第四章　反向传播网络

反向传播网络（ Back-Propagation Network，简称 BP 网络）是将 W-H 学习规则一般化，对非线性可微分函数进行权值训练的多层网络。

BP 网络主要用于：

1）函数逼近：用输入矢量和相应的输出矢量训练一个网络逼近一个函数；

2）模式识别：用一个特定的输出矢量将它与输入矢量联系起来；

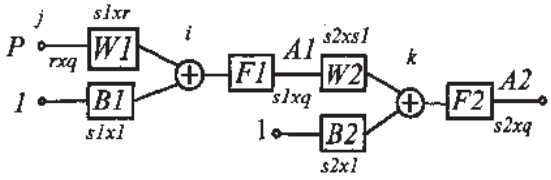
3）分类：把输入矢量以所定义的合适方式进行分类；

4）数据压缩：减少输出矢量维数以便于传输或存储。

在人工神经网络的实际应用中，80%~90%的人工神经网络模型是采用 BP 网络或它的变化形式，它也是前向网络的核心部分，体现了人工神经网络最精华的部分。在人们掌握反向传播网络的设计之前，感知器和自适应线性元件都只能适用于对单层网络模型的训练，只是后来才得到了进一步拓展。

4.1　BP 网络模型与结构

一个具有 r 个输入和一个隐含层的神经网络模型结构如图 4.1 所示。



$i = 1, 2, ..., s1$；　　$k = 1, 2, ..., s2$；

$j = 1, 2, ..., r$.

图 4.1　具有一个隐含层的神经网络模型结构图

感知器和自适应线性元件的主要差别在激活函数上：前者是二值型的，后者是线性的。BP 网络具有一层或多层隐含层，除了在多层网络上与前面已介绍过的模型有不同外，其主要差别也表现在激活函数上。 BP 网络的激活函数必须是处处可微的，所以它就不能采用二值型的阀值函数{0, 1}或符号函数{-1, 1}， BP 网络经常使用的是 S 型的对数或正切激活函数和线性函数。

图 4.2 所示的是 S 型激活函数的图形。可以看到 $f(\cdot)$ 是一个连续可微的函数，其一阶导数存在。对于多层网络，这种激活函数所划分的区域不再是线性划分，而是由一个非线性的超平面组成的区域。它是比较柔和、光滑的任意界面，因而它的分类比线性划分精确、合理，这种网络的容错性较好。另外一个重要的特点是由于激活函数是连续可微的，它可

以严格利用梯度法进行推算，它的权值修正的解析式十分明确，其算法被称为误差反向传播法，也简称 BP 算法，这种网络也称为 BP 网络。

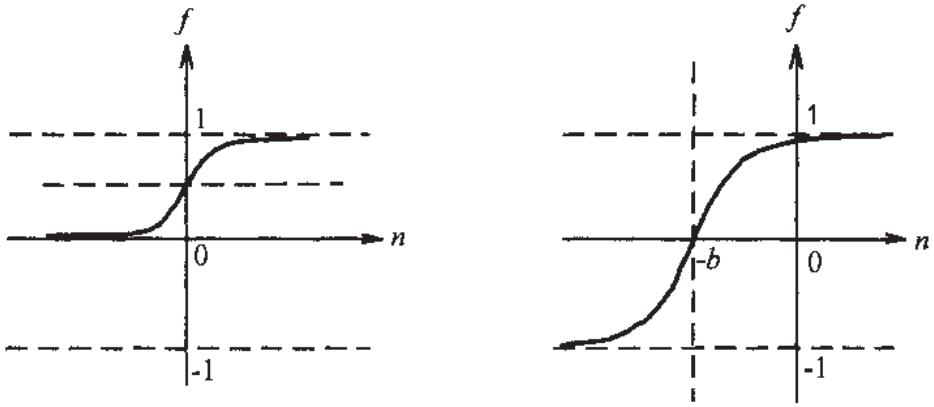
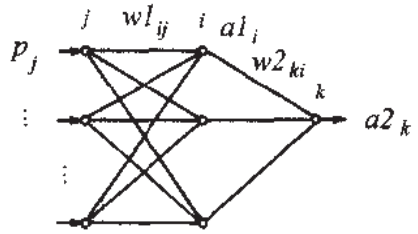


图 4.2　BP 网络 S 型激活函数

因为 S 型函数具有非线性放大系数功能，它可以把输入从负无穷大到正无穷大的信号，变换成-1 到 1 之间输出，对较大的输入信号，放大系数较小；而对较小的输入信号，放大系数则较大，所以采用 S 型激活函数可以去处理和逼近非线性的输入/输出关系。不过，如果在输出层采用 S 型函数，输出则被限制到一个很小的范围了，若采用线性激活函数，则可使网络输出任何值。所以只有当希望对网络的输出进行限制，如限制在 0 和 1 之间，那么在输出层应当包含 S 型激活函数，在一般情况下，均是在隐含层采用 S 型激活函数，而输出层采用线性激活函数。

4.2　BP 学习规则

BP 网络的产生归功于 BP 算法的获得。BP 算法属于δ 算法，是一种监督式的学习算法。其主要思想为：对于 q 个输入学习样本：$P^1, P^2, ..., P^q$，已知与其对应的输出样本为：$T^1, T^2, ..., T^q$。学习的目的是用网络的实际输出 $A^1, A^2, ..., A^q$ 与目标矢量 $T^1, T^2, ..., T^q$ 之间的误差来修改其权值，使 A^l，（$l = 1, 2, ..., q$）与期望的 T^l 尽可能地接近；即：使网络输出层的误差平方和达到最小。它是通过连续不断地在相对于误差函数斜率下降的方向上计算网络权值和偏差的变化而逐渐逼近目标的。每一次权值和偏差的变化都与网络误差的影响成正比，并以反向传播的方式传递到每一层的。



$k = 1, 2, ..., s2;$　　$i = 1, 2, ..., s1;$

$j = 1, 2, ..., r.$

图 4.3　具有一个隐含层的简化网络图

BP 算法是由两部分组成：信息的正向传递与误差的反向传播。在正向传播过程中，输入信息从输入经隐含层逐层计算传向输出层，每一层神经元的状态只影响下一层神经元的状态。如果在输出层没有得到期望的输出，则计算输出层的误差变化值，然后转向反

向传播，通过网络将误差信号沿原来的连接通路反传回来修改各层神经元的权值直至达到期望目标。

为了明确起见，现以图 4.1 所示两层网络为例进行 BP 算法推导，其简化图如图 4.3 所示。

设输入为 P，输入神经元有 r 个，隐含层内有 $s1$ 个神经元，激活函数为 $F1$，输出层内有 $s2$ 个神经元，对应的激活函数为 $F2$，输出为 A，目标矢量为 T。

4.2.1 信息的正向传递

1）隐含层中第 i 个神经元的输出为：

$$a1_i = f1(\sum_{j=1}^{r} w1_{ij} p_j + b1_i), i = 1, 2, ..., s1 \tag{4.1}$$

2）输出层第 k 个神经元的输出为：

$$a2_k = f2(\sum_{i=1}^{s1} w2_{ki} a1_i + b2_k), k = 1, 2, ..., s2 \tag{4.2}$$

3）定义误差函数为：

$$E(W, B) = \frac{1}{2}\sum_{k=1}^{s2}(t_k - a2_k)^2 \tag{4.3}$$

4.2.2 利用梯度下降法求权值变化及误差的反向传播

（1）输出层的权值变化

对从第 i 个输入到第 k 个输出的权值有：

$$\begin{aligned}\Delta w2_{ki} &= -\eta\frac{\partial E}{\partial w2_{ki}} = -\eta\frac{\partial E}{\partial a2_k}\frac{\partial a2_k}{\partial w2_{ki}} \\ &= \eta(t_k - a2_k)\ f2' \quad a1_i = \eta\ \delta_{ki}\ a1_i\end{aligned} \tag{4.4}$$

其中：

$$\delta_{ki} = (t_k - a2_k) = e_k\ f2' \tag{4.5}$$

$$e_k = t_k - a2_k \tag{4.6}$$

同理可得：

$$\begin{aligned}\Delta b2_{ki} &= -\eta\frac{\partial E}{\partial b2_{ki}} = -\eta\frac{\partial E}{\partial a2_k}\cdot\frac{\partial a2_k}{\partial b2_{ki}} \\ &= \eta(t_k - a2_k)\cdot f2' \quad = \eta\cdot\delta_{ki}\end{aligned} \tag{4.7}$$

（2）隐含层权值变化

对从第 j 个输入到第 i 个输出的权值，有：

$$\begin{aligned}\Delta w1_{ij} &= -\eta\frac{\partial E}{\partial w1_{ij}} = -\eta\frac{\partial E}{\partial a2_k}\cdot\frac{\partial a2_k}{\partial a1_i}\cdot\frac{\partial a1_i}{\partial w1_{ij}} \\ &= \eta\sum_{k=1}^{s2}(t_k - a2_k)\cdot f2' \quad \cdot w2_{ki}\ \ f1' \quad \cdot p_j = \eta\cdot\delta_{ij}\cdot p_j\end{aligned} \tag{4.8}$$

其中:

$$\delta_{ij} = e_i \cdot f1', \quad e_i = \sum_{k=1}^{s2} \delta_{ki} w2_{ki} \tag{4.9}$$

同理可得:

$$\Delta b1_i = \eta \delta_{ij} \tag{4.10}$$

在 MATLAB 工具箱中，上述公式的计算均已编成函数的形式，通过简单的书写调用即可方便地获得结果。具体有:

1 ）对于(4.1)式所表示的隐含层输出，若采用对数 S 型激活函数，则用函数 **logsig.m** ；若采用双曲正切 S 型激活函数，则用函数 **tansig.m** ；

2 ）对于(4.2)式所表示的输出层输出，若采用线性激活函数有 **purelin .m** 与之对应;

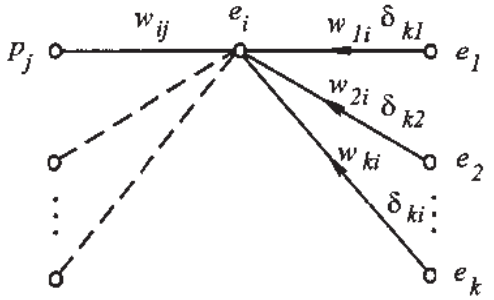
3 ）对于(4.3)式所表示的误差函数，可用函数 **sumsqr.m** 求之;

4 ）有 **learnbp.m** 函数专门求(4.4)、(4.7)、(4.8)和(4.10)式所表示的输出层以及隐含层中权值与偏差的变化量;

5 ）由(4.5)和(4.9)式所表示的误差的变化有函数 **deltalin.m** ， **deltatan.m** 和 **deltalog.m** 来解决。它们分别用于线性层、双曲正切层和对数层。

4.2.3 误差反向传播的流程图与图形解释

误差反向传播过程实际上是通过计算输出层的误差 e_k ，然后将其与输出层激活函数的一阶导数 $f2'$ 相乘来求得 δ_{ki} 。由于隐含层中没有直接给出目标矢量，所以利用输出层的 δ_{ki} 进行误差反向传递来求出隐含层权值的变化量 $\Delta w2_{ki}$ 。然后计算 $e_i = \sum_{k=1}^{s2} \delta_{ki} \ w2_{ki}$ ，并同样通过将 e_i 与该层激活函数的一阶导数 $f1'$ 相乘，而求得 δ_{ij} ，以此求出前层权值的变化量 $\Delta w1_{ij}$ 。如果前面还有隐含层，沿用上述同样方法依此类推，一直将输出误差 e_k 一层一层的反推算到第一层为止。图 4.4 给出了形象的解释。



$k = 1, 2, ..., s2;$ $i = 1, 2, ..., s1;$

$j = 1, 2, ..., r.$

图 4.4 误差反向传播法的图形解释

BP 算法要用到各层激活函数的一阶导数，所以要求其激活函数处处可微。对于对数 S 型激活函数 $f(n)=\frac{1}{1+e^{-n}}$，其导数为:

$$f'(n)=\frac{0-e^{-n}(-1)}{(1+e^{-n})^2}=\frac{1}{(1+e^{-n})^2}(1+e^{-n}-1)$$

$$=\frac{1}{1+e^{-n}}(1-\frac{1}{1+e^{-n}})=f(n)[1-f(n)]$$

对于线性函数的导数有:

$$f'(n)=n'=1$$

所以对于具有一个 S 型函数的隐含层，输出层为线性函数的网络，有:

$$f2'=1,\quad f1'=a(1-a)$$

4.3 BP 网络的训练过程

为了训练一个 BP 网络，需要计算网络加权输入矢量以及网络输出和误差矢量，然后求得误差平方和。当所训练矢量的误差平方和小于误差目标，训练则停止，否则在输出层计算误差变化，且采用反向传播学习规则来调整权值，并重复此过程。当网络完成训练后，对网络输入一个不是训练集合中的矢量，网络将以泛化方式给出输出结果。

在动手编写网络的程序设计之前，必须首先根据具体的问题给出的输入矢量 P 与目标矢量 T，并选定所要设计的神经网络的结构，其中包括以下内容:

①网络的层数; ②每层的神经元数; ③每层的激活函数。

由于 BP 网络的层数较多且每层神经元也较多，加上输入矢量的组数庞大，往往使得采用一般的程序设计出现循环套循环的复杂嵌套程序，从而使得程序编得既费时，又不易调通，浪费了大量的时间在编程中而无暇顾及如何设计出具有更好性能的网络来。在这点上 MATLAB 工具箱充分展示出其神到之处。它的全部运算均采用矩阵形式，使其训练既简单，又明了快速。为了能够较好地掌握 PB 网络的训练过程，下面我们仍用两层网络为例来叙述 BP 网络的训练步骤。

1 ）用小的随机数对每一层的权值 W 和偏差 B 初始化，以保证网络不被大的加权输入饱和; 并进行以下参数的设定或初始化:

a)期望误差最小值 err_goal ;

b)最大循环次数 max_epoch ;

c)修正权值的学习速率 lr，一般情况下 lr = 0.01 ~ 0.7 ;

f)从 1 开始的循环训练: for epoch = 1 : max_epoch ;

2 ）计算网络各层输出矢量 A1 和 A2 以及网络误差 E:

```
A1 = tansig ( W1*P, B1);
```

```
A2 = purelin ( W2*A1, B2);
E = T - A;
```

3 ）计算各层反传的误差变化 D2 和 D1 并计算各层权值的修正值以及新权值:

```
D2 = deltalin ( A2, E );
D1 = deltatan ( A1, D2, W2 );
[ dW1, dB1 ] = learnbp ( P, D1, lr );
[ dW2, dB2 ] = learnbp ( A1, D2, lr );
W1 = W1 + dW1; B1 = B1 + dB1;
W2 = W2 + dW2; B2 = B2 + dB2;
```

4 ）再次计算权值修正后误差平方和:

```
SSE = sumsqr(T - purelin(W2*tansig(W1*P,B1),B2));
```

5 ）检查 SSE 是否小于 err_goal,若是，训练结束；否则继续。

以上就是 BP 网络在 MATLAB 中的训练过程。可以看出其程序是相当简单明了的。即使如此，以上所有的学习规则与训练的全过程，仍然可以用函数 **trainbp.m** 来完成。它的使用同样只需要定义有关参数：显示间隔次数，最大循环次数，目标误差，以及学习速率，而调用后返回训练后权值，循环总数和最终误差:

```
TP = [ disp_fqre   max_epoch   err_goal   lr ];
[ W, B, epochs, errors ] = trainbp ( W, B,'F', P, T, TP );
```

函数右端的'F'为网络的激活函数名称。

当网络为两层时，可从第一层开始，顺序写出每一层的权值初始值，激活函数名，最后加上输入、目标输出以及 TP ，即:

```
[W1,B1,W2,B2,W3,B3,epochs,errors] = trainbp(W1,B1,'F1',W2,B2,'F2',W3,B3,'F3',P,T,TP);
```

神经网络工具箱中提供了两层和三层的 BP 训练程序，其函数名是相同的，都是 **trainbp.m** ，用户可以根据层数来选取不同的参数。

【例 4.1 】用于函数逼近的 BP 网络的设计。

一个神经网络最强大的用处之一是在函数逼近上。它可以用在诸如被控对象的模型辨识中，即将过程看成一个黑箱子，通过测量其输入/输出特性，然后利用所得实际过程的输入/输出数据训练一个神经网络，使其输出对输入的响应特性具有与被辨识过程相同的外部特性。

下面给出一个典型的用来进行函数逼近的两层结构的神经网络，它具有一个双曲正切

型的激活函数隐含层，其输出层采用线性函数。

这里有 21 组单输入矢量和相对应的目标矢量，试设计神经网络来实现这对数组的函数关系。

```
P = -1 : 0.1 :1;
T = [ -0.96    0.577   -0.0729  0.377    0.641    0.66    0.461   0.1336 ...
      -0.201   -0.434    -0.5    -0.393   -0.1647  0.0988  0.3072 ...
      0.396    0.3449  0.1816 -0.0312   -0.2183   -0.3201 ];
```

为此，我们选择隐含层神经元为 5 。较复杂的函数需要较多的隐含层神经元，这可以根据试验或经验来确定。

图 4.5 给出了目标矢量相对于输入矢量的图形。一般在测试中常取多于训练用的输入矢量来对所设计网络的响应特性进行测试。在函数逼近的过程中，画出网络输出相对于输入矢量的图形，可以观察到网络在训练过程中的变化过程。为了达到这一目的，我们定义一个密度较大的第二个输入矢量:

```
P2 = -1 : 0.025 :1;
```

首先对网络进行初始化:

```
[ R, Q ] = size (P);        [ S2, Q]= size (T);        S1 = 5;
[ W1, B1 ] = rands ( S1, R );
[ W2, B2 ] = rands ( S2, S1 );
```

通过测试，用输入矢量 P2 来计算网络的输出:

```
A2 = purelin ( W2*tansig ( W1*P2, B1 ), B2 );
```

可以画出结果来观察初始网络是如何接近所期望训练的输入/输出关系，如图 4.6 所示。其中，初始网络的输出值用实线给出。

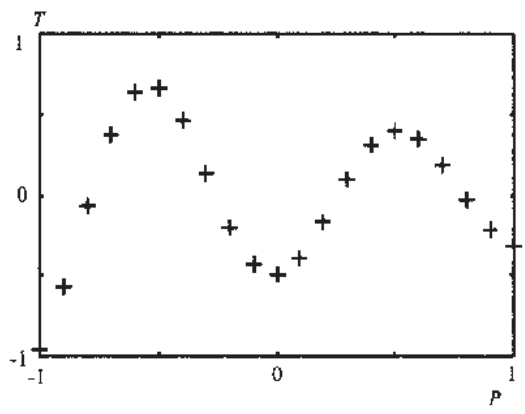


图 4.5 以目标矢量相对于输入矢量的图形

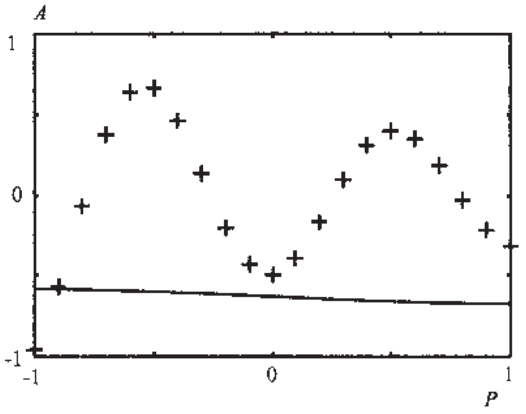


图 4.6 初始网络的输出曲线

网络训练前的误差平方和为 11.9115 ，其初始值为:

```
W10 = [ 0.7771   0.5336   -0.3874   -0.2980   0.0265];
B10 = [0.1822;0.6920;-0.1758;0.6830;-0.4614];
W20 = [-0.1692   0.0746   -0.0642   -0.4256   -0.6433];
B20 = [-0.6926];
```

下面定义训练参数并进行训练:

```
disp_fqre = 10;      max_epoch = 8000; err_goal = 0.02;      lr = 0.01;
TP = [ disp_fqre   max_epoch   err_goal   lr ];
trainbp ( W1, B1, 'tansig', W2 B2, 'purelin', P, T, TP ]
```

由此可以返回训练后的权值、训练次数和偏差。

图 4.7 至图 4.10 给出了网络输出值随训练次数的增加而变化的过程。每个图中标出了循环数目以及当时的误差平方和。

图 4.11 给出了 6801 次循环训练后的最终网络结果，网络的误差平方和落在所设定的 0.02 以内(0.0199968)。

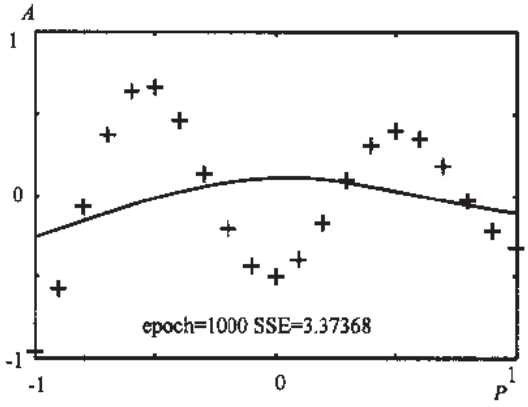


图 4.7 训练 1000 次的结果

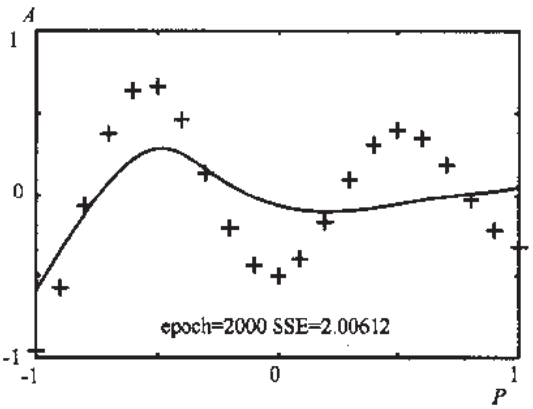


图 4.8 训练 2000 次的结果

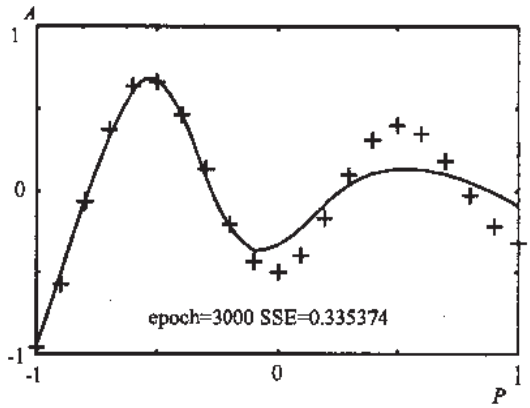


图 4.9 训练 3000 次的结果

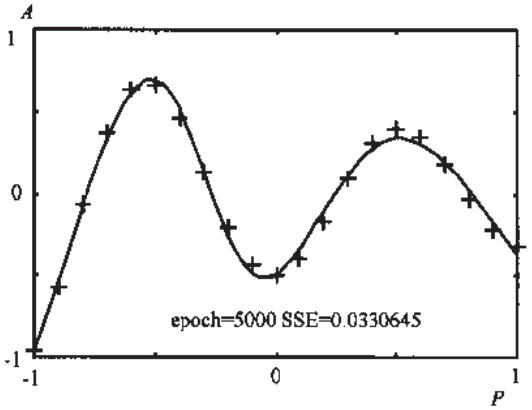
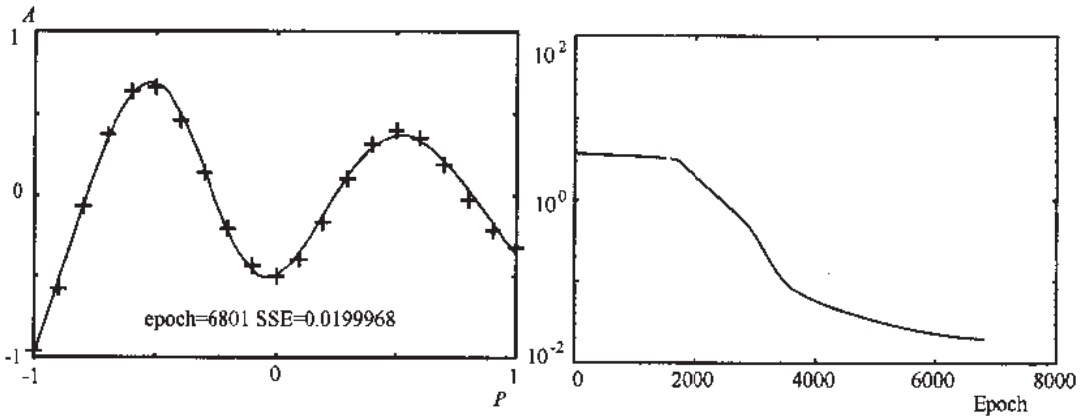


图 4.10 训练 5000 次的结果



(a) 训练结束后的网络输出结果　　(b) 训练结束后的网络误差记录

图 4.11　训练结束后的网络输出与误差结果

因为反向传播法采用的是连续可微函数，所以网络对输入/输出的逼近是平滑的。另外，虽然网络仅在输入值-1, -0.9, -0.8, ..., 0.9, 1.0 处进行训练，但对于其他输入值的出现，例如，对训练后的网络输入 $p = 0.33$ 的值，网络的输出端可得其对应输出为:

```
》 A1 = tansig(W1*0.33,B1);
》 A2 = purelin(W2*A1,B2)
A2 =
    0.1659
```

正如所希望的那样，这个值落在输入矢量为 0.3 和 0.4 所对应的输出矢量之间。网络的这个能力使其平滑地学习函数，使网络能够合理地响应被训练以外的输入。这性质称为泛化性能。要注意的是，泛化性能只对被训练的输入/输出对最大值范围内的数据有效，即网络具有内插值特性，不具有外插值性。超出最大训练值的输入必将产生大的输出误差。

4.4　BP 网络的设计

在进行 BP 网络的设计时，一般应从网络的层数、每层中的神经元个数和激活函数、初始值以及学习速率等几个方面来进行考虑。下面讨论一下各自选取的原则。

4.4.1　网络的层数

理论上已经证明：具有偏差和至少一个 S 型隐含层加上一个线性输出层的网络，能够逼近任何有理函数。这实际上已经给了我们一个基本的设计 BP 网络的原则。增加层数主要可以更进一步的降低误差，提高精度，但同时也使网络复杂化，从而增加了网络权值的训练时间。而误差精度的提高实际上也可以通过增加隐含层中的神经元数目来获得，其训

练效果也比增加层数更容易观察和调整。所以一般情况下，应优先考虑增加隐含层中的神经元数。

另外还有一个问题：能不能仅用具有非线性激活函数的单层网络来解决问题呢？结论是：没有必要或效果不好。因为能用单层非线性网络完美解决的问题，用自适应线性网络一定也能解决，而且自适应线性网络的运算速度还更快。而对于只能用非线性函数解决的问题，单层精度又不够高，也只有增加层数才能达到期望的结果。

这主要还是因为一层网络的神经元数被所要解决的问题本身限制造成的。下面给出两个例题来说明此问题。

【例 4.2】考虑两个单元输入的联想问题。其输入和输出矢量分别为：

$$P=[\,-3 \quad 2\,], \quad T=[\,0.4 \quad 0.8\,]$$

解：

当采用含有一个对数 S 型单层网络求解时，可求得解为：

$$w=0.3350$$

$$b=0.5497$$

此时所达到的误差平方和 err_goal < 0.001 。若将这个误差转换成输出误差时，其绝对误差约为 0.02 。

若采用自适应线性网络来实现此联想，得解为：

$$w=0.08$$

$$b=0.64$$

此网络误差为：$e=T-Y=0$ 。

我们还可以从点拟合这个角度来看这个问题。以输入 P 和输出 A 分别为横坐标和纵坐标，可得图 4.12，其中，虚线为采用非线性网络拟合的结果。求两点的联想问题实质上是求以此两点所拟合的直线，所以当采用自适应线性网络来求解此问题时，可以得到零误差的完美解。而当采用 S 型函数对此两点进行拟合，因为点数太少，所以反而产生一定的误差。

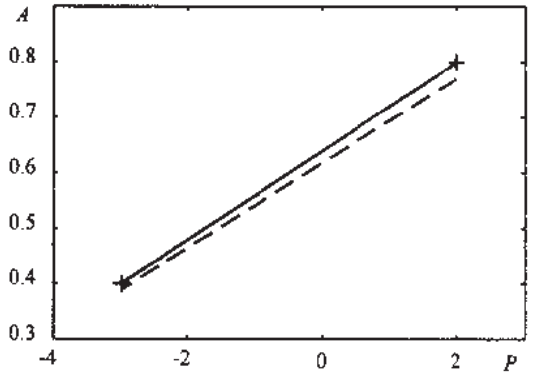


图 4.12　两种不同网络的解

【例 4.3】用一层网络来实现下面的输入/输出关系：

$$P=[\,-6 \quad -6.1 \quad -4.1 \quad -4 \quad 4 \quad 4.1 \quad 6 \quad 6.1\,],$$

$$T=[\,0.0 \quad 0.0 \quad 0.97 \quad 0.99 \quad 0.01 \quad 0.03 \quad 1 \quad 1\,],$$

解:

这是一个约束大于变量的代数问题，这是因为由输入/输出对可得:

矢量的维数为: $P_{rxq} = P_{1x8}$, $T_{sxq} = T_{1x8}$ ，所以约束等式有 8 个。

而权值的维数为: $W_{rxm} = W_{1x1}$ ，加上一个偏差值 b ，一共只有两个变量。

当采用单层对数非线性网络求解时，求得的一个解为:

$$w = 0.079$$

$$b = -0.1616$$

此时具有误差平方和为 1.85 ，总共进行了 380 次循环修正。

当采用自适当线性网络求解本题时，在具有相同误差平方和时，只用了 40 次循环修正，即得出一解为:

$$w = 0.0204$$

$$b = 0.4537$$

同样可以采用横纵坐标来画出输入和目标输出矢量，如图 4.13 所示。并分别画出两个网络对输入矢量的响应。图中，将输入/目标矢量对用 "+" 号表示，因为两个网络具有相同的误差，所以在图中几乎看不出它们之间的差别来。

网络的输出反映的是在该网络下所达到的最小误差平方和时，对输入节点的响应。

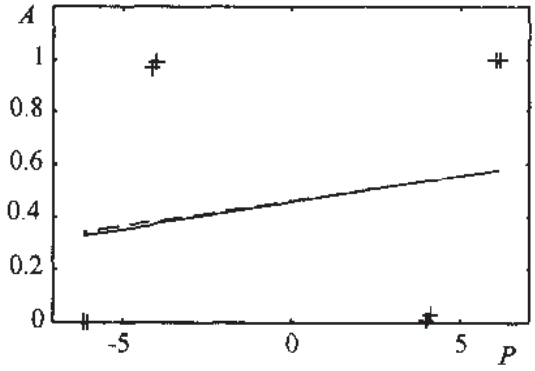


图 4.13　网络求解结果图

很明显本例题的误差不为零，而且非线性与线性网络的响应几乎是完全相同的。这可以使我们得出结论: 既使是采用非线性激活函数，在采用单层网络时，并没有体现出特殊的优越性，甚至还不如线性网络的效果。从图 4.13 中，我们还可以看出，网络的输出直线实际上起到的是一种分类功能。它把四个点分成上下两类。

通过上面两个例题可以看出，对于一般可用一层解决的问题，应当首先考虑用感知器，或自适应线性网络来解决，而不采用非线性网络，因为单层不能发挥出非线性激活函数的特长。

输入神经元数可以根据需要求解的问题和数据所表示的方式来确定。如果输入的是电压波形，那么可根据电压波形的采样点数来决定输入神经元的个数，也可以用一个神经元，使输入样本为采样的时间序列。如果输入为图像，那么则输入可以用图像的象素，也可以为经过处理后的图像特征来确定其神经元个数。总之问题确定后，输入与输出层的神经元数就随之定了。在设计中应当注意尽可能地减少网络模型的规模，以便减少网络的训练时

间。下面我们通过同样简单的单层非线性网络所形成的网络误差曲面，并通过与线性的情况的对比，来进一步了解非线性网络的特性、功能及其优缺点。

【例 4.4 】非线性误差曲面特性观察。

理解和掌握反向传播网络工作的最好方式就是观察它是如何工作的。像 W-H 学习规则一样，反向传播法也是企图通过调整每一个正比于误差的变化来修正权值而使误差函数最小化，也是属于梯度下降法。一个简单的比喻就是好像在观察一个没有惯性的弹子（代表网络）在一个粗糙不平的面上滚动。弹子总是在最速方向上滚动直到停留在一个谷的底部。（即误差的极小值）。

为了能够观察非线性误差曲面的形状以及训练时误差的走向，现仍然采用单层非线性网络来求解【例 4.2 】，首先观察误差的立体图。它的 MATLAB 编程如下：

```
Wrange = -4 : 0.4 :4;                          % 限制坐标范围
Brange = -4 : 0.4 :4;
ES = errsurf ( P, T, Wrange, Breang, 'logsig' );   % 计算误差曲面
view ( [ 60 30 ] );                            % 显示误差曲面图形
xlabel ('W'),                                  % 写 x 坐标说明
ylabel ('B'),                                  % 写 y 横坐标说明
zlabel ('Sum Squared Error'),                  % 写 z 坐标说明
title ('Error Surface Graph'),                 % 写图标题
```

误差的等高线图用下列命令绘出：

```
contour ( ES, 25, Wrange, Brange);
axis ('equal'),
```

两图分别如图 4.14 和图 4.15 所示，其中图 4.14 中的 z 坐标为误差值。两图中的黑点对应着相同的点。对照两图可以分别看出不同的网络权值所对应的误差值。

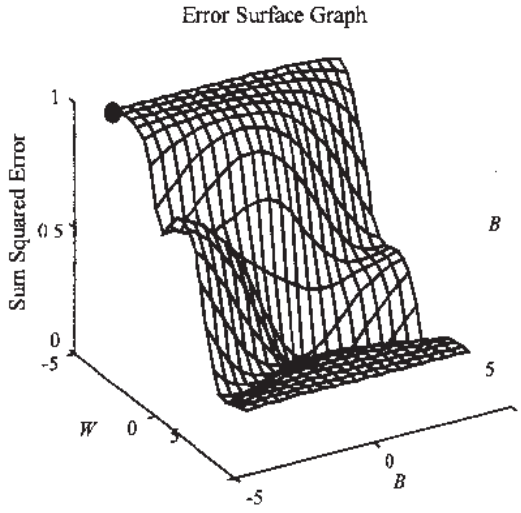


图 4.14 误差曲面图形

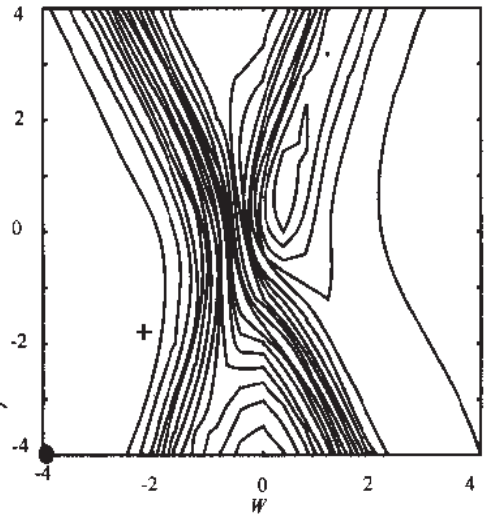
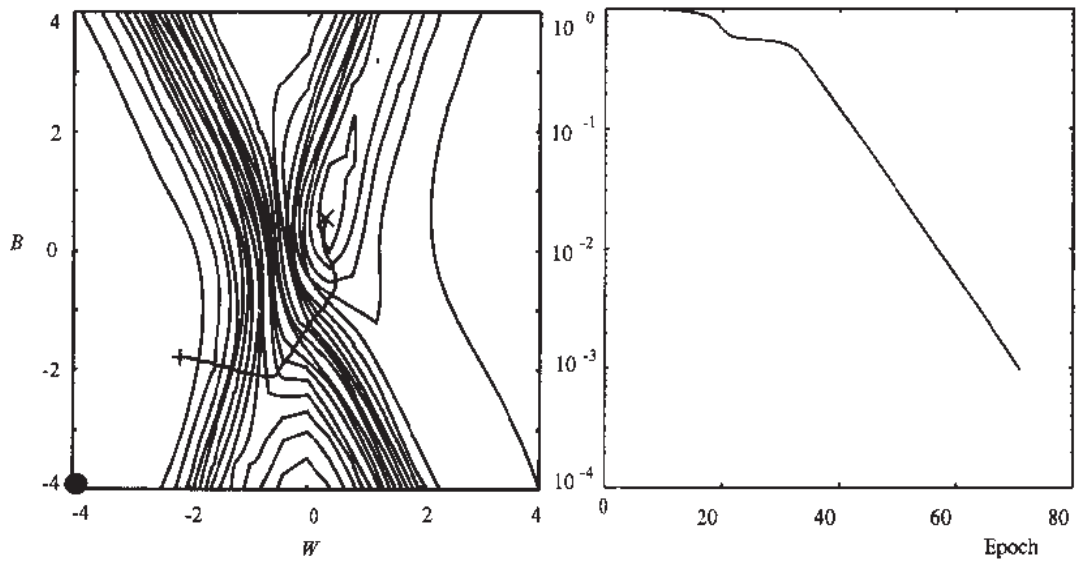


图 4.15 误差的等高线图



(a) 训练过程中参数变化过程　　(b) 训练过程中误差记录

图 4.16　训练过程中参数变化过程和误差记录

两图以不同的方式画出了误差的分布。从图 4.14 中可以看出一个较高的平原以及一个较低的扁平层出现在两个边缘。在等高线图中，平原面覆盖了左半边的大部分，而一个较大的“丘”地是在中心的底部，一个较小的“丘”在顶部。最小误差是在平面和丘之间：两角之间的交点给出了最小误差的网络权值。

训练一个网络的工作始于随机初始化的权值。本例中在误差等高线中用符号“+”画出了随机选取的误差位置。图 4.16(a)给出训练过程中误差从初始值移动到最终结果的过程记录。这个记录是由变量 errors 在整个训练过程中保存。这些误差可以用下面的命令画出：**ploterr (errors)**，如图 4.16(b)所示。注意权值变化在误差等高线图中的走向。它是与等高线呈垂直的角度穿越等高线向最小值逼近的。

4.4.2　隐含层的神经元数

网络训练精度的提高，可以通过采用一个隐含层，而增加其神经元数的方法来获得。这在结构实现上，要比增加更多的隐含层要简单得多。那么究竟选取多少个隐含层节点才合适？这在理论上并没有一个明确的规定。在具体设计时，比较实际的做法是通过对不同神经元数进行训练对比，然后适当地加上一点余量。

为了对隐含层神经元数在网络设计时所起的作用有一个比较深入的理解，下面先给出一个有代表性的实例，然后从中得出几点结论。

【例 4.5 】用两层 BP 网络实现“异或”功能。

这是一个很能说明问题的例子。我们已经知道，单层感知器不可能实现“异或”功能，即使网络实现如下的输入/输出的功能：

$$P=\begin{bmatrix}0 & 0 & 0 & 1\\ 0 & 1 & 0 & 1\end{bmatrix},\quad T=[0 \quad 1 \quad 1 \quad 0]$$

很明显，希望在输入平面上，用一条直线将目标矢量的四个点分成两部分是不可能的，但两条直线则可以解决的问题。

当然，三条直线、四条直线均能解决此问题。另外我们知道，对于一个二元输入网络来说，神经元数即为分割线数。所以，采用 2,3 和 4 等数作为隐含层神经元均能解决此问题。由此，可以画出实现“异或”功能的四种可能的 BP 网络结构图。

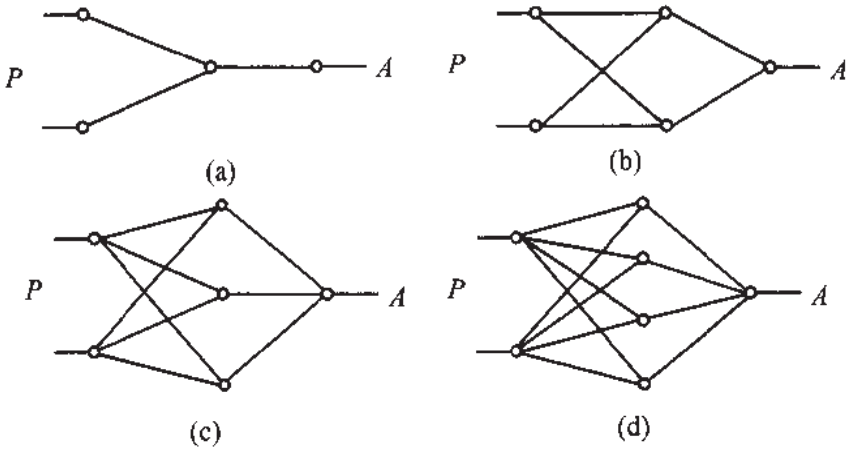


图 4.17　s1 分别为 1, 2, 3 和 4 时的求解“异或”功能的网络结构

图 4.17(a)中，输出节点与隐含层节点相同，显然该输出层是多余的，该网络也不能解决问题，因此需要增加隐含层的节点数。

更加极端的情况是多于 4 个隐含层的节点时，会有什么结果？在此例中，隐含层中神经元数为多少时为最佳？为了弄清这些问题我们针对 $s1 = 2, 3, 4, 5, 6$ 以及为 20 、 25 和 30 时对网络进行了设计。为了节省训练时间，选择误差目标为 err_goal = 0.02 ，并通过对网络训练时所需的循环次数和训练时间的情况来观察网络求解效果。各网络的训练结果如表 4.1 所示。

图 4.18 给出了表 4.1 所对应的训练误差的记录。从中可以看到其误差曲线下降的过程。撇开初始值的影响，误差在 $s1$ 较少时显得比较平，且下降速度也慢，而随着 $s1$ 的增大，下降速度增快。不过，由于隐含层节点数的增加，其初始误差值随之增加，这一点从图 4.18(b)中可以清楚地看出。

表 4.1　$s1 = 2, 3, 4, 5, 6, 20, 25$ 和 30 时的网络训练结果

$s1$	时间(秒)	循环次数
2	5.71	118
3	4.40	90
4	4.39	88
5	4.45	85
6	4.62	85
20	3.57	68
25	4.06	72
30	5.11	96

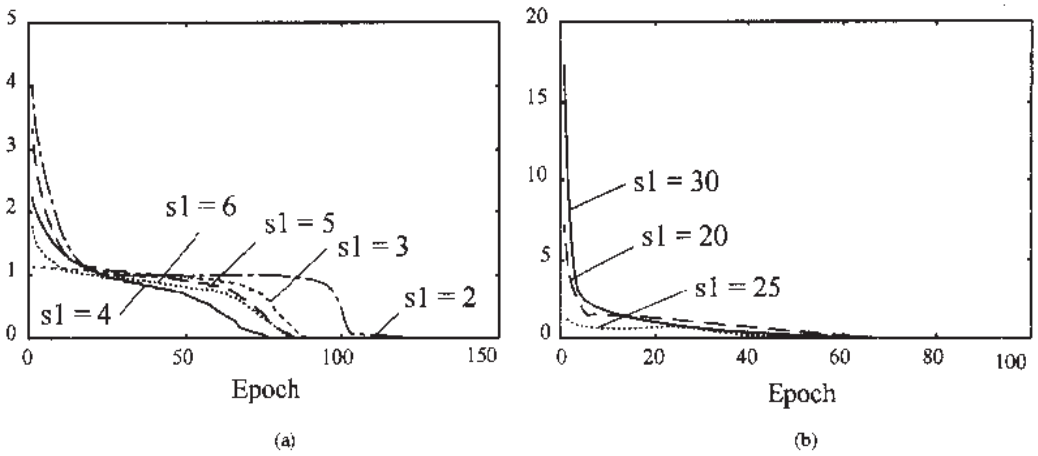


图 4.18 “异或”网络训练误差的记录

我们评价一个网络设计的好坏，首先是它的精度，再一个就是训练时间。时间包含有两层：一是循环次数，二是每一次循环中计算所花费的时间。

从表 4.1 和图 4.18 中可以看出下面几种情况：

1 ）神经元太少，网络不能很好地学习，需要训练的次数也多，训练精度也不高，如取 $s1 = 2$ 时，虽然能够解决问题，但为最糟糕的情况，因为很难达到较高的训练精度；

2 ）一般而言，网络隐含层神经元的个数 $s1$ 越多，功能越大，但当神经元数太多，一是循环次数，也就是训练时间随之增加。另外，还会产生其他问题。实际上，在进行函数逼近时，$s1$ 过大，可能导致不协调的拟合。表 4.1 中当取 $s1$ 大于 25 以后，网络解决问题的能力就开始出现问题；

3 ）当 $s1 = 3, 4, 5$ 时，其输出精度都相仿，而 $s1 = 3$ 时的训练次数最多。$s1$ 的增加能够加速误差的下降，不过从另一方面来看，随着 $s1$ 的增加，在每一次循环过程中所要进行的计算量也随着增加，所以所需要的训练时间并不一定也随之减少，这一点从 $s1 = 5$ 和 $s1 = 6$ 的情况中看得很清楚：两者所用训练时间相同，而后者比前者所用训练时间较长。对于本题，可以说选 $s1 = 3, 4, 5$，直至 15 均可，当然以 $s1 = 5$ 和 6 时为最佳。由此可以看出，网络隐含层节点数的选择是有一个较广的范围的。不过从网络实现的角度上说，倾向于选择较少的节点数。一般地讲，网络 $s1$ 的选择原则是：在能够解决问题的前提下，再加上 1 到 2 个神经元以加快误差的下降速度即可。

4.4.3 初始权值的选取

由于系统是非线性的，初始值对于学习是否达到局部最小、是否能够收敛以及训练时间的长短的关系很大。如果初始权值太大，使得加权后的输入和 n 落在了 S 型激活函数的饱和区，从而导致其导数 $f'(s)$非常小，而在计算权值修正公式中，因为 $\delta \propto f'(n)$，当 $f'(n) \to 0$ 时，则有 $\delta \to 0$。这使得 $\Delta w_{ij} \to 0$，从而使得调节过程几乎停顿下来。所以，一般总是希望经过初始加权后的每个神经元的输出值都接近于零，这样可以保证每个神经元的权值都能够在它们的 S 型激活函数变化最大之处进行调节。所以，一般取初始权值在(-1,1)之间的随机数。另外，为了防止上述现象的发生，威得罗等人在分析了两层网络是如何对一

个函数进行训练后，提出一种选定初始权值的策略：选择权值的量级为$\sqrt{S1}$，其中$s1$为第一层神经元数目。利用他们的方法可以在较少的训练次数下得到满意的训练结果。在MATLAB工具箱中可采用函数**nwlog.m**或**nwtan.m**来初始化隐含层权值W1和B1。其方法仅需要使用在第一隐含层的初始值的选取上，后面层的初始值仍然采用随机取数。

【例4.6】较好的初始值时的训练效果的观察。

以前面的【例4.1】为例，当改用下列初始值：

```
[ W1,B1 ] = nwtan(S1,R);
[ W2,B2 ] = rands(S2,S1)*0.5;
```

在这个初始值函数下，获得的一组初始值为：

```
W1 = [3.5 3.5 3.5 3.5 3.5 3.5 ];
B1 = [ -2.8562; 1.0774; -0.5880; 1.4083; 2.8722 ];
W2 = [0.2622 -0.2375 -0.4525 0.2361 -0.1718 ];
B2 = [0.1326 ];
```

重新训练网络后，相对于原先随机初始值时的6 801次的训练，仅用了454次，就达到了同样的目标误差0.02 (0.0198409)。这比标准的反向传播法快了10倍以上，明显地节省了大量的训练时间。图4.19和图4.20给出了初始权值时的输入/输出图以及网络训练结束后的误差记录。网络训练结束后的权值为：

```
W1 = [ 3.6149 3.8072 3.6115 3.3737 3.7562];
B1 = [ -2.7713; 1.1006; -0.7192; 1.5936; 2.9815];
W2 = [-0.5747 -1.1234 0.8447 -0.0467 1.3260];
B2 = -0.9116
```

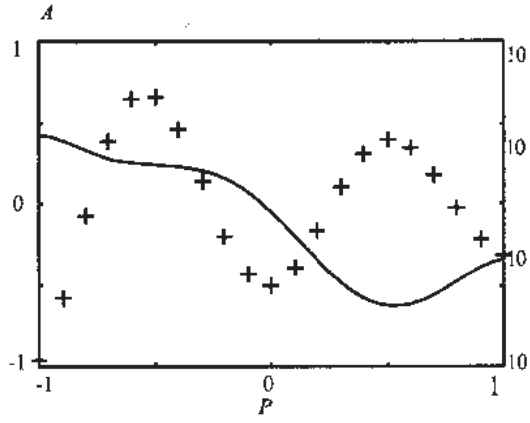


图4.19　网络训练的初始权值

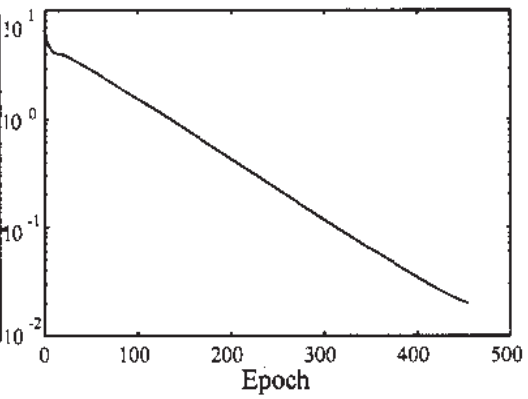


图4.20　网络训练的误差记录

4.4.4　学习速率

学习速率决定每一次循环训练中所产生的权值变化量。大的学习速率可能导致系统的不稳定；但小的学习速率导致较长的训练时间，可能收敛很慢，不过能保证网络的误差值不跳出误差表面的低谷而最终趋于最小误差值。所以在一般情况下，倾向于选取较小的学

习速率以保证系统的稳定性。学习速率的选取范围在 0.01 ~ 0.8 之间。

和初始权值的选取过程一样，在一个神经网络的设计过程中。网络要经过几个不同的学习速率的训练，通过观察每一次训练后的误差平方和 Σe^2 的下降速率来判断所选定的学习速率是否合适。如果 Σe^2 下降很快，则说明学习速率合适，若 Σe^2 出现振荡现象，则说明学习速率过大。对于每一个具体网络都存在一个合适的学习速率。但对于较复杂网络，在误差曲面的不同部位可能需要不同的学习速率。为了减少寻找学习速率的训练次数以及训练时间，比较合适的方法是采用变化的自适应学习速率，使网络的训练在不同的阶段自动设置不同学习速率的大小。这一方法将在后面讨论。

【例 4.7 】观察学习速率太大的影响。

具有非线性激活函数的各层网络对大的学习速率很敏感。对于非线性网络，不像线性网络那样可以选择一个最优学习速率。层数越多，网络训练的学习速率只能越低。较大的学习速率极易产生振荡而使其很难达到期望的目标，有的甚至发散。为了演示使用太大学习速率的后果，我们仍然采用简单的【例 4.2 】的输入/目标输出来进行观察和对比。取学习速率为:

lr = 4;

再次训练网络并观察其训练的不同权值在误差等高线图中的轨迹，如图 4.21 所示。图 4.22 给出学习速率的记录，图中纵轴表示学习速率，横轴为训练次数。从中可以看到，较大的学习速率在训练初始阶段并不成问题，且能够加速误差的减少，能比一般的训练学习速率产生更佳的误差减小率。但是随着训练的不断深入则出现了问题。由于学习速率过大，使网络每一次的修正值太大，从而导致在权值的修正过程中超出误差的最小值而永不收敛。

避免这一情况发生的办法就是减少学习速率。当然，经验在选取学习速率时是很有帮助的。再就是采用后面将要讲到的自适应学习速率，使网络根据不同的训练阶段自动调节其学习速率。

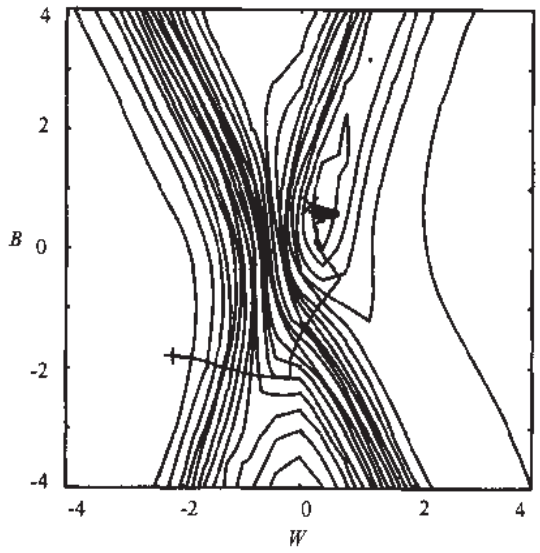


图 4.21　网络训练过程记录

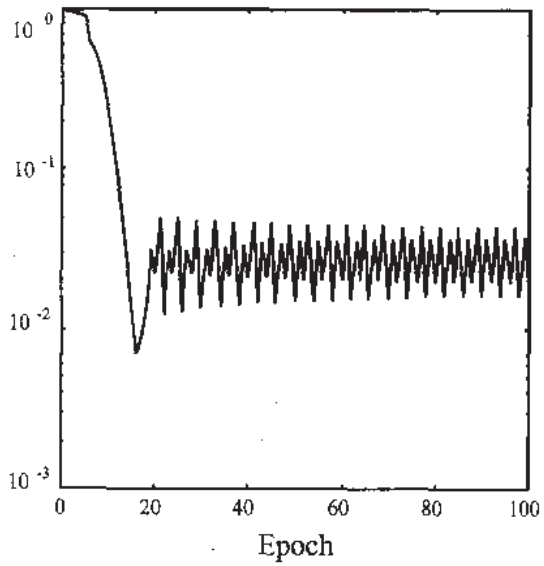


图 4.22　学习速率的记录

4.4.5 期望误差的选取

在设计网络的训练过程中，期望误差值也应当通过对比训练后确定一个合适的值，这个所谓的“合适”，是相对于所需要的隐含层的节点数来确定，因为较小的期望误差值是要靠增加隐含层的节点，以及训练时间来获得的。一般情况下，作为对比，可以同时对两个不同期望误差值的网络进行训练，最后通过综合因素的考虑来确定采用其中一个网络。

4.5 限制与不足

虽然反向传播法得到广泛的应用，但它也存在自身的限制与不足，其主要表现在于它的训练过程的不确定上。具体说明如下：

（1）需要较长的训练时间

对于一些复杂的问题，BP 算法可能要进行几小时甚至更长的时间的训练。这主要是由于学习速率太小所造成的。可采用变化的学习速率或自适应的学习速率来加以改进。

（2）完全不能训练

这主要表现在网络出现的麻痹现象上。在网络的训练过程中，当其权值调得过大，可能使得所有的或大部分神经元的加权总和 n 偏大，这使得激活函数的输入工作在 S 型转移函数的饱和区，从而导致其导数 $f'(n)$ 非常小，从而使得对网络权值的调节过程几乎停顿下来。通常为了避免这种现象的发生，一是选取较小的初始权值，二是采用较小的学习速率，但这又增加了训练时间。

（3）局部极小值

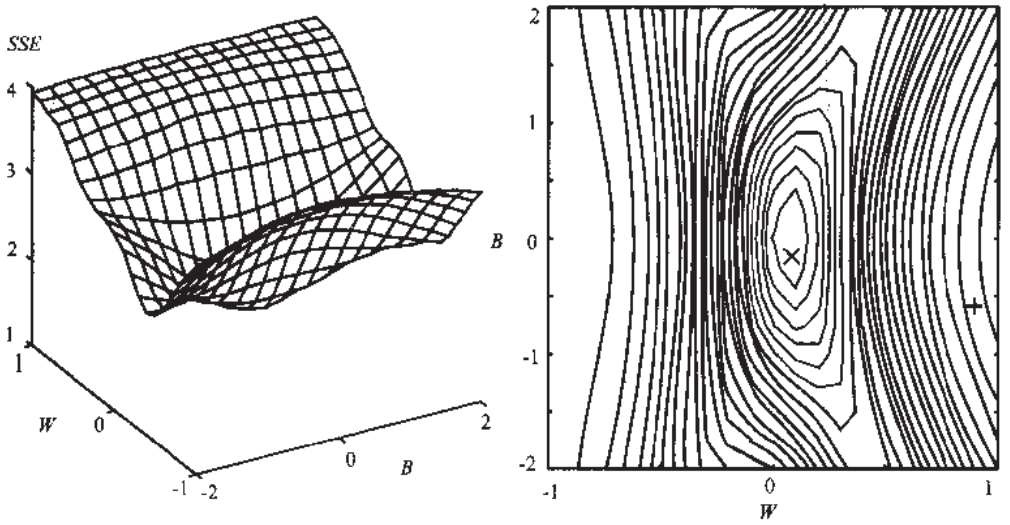
BP 算法可以使网络权值收敛到一个解，但它并不能保证所求为误差超平面的全局最小解，很可能是一个局部极小解。这是因为 BP 算法采用的是梯度下降法，训练是从某一起始点沿误差函数的斜面逐渐达到误差的最小值。对于复杂的网络，其误差函数为多维空间的曲面，就像一个碗，其碗底是最小值点。但是这个碗的表面是凹凸不平的，因而在对其训练过程中，可能陷入某一小谷区，而这一小谷区产生的是一个局部极小值。由此点向各方向变化均使误差增加，以致于使训练无法逃出这一局部极小值。

如果对训练结果不满意的话，通常可采用多层网络和较多的神经元，有可能得到更好的结果。然而，增加神经元和层数，同时增加了网络的复杂性以及训练的时间。在一定的情况下可能是不明智的。可代替的办法是选用几组不同的初始条件对网络进行训练，以从中挑选它们的最好结果。

【例 4.8】误差的局部和全局最小值的观察。

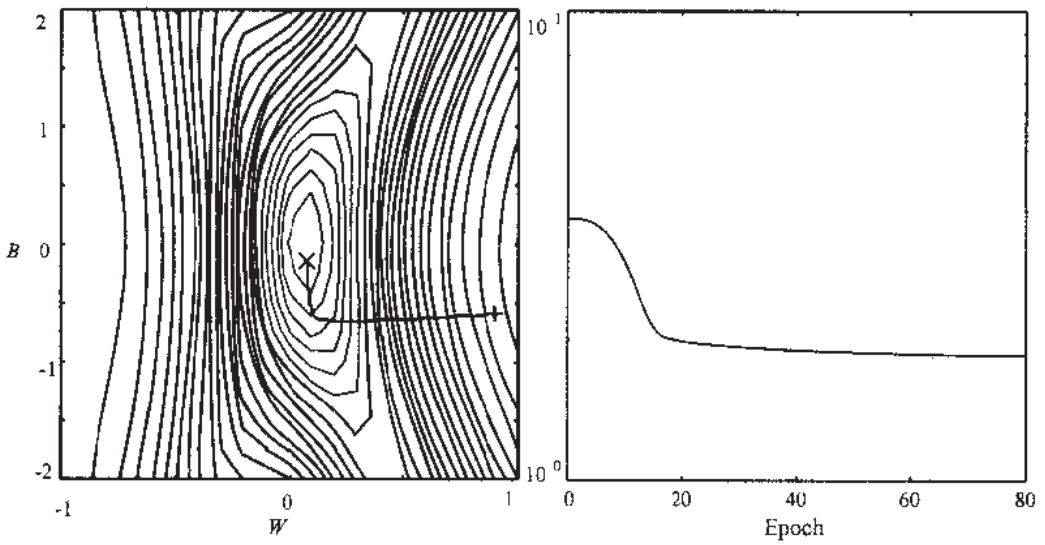
非线性网络引进了线性网络所没有的复杂性。线性网络在其误差曲面上只有唯一的一个最小值，而非线性网络可能存在几个局部极小值。这些极小值是各不相同的。好比在同一个海拔上有几个深度不同的峡谷。理想的网络应当具有最小误差的网络，即全局误差最小的网络。但所采用的基于梯度下降法的 BP 算法并不能保证做到这一点。网络有可能被陷入到误差曲面的一个较高的谷中，即局部极小值点。

这里沿用【例 4.3 】中的输入矢量和目标矢量来观察误差的局部与全局最小值的问题。其误差曲面以及误差的等高线图形如图 4.23 所示。从图中可看到这两个误差表面在图的中心有全局最小值，谷的两边均有局部极小值。这个情况在误差等高线上的右顶部和右底部能够清楚地找到。初始权值为： W0 = 0.8951; B0 = -0.5897 。等高线图形中的 “+”表示初始权值时的误差值。用其开始训练网络如图 4.24(a)轨迹中可以看到，网络沿着误差梯度直落到全局最小值。若全局最小值比期望误差值小，那么即可解决问题。图 4.24(b)给出了训练过程中的误差记录。



(a) 网络误差曲面图　　(b) 网络误差等高线图

图 4.23　网络误差曲面图与等高线图

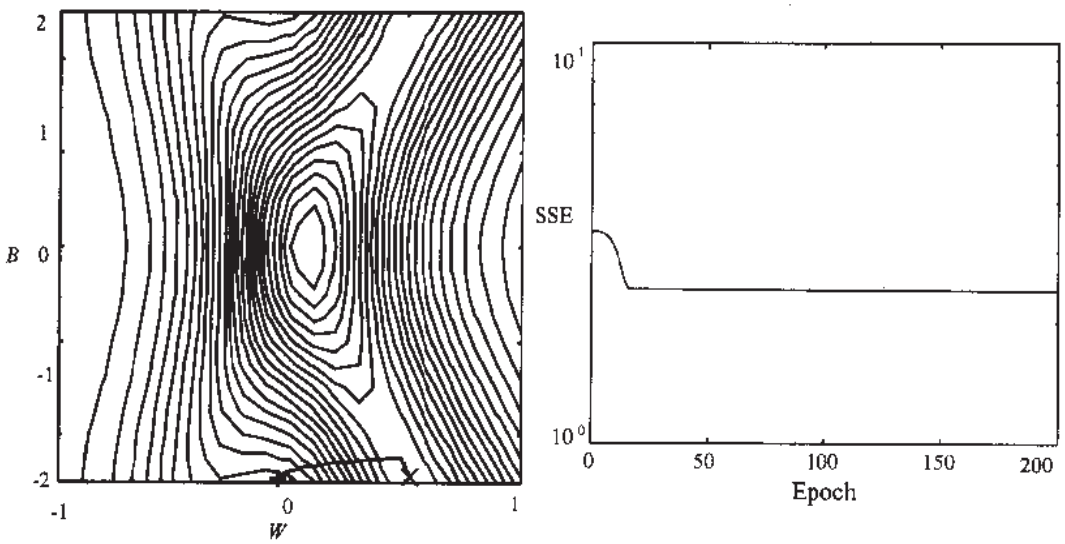


(a) 网络权值训练过程记录　　(b) 误差过程记录

图 4.24　网络权值训练过程及其误差记录

当网络选取不同的初始值，如： W0 = -0.0511;　　B0 = -2.4462 ; 则导致不同的训练结果如图 4.25 所示。

此时，网络达不到全局最小值，而是落入了谷中的一个局部极小值。由于这个谷是很浅的，所以导致网络的学习减速，最终达到局部极小值。



(a) 局部极小值训练过程记录　　(b) 误差过程记录

图 4.25　网络权值变化过程及其误差记录

我们如何来对待并解决局部极小值的问题？希望产生一个完全避免极小值的算法至今还没有可能。一个较现实的做法是，训练网络使其具有足够低的可接受的误差。这样，无论网络是否落入局部极小值，问题都得到了解决。可行方法之一就是采用比所需要的更强大的网络来解决问题。例如，增加较多的神经元层，较多的神经元数，这些都常常有助于达到期望的误差。另一个可用来跳出较浅的凹凸不平区域的局部极小值的技术是下面要介绍的附加动量法。

4.6　反向传播法的改进方法

由于在人工神经网络中，反向传播法占据了非常重要的地位，所以近十几年来，许多研究人员对其做了深入的研究，提出了很多改进的方法。主要目标是为了加快训练速度，避免陷入局部极小值和改善其它能力。本节只讨论前两种性能的改进方法的有关内容。

4.6.1　附加动量法

附加动量法使网络在修正其权值时，不仅考虑误差在梯度上的作用，而且考虑在误差曲面上变化趋势的影响，其作用如同一个低通滤波器，它允许网络忽略网络上的微小变化特性。在没有附加动量的作用下，网络可能陷入浅的局部极小值，利用附加动量的作用则有可能滑过这些极小值。

该方法是在反向传播法的基础上在每一个权值的变化上加上一项正比于前次权值变化

量的值，并根据反向传播法来产生新的权值变化。带有附加动量因子的权值调节公式为：

$$\begin{aligned}\Delta w_{ij}(k+1) &= (1-mc)\eta\delta_i p_j + mc\Delta w_{ij}(k)\\ \Delta b_i(k+1) &= (1-mc)\eta\delta_i + mc\Delta b_i(k)\end{aligned} \tag{4.11}$$

其中 k 为训练次数，mc 为动量因子，一般取 0.95 左右。

附加动量法的实质是将最后一次权值变化的影响，通过一个动量因子来传递。当动量因子取值为零时，权值的变化仅是根据梯度下降法产生；当动量因子取值为 1 时，新的权值变化则是设置为最后一次权值的变化，而依梯度法产生的变化部分则被忽略掉了。以此方式，当增加了动量项后，促使权值的调节向着误差曲面底部的平均方向变化，当网络权值进入误差曲面底部的平坦区时，δ_i 将变得很小，于是，$\Delta w_{ij}(k+1) \approx \Delta w_{ij}(k)$，从而防止了 $\Delta w_{ij} = 0$ 的出现，有助于使网络从误差曲面的局部极小值中跳出。

在 MATLAB 工具箱中，带有动量因子的权值修正法是用函数 **learnbpm.m** 来实现的。在使用此函数之前，先需将初始权值的变化置零：

dW = 0*W;　dB =0*B;

然后，权值的变化可以根据当前层的输入（比如 P），误差变化（ D = **deltalog.m, deltatan.m, deltalin.m** ），学习速率 lr，以及动量因子 mc 求得：

[dW, dB] = learnbpm (P, D, lr, mc, dW, dB);

函数 **learnbpm.m** 返回一个新的权值变化和偏差变化矢量。当要训练一个没有偏差或具有固定偏差的网络时，dB 项可以从函数中消失，这样，网络的偏差则不被修正。

根据附加动量法的设计原则，当修正的权值在误差中导致太大的增长结果时，新的权值应被取消而不被采用，并使动量作用停止下来，以使网络不进入较大误差曲面；当新的误差变化率对其旧值超过一个事先设定的最大误差变化率时，也得取消所计算的权值变化。其最大误差变化率可以是任何大于或等于 1 的值。典型的值取 1.04 。所以在进行附加动量法的训练程序设计时，必须加进条件判断以正确使用其权值修正公式。

训练程序中对采用动量法的判断条件为：

$$mc = \begin{cases} 0 & \text{当 } SSE(k) > SSE(k-1)\cdot 1.04 \\ 0.95 & \text{当 } SSE(k) < SSE(k-1) \\ mc & \text{其他} \end{cases} \tag{4.12}$$

所有这些判断过程细节均包含在 MATLAB 工具箱中的函数 **trainbpm.m** 中，它可以训练一层直至三层的带有附加动量因子的反向传播网络。以调用其他函数的同样方式调用 **trainbpm.m**，只是对变量 TP 需提供较多的参数。下面是对单层网络使用函数 **trainbpm.m** 的情形：

[W, B, epochs, errors] = trainbpm (W, B, 'F', P, T, TP);

TP 行矢量中的训练函数是训练过程中所需用到的参数，它们依次为：显示结果的频率 disp_freq，期望的误差目标 err_goal，学习速率 lr，动量因子 mc 以及最大误差变化率

err_ratio 。

为了能够观察附加动量法的作用效果，我们特地选取了一层网络的训练作为例子，并且令其偏差固定不变。在训练过程中，通过绘出权值与输出误差的函数变化图形显示出每训练一次后网络输出误差的走向，以动态变化的形式形象地让读者感受到附加动量在网络训练过程中所产生的作用。另外，在训练过程中，由此可以观察到任意初始值下的训练过程。在实际网络的设计过程中，除非网络比较简单，加上经验比较丰富，一般情况下是不直接采用网络的训练函数来训练网络，因为训练函数只有在整个训练完成之后才给出结果。对于复杂网络，在其训练过程中，往往可能会由于一些参数选取不当，如学习速率可能太大等原因，而使得训练不合适，如果设置的训练次数较大，如几千次，在这种情况下，花了很长时间获得的训练结果很可能是无效的。较好的做法是写出训练过程，并在其中加上监视作图程序，这样可以使设计者在发现由于参数设计不当而进行无用训练时及时中止训练而进行调整与改进。

【例 4.9 】采用附加动量法的反向传播网络的训练。

下面以【例 4.3 】数据为例编写出带有附加动量的反向传播法以及具有上述观察效果的 MATLAB 程序。

```
% bp9.m
%
clf reset
pausetime = 0.1
% 初始化及赋初值
P = [ -6.0 -6.1 -4.1   -4.0   5.0   -5.1 6.0 6.1 ];
T = [   0   0   0.97 0.99   0.01 0.03 1.0 1.0 ];
[ R, Q ] =size(P);   [ S, Q ] = size(T);
disp('The bias B is fixed at 3.0 and will not learn.')
Z1 = meun('Intialize Weight with:', ...
    'W0 = [ -0.9 ]; B0 = 3;', ...                  % 按给定初始值
    'Pick Values with Mouse/Arrow Keys', ...       % 用鼠标在图上任点初始值
    'Random Intial Condition [Default];            % 随机初始值（缺省情况）
disp('')
B0 = 3;
if Z1 = 1   W0 = [ -0.9 ];
elseif Z1 = =3   W0 = rand (S,R);
end
Z2 menu ('Use momentium constant of :', ...
    '0.0', ...
    '0.95 [ Default ]');
disp('')
if Z2 == 2   [ W0, dummy ] = ginput(1);
```

```
end
% 作权值一误差关系图并标注初始值
A = logsig(W0*P,B0);   E = T - A;      SSE = sumsqr(E):
h = plot (W0,SSE,'.m');
set (h,'markersize',12);
pause2(pausetime);
hold on
% 训练网络
disp_fqre = 5;  max_epoch = 500;  err_goal = 0.01;     lr = 0.05;
if Z2 ==1, momentum = 0; else momentum = 0.98; end
err_ratio = 1.04;     W = W0; B = B0;
A = logsig(W0*P,B0);   E = T - A;      SSE = sumsqr(E):
mc = 0; dW = 0*dW;                                          % 初始化动量因子
for epoch = 1 : max_epoch
if SSE < err_goal, epoch = epoch - 1; break, end
D = deltalog ( A, E ); dW = learnbpm (P,D,lr,mc,dW); TW = W + dW;
TA = logsig (TW*P,B); TE = T - TA; TSSE = sumsqr(TE);
if TSSE > SSE*err_ratio, mc = 0;                            % 判断动量效果
else if TSSE < SSE, mc = momentum; end
W = TW; A = TA; E = TE; SSE = TSSE; end
errors = [ error SSE ];
% 显示结果
plot(W,SSE,'xg');pause,hold off
ploterr(errors),pause
W,B
SSE = sumsqr(T-logsig(W*P,B))
end
```

一维误差曲线图如图 4.26 所示。可以看到在误差曲线上有两个误差最小值，一个为局部极小值在左边，右边的为全局最小值。本例中的缺省初始值为 W0 = -0.9，并表示在图的左边以“ o”表示。如果动量因子 mc 取为 0，网络则以纯梯度法进行训练，此举的结果如图 4.27 所示。其误差的变化趋势是以简单的方式“滚到”局部极小值的底部就再也停止不动了。图 4.28 给出了误差记录。

当采用附加动量法后，网络的训练则可以自动地避免陷入这个局部极小值。这个结果如图 4.29 所示。

网络的训练误差先落入局部极小值，在附加动量的作用下，继续向前产生一个正向斜率的运动，并跳出较浅的峰值，落入了全局最小值。然后，仍然在附加动量的作用下，达到一定的高度后（即产生了一个 SSE > SSE*1.04 ）自动返回，并像弹子滚动一样来回左右摆动，直至停留在最小值点上。

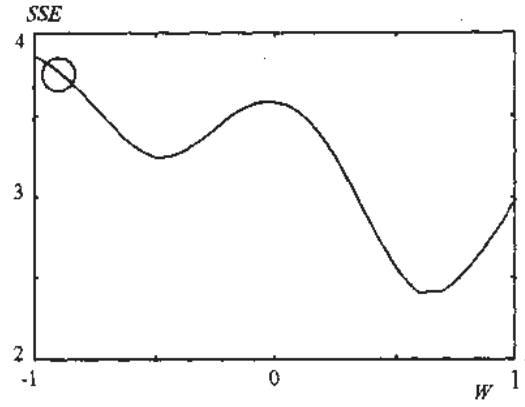


图 4.26　网络误差曲线图

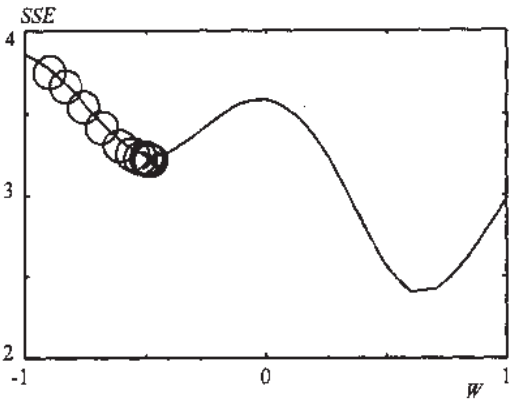


图 4.27　mc = 0 时的训练结果

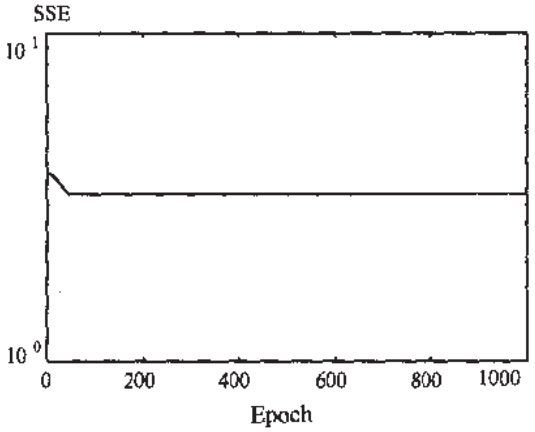


图 4.28　训练误差记录

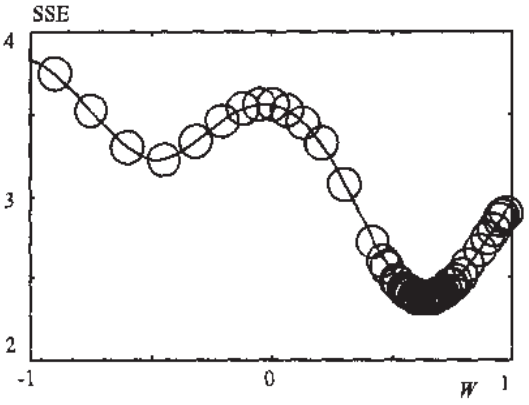


图 4.29　采用附加动量法的训练结果

读者可以执行上面的程序来观察带有附加动量法的网络训练的全过程，并可以任意选择初始权值来观察其训练结果。

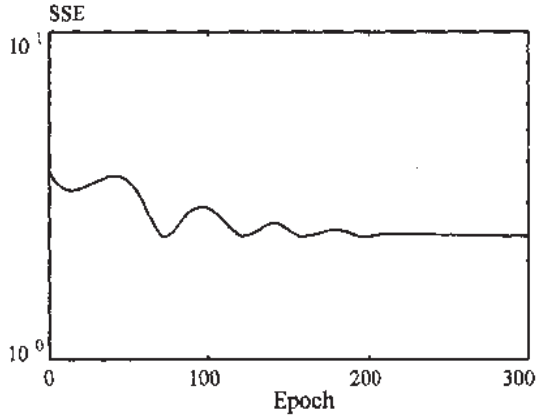


图 4.30　采用动量法时的训练误差记录

通过观察可以发现，训练参数的选择对训练效果的影响是相当大的。如学习速率太大，将导致其误差值来回振荡；学习速率太小，则导致太小动量能量，从而使其只能跳出很浅的“坑”。对于较大的“坑”或“谷”将无能为力。而从另一方面来看，其误差曲线（面）

的形状与凹凸性是由问题的本身决定的，所以每个问题都是不相同的。这必然对学习速率的选择带来了困难。一般情况下只能采用不同的学习速率进行对比尝试（典型的值取为0.05）。

通过观察带有附加动量法的网络训练的过程还使我们感到，对于这种训练必须给予足够的训练次数，以使其训练结果为最后稳定到最小值时的结果，而不是得到一个正好摆动到较大误差值时的网络权值。

此训练方法也存在有缺点。它对训练的初始值有要求，必须使其值在误差曲线上的位置所处误差下降方向与误差最小值的运动方向一致。如果初始误差点的斜率下降方向与通向最小值的方向背道而驰，则附加动量法失效。训练结果将同样落入局部极小值而不能自拔。初始值选得太靠近局部极小值时也不行，所以建议多用几个初始值先粗略训练几次以找到合适的初始位置。另外，学习速率太小也不行，比如对于本例题，选择 lr = 0.01，网络则没有足够的能量跳过低“谷”。

4.6.2 误差函数的改进

前面定义的函数是一个二次函数:

$$E=\frac{1}{2}\sum_{k}(t_k - a_k)^2$$

当 a_k 趋向 1 时，E 趋向一个常数，即处于 E 的平坦区，从而造成了不能完全训练的麻痹现象。所以当一个网络的误差曲面存在着平坦区时，可以选用别的误差函数 $f(t_k,a_k)$ 来代替 $(t_k - a_k)^2$ 的形式，只要其函数在 $a_k = t_k$ 时能够达到最小值即可。

包穆(Baum)等人于 1988 年提出一种误差函数为:

$$E=\sum_{k}[\frac{1}{2}(1+t_k)\log\frac{1+t_k}{1+a_k}+\frac{1}{2}(1- t_k)\log\frac{1- t_k}{1- a_k}]$$

该式同样满足当 $a_k = t_k$ 时，$E = 0$，不过，当 $a_k \to \pm 1$ 时，该式发散。所以能够克服麻痹现象。如果采用双曲正切函数来作为激活函数，即取:

$$f(n)=\tanh(n)=\frac{1- e^{-2n}}{1+e^{-2n}}=\frac{e^{n}- e^{-n}}{e^{n}+e^{-n}}$$

又因为

$$f^2(n)=\tanh^2(n)=\frac{e^{2n}- 2e^{n}e^{-n}+e^{-2n}}{(e^{n}+e^{-n})^2}=\frac{- 2}{(e^{n}+e^{-n})^2}$$

而

$$f'(n)=\tanh(n)=\frac{(e^{n}+e^{-n})(e^{n}+e^{-n})- (e^{n}- e^{-n})(e^{n}- e^{-n})}{(e^{n}+e^{-n})^2}$$

$$=\frac{4}{(e^{n}+e^{-n})^2}=\frac{(e^{n}+e^{-n})^2- 2}{(e^{n}+e^{-n})^2}=1- f^2(n)$$

求误差函数对输出层的变量 n 求一阶导数并同时考虑关系式：$a_k' = 1 - a_k^2$：

$$\frac{\partial E}{\partial n} = \frac{1}{2}(1+t_k)\frac{1+a_k}{1+t_k}\frac{(1+t_k)(-a_k)}{(1+a_k)^2}(1-a^2) + \frac{1}{2}(1-t_k)\frac{1-a_k}{1-t_k}\frac{(1-t_k)a_k}{(1-a_k)^2}(1-a^2)$$
$$= t_k - a_k = \delta_{ki}$$

与常规的误差函数的情况 $\delta_{ij} = f'(n)(t_k - a_k)$ 相比较，其中的 $f'(n)$ 项消失了。这样，当 n 增大，进入激活函数的平坦区，使 $f'(n) \to 0$ 时，不会产生不能完全训练的麻痹现象。但由于失去了 $f'(n)$ 对 Δw 的控制作用，过大的 Δw 又有可能导致网络过调或振荡。为了解决这个问题，1989 年，范尔曼(S. Fahlman)提出一种折中的方案，即取 $\delta_k = [f'(n) + 0.1](t_k - a_k)$，该式一方面恢复了 $f'(n)$ 的某些影响，另一方面当 $|n|$ 变大时，仍能保持 δ_k 有一定的大小，从而避免了麻痹现象的发生。

4.6.3 自适应学习速率

对于一个特定的问题，要选择适当的学习速率不是一件容易的事情。通常是凭经验或实验获取，但即使这样，对训练开始初期功效较好的学习速率，不见得对后来的训练合适。为了解决这一问题，人们自然会想到在训练过程中，自动调整学习速率。通常调节学习速率的准则是：检查权值的修正值是否真正降低了误差函数，如果确实如此，则说明所选取的学习速率值小了，可以对其增加一个量；若不是这样，而产生了过调，那么就应该减小学习速率的值。下式给出了一种自适应学习速率的调整公式：

$$\eta(k+1) = \begin{cases} 1.05\eta(k) & SSE(k+1) < SSE(k) \\ 0.7\eta(k) & SSE(k+1) > 1.04\ SSE(k) \\ \eta(k) & \text{其他} \end{cases} \tag{4.13}$$

初始学习速率 $\eta(0)$ 的选取范围可以有很大的随意性。

【例 4.10】采用自适应学习速率训练网络。

用【例 4.6】中所提出的初始条件法能够产生比【例 4.1】快得多的网络训练结果，说明采用较好的权值和偏差的初始值，能够加快训练速率。如果再加用自适应学习速率，则能够更进一步地减少训练时间。

与采用附加动量法时的判断条件相仿，当新误差超过旧误差一定的倍数时，学习速率将减少；否则其学习速率保持不变；当新误差小于旧误差时，学习速率将被增加。此方法可以保证网络总是以最大的可接受的学习速率进行训练。当一个较大的学习速率仍能够使网络稳定学习，使其误差继续下降，则增加学习速率，使其以更大的学习速率进行学习。一旦学习速率调得过大，而不能保证误差继续减少，则减少学习速率直到使其学习过程稳定为止。

MATLAB 工具箱中带有自适应学习速率进行反向传播训练的函数为:**trainbpa.m** 。它可以训练直至三层网络。使用方法为:

```
[W,B,epochs,TE] = trainbpa(W,B,'F',P,T,TP)
```

在行矢量 TP 中的参数依次为：显示频率 disp_freq，最大训练次数 max_epoch，目标误差 err_goal，初始学习速率 lr，递增乘因子 lr_inc，递减乘因子 lr_dec 和误差速率 err_ratio。

函数在训练结束后返回最终权值 W 和偏差 B，训练网络所用次数 epochs 和训练误差记录 TE。TE 是两个行矢量，第一行为网络的训练误差，第二行为所对应的学习速率。

同其他训练函数的调用方法一样，这个训练过程函数的应用非常简单，整个网络的设计训练过程只需要以下几行程序：

```
disp_freq = 10;
max_epoch = 2000;
err_goal =0.02;
lr = 0.02;
lr_inc = 1.05;
lr_idec = 0.7;
err_ratio = 1.04;
TP = [disp_freq max_epoch err_goal lr lr_inc lr_dec err_ratio];
[W1,B1,W2,B2,epochs,TE] = trainbpa(W1,B1,'tansig',W2,B2,'purelin',P,T,TP];
```

仅训练了 120 次就达到了目标误差 0.02 的目的。这个结果只是采用较好初始条件情况下的 454 次训练的近四分之一倍。它同时是固定学习速率为 0.01 时的五十分之一(【例 4.1】中的训练次数为 6801)。

图 4.31 给出了学习速率在训练过程中的记录。误差平方和的记录在图 4.32 中。

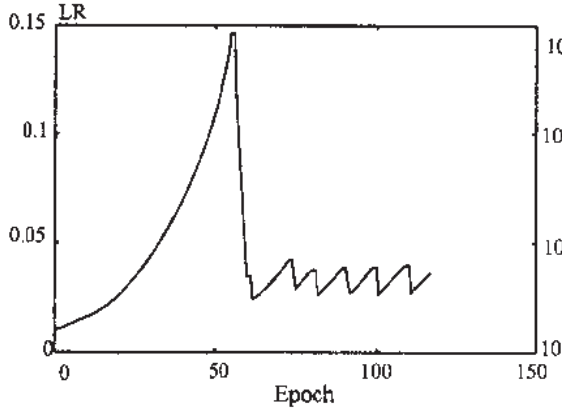


图 4.31 训练中的学习速率

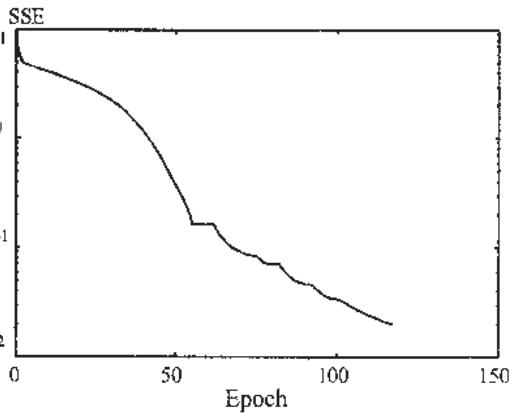


图 4.32 训练中的误差记录

从中可以看出：在训练的初始阶段，学习速率是以直线形式上升。当接近误差最小值时，学习速率又自动地下降。注意误差曲面有少许的波动。

可以将动量法和自适应学习速率结合起来以利用两方面的优点。这个技术已编入了函数 **trainbpx.m** 之中。这个函数的调用和其他函数一样，只是需要更多的初始参数而已：

```
TP = [disp_freq max_epoch err_goal lr lr_inc lr_dec mom_const err_ratio];
[ W,B,epochs,[errors;lr]] = trainbpx ( W,B,F,P,T,TP )
```

4.6.4 双极性 S 型压缩函数法

另外一种改进训练性能的简单方法为双极性S型压缩函数反向传播法。一般对数S型压缩函数的输出动态范围为(0,1)，这不是最佳的，因为从权值调节公式可知，权值的变化也正比于前一层的输出，而因其中一半是趋向0的一边，这必然引起权值调节量的减少或不调节，从而加长了训练时间。为了解决这个问题，可将输入范围变为 1/2，同时也使S型函数置偏的输出范围也变为 ± 1/2，即由

$$a_{原}=\frac{1}{1+e^{-n}}$$

改为

$$a_{新}=-\frac{1}{2}+\frac{1}{1+e^{-n}}$$

要注意的是，当利用反向传播法时，因为

$$a^2=(-\frac{1}{2}+\frac{1}{1+e^{-n}})^2=\frac{1}{4}-\frac{1}{1+e^{-n}}+(\frac{1}{1+e^{-n}})^2,$$

激活函数的一阶导数此时为:

$$a=\frac{-e^{-n}}{(1+e^{-n})^2}=\frac{-(-1+1+e^{-n})}{(1+e^{-n})^2}=\frac{1}{(1+e^{-n})^2}-\frac{1}{1+e^{-n}}$$

$$=\frac{1}{4}-[\frac{1}{4}-\frac{1}{1+e^{-n}}+(\frac{1}{1+e^{-n}})^2]=\frac{1}{4}-a^2$$

实验证明，采用此方法，收敛时间平均可以减少 30~50%。当然，若采用此方法来训练网络，其训练程序需要设计者自己编写一部分。

4.7 本章小结

1 ）反向传播法可以用来训练具有可微激活函数的多层前向网络以进行函数逼近，模式分类等工作;

2 ）反向传播网络的结构不完全受所要解决的问题所限制。网络的输入神经元数目及输出层神经元的数目是由问题的要求所决定的，而输入和输出层之间的隐含层数以及每层的神经元数是由设计者来决定的;

3 ）已证明，两层 S 型/线性网络，如果 S 型层有足够的神经元，则能够训练出任意输入和输出之间的有理函数关系;

4 ）反向传播法沿着误差表面的梯度下降，使网络误差最小，网络有可能陷入局部极小值;

5 ）附加动量法使反向传播减少了网络在误差表面陷入低谷的可能性并有助于减少训练时间;

6 ）太大的学习速率导致学习的不稳定，太小值又导致极长的训练时间。自适应学习速率通过在保证稳定训练的前提下，达到了合理的高速率，可以减少训练时间；

7 ） 80%~90% 的实际应用都是采用反向传播网络的。改进技术可以用来使反向传播法更加容易实现并需要更少的训练时间。

习 题

〖4.1〗运用将附加动量法和自适应学习速率相结合的技术的算法函数 **trainbpx.m** 训练【例 4.1 】并与其他方法的训练结果相比较。

〖4.2〗根据所学过的 BP 网络设计及改进方案设计实现模糊控制规则为 $T = 1/2*(e + ec)$ 的模糊神经网络控制器。设计要求为:

①输人、输出矢量及问题的阐述;

②给出网络结构;

③学习方法（包括所采用的改进方法）;

④初始化及必要的参数选取;

⑤最后的结果，循环次数，训练时间，其中着重讨论:

a)不同隐含层 S1 时的收敛速度与误差精度的对比分析，

b)当 S1 设置为较好的情况下，在训练过程中取始终不变的学习速率 lr 值时，对 lr 值为不同值时的训练时间，包括稳定性进行观察比较，

c)当采用自适应值学习速率时，与单一固定的学习速率 lr 中最好的情况进行对比训练的观察，

d)给出结论或体会。

⑥验证，采用插值法选取多于训练时的输人，对所设计的网络进行验证，给出验证的 A 与 T 值。