“阈值输入”，x_4≡1）。sigmoid单元2把sigmoid单元1的输出以及x_1、x_2、x_3和x_4作为其输入。所有sigmoid单元的输入均用可调节权值加权。基于网络输出与训练输入向量集合的标号之间的二次方差最小化，为该网络推导出一个适当实例化的反向传播式的增量调节程序。答案可用向量表达方式，即你无需具体到输入向量的分量x_i。



3.8 一个点积单元（DPU）计算一个输入向量 W和一个权向量 W的向量点积。它没有阈值；它的输出可简单地写为矩阵等式 WX，其中，W是行向量而X是列向量。考虑图中的DPU网络。证明整个网络等同于一个单DPU。这一DPU的权向量是什么？



3.9 1) 为一个sigmoid单元制定一个递增、梯度下降的权变化规则。在单元里，误差函数为ε=|d-f |，d为所希望的输出，f =1/(1-e^{-s})，s=X · W，X为一个n+1维输入向量，且W为可被改变的n+1维权向量。

2) 解释一下你的规则与用二次方差作为标准的规则的不同之处。

3) 若采用这一新的误差标准，应对多层前向网络的反向传播规则做哪些改动？