第 3 章

线性神经网络

线性神经网络是由一个或者多个线性神经元组成的网络，它和感知器的区别在于每个线性神经元的传递函数都是线性函数，输出是一段区间值，而感知器的传递函数是符号函数，输出为二值量－1 或 1。线性神经网络主要应用领域有：函数拟合与逼近、预测、模式识别等。

线性神经网络的输出表达式为

$$y = \text{purelin}(w\boldsymbol{p} + b)$$

其中，$\boldsymbol{p}$ 是输出向量；y 是输出值；w 为权值；b 是阈值或者偏置；purelin 为线性传递函数，为过零点斜率为 1 的线性函数。

线性神经网络的学习规则采用的是 LMS(Least Mean Square，最小均方差)算法，这种学习规则的基本思想是：寻找最佳的权值和阈值，使得各个神经元的输出均方误差最小。

3.1 线性神经元模型及结构

3.1.1 神经元模型

线性神经元模型如图 3-1 所示，其中，R 为输入向量元素的数目。

从网络结构上看，和感知器神经网络结构类似，不同的是神经元的传递函数是线性传递函数 purelin，如图 3-2 所示。

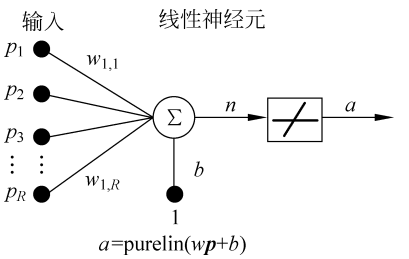

图 3-1 线性神经元模型

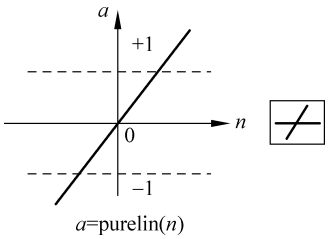

图 3-2 线性传递函数

由于线性神经网络中神经元的传递函数为线性函数，其输入/输出之间是简单的比例关系。单个线性神经元可以通过下式计算：

$$a=\text{purelin}(n)=\text{purelin}(w\boldsymbol{p}+b)=w\boldsymbol{p}+b$$

3.1.2　线性神经网络结构

图 3-3 中给出一个具有 R 个输出 S 个神经元的单层线性神经元网络的形式，输出向量数目和神经元数目相等，也是 S 个。权值矩阵为 $\boldsymbol{W}$，阈值为 b，这种网络也称为 Madaline 网络。

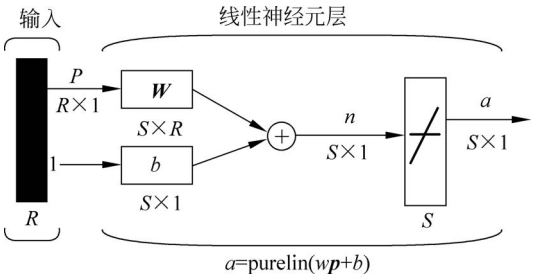

图 3-3　线性神经网络

这里介绍的单层线性神经网络结构，和多层线性神经网络一样有用。因为对于每一个多层线性神经网络而言，都可以设计出一个性能相当的单层线性神经网络。

3.2　LMS 学习算法

LMS 学习算法是基于负梯度下降的原则来减小网络的训练误差。最小均方误差学习算法也属于监督类学习算法。令 $\boldsymbol{p}_k=(p_1(k),p_2(k),\cdots,p_R(k))$ 表示网络的输入向量，$\boldsymbol{d}_k=(d_1(k),d_2(k),\cdots,d_s(k))$ 表示网络的期望输出向量，$\boldsymbol{y}_k=(y_1(k),y_2(k),\cdots,y_s(k))$ 表示网络的实际输出向量。其中，$k=1,2,\cdots,m$，表示输入向量与对应的期望输出向量样本对的数量，计算出实际输出向量与相应的期望输出向量的误差，并且依据误差来调整网络的权值和阈值，使该误差逐渐减小。LMS 学习规则就是要减小这些误差平方和的均值，定义如下。

$$\text{mse}=\frac{1}{m}\sum_{k=1}^{m}e^2(k)=\frac{1}{m}\sum_{k=1}^{m}(d(k)-y(k))^2$$

从最小均方误差的定义可以看出，它的性能指标是一个二次方程，所以它要么具有全局最小值，要么没有最小值，而选择什么样的输入向量恰恰会决定网络的性能指标会有什么样的最小值。

如果考虑第 k 次循环时训练误差的平方对网络权值和阈值的二阶偏微分，会得到公式：

$$\frac{\partial e^2(k)}{\partial w_{ij}}=2e(k)\frac{\partial e(k)}{\partial w_{ij}}$$

其中，$j=1,2,\cdots,S$。

$$\frac{\partial e^2(k)}{\partial b}=2e(k)\frac{\partial e(k)}{\partial b}$$

再计算此时的训练误差对网络权值和阈值的一阶偏微分：

$$\frac{\partial e(k)}{\partial w_{ij}}=\frac{\partial[d(k)-y(k)]}{\partial w_{ij}}=\frac{\partial e}{\partial w_{ij}}[d(k)-(\boldsymbol{W}\boldsymbol{p}(k)+b)]$$

或者

$$\frac{\partial e(k)}{\partial w_{ij}}=\frac{\partial e}{\partial w_{ij}}\left[d(k)-\left(\sum_{i=1}^{R}w_{ij}\boldsymbol{p}_i(k)+b\right)\right],\quad j=1,2,\cdots,S$$

其中，$p_i(k)$表示第 k 次循环中的第 i 个输入向量。则有：

$$\frac{\partial e(k)}{\partial w_{ij}}=-\boldsymbol{p}_i(k)$$

$$\frac{\partial e(k)}{\partial b}=-1$$

根据负梯度下降的原则，网络权值和阈值的改变量应该是 $2\eta e(k)\boldsymbol{p}(k)$和 $2\eta e(k)$。所以网络权值和阈值修正公式如下。

$$w(k+1)=w(k)+2\eta e(k)\boldsymbol{p}^{\mathrm{T}}(k)$$

$$b(k+1)=b(k)+2\eta e(k)$$

式中：η——学习率。

当 η 取较大值时，可以加快网络的训练速度，但是如果 η 的值太大，会导致网络稳定性的降低和训练误差的增加。所以，为了保证网络进行稳定的训练，学习率 η 的值必须选择一个合适的值。

重复以上求解过程，直到达到预定的精度，算法结束。

3.3 LMS 学习率的选择

如果在线性神经网络中，学习率参数 η 的选择非常重要，直接影响了神经网络的性能和收敛性。

3.3.1 稳定收敛的学习率

如前所述，η 越小，算法的运行时间就越长，算法也就记忆了更多过去的数据。因此，η 的倒数反映了 LMS 算法的记忆容量大小。

η 往往需要根据经验选择，且与输入向量的统计特性有关。尽管我们小心翼翼地选择学习率的值，仍有可能选择了一个过大的值，使算法无法稳定收敛。

1996 年，Hayjin 证明，只要学习率 η 满足下式，LMS 算法就是按方差收敛的。

$$0<\eta<\frac{2}{\lambda_{\max}}$$

其中，$\lambda_{\max}$ 是输入向量 $\boldsymbol{x}(n)$组成的自相关矩阵 $\boldsymbol{R}$ 的最大特征值。由于 $\lambda_{\max}$ 常常不可知，因此往往使用自相关矩阵 $\boldsymbol{R}$ 的迹来代替。按定义，矩阵的迹是矩阵主对角线元素之和：

$$\mathrm{tr}(\boldsymbol{R})=\sum_{i=1}^{O}\boldsymbol{R}(i,i)$$

同时，矩阵的迹又等于矩阵所有特征值之和，因此一般有 $\mathrm{tr}(\boldsymbol{R})>\lambda_{\max}$。只要取

$$0<\eta<\frac{2}{\mathrm{tr}(\boldsymbol{R})}<\frac{2}{\lambda_{\max}}$$

即可满足条件。按定义，自相关矩阵的主对角线元素就是各输入向量的均方值。因此公式又可以写成：

$$0<\eta<\frac{2}{\text{向量均方值之和}}$$

3.3.2　学习率逐渐下降

在感知器学习算法中曾提到，学习率 η 随着学习的进行逐渐下降比始终不变更加合理。在学习的初期，用比较大的学习率保证收敛速率，随着迭代次数增加，减小学习率以保证严谨，确保收敛。一种可能的学习率下降方案为

$$\eta=\frac{\eta_0}{n}$$

在这种方法中，学习率会随着迭代次数的增加较快下降。另一种方法是指数式下降：

$$\eta=c^n\eta_0$$

c 是一个接近 1 而小于 1 的常数。Darken 与 Moody 于 1992 年提出搜索-收敛(Search-then-Converge Schedule)方案，公式为

$$\eta=\frac{\eta_0}{1+\left(\frac{n}{\tau}\right)}$$

其中，η_0 与 τ 均为常量。当迭代次数较小时，学习率 $\eta\approx\eta_0$，随着迭代次数增加，学习率逐渐下降，公式近似于：

$$\eta=\frac{\eta_0}{0+\left(\frac{n}{\tau}\right)}=\frac{\tau\eta_0}{n}$$

LMS 算法的一个缺点是，它对输入向量自相关矩阵 $\boldsymbol{R}$ 的条件数敏感。当一个矩阵的条件数比较大时，矩阵就称为病态矩阵，如果这种矩阵中的元素做微小改变，可能会引起相应线性方程的解的很大变化。

3.4　线性神经网络的构建

3.4.1　生成线性神经元

构造一个如图 3-4 所示的具有两个输入端的单线性神经元网络。其权值矩阵 $\boldsymbol{w}$ 是一个行向量，网络输出为

$$a=\mathrm{purelin}(n)=\mathrm{purelin}(w\boldsymbol{p}+b)=w\boldsymbol{p}+b \tag{3-1}$$

或者

$$a=w_{1,1}\boldsymbol{p}_1+w_{1,2}\boldsymbol{p}_2+b \tag{3-2}$$

和感知器网络一样，线性神经网络也具有一个分界线，由输入向量决定，即 $n=0$ 时，方程 $w\boldsymbol{p}+b=0$，其分类示意图如图 3-5 所示。

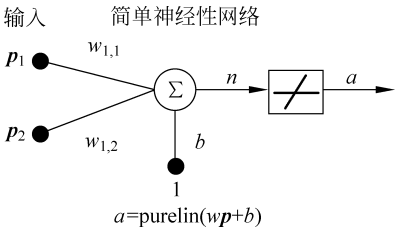

图 3-4　二输入单神经元线性网络结构

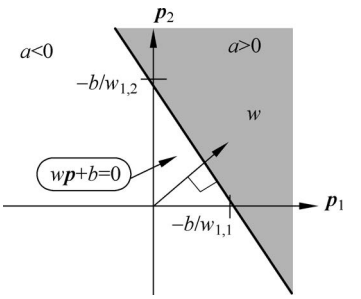

图 3-5　二输入线性神经网络分类示意图

输入向量在分界线右上部时，输出大于 0；输入向量在左下部时，输出则小于 0。这样，线性神经网络就可以用来研究分类问题。然而应用线性神经网络进行分类的前提是：进行分类的问题是线性可分的，这方面和感知器网络的局限性是相同的。

【例 3-1】 应用 newlin()函数设计一个双输入单输出线性神经网络。

```
>> clear all;
net = newlin([ - 1,1; - 1,1],1);
W = net.IW{1,1}                          % 网络权值默认为 0
W =
     0     0
>> b = net.b{1}
b =
     0
```

也可以给定权值与阈值，如：

```
>> net.b{1} = [ - 9];
>> b = net.b{1}
b =
    - 9
>> % 对输入向量 p 应用函数 sim()进行仿真，并输出仿真结果
>> p = [3;5];
>> a = sim(net,p)
a =
    38
```

从上述可见，应用 newlin()函数可以构建一个线性神经网络并随意调整权值和阈值，还可以利用 sim()函数进行仿真。

3.4.2　线性滤波器

首先需要了解一下应用于线性神经网络中的触发延迟线，如图 3-6 所示。从左端接入输入向量，通过 TDL，发生(N－1)延迟。TDL 的输出是一个 N 维向量，相当于当前输入向量的前一时刻的输入信号。

如果在线性神经网络中应用了触发延迟线，则将产生如图 3-7 所示的线性滤波器。线性滤波器的输出为

$$a(k)=\text{purelin}(w\boldsymbol{p}+b)=\sum_{i=1}^{R}w_{1,i}a(k-i+1)+b \tag{3-3}$$

这样的网络可以应用于信号处理滤波，下面举例说明。

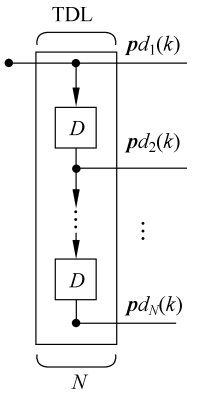

图 3-6　触发延迟线

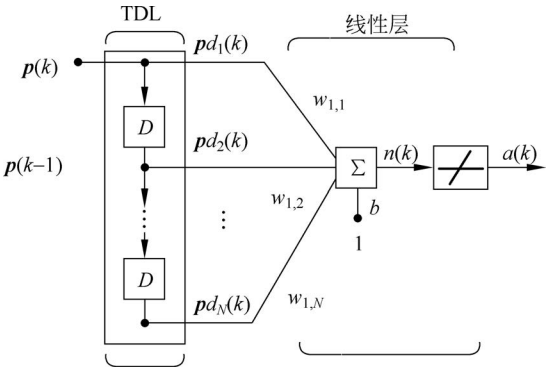

图 3-7　线性滤波器

【例 3-2】 假设输入向量 $\boldsymbol{p}$，期望输出向量 $\boldsymbol{T}$，以及初始输入延迟 P_1。

```
>> clear all;
P = {1 2 1 3 3 2};
Pi = {1 3};
T = {5.0 6.1 4.0 6.0 6.9 8.0};
 % 应用 newlind()构建一个网络以满足上面的输入/输出关系和延迟条件
net = newlind(P,T,Pi);
 % 验证下一个网络的输出
Y = sim(net,P,Pi)
```

运行程序，输出如下。

```
Y =
    [4.9824]    [6.0851]    [4.0189]    [6.0054]    [6.8959]    [8.0122]
```

由此可见，网络输出和期望输出有一定的差距，但它们是合理的。任何情况下，均方误差都是最小的。

3.4.3　自适应线性滤波

自适应滤波网络的生成可以采用两种方式：一种是通过调用 newlin()函数直接生成带有延迟链的自适应滤波网络；另一种则是首先利用 newlin()函数生成不带延迟链的线性网络，然后通过网络重定义将延迟链加入预先生成的线性网络中。图 3-8 为一个单神经元自适应滤波网络的结构示意图。

该自适应滤波网络可以采用如下两种方式来生成(假设网络输入范围为[0,10])。

方式一：

```
net = newlin([0, 10], 1,[0 1 2]);
```

方式二：

```
net = newlin([0, 10], 1);
net.inputWeights{1,1}.delays = [0 1 2];
```

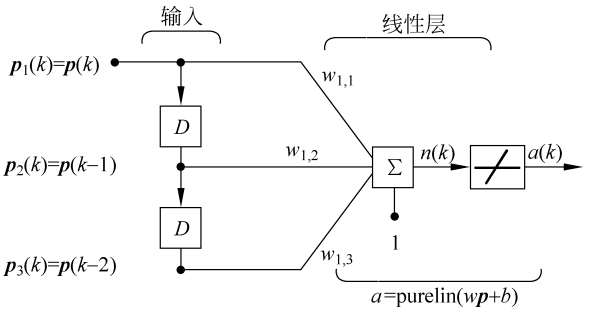

图 3-8　单神经元自适应滤波网络

其中，[0 1 2]中各元素分别表示自适应滤波网络各维输入所对应的延迟量。下面对所生成的自适应滤波网络分别进行初始化、仿真和训练。

自适应滤波网络的初始化与一般的线性神经网络基本相同，只是在初始化网络权值和阈值的同时，自适应滤波网络还要对延迟输入的初始值进行设置。

3.5　线性神经网络的训练

自适应线性元件的网络训练过程可以归纳为以下三个步骤。

（1）表达：计算训练的输出向量 $\boldsymbol{A}=\boldsymbol{W}*\boldsymbol{P}+\boldsymbol{B}$，以及与期望输出之间的误差 $\boldsymbol{E}=\boldsymbol{T}-\boldsymbol{A}$。

（2）检查：将网络输出误差的平方和与期望误差相比较，如果其值小于期望误差，或训练已达到事先设定的最大训练次数，则停止训练，否则继续。

（3）学习：采用 W-H 学习规则计算新的权值和偏差，并返回(1)。

每进行一次上述三个步骤，被认为是完成一个训练循环次数。

如果网络训练获得成功，那么将一个不在训练中的输入向量输入网络中时，网络的输出趋于与其相关联的输出向量。这个特性被称为泛化，这在函数逼近以及输入向量分类的应用中是相当有用的。

如果经过训练，网络仍不能达到期望目标，可以有两种选择：或检查一下所要解决的问题，是否适用于线性网络；或对网络进行进一步的训练。

虽然只适用于线性网络，W-H 学习规则仍然是重要的，因为它展现了梯度下降法是如何来训练一个网络的，此概念后来发展成反向传播法，使之可以训练多层非线性网络。

3.6　线性神经网络与感知器的对比

不同神经网络有不同的特点和适用领域。尽管感知器与线性神经网络在结构和学习算法上都没有什么太大的差别，甚至是大同小异，但仍能从细小的差别上找到其功能的不同点。它们的差别主要表现在以下两点。

1. 网络传输函数

LMS 算法将梯度下降法用于训练线性神经网络，这个思想后来发展成反向传播法，

具备可以训练多层非线性网络的能力。

感知器与线性神经网络在结构上非常相似，唯一的区别在于传输函数：感知器传输函数是一个二值阈值元件，而线性神经网络的传输函数是线性的，这就决定了感知器只能做简单的分类，而线性神经网络还可以实现拟合或逼近。在应用中也确实如此，线性神经网络可以通过对网络的训练，得出线性逼近关系，这一特点可以在系统辨识或模式联想中得到应用。

2. 学习算法

学习算法要与网络的结构特点相适应。感知器的学习算法是最早提出的可收敛的算法，LMS 算法与它关系密切，形成上也非常类似，它们都采用了自适应的思想。

在计算上，从表面看 LMS 算法似乎与感知器学习算法没什么两样。这里需要注意一个区别：LMS 算法得到的分类边界往往处于两类模式的正中间，而感知器学习算法在刚刚能正确分类的位置就停下来了，从而使分类边界离一些模式距离过近，使系统对误差更敏感。这一区别与两种神经网络的不同传输函数有关。

3.7　线性神经网络函数

MATLAB 神经网络工具箱中为线性神经网络提供了大量的函数，它们可分别用于线性网络的设计、创建、分析、训练及仿真等，下面给予介绍。

3.7.1　创建函数

在 MATLAB 神经网络工具箱中，提供了三个函数用于线性神经网络的创建。

1. newlin()函数

newlin()函数用于创建一个线性层，在 MATLAB 中推荐使用 linearlayer()函数。线性层是一个单独的层次，它的权函数为 dotprod()，输入函数为 netsum()，传递函数为 purelin()。线性层一般用作信号处理和预测中的自适应滤波器。函数的调用格式为

```
net = newlin(PR,S,ID,LR)
```

其中，PR 为由 R 个输入元素的最大值和最小值组成的 $R\times 2$ 维矩阵；S 为输出向量的数目；ID 为输入延迟向量，默认为[0]；LR 为学习速率，默认为 0.01；net 函数返回值，一个新的线性层。

```
net = newlin
```

表示在一个对话框中创建一个新的网络

【例 3-3】 应用 newlin()创建线性网络。

```
>> clear all;
P1 = {0 -1 1 1 0 -1 1 0 0 1};
T1 = {0 -1 0 2 1 -1 0 1 0 1};
net = newlin(P1,T1,[0 1],0.01);
Y1 = net(P1)
```

```
P2 = {1 0 -1 -1 1 1 1 0 -1};
T2 = {2 1 01 -2 0 2 2 1 0};
net = newlin(P2,T2,[0 1],0.01);
Y2 = net(P2)
net = init(net);
P3 = [P1 P2];
T3 = [T1 T2];
net.trainParam.epochs = 200;
net.trainParam.goal = 0.01;
net = train(net,P3,T3);
Y3 = net([P1 P2])
```

运行程序,输出如下,网络训练过程如图 3-9 所示。

```
Y1 =
    [0]    [0]    [0]    [0]    [0]    [0]    [0]    [0]    [0]    [0]
Y2 =
    [0]    [0]    [0]    [0]    [0]    [0]    [0]    [0]    [0]
Y3 =
 1～8 列
 [0.2061] [-0.5191] [-0.0458] [1.9084] [1.1832] [-0.5191] [-0.0458] [1.1832]
 9～15 列
 [0.2061]  [0.9313]  [1.9084]  [1.1832]  [-0.5191]  [-1.4962]  [-0.0458]
16～19 列
 [1.9084]  [1.9084]  [1.1832]  [-0.5191]
```

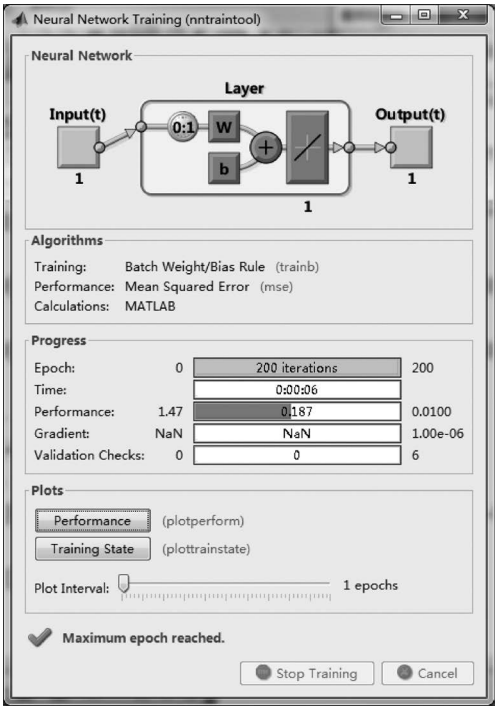

图 3-9　网络训练过程图

2. linearlayer()函数

在 MATLAB R2014b 后的版本的神经网络工具箱中，此函数用来替代 newlin()函数。该函数用于设计静态或动态的线性系统。函数的调用格式为

```
linearlayer(inputDelays,widrowHoffLR)
```

其中，inputDelays 为输入延迟的行向量，默认为 1：2；widrowHoffLR 为学习速率，默认为 0.01。

【例 3-4】 使用默认参数，利用 linearlayer()函数训练一个神经网络。

```
>> clear all;
x = {0 -1 1 1 0 -1 1 0 0 1};
t = {0 -1 0 2 1 -1 0 1 0 1};
net = linearlayer(1:2,0.01);
[Xs,Xi,Ai,Ts] = preparets(net,x,t);
net = train(net,Xs,Ts,Xi,Ai);
view(net)
Y = net(Xs,Xi);
perf = perform(net,Ts,Y)
```

运行程序，输出如下，网络训练过程如图 3-10 所示，网络训练结果如图 3-11 所示。

```
perf =
    0.2396
```

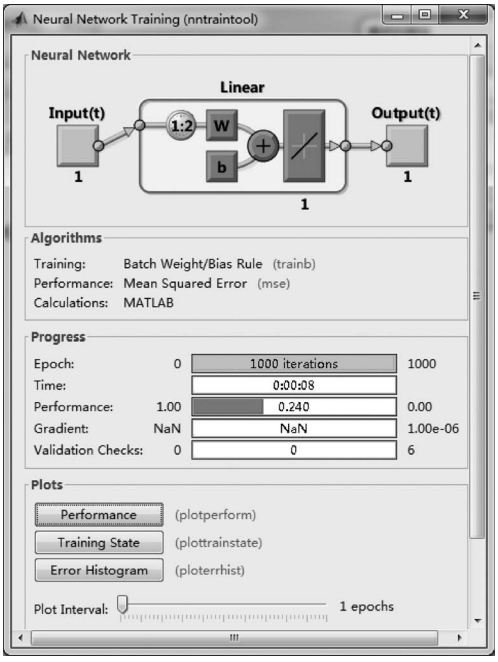

图 3-10　网络训练过程

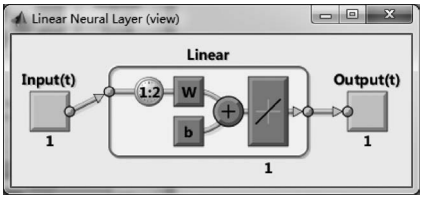

图 3-11　网络创建线性神经网络层

3. newlind()函数

该函数可以设计一个线性层，它通过输入向量和目标向量来计算线性层的权值和阈

值。函数的调用格式为

```
net = newlind(P,T,Pi)
```

其中，$\boldsymbol{P}$ 为 Q 组输入向量组成的 $R\times Q$ 维矩阵；$\boldsymbol{T}$ 为 Q 组目标分类向量组成的 $S\times Q$ 维矩阵；$\boldsymbol{P}_i$ 为初始输入延迟状态的 ID 个单元阵列，每个元素 $\boldsymbol{P}_i\{i,k\}$ 都是一个 $R_i\times Q$ 维的矩阵，默认为空；net 为一个线性层，它的输出误差平方和对于输入 $\boldsymbol{P}$ 来说具有最小值。

【例 3-5】 利用 newlind()创建一个线性神经网络，并进行仿真及直线拟合。

```
>> clear all;
x = -5:5;
y = 3*x-7;                                  % 直线方程为 y = 3x-7
randn('state',2);                           % 设置种子,便于重复执行
y = y+randn(1,length(y))*1.55;              % 加入噪声的直线
plot(x,y,'ro');
P = x;T = y;
net = newlind(P,T);                         % 用 newlind()建立线性层
 % 新的输入样本
new_x = -5:0.2:5;
new_y = sim(net,new_x);                     % 仿真
hold on;
plot(new_x,new_y);
legend('原始数据点','最小二乘拟合直线');
title('newlind()函数用于最小二乘拟合直线');
disp('线性网络的权值: ')
net.iw
disp('线性网线的阈值: ')
net.b
```

运行程序，输出如下，效果如图 3-12 所示。

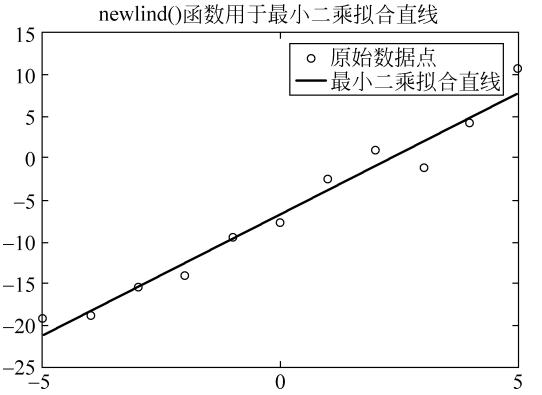

图 3-12　newlind()用于最小二乘拟合直线效果图

线性网络的权值：

```
ans =
    [2.9193]
```

线性网线的阈值：

```
ans =
    [ - 6.6690]
```

拟合得到的直线方程为 $y=2.9193x-6.6690$ 与原方程 $y=3*x-7$ 比较接近。

3.7.2　传输函数

在 MATLAB 中，也提供了相关 purelin()函数实现线性网络的传输。神经元最简单的传输函数是简单地从神经元输入到输出的线性传输函数，输出仅被神经元所附加的偏差所修正。线性传输函数常用于由 Widrow-Hoff 或 BP 准则来训练的神经网络中，函数的调用格式为

$\boldsymbol{A}$ = purelin($\boldsymbol{N}$,FP)：$\boldsymbol{N}$ 为 $S\times Q$ 维的网络输入(列)向量矩阵，FP 为性能参数(可忽略)，返回网络输入向量 $\boldsymbol{N}$ 的输出矩阵 $\boldsymbol{A}$。

info = purelin('code')：依据 code 值的不同，返回不同的信息。

- 当 code=name 时表示返回传输函数的全称。
- 当 code=active 时返回有传输函数最小、最大值的有效输入范围。
- 当 code=output 时返回有传输函数的最小、最大值的二元向量。
- 当 code=fullderiv 时，返回 1 或 0，具体取决于 dA_dN 是 $S\times S\times Q$ 还是 $S\times Q$。
- 当 code=fpnames 时返回函数参数的名称。
- 当 code=fpdefaults 时返回默认的函数参数。

【例 3-6】 产生一个线性 S 型传输函数。

```
>> clear all;
n = - 5:0.1:5;
a = purelin(n);
plot(n,a)
```

运行程序，效果如图 3-13 所示。

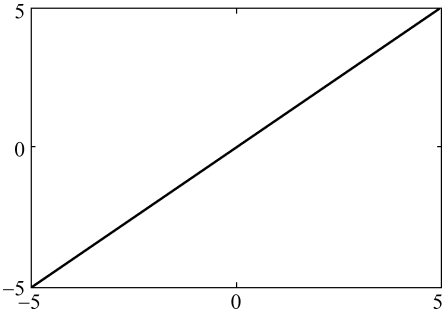

图 3-13　线性 S 型传输函数

3.7.3　学习函数

在 MATLAB 神经网络工具箱中，提供了 learnwh()函数及 maxlinlr()函数用于线性神经网络的学习，下面给予介绍。

1. learnwh()函数

该函数为 W-H 学习函数，也称为 Delta 准则或最小方差准则学习函数。它可以修改神经元的权值和阈值，使输出误差的平方和最小。它沿着误差平方和的下降最快方向连续调整网络的权值和阈值，由于线性神经网络的误差性能表面是抛物面，仅有一个最小值，因此可以保证网络是收敛的，前提是学习速率不超出由函数 maxlinlr()计算得到的最大值。函数的调用格式为

[dW,LS] = learnwh(***W***,***P***,***Z***,***A***,***N***,***T***,***E***,gW,gA,D,LP,LS)：参数 ***W*** 为 $S\times R$ 维的权值矩阵(或 $S\times 1$ 维的阈值向量)；***P*** 为 Q 组 R 维的输入向量矩阵(或 Q 组单个输入)；***Z*** 为 Q 组 S 维的加权输入向量；***A*** 为 $S\times Q$ 的输出向量，***N*** 为 Q 组 S 维的网络输入向量；***E*** 为 Q 组 S 维的误差向量(***E*** = ***T*** − ***Y***，***T*** 为网络的目标向量，***Y*** 为网络的输出向量)；gW 为 $S\times R$ 维的性能参数的梯度；gA 为 Q 组 S 维的性能参数的输出梯度；D 为 $S\times R$ 维的神经网元间隔距离；LP 为学习参数，如果没有则为[]；LS 为学习状态，初始值为[]。输出参数 dW 为 $S\times R$ 维权值变化矩阵；LS 为新的学习状态。

info = learnwh('code')：针对不同的 code 返回相应的有用信息。

- pnames：表示返回学习参数的名称。
- pdefaults：表示返回默认的学习参数。
- needg：表示如果函数使用了 gW 或 gA，则返回 1。

【例 3-7】 用 leanrwh()实现一个线性神经网络，解决例 3-5 中的直线拟合问题。

```
>> clear all;
x = -5:5;                                   % 定义数据
y = 3 * x - 7;                              % 直线方程为 y = 3x - 7
randn('state',2);                           % 设置种子,便于重复执行
y = y + randn(1,length(y)) * 1.55;          % 加入噪声的直线
x = [ones(1,length(x));x]                   % x 加上偏置
lp.lr = 0.01;                               % 学习率
Max = 150;                                  % 最大迭代次数
ep1 = 0.1;                                  % 均方差终止阈值
ep2 = 0.0001;                               % 权值变化终止阈值
 % 初始化
w = [0 0];
 % 循环更新
for i = 1:Max
    fprintf('第 %d 次迭代: \n',i)
    e = y - purelin(w * x);                 % 求得误差向量
    ms(i) = mse(e);                         % 均方差
    ms(i)
    if(ms(i)< ep1)                          % 如果均方差小于某个值,则算法收敛
        fprintf('均方差小于指定数而终止\n');
        break;
    end
    dW = learnwh([],x,[],[],[],[],e,[],[],[],lp,[]);    % 权值调整量
    if(norm(dW)< ep2)                       % 如果权值变化小于指定值,则算法收敛
        fprintf('权值变化小于指定数而终止\n');
        break;
    end
```

```
    w = w + dW                                    % 用 dW 更新权值
end
 % 显示
fprintf('算法收敛于: \nw = ( % f, % f),MSE: % f\n',w(1),w(2),ms(i));
figure;
subplot(211);                                     % 绘制散点和直线
plot(x(2,:),y,'o');  title('散点与直线拟合结果');
xlabel('x');ylabel('y');
axis([ - 6 6,min(y) - 1,max(y) + 1]);
x1 = - 5:0.2:5;
y1 = w(1) + w(2) * x1;
hold on;
plot(x1,y1);
subplot(2,1,2);semilogy(1:i,ms,' - o');      % 绘制均方差下降曲线
xlabel('迭代次数');ylabel('MSE');title('均方差下降曲线');
```

运行程序,输出如下,效果如图 3-14 所示。

```
x =
      1      1      1      1      1     1     1     1     1     1     1
    - 5    - 4    - 3    - 2    - 1     0     1     2     3     4     5
```

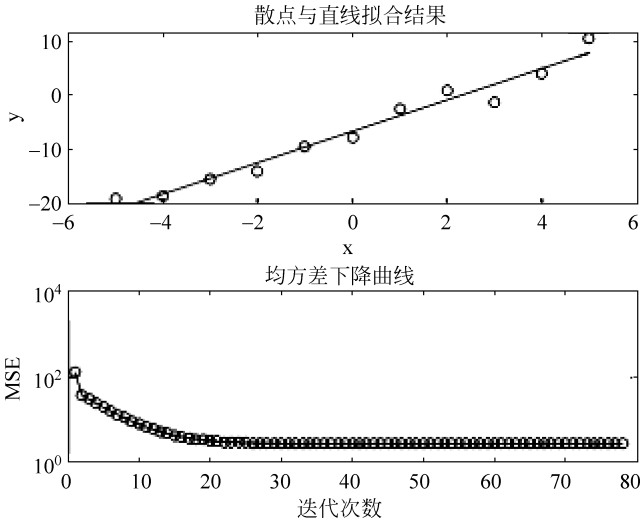

图 3-14　learnwh()拟合直线

第 1 次迭代：

```
ans =
  132.5929
w =
   - 0.7336    3.2112
```

第 2 次迭代：

```
ans =
   38.9751
```

```
w =
    -1.3865    2.8901
…
```

第 78 次迭代：

```
ans =
    2.8933
```

权值变化小于指定数而终止

算法收敛于：

```
w=(-6.668192,2.919306),MSE:2.893348
```

2. maxlinlr()函数

前面介绍到，线性神经网络的学习率应小心选择，否则可能出现收敛过慢或无法稳定收敛的问题。MATLAB 提供了一个计算最大学习率的函数，其公式为

$$0 < \eta < \frac{2}{\lambda_{\max}}$$

其中，$\lambda_{\max}$ 为输入向量自相关矩阵的最大特征值。

maxlinlr()函数的调用格式为

lr = maxlinlr(***P***)：输入参数 ***P*** 为 $R \times Q$ 维矩阵，对不带阈值的线性层得到一个所需要的最大学习速率 lr。

在命令窗口中输入 type maxlinlr 命令可看到函数的计算代码为

```
lr = 0.9999/max(eig(p*p'));
```

eig()函数用于计算矩阵的特征值。

lr=maxlinlr(P, 'bias')：针对带有阈值的线性层得到一个所需要的最大学习速率。

计算代码为

```
p2=[p; ones(1,size(p,2))];
lr = 0.9999/max(eig(p2*p2'));
```

【例 3-8】 在给定输入 P 的情况下，分为“带阈值”与“不带阈值”两种情况求得该线性层所需的最大学习速率。

```
>> clear all;
P = [1 2 -4 7; 0.1 3 10 6];
disp('不带阈值速率为: ')
lr1 = maxlinlr(P)
disp('带阈值速率为: ')
lr2 = maxlinlr(P,'bias')
```

运行程序，输出如下。

不带阈值速率为：

```
lr1 =
    0.0069
```

带阈值速率为：

```
lr2 =
    0.0067
```

3.7.4　均方误差性能函数

mse()函数是线性神经网络的性能函数，以均方误差来评价网络的精确程度。函数的调用格式为

perf = mse(net,*t*,*y*,ew)：输入参数 net 为网络；*t* 为目标矩阵或单元阵列；*y* 为输出矩阵或单元数组；ew 为错误的权重(可选)；输出参数 perf 平均绝对误差。

【例 3-9】 计算一个矩阵的均方误差。

```
>> clear all;
>> randn('seed',2);                    % 设定随机种子
>> a = randn(3,4)                      % 3×4 矩阵
a =
     0.2820    -0.7123    1.1219    1.8966
    -0.8983    -1.1757    1.0644    1.2472
     1.1428     0.1415    1.5076    1.0075
>> mse(a)                              % 计算均方误差
ans =
    1.2445
>> b = a(:);                           % 把矩阵 a 整理成向量
>> sum(b.^2)/length(b)                 % 手工计算均方误差
ans =
    1.2445
>> mse(b)                              % 对向量 b 计算均方误差
ans =
    1.2445
```

由以上示例可看出，mse()会对矩阵或数组中的所有元素计算均方误差，最终返回一个标量值，所以数组的形状、元素的位置变化不影响 mse()函数的结果，假设所有形式的输入都被转换为向量 $\boldsymbol{x}$，则计算公式为

$$\mathrm{mse}(\boldsymbol{x})=\frac{1}{N}\sum_{i=1}^{N}x_i^2$$

式中，N 为向量长度。

3.8　线性神经网络的局限性

线性神经网络只能反映输入和输出样本矢量间的线性映射关系，和感知器神经网络一样，它也只能解决线性可分问题。由于线性神经网络的误差曲面是一个多维抛物面，所以在学习速率足够小的情况下，对于基于最小二乘梯度下降原理进行训练的线性神经网络总可以找到一个最优解。但是，尽管如此，对线性神经网络的训练并不一定总能达到零误差。线性神经网络的训练性能要受网络规则和训练样本集大小的限制。如果

线性神经网络的自由度(即神经网络所有权值和阈值的个数总和)小于训练样本集中“输入-目标”向量的对数,且各样本矢量线性无关,则网络训练不可能达到零误差,而只能得到一个使网络误差最小的解;反之,如果网络自由度大于样本集的个数,则会得到无穷多个使网络训练误差为零的解。此外,值得注意的是,线性神经网络的训练和性能要受到学习速率参数的影响,过大的学习速率可能会导致网络性能发散。

3.8.1 线性相关向量

在应用线性神经网络解决问题前,首先要判断该问题是否能用线性网络来解决。通常情况下,线性网络的自由度(权值和阈值的总和,即 $S\times R+S$)至少要等于约束的数目(输入/输出样本数 Q),这样才可以应用。但是当输入样本线性相关或没有阈值的时候,这种要求就不一定成立了。如果线性相关的输入量与期望输出向量之间并不匹配,则此问题是一个非线性问题,且这个问题得不到零误差解。

【例 3-10】 给定一个输入向量和期望输出向量 $\boldsymbol{T}$ 进行训练。

```
>> clear all;
P = [1.0 2.0 3.0;4.0 5.0 6.0];
T = [0.5 1.0 -1.0];
lr = maxlinlr(P,'bias');
net = newlin([0 10;0 10],1,[0],lr);
net.trainParam.show = 50;
net.trainParam.epochs = 500;
net.trainParam.goal = 0.001;
[net,tr] = train(net,P,T);
tr
```

运行程序,输出如下,训练过程如图 3-15 所示。

```
tr =
        trainFcn: 'trainb'
      trainParam: [1x1 struct]
      performFcn: 'mse'
    performParam: [1x1 struct]
        derivFcn: 'defaultderiv'
       divideFcn: 'dividetrain'
      divideMode: 'sample'
     divideParam: [1x1 struct]
        trainInd: [1 2 3]
          valInd: []
         testInd: []
            stop: 'Maximum epoch reached.'
      num_epochs: 500
       trainMask: {[1 1 1]}
         valMask: {[NaN NaN NaN]}
        testMask: {[NaN NaN NaN]}
      best_epoch: 499
            goal: 1.0000e-03
          states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'val_fail'}
           epoch: [1x501 double]
```

```
            time: [1x501 double]
            perf: [1x501 double]
           vperf: [1x501 double]
           tperf: [1x501 double]
        val_fail: [1x501 double]
       best_perf: 0.3474
      best_vperf: NaN
      best_tperf: NaN
```

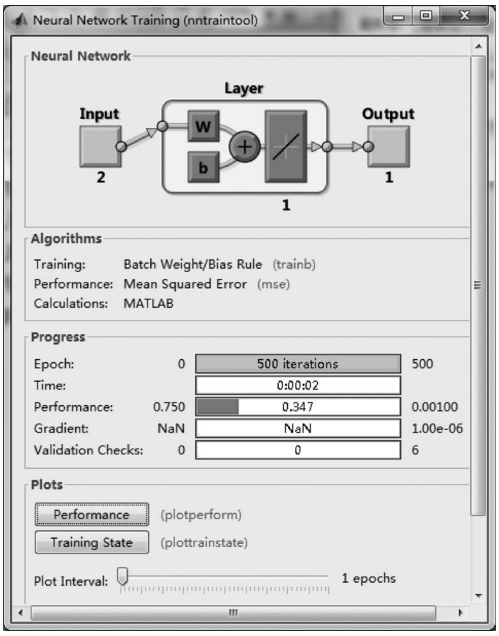

图 3-15　训练过程图

由此可见，输入向量具有相关性，而期望输出向量并不具有这种相关性，即输入/输出不匹配。应用 maxlinlr()函数寻找训练用最快速最稳定的学习率，然后用 newlin()函数生成一个线性神经元，并设置训练次数、训练精度、显示情况等进行训练。

达到最大训练次数后，仿真停止，但是训练精度并未达到。应用函数 sim()求出给定输入向量的输出向量，对网络进行验证：

```
>> p = [1.0;4];
>> a = sim(net,p)
a =
    0.8971
```

仿真结果表明，网络输出值不等于期望输出值，且相差较大。由此可以得出结论：线性网络不能适应输入向量之间具有线性相关性的非线性问题。

3.8.2　学习速率过大

在网络设计中，学习速率的选取是影响收敛速度以及训练结果的一个很重要的因素。当学习速率过小的时候，依据 W-H 学习规则总能够训练网络满足精度要求；但是

当学习速率较大时，则可能导致训练过程不稳定。MATLAB 工具箱函数给出了一个正确求解学习速率的函数 maxlinlr()。

下面以不同的学习速率再次训练网络以展现两种不希望的学习速率带来的影响。

【例 3-11】 为了能够清楚地观察到网络训练过程中权值的修正所带来的误差的变化情况，因此在程序中加入误差等高线图，并在其中显示每一次权值修正后的误差值位置。

```
>> clear all;
P = [1 -1.2];
T = [0.5 1];
[R,Q] = size(P);
[W,B] = size(T);
 %绘制误差曲面图
wrange = -2:0.2:2;                          %限定 W 值的坐标范围
brange = -2:0.2:2;                          %限定 B 值的坐标范围
ES = errsurf(P,T,wrange,brange,'purelin');  %求神经元的误差平面
mesh(ES,[60,30]);                           %求神经元的误差平面,效果如图 3-16 所示
set(gcf,'color','w');                       %将曲面图背景设为白色
```

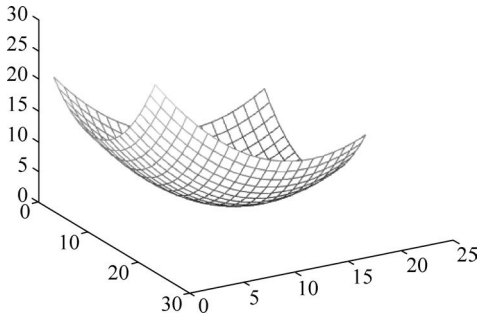

图 3-16 线性网络的误差曲面图

图 3-16 是由权值 w 和偏差 b 所决定的线性网络误差图，可以看到它是一个抛物线型的。

```
>> %设计网络权值并绘制投影图
figure;
net = newlind(P,T);                         %求理想的权值和偏差
dw = net.iw{1,1};                           %赋理想的权值和偏差
db = net.b{1};                              %赋理想的偏差
%作等高线图,ES 为高,返回等高线矩阵 C,列向量 h 是线或对象的句柄
[C,h] = contour(wrange,brange,ES);
clabel(C,h);                                %一条线一个句柄,被作为输入
colormap cool;                              %桌面的颜色 cool(青和洋色)
axis('equal');
hold on;
plot(dw,db,'rp','LineWidth',2.5);
xlabel('W'); ylabel('B');
```

图 3-17 为图 3-16 的从上往下的投影图，图中的曲线称为等高线，线上的点具有相同的误差值。图中的“星号”代表用函数 newlind()求出的误差最小值。

当运行以下程序，弹出如图 3-18 所示的选择列表框。

```
lr = menu('选择学习速率: ',…
    '1.2 * maxlinr',…
    '2.8 * maxlinr');
```

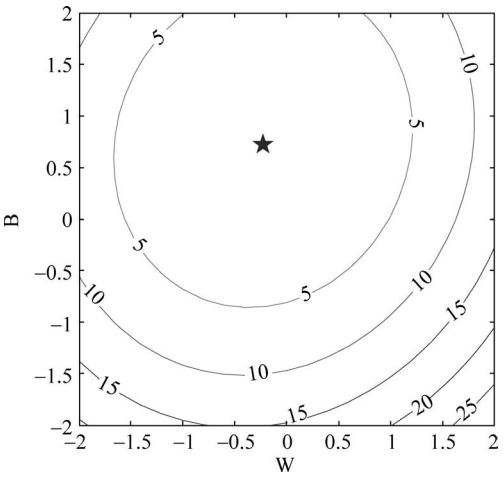

图 3-17　误差等高线图

图 3-18　选择列表框

单击图 3-18 中的 1.2 * maxlinlr 按钮，并运行以下代码。

```
>> disp('')
 % 训练权值
disp_freq = 1;
max_epoch = 28;
err_goal = 0.001;
if lr == 1
    lp.lr = 1.2 * maxlinlr(P,'bias');
else
    lp.lr = 2.8 * maxlinlr(P,'bias');
end
a = W + P + B;
A = purelin(a);
E = T - A;                                  % 求误差
sse = sumsqr(E);                            % 求误差矩阵元素的平方和
errors = [sse];
for epoch = 1:max_epoch                     % 训练权值
    if sse < err_goal
        epoch = epoch - 1;
        break;
    end
    lw = W;      lb = B;
    dw = learnwh([],P,[],[],[],[],E,[],[],[],lp,[]);
    db = learnwh(B,ones(1,Q),[],[],[],[],E,[],[],[],lp,[]);
    W = W + dw;
```

```
        B = B + db;
        a = W * P + B;
        A = purelin(a);
        E = T - A;
        sse = sumsqr(E);
        errors = [errors sse];                    % 把误差变为一个行向量
        if rem(epoch,disp_freq) == 0
            plot([lw,W],[lb,B],'g - ');           % 显示权值与偏差向量训练图,效果如图 3-19 所示
            drawnow
        end
    end
    hold off;
    m = W * P + B;
    a = purelin(m);
    plot(a);                                      % 作每次训练的误差图,效果如图 3-20 所示
```

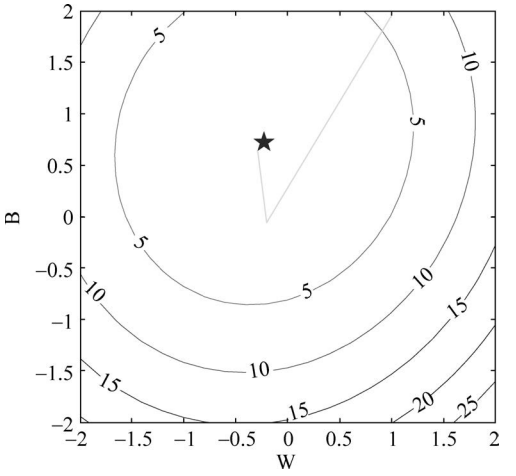

图 3-19　lr=1.2 * maxlinlr 权向量修正的变化过程

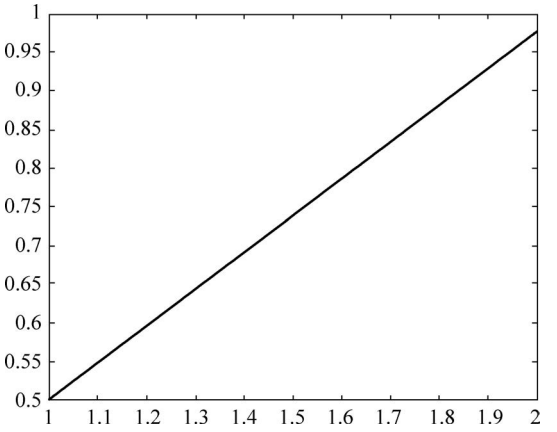

图 3-20　网络训练过程中的误差记录

当单击图 3-18 中的 2.8 * maxlinlr 按钮时，并执行以下代码，可得到如图 3-21 和图 3-22 所示效果图。

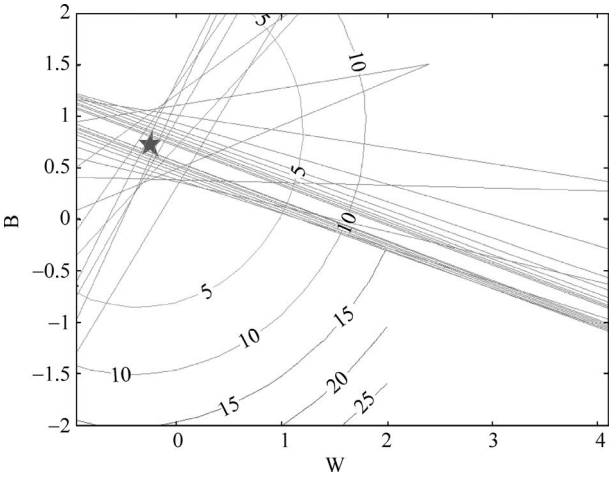

图 3-21　lr＝2.8 * maxlinlr 权向量修正的变化过程

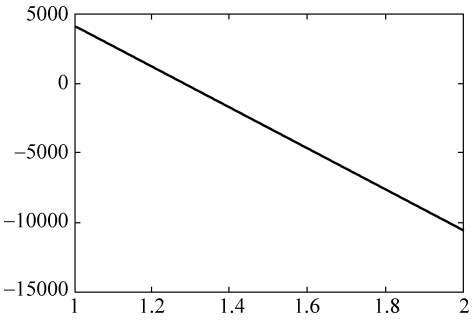

图 3-22　网络训练过程中的误差记录

正如所看到的，由于学习速率太大，虽然能够使权值得到较大的修正，但由于过量而产生了振荡。不过误差在其权值振荡过程中仍然能够直线下降，在经过十多次循环后就达到了误差的最小值。如果学习速率再加大，可能就不是这么幸运了。

由图 3-21 和图 3-22 可看出，由于其学习速率太大，网络的权值修正过程总是在最小误差方向上运动，但每一次都由于过大的调整使其偏离期望目标越来越远，其误差是向增加而不是减少的方向移动。由此可得出结论：应选取较小的学习速率以保证网络收敛，而不应选太大的学习速率而使其发散。

3.8.3　不定系统

如果单线性神经网络只有一个输入向量，应用一组输入/输出样本进行训练。而此时，对应单输入、单输出网络，有两个可调整变量，即一个权值和一个阈值。那么问题是：约束条件只有一个，变量数目大于约束数目，这就是不定系统。

【例 3-12】　用线性神经网络演示不定系统。

```
>> clear all;
 %输入向量及期望输出向量
P = [ + 1.0];
```

```
T = [ + 0.5];
% 给出权值和阈值的范围并绘制误差曲线及误差曲面等高线
w_range = - 1:0.2:1;
b_range = - 1:0.2:1;
ES = errsurf(P,T,w_range,b_range,'purelin');
plotes(w_range,b_range,ES);                                    % 效果如图 3-23 所示
% 寻找训练用最快速稳定的学习率,创建生成一个线性神经元,并设置训练次数
maxlr = maxlinlr(P,'bias');
net = newlin([ - 2 2],1,[0],maxlr);
net.trainParam.goal = 1e - 10;
% 设定训练参数,对网络进行训练,默认的学习规则为 W-H 规则,也可以在训练参数中修改
net.trainParam.epochs = 1;
net.trainParam.show = NaN;
h = plotep(net.IW{1},net.b{1},mse(T - sim(net,P)));
[net,tr] = train(net,P,T);
r = tr;
epoch = 1;
while true
    epoch = epoch + 1;
    [net,tr] = train(net,P,T);                                 % 效果如图 3-24 所示
    if length(tr.epoch)> 1
        h = plotep(net.IW{1},net.b{1},tr.perf(2),h);           % 效果如图 3-25 所示
        r.epoch = [r.epoch epoch];
        r.perf = [r.perf tr.perf(2)];
        r.vperf = [r.vperf NaN];
        r.tperf = [r.tperf NaN];
    else
        break
    end
end
tr = r;
```

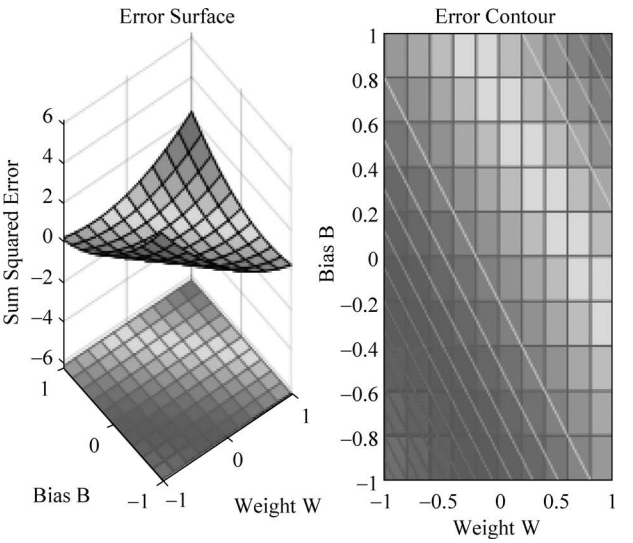

图 3-23 误差曲面及等高线效果图

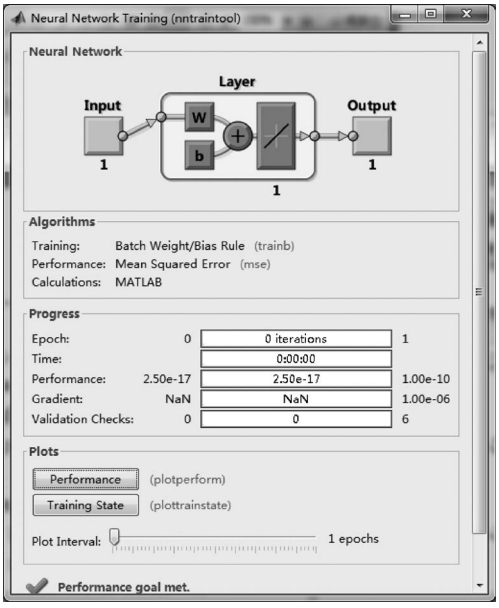

图 3-24　训练过程图

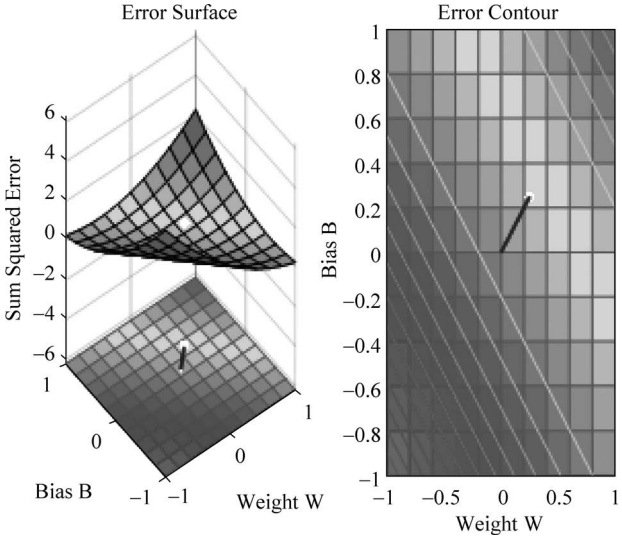

图 3-25　训练后误差曲面及等高线

下面再应用 newlind()函数来求解。前面训练时应用 train()函数，而此时应用 solvelin()函数，结论不同。train()在不同的初始条件下就会得到不同的结果，而 solvelin()每一次的结果都相同。

```
>> solvednet = newlind(P,T);
hold on;
plot(solvednet.IW{1,1},solvednet.b{1},'ro')                    %效果如图 3-26 所示
%绘制训练误差随时间变化的曲线，训练误差与训练精度有很大关系
hold off;
```

```
plotperf(tr,net.trainParam.goal);          %绘制出的误差响应曲线如图 3-27 所示
>> %对网络进行验证
p = 1.0;
a = sim(net,p)
a =
    0.5000
```

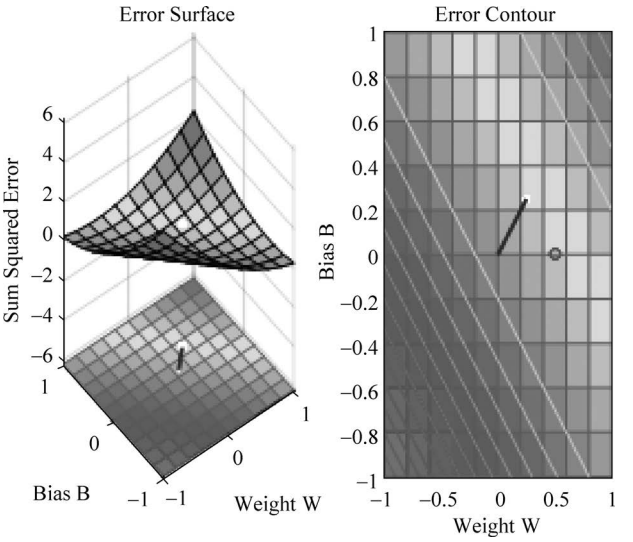

图 3-26　两种训练结果效果图

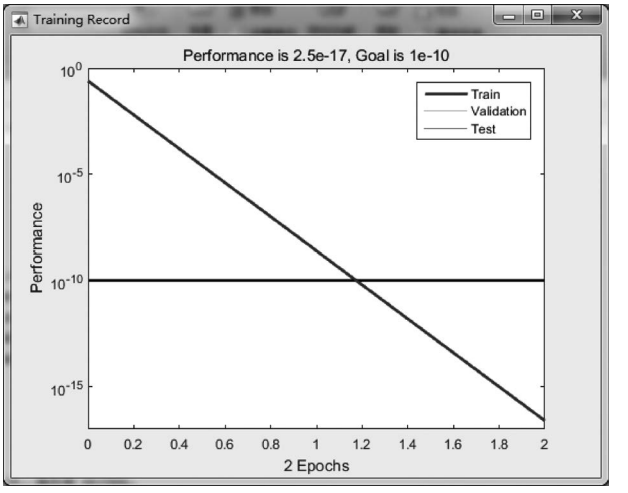

图 3-27　误差响应曲线效果图

3.9　线性神经网络的应用

线性神经网络只能实现线性运算，这一点与单层感知器比较类似，因此线性神经网络只能用于解决比较简单的问题。

单个线性神经网络只能解决线性可分问题，与或逻辑就是典型的线性可分问题，这里选择与逻辑作为实例。而异或逻辑则线性不可分，因此要使用多个线性神经网络问题间接实现。

3.9.1　逻辑与

逻辑与有两个输入、一个输出，因此对应的线性网络拥有两个输入节点、一个输出节点，如图 3-28 所示。

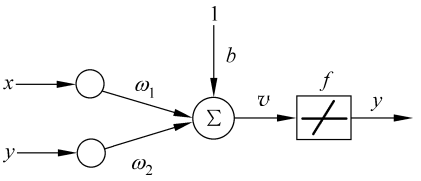

图 3-28　网络结构

包括偏置，网络的训练中共需确定 3 个自由变量，而输入的训练向量则有 4 个，因此可以形成一个线性方程组：

$$\begin{cases} 0\times x+0\times y+1\times b=0 \\ 0\times x+1\times y+1\times b=0 \\ 1\times x+0\times y+1\times b=0 \\ 1\times x+1\times y+1\times b=1 \end{cases}$$

由于方程的个数超过了自变量的个数，因此方程没有精确解，只有近似解，用伪逆的方法可以求得权值向量的值。

下面以分步的方式用线性神经网络得到相同的结果。

网络中需要求解的是权值 ω_1、ω_2 和偏置 b。把偏置加入权值中统一计算，定义总的权值向量

$$\boldsymbol{\omega}=[b,\omega_1,\omega_2]^{\mathrm{T}}$$

定义输入

$$\boldsymbol{P}=\begin{bmatrix} 1,x_1,y_1 \\ 1,x_2,y_2 \\ \cdots \end{bmatrix}$$

期望输出

$$\boldsymbol{d}=[0,0,0,1]$$

初始化，将权值和偏置初始化为零。令$\boldsymbol{\omega}=[b,\omega_1,\omega_2]^{\mathrm{T}}=[0,0,0]^{\mathrm{T}}$。

根据以上叙述，使用工具箱函数可以实现与逻辑，代码为

```
>> clear all;
 % 定义变量
p = [0 0 1 1;0 1 0 1];                          % 输入向量
d = [0 0 0 1];                                  % 期望输出向量
lr = maxlinlr(p,'bias')                         % 根据输入矩阵求解最大学习率
 % 线性网络实现
net1 = linearlayer(0,lr);                       % 创建线性网络
net1 = train(net1,p,d);                         % 线性网络训练
 % 感知器实现
net2 = newp([ - 1 1; - 1 1],1,'hardlim');       % 创建感知器
net2 = train(net2,p,d);                         % 感知器学习
 % 显示
disp('线性网络输出: ')
```

```
Y1 = sim(net1,p)
disp('线性网络二值输出: ');
Y11 = Y1 >= 0.5
disp('线性网络最终权值: ')
w1 = [net1.iw{1,1},net1.b{1,1}]
disp('感知器输出: ')
Y2 = sim(net2,p)
disp('感知器二值输出: ')
Y22 = Y2 >= 0.5
disp('感知器最终权值: ')
w2 = [net2.iw{1,1},net2.b{1,1}]
 %图形窗口输出
plot([0,0,1],[0,1,0],'ro');
hold on;
plot(1,1,'d');
x = -2:0.2:2;
y1 = 1/2/w1(2) - w1(1)/w1(2) * x - w1(3)/w1(2); %1/2是区分0和1的阈值
plot(x,y1,'-');
y2 = -w2(1)/w2(2) * x - w2(3)/w2(2);          %hardlim()函数以0为阈值,分别输出0和1
plot(x,y2,'--');
axis([-0.5 2 -0.5 2]);
xlabel('x');ylabel('ylabel');
title('线性神经网络用于求解与逻辑')
legend('0','1','线性神经网络分类面','感知器分类面')
```

运行程序,输出如下,训练过程如图 3-29 所示,得到分类面如图 3-30 所示。

```
lr =
    0.1569
```

线性网络输出:

```
Y1 =
   -0.2500    0.2500    0.2500    0.7500
```

线性网络二值输出:

```
Y11 =
     0     0     0     1
```

线性网络最终权值:

```
w1 =
    0.5000    0.5000   -0.2500
```

感知器输出:

```
Y2 =
     0     0     0     1
```

感知器二值输出:

```
Y22 =
     0     0     0     1
```

感知器最终权值：

```
w2 =
     2     1     -3
```

因此，线性网络得到的决策面为直线$\frac{1}{2}(x+y)-\frac{1}{4}=\frac{1}{2}$，感知器得到的决策面为直线 $2x+1-3=0$。

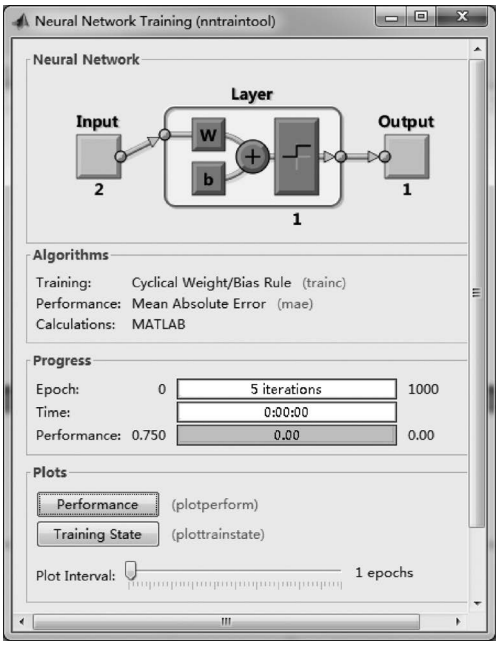

图 3-29　网络训练过程

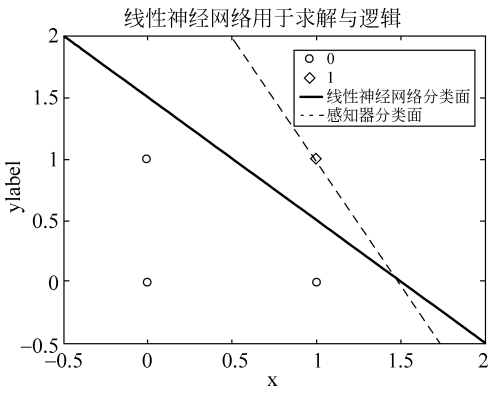

图 3-30　线性网络与感知器的比较

显然，线性网络得到的分类面大致位于两类坐标点的中间位置，而感知器得到的分类面恰好穿过其中一个坐标点。在另外一些场合，感知器得到的分类面也离训练的模式很近，从这一点上说，线性神经网络优于感知器。造成这种差别的原因并不在于学习算法，尽管通常认为 LMS 算法更合理，但两者在计算上几乎拥有相同的形式。

感知器更新权值向量：$\omega(n+1)=\omega(n)+\eta[d(n)-y(n)]x(n)$

线性神经网络更新权值向量：$\omega(n+1)=\omega(n)+\eta \boldsymbol{x}^{\mathrm{T}}(n)e(n)$

根本原因在于，感知器使用了二值阈值元件，输出值只能为两种(0/1 或 1/－1)，因此，只要有一次达到某个值，该值使得二值化后的输出是正确的，那么误差 $e(n)=[d(n)-y(n)]$就等于零，根据上面的公式，权值向量就不会再有实质性的更新了。而线性神经网络用线性的传输函数将结果直接输出，误差值可以根据输出值不断变化，从而使得权值根据误差变化不断获得更新，最终获得最小均方误差意义下的最优解。

3.9.2　逻辑异或

异或属于线性不可分问题，在此通过两种方法来实现。

1. 添加非线性输入

添加非线性输入的代价是输入向量维数变大，运算复杂度变大。结构如图 3-31 所示。

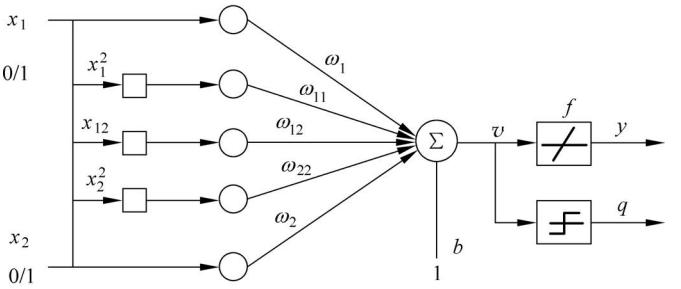

图 3-31　添加非线性输入

这种方法的思路是，既然运算过程中无法引入非线性运算的特性，那么就在输入端添加非线性成分。实现的代码为

```
>> clear all;
 % 定义变量
p1 = [0 0 1 1;0 1 0 1];                    % 原始输入向量
p2 = p1(1,:).^2;
p3 = p1(1,:). * p1(2,:);
p4 = p1(2,:).^2;
p = [p1(1,:);p2;p3;p4;p1(2,:)]             % 添加非线性成分后的输入向量
d = [0 1 1 0];                             % 期望输出向量
lr = maxlinlr(p,'bias')                    % 根据输入矩阵求解最大学习率
 % 线性网络实现
net = linearlayer(0,lr);                   % 创建线性网络
net = train(net,p,d);                      % 线性网络训练
 % 显示
disp('网络输出: ')
Y1 = sim(net,p)
disp('网络二值输出: ')
Y11 = Y1 > = 0.5
disp('最终权值: ')
w1 = [net.iw{1,1},net.b{1,1}]
 % 图形窗口输出
plot([0,1],[0,1],'ro');
hold on;
plot([0 1],[1 0],'d');
axis([ - 0.1 1.1  - 0.1 1.1]);
xlabel('x');ylabel('y');
title('线性神经网络用于求解异或逻辑');
x = - 0.1:0.1:1.1;
y = - 0.1:0.1:1.1;
N = length(x);
X = repmat(x,1,N);
Y = repmat(y,N,1);
Y = Y(:);
Y = Y';
p = [X;X.^2;X. * Y;Y.^2;Y];
yy = net(p);
```

```
y1 = reshape(yy,N,N);
[C,h] = contour(x,y,y1);
clabel(C,h,0.5);
legend('0','1','线性神经网络分类面');
```

运行程序，输出如下，训练过程如图 3-32 所示，决策面如图 3-33 所示。

```
p =
     0     0     1     1
     0     0     1     1
     0     0     0     1
     0     1     0     1
     0     1     0     1
lr =
    0.1033
```

网络输出：

```
Y1 =
    0.0000    1.0000    1.0000    0.0000
```

网络二值输出：

```
Y11 =
     0     1     1     0
```

最终权值：

```
w1 =
    0.5000    0.5000   -2.0000    0.5000    0.5000    0.0000
```

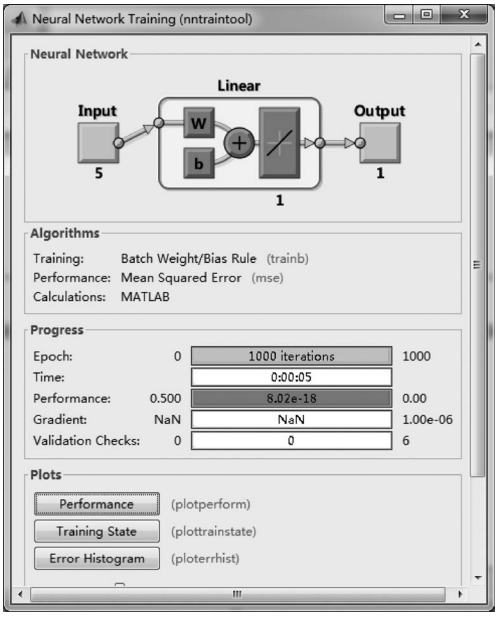

图 3-32　训练过程图

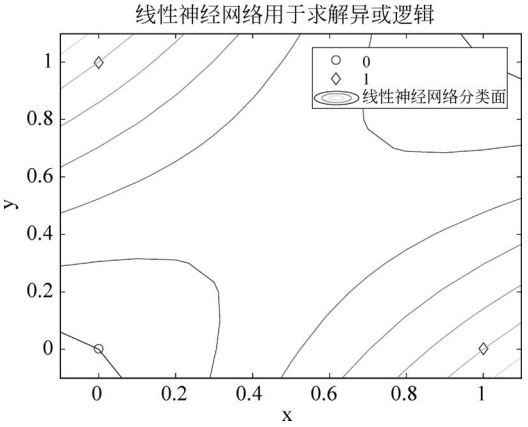

图 3-33　异或问题的决策面

2. 使用 Madaline

Madaline 的核心思想是使用多个线性神经元。在这里使用两个神经元，分别得到输出后再对输出值进行判断，得到最终的分类结果，其结构如图 3-34 所示。

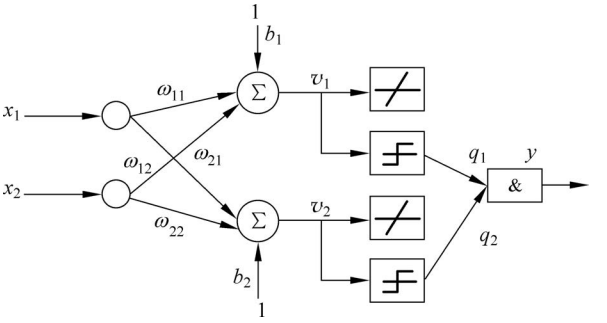

图 3-34　Madaline 结构图

对于异或问题，解决的方法是将其分为两个子问题，分别用一个线性神经元实现。两个神经元的期望输出分别如图 3-35 和图 3-36 所示。

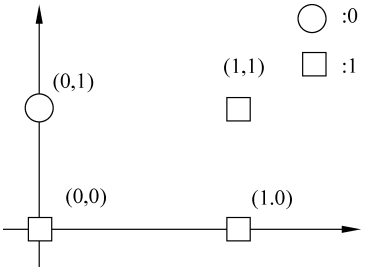

图 3-35　第一个神经元

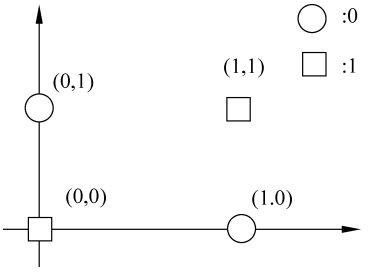

图 3-36　第二个神经元

按照图中 1 与 0 的定义，两个神经元的输出必须取一次与非，最后算得的结果才符合异或运算的定义。其实现的 MATLAB 代码为

```
>> clear all;
 % 第一个神经元
P1 = [0,0,1,1;0,1,0,1];                    % 输入向量
d1 = [1,0,1,1];                            % 期望输出向量
lr = maxlinlr(P1,'bias');                  % 根据输入矩阵求解最大学习率
net1 = linearlayer(0,lr);                  % 创建线性网络
net1 = train(net1,P1,d1);                  % 线性网络训练
 % 第二个神经元
P2 = [0,0,1,1;0,1,0,1];                    % 输入向量
d2 = [1,1,0,1];                            % 期望输出向量
lr = maxlinlr(P2,'bias');                  % 根据输入矩阵求解最大学习率
net2 = linearlayer(0,lr);                  % 创建线性网络
net2 = train(net2,P2,d2);                  % 线性网络训练
Y1 = sim(net1,P1);Y1 = Y1 > = 0.5;
Y2 = sim(net2,P2);Y2 = Y2 > = 0.5;
```

```
Y = ～(Y1&Y2);
 % 显示
disp('第一个神经元最终权值: ')
w1 = [net1.iw{1,1}, net1.b{1,1}]
disp('第二个神经元最终权值: ')
w2 = [net2.iw{1,1}, net2.b{1,1}]
disp('第一个神经元测试输出: ')
Y1
disp('第二个神经元测试输出: ');
Y2
disp('最终输出: ');
Y
plot([0,1],[0,1],'ro');                          % 图形窗口输出
hold on;
plot([0,1],[1,0],'d');
x = -2:2:2;
y1 = 1/2/w1(2) - w1(1)/w1(2) * x - w1(3)/w1(2); % 第一条直线,1/2 是区分 0 和 1 的阈值
plot(x,y1,'-');
y2 = 1/2/w2(2) - w2(1)/w2(2) * x - w2(3)/w2(2); % 第二条直线,1/2 是区分 0 和 1 的阈值
plot(x,y2,'m:');
axis([-0.1,1.1,-0.1,1.1])
xlabel('x');ylabel('y');
title('Madaline 用于求解异或逻辑')
legend('0','1','第一条直线','第二条直线');
```

运行程序,输出如下,训练过程如图 3-37 所示,解决异或问题效果如图 3-38 所示。

第一个神经元最终权值:

```
w1 =
    0.5000    -0.5000    0.7500
```

第二个神经元最终权值:

```
w2 =
   -0.5000    0.5000    0.7500
```

第一个神经元测试输出:

```
Y1 =
     1     0     1     1
```

第二个神经元测试输出:

```
Y2 =
     1     1     0     1
```

最终输出:

```
Y =
     0     1     1     0
```

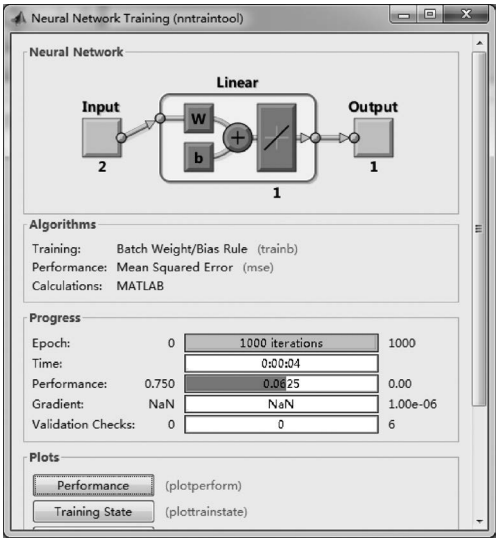

图 3-37　网络训练过程

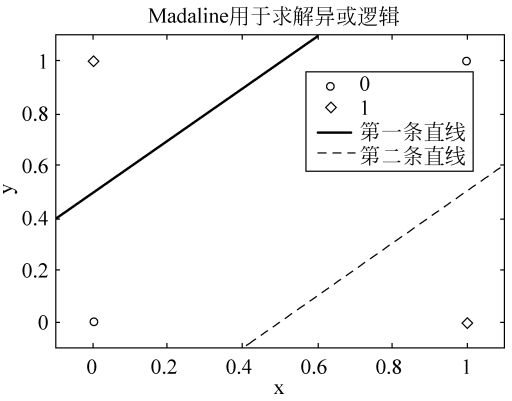

图 3-38　Madaline 解决异或问题的效果图

3.9.3　在噪声对消中的应用

使用自适应滤波器的线性滤波功能可以进行网络消噪。下面以飞行中飞机机长同乘客之间的广播模型为例来说明其工作原理。在飞行的飞机中，如果飞行员对着麦克风说话，驾驶座舱的引擎噪声将会混杂入他的声音信号中，这使得旅客听到的是非常嘈杂的声音。我们需要的是飞行员的声音而不是飞机引擎的噪声。如果可以获得引擎噪声样本并把它作为自适应滤波器的输入，那么就可以通过自适应滤波器来消除引擎噪声的影响。消除噪声的自适应滤波器的训练及其工作结构图如图 3-39 所示。

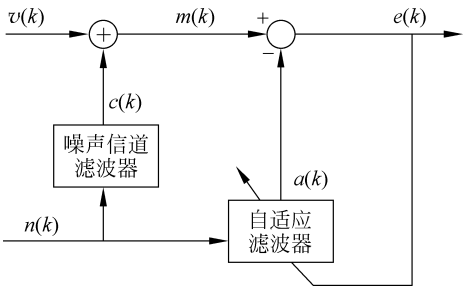

图 3-39　自适应滤波器消噪的训练及其工作结构图

这里采用自适应神经元线性网络去逼近带有引擎噪声信号 $n(k)$ 的飞行员声音信号 $m(k)$。由引擎噪声信号 $n(k)$ 给网络提供输入信息，从图 3-39 中可以看到，网络通过自适应滤波器的输出信号去逼近混杂在声音信号中的噪声部分，调整自适应滤波器以便使误差 $e(k)$ 最小。由图 3-39 可得以下关系式：

$$m(k)=v(k)+c(k)$$

$$e(k)=m(k)-a(k)$$

由此可得：

$$e(k)=v(k)+c(k)-a(k)$$

当自适应滤波器成功地逼近信号 $c(k)$ 时，在得到的 $e(k)$ 信号中就只剩下了所需要的机长的声音信号了。

在进行网络的训练过程中，进行网络的训练过程中，将机长的声音信号置为0，并调节网络的权值，然后将训练完成后获得的网络再以同样的方式接入系统中加以应用。当网络工作时，由于网络消除了飞行员声音信号中的噪声信号，使其输出 $e(k)$ 仅为飞行员的声音。

这种去噪方法优于传统的滤波器，它把噪声从信号中之中近似完全地消去，而传统的低通滤波器只是通过高频时很小的放大倍数把噪声抑制掉。所以网络的逼近精度越高，消噪的效果越好。

下面通过示例来演示自适应滤波消噪处理。

【例 3-13】 对于一个最优的滤波器，希望通过滤波将信号中的噪声去掉，一般的滤波器很难完全做到。利用自适应线性网络实现噪声对消的原理如图 3-40 所示。

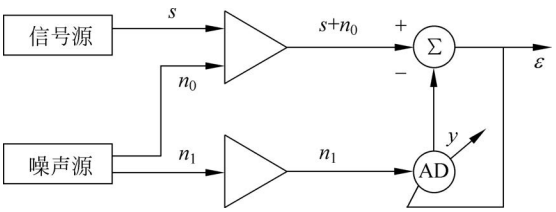

图 3-40　噪声对消原理框图

图中 s 为原始输入信号，假设为平稳的零均值随机信号；n_0 为与 s 不相关的随机噪声；n_1 为与 n_0 相关的信号；系统输出为 ε；$s+n_0$ 为 Adaline 神经元的预期输出，y 为 Adaline 神经元的输出。则

$$\begin{aligned}\varepsilon &= s + n_0 - y\\ E[\varepsilon^2] &= E[(s+n_0-y)^2]\\ &= E[s^2] + 2E[s\cdot(n_0-y)] + E[(n_0-y)^2]\\ &= E[\varepsilon^2] + E[(n_0-y)^2]\end{aligned}$$

通过 Adaline 调节，得到

$$E_{\min}[s^2] = E_{\min}[s^2] + E_{\min}[(n_0-y)^2]$$

上式中，当 $E_{\min}[(n_0-y)^2]\to 0$ 时，$y\to n_0$，其输出 ε 为 s，则噪声被抵消。

采用这种系统来完成对胎儿心率的检测，可以得到十分满意的结果。由于测量胎儿的心率一定会受到母体心率的干扰，而且母亲心率很强，但与胎儿心率是相互独立的，所以可将母体心率作为噪声源 n_1 输入 Adaline 中，混有噪声的胎儿心率信号为目标响应，通过对消后，系统就可以得到清晰的胎儿心率。

这种系统还可以应用于电话中的回音对消。在电话通话的过程中，如果没有回音对消措施，那么，我们自身的声音会与来自对方的声音一起传到听筒中，而且自身的声音更强，影响通话的质量。可将自身的声音作为噪声源 n_1 输入 Adaline 中，混有对方声音的信号作为目标响应，通过对消后，系统就可以得到清晰的来自对方的声音信号。

这里假设传输信号为正弦波信号，噪声为随机噪声，进行自适应线性神经网络设计。

根据以上分析，Adaline 自适应线性神经元的输入向量为随机噪声 n_l；正弦波信号

与随机噪声之和为 Adaline 神经元的目标向量；输出信号为网络调整过程中的误差信号。

其实现的 MATLAB 代码为

```
>> clear all;
% 定义输入向量和目标向量
time = 0.01:0.01:10;                          % 时间变量
noise = (rand(1,1000) - 0.5) * 4;             % 随机噪声
input = sin(time);                            % 信号
p = noise;                                    % 将噪声作为 Adaline 的输入向量
t = input + noise;                            % 将噪声 + 信号作为目标向量
% 创建线性神经网络
net = newlin([ - 1 1],1,0,0.0005);
% 线性神经网络的自适应调整(训练)
net.adaptParam.passes = 70;
[net,y,output] = adapt(net,p,t);              % 输出信号 output 为网络调整过程中的误差
% 绘制信号,叠加随机噪声的信号,输出信号的波形
hold on;
% 绘制信号的波形
subplot(3,1,1);plot(time,input,'r');
title('信号波形 sin(t)');
subplot(3,1,2);plot(time,t,'m');              % 绘制叠加随机噪声信号的波形
xlabel('t');title('随机噪声波形 sin(t) + noise(t)');
% 绘制输出信号的波形
subplot(3,1,3);plot(time,output,'b');
xlabel('t');title('输出信号波形 y(t)');
```

运行程序，如图 3-41 所示。

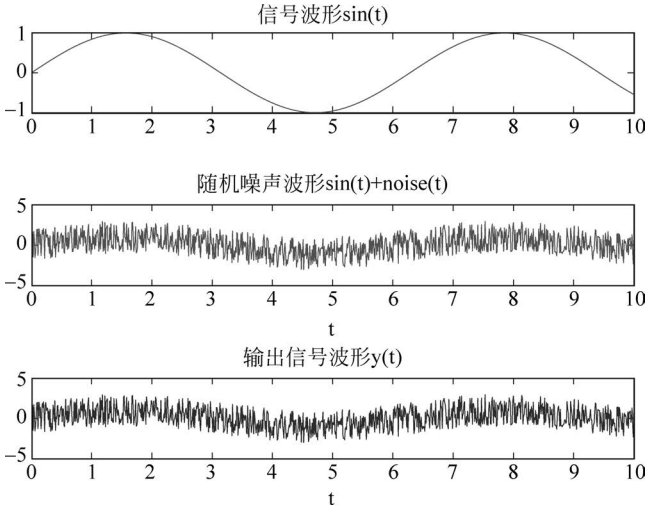

图 3-41　对消噪声的效果图

从图中可以看出，输出信号除了含有一定直流分量外，其波形与输入信号波形基本一致，消除了叠加的随机噪声。

3.9.4　在信号预测中的应用

实际上，物理系统的输入/输出关系式往往比较复杂，一般的形式为

$$y(k)=f(x(k-n),x(k-n+1),\cdots,x(k))$$

由上式，通过适当的设计网络的过程，使下式成立：

$$a(k)=x(k)=f(x(k-1),\cdots,x(k-n))$$

可以通过$(k-1)$及以前测量的数据来完成 k 时刻的预测任务。设计过程如图 3-42 所示。

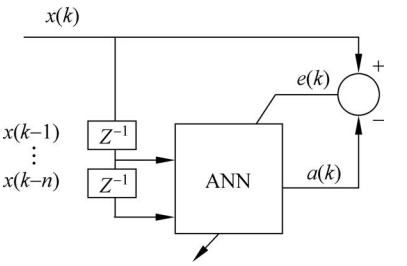

图 3-42　神经网络进行预测应用的设计结构图

在这里需要特别注意，$x(k)$信号没有加在输入端。由此可见，训练网络的结构设计不是一成不变、生搬硬套的，一定要根据具体问题灵活应用和掌握。

设计的目的是使 $x(k)-a(k)=e(k)=0$，即当求得 $e=0$ 时，就可以得到网络输出 $x(k)=a(k)$的预测。预测的应用中，同样有一个需要确定输出与前几次延时阶次有关的问题，只有 r 确定正确了，预测的结果才准确。预测功能可被用在已知最近几年的产量或数据，从而预测明年的数值这类问题上。注意只有确认所要预测的关系式确实是具有线性关系时，方可采用自适应线性网络来实现。

由此可见，都是自适应线性元件网络，不同的网络设计结构就能完成不同的任务，解决不同问题。灵活应用这种神经网络，就是要学会把不同的问题和网络结合起来，使之从数学理论上完成某种功能。虽然网络训练的都只是外部的输入/输出特性，但具体到每个应用中时是有很大的不同的。

【例 3-14】 实现自适应预测的线性网络。设自适应滤波器如图 3-43 所示，该滤波器的目的是要从输入信号的前两个时刻的值预测当前时刻的值。

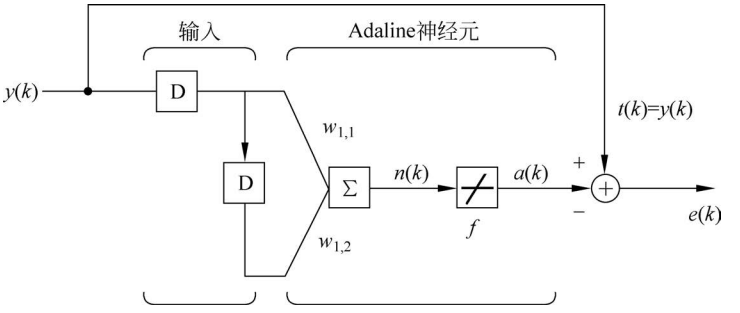

图 3-43　自适应滤波器示意图

图中 D 为延迟单元，多个延迟单元可以构成抽头延迟线，如图 3-44 所示。

设输入信号为一随机序列，试编写 MATLAB 程序，画出上述自适应滤波器的输入输出波形。

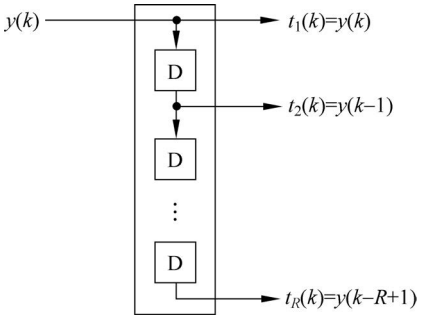

图 3-44　抽头延迟线

其实现的 MATLAB 程序代码如下。

```
>> clear all;
 % 定义输入向量和目标向量
time = 0.5:0.5:20;                          % 时间变量
y = (rand(1,40) - 0.5) * 4;                 % 定义随机输入信号
p = con2seq(y);                             % 将随机输入向量转换为串行向量
delays = [1 2];                             % 定义 Adaline 神经元输入延迟量
t = p;                                      % 定义 Adaline 神经元的数目向量
 % 创建线性神经网络
net = newlin(minmax(y),1,delays,0.0005);
 % 线性神经网络的自适应调整(训练)
net.adaptParam.passes = 70;
[net,a,output] = adapt(net,p,t);            % 输出信号 output 为网络调整过程中的误差
 % 绘制随机输入信号/输出信号的波形
hold on;
subplot(3,1,1);plot(time,y,'r * - ');       % 输出信号 output 为网络调整过程中的误差
xlabel('t','position',[20.5, - 1.8]);
ylabel('随机输入信号 s(t)');
axis([0 20  - 2 2]);
subplot(3,1,2);
output = seq2con(output);
plot(time,output{1},'ko - ');               % 绘制预测输出信号的波形
xlabel('t','position',[20.5, - 1.8]);
ylabel('预测输出信号 y(t)');
axis([0 20  - 2 2]);
subplot(3,1,3);
e = output{1} - y;
plot(time,e,'k - ');                        % 绘制误差曲线
xlabel('t','position',[20.5, - 1.8]);
ylabel('误差曲线 e(t)');
axis([0 20  - 2 2]);
hold off;
```

运行程序，效果如图 3-45 所示。

从图中可以看出，输出信号波形与输入信号波形基本一致，误差较小，输出波形较好地预测了输入波形。

值得一提的是，在程序设计中，需要注意学习率和训练步长的选择。学习率过大，学

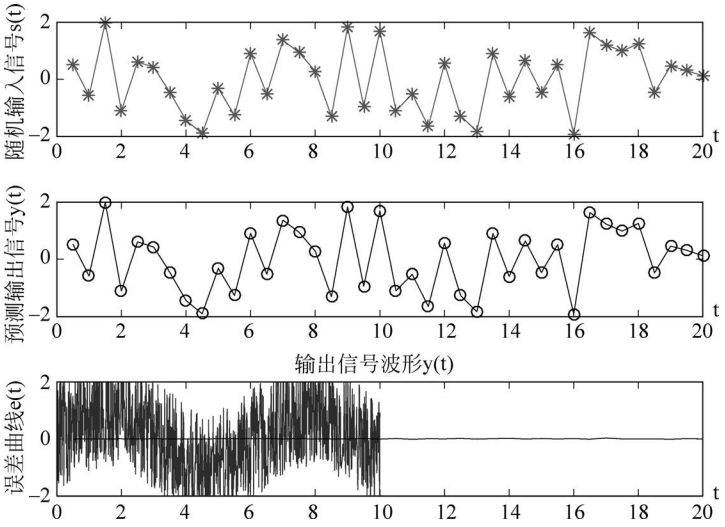

图 3-45　线性神经网络信号预测效果

习的过程将不稳定，且误差会更大；学习率过小，学习的过程将变慢，需要的训练步长数将加大。选择不当，将得不到满意的结果。