

### 三、*BP*网及*BP*算法

1. *BP*网络结构
2. *BP*学习算法
3. *BP*算法应用例子
4. 有关*BP*网和*BP*算法的讨论
5. *BP*算法的改进

# 1. *BP*网络结构

*BP*网与线性阈值单元组成的多层感知器网络结构完全相同，只是各隐节点的激活函数使用了第二章介绍的*Sigmoidal*函数，所以*BP*网也称隐节点激活函数采用*Sigmoidal*函数的多层感知器。*BP*网输出节点的激活函数根据应用的不同而异：如果多层感知器用于分类，则输出层节点一般用*Sigmoidal*函数或硬极限函数；如果多层感知器用于函数逼近，则输出层节点应该用线性函数。

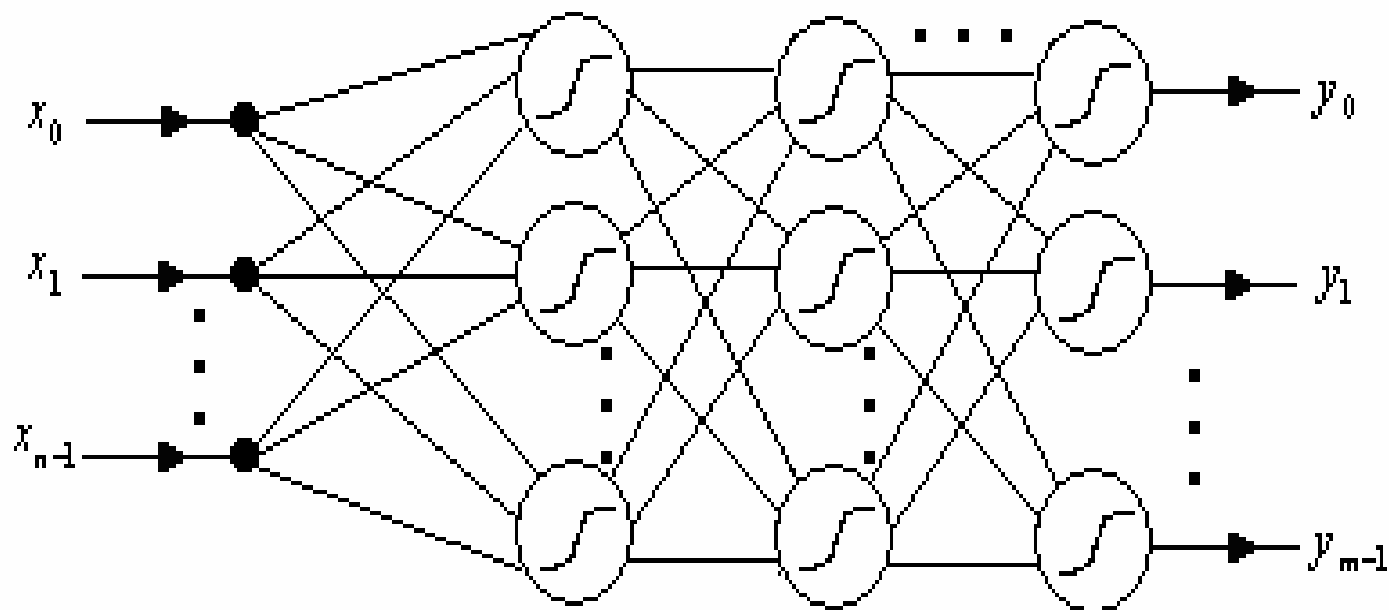


图3.8 隐节点和输出节点都使用*Sigmoidal*函数的BP网

与线性阈值单元组成的多层感知器一样，*BP*网采用多层结构，包括输入层、多个隐含层、输出层，各层间实现全连接。如图3.8所示为隐节点和输出节点都使用*Sigmoidal*激活函数的*BP*网（注意：网络中没有画出各神经元的阈值）。

下面讨论图3.8所示BP网的权值调节算法,即著名的BP算法。此时,对BP网的各计算节点,我们有:

$$u_j = \sum w_i x_i - \theta_j$$

$$y_j = f(u_j) = 1/(1 + \exp(-\lambda u_j))$$

( $f(\cdot)$ 是双极性S函数亦可,实际上,连续可微单调上升的S型函数均可)。注意

$$\begin{aligned} f'(u_j) &= \frac{\lambda \exp(-\lambda u_j)}{1 + \exp(-\lambda u_j)} \cdot \frac{1}{1 + \exp(-\lambda u_j)} \\ &= \lambda (1 - f(u_j)) f(u_j) \end{aligned}$$

假设BP网的输入矢量为  $x \in R^n$ ，其中  $x = (x_0, x_1, \dots, x_{n-1})^T$ ；

第一隐层有  $n_1$  个神经元，它们的输出为  $x' \in R^{n_1}$ ，其中

$x' = (x'_0, x'_1, \dots, x'_{n_1-1})^T$ ；第二隐层有  $n_2$  个神经元，输出为  $x'' \in R^{n_2}$ ， $x'' = (x''_0, x''_1, \dots, x''_{n_2-1})^T$ ；输出层有个  $m$  神经元，输出  $y \in R^m$ ， $y = (y_0, y_1, \dots, y_{m-1})^T$ 。又设输入层到第一隐层的权为  $w_{ij}$ ，阈值为  $\theta_j$ ；第一隐层到第二隐层的权为  $w'_{jk}$ ，阈值为  $\theta'_k$ ；第二隐层到输出层的权为  $w''_{kl}$ ，阈值为  $\theta''_l$ 。于是各层神经元输出为：

$$\begin{cases} y_l = f\left(\sum_{k=0}^{n_2-1} w_{kl}'' x_k'' - \theta_l''\right), l = 0, 1, \dots, m-1 \\ x_k'' = f\left(\sum_{j=0}^{n_1-1} w_{jk}' x_j' - \theta_k'\right), k = 0, 1, \dots, n_2-1 \\ x_j' = f\left(\sum_{i=0}^{n-1} w_{ij} x_i - \theta_j\right), j = 0, 1, \dots, n_1-1 \end{cases} \quad (3.16)$$

显然，它将完成 $n$ 维空间矢量到 $m$ 维空间的映射。

## 2. *BP*学习算法

*BP*算法也称误差反向传播算法（*Error Back-propagation Algorithm*），是一类有导学习算法，用于*BP*网的权值和阈值学习。

下面讨论所有训练样本同时用于网络权值调节（即批处理）的BP算法。设有 $P$ 个学习样本矢量，对应的期望输出为 $d^{(1)}, d^{(2)}, \dots, d^{(p)}$ ，学习是通过误差校正权值，使各 $y^{(p)}$ 接近 $d^{(p)}$ 。为简化推导，我们把各计算节点的阈值并入权矢量，即设 $\theta_l'' = w_{n_2 l}''$ ， $\theta_k' = w_{n_1 k}'$ ， $\theta_j = w_{nj}$ ， $x_{n_2}'' = x_{n_1}' = x_n = -1$ ，则（3.16）式中相应的矢量 $w$ ， $w'$ ， $w''$ ， $x$ ， $x'$ ， $x''$ 维数均增加1。



## 1) *BP*算法原理

该学习规则的推导仍是基于最小均方误差准则。

当一个样本（设为第  $p$  个样本）输入网络，并产生输出时，均方误差应为各输出单元误差平方之和：

$$E^{(p)} = \frac{1}{2} \sum_{l=0}^{m-1} \left( d_l^{(p)} - y_l^{(p)} \right)^2 \quad (3.17)$$

当所有样本都输入一次后，总误差为：

$$E_A = \sum_{p=1}^P E^{(p)} = \frac{1}{2} \sum_{p=1}^P \sum_{l=0}^{m-1} \left( d_l^{(p)} - y_l^{(p)} \right)^2 \quad (3.18)$$

设 $w_{sq}$ 为网络中的任一连接权值，则根据梯度下降法，误差修正量应为：

$$\Delta w_{sq} = -\eta \frac{\partial E_A}{\partial w_{sq}} \text{-----批处理，或}$$

$$\Delta w_{sq} = -\eta \frac{\partial E_p}{\partial w_{sq}} \text{-----增量修正}$$

以批处理为例：

(1) 对输出层：

$$w_{kl}''(n_0 + 1) = w_{kl}''(n_0) - \eta \frac{\partial E_A}{\partial w_{kl}''} \quad (3.19)$$

其中， $n_0$ 为迭代次数。 $\frac{\partial E_A}{\partial w_{kl}''}$ 求解过程如图3.9：

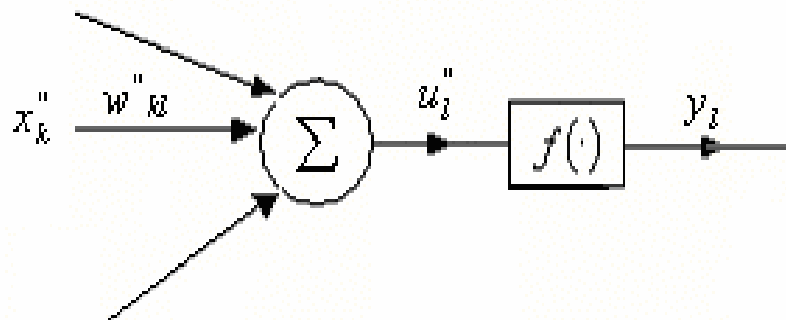


图3.9 输出层权值增量求解示意图

$$\frac{\partial E_A}{\partial w_{kl}''} = \sum_{p=1}^P \frac{\partial E^{(p)}}{\partial w_{kl}''} \quad (E^{(p)} \text{ 是 } y_0, y_1, \dots, y_{m-1} \text{ 的函数, 但 } w_{kl}'' \text{ 只影响 } y_l)$$

$$= \sum_{p=1}^P \frac{\partial E^{(p)}}{\partial y_l^{(p)}} \frac{\partial y_l^{(p)}}{\partial u_l^{''(p)}} \frac{\partial u_l^{''(p)}}{\partial w_{kl}''}$$

$$\left( u_l^{''(p)} = \sum_{k=0}^{n_2} w_{kl}'' x_k^{''(p)}, y_l^{(p)} = f(u_l^{''(p)}), E^{(p)} = \frac{1}{2} \sum_{l=0}^{m-1} (d_l^{(p)} - y_l^{(p)})^2 \right)$$

$$= - \sum_{p=1}^P (d_l^{(p)} - y_l^{(p)}) f'(u_l^{''(p)}) x_k^{''(p)} \quad (f'(u_l^{''(p)}) = y_l^{(p)}(1 - y_l^{(p)}))$$

$$= - \sum_{p=1}^P (d_l^{(p)} - y_l^{(p)}) y_l^{(p)} (1 - y_l^{(p)}) x_k^{''(p)}$$

$$= - \sum_{p=1}^P \delta_{kl}^{(p)} x_k^{''(p)} \quad \text{----- } \delta \text{ 学习规则}$$

其中：  $\delta_{kl}^{(p)} = (d_l^{(p)} - y_l^{(p)}) y_l^{(p)} (1 - y_l^{(p)})$

所以有：  $w_{kl}''(n_0 + 1) = w_{kl}''(n_0) + \eta \sum_{p=1}^P \delta_{kl}^{(p)} x_k''^{(p)}$  (3.20)

(2) 对中间隐层：

$$w_{jk}'(n_0 + 1) = w_{jk}'(n_0) - \eta \frac{\partial E_A}{\partial w_{jk}'} \quad (3.21)$$

其中  $\frac{\partial E_A}{\partial w_{jk}'}$  求解过程如下图3.10:

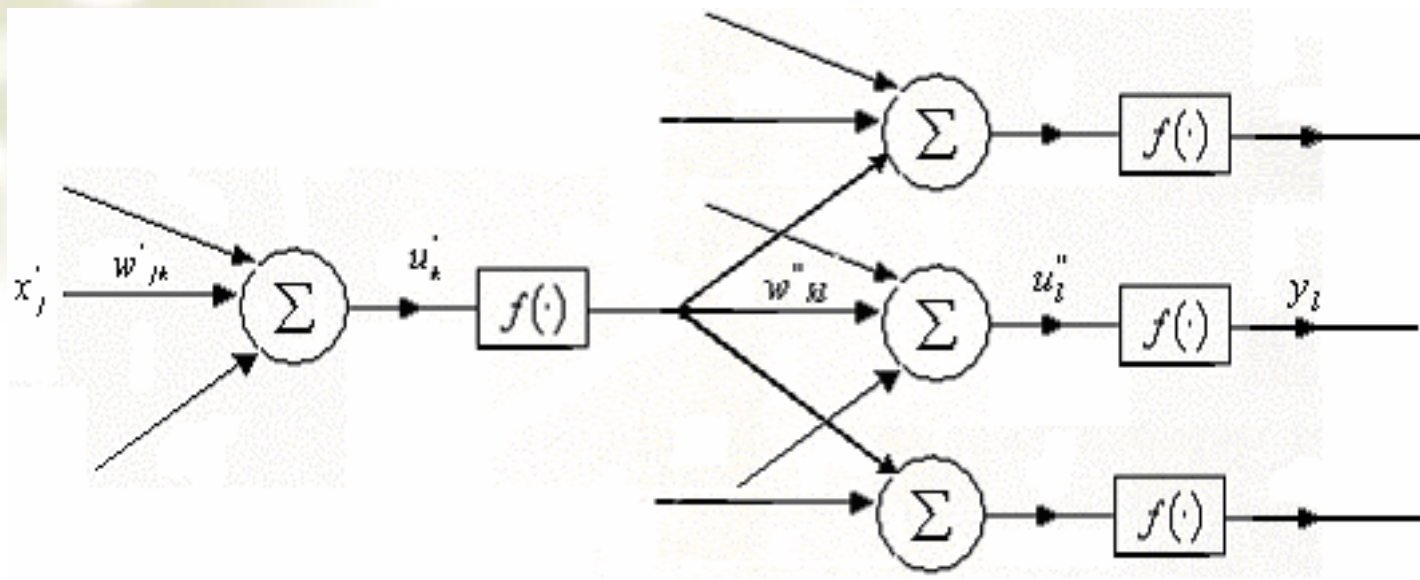


图3.10 中间隐层权值增量求解示意图

$$\frac{\partial E_A}{\partial w'_{jk}} = \sum_{p=1}^P \frac{\partial E^{(p)}}{\partial w'_{jk}} \quad (E(p) \text{ 是 } y_0, y_1, \dots, y_{m-1} \text{ 的函数, 每个 } y_l \text{ 都受 } w'_{jk} \text{ 影响})$$

$$= \sum_{p=1}^P \sum_{l=0}^{m-1} \frac{\partial E^{(p)}}{\partial y_l^{(p)}} \frac{\partial y_l^{(p)}}{\partial u_l^{''(p)}} \frac{\partial u_l^{''(p)}}{\partial x_k^{''(p)}} \frac{\partial x_k^{''(p)}}{\partial u_k^{'(p)}} \frac{\partial u_k^{'(p)}}{\partial w'_{jk}}$$

$$= - \sum_{p=1}^P \sum_{l=0}^{m-1} (d_l^{(p)} - y_l^{(p)}) f'(u_l^{''(p)}) w_{kl}'' x_k^{''(p)} (1 - x_k^{''(p)}) x_j^{'(p)}$$

$$= - \sum_{p=1}^P \sum_{l=0}^{m-1} \delta_{kl}^{(p)} w_{kl}'' x_k^{''(p)} (1 - x_k^{''(p)}) x_j^{'(p)}$$

$$= - \sum_{p=1}^P \delta_{jk}^{(p)} x_j^{'(p)}$$

其中：
$$\delta_{jk}^{(p)} = \sum_{l=0}^{m-1} \delta_{kl}^{(p)} w_{kl}'' x_k^{''(p)} (1 - x_k^{''(p)})$$

所以有 
$$w_{jk}'(n_0 + 1) = w_{jk}'(n_0) + \eta \sum_{p=1}^P \delta_{jk}^{(p)} x_j^{'(p)} \quad (3.22)$$

同理可得第一隐层的权值修正公式为：

$$w_{ij}(n_0 + 1) = w_{ij}(n_0) + \eta \sum_{p=1}^P \delta_{ij}^{(p)} x_i^{(p)} \quad (3.23)$$



其中：

$$\delta_{ij}^{(p)} = \sum_{k=0}^{n_2} \delta_{jk}^{(p)} w'_{jk} x_j^{(p)} (1 - x_j^{(p)})$$

注意，对增量式修正，上面各式中各权值的修正量是一项，而不是 $p$ 从1至 $P$ 的 $\Sigma$ 求和项。

显然，学习分两个阶段：（1）由前向后正向计算各隐层和输出层的输出；（2）由后向前误差反向传播以用于权值修正。

## 2) *BP*算法的各步骤

- (1) 权值初始化:  $w_{sq} = \text{Random}(\cdot)$  ,  $sq$  为  $ij$ ,  $jk$  或  $kl$ 。
- (2) 依次输入  $P$  个学习样本。设当前输入第  $p$  个样本。
- (3) 依次计算各层的输出:  $x'_j$ ,  $x''_k$  及  $y_l$  ,  $j=0,1,\dots,n_1$ ,  
 $k=0,1,\dots,n_2$ ,  $l=0,1,\dots,m-1$ 。
- (4) 求各层的反传误差:

$$\delta_{kl}^{(p)} = (d_l^{(p)} - y_l^{(p)}) y_l^{(p)} (1 - y_l^{(p)}) \quad , \quad l = 0, 1, \dots, m-1$$

$$\delta_{ij}^{(p)} = \sum_{k=0}^{n_2} \delta_{jk}^{(p)} w'_{jk} x_j^{(p)} (1 - x_j^{(p)}) \quad , \quad k = 0, 1, \dots, n_2$$

$$\delta_{jk}^{(p)} = \sum_{l=0}^{m-1} \delta_{kl}^{(p)} w''_{kl} x_k^{(p)} (1 - x_k^{(p)}) \quad , \quad j = 0, 1, \dots, n_1$$

并记下各个  $x_k^{(p)}$ ,  $x_j^{(p)}$ ,  $x_i^{(p)}$  的值。

(5) 记录已学习过的样本个数 $p$ 。如果 $p < P$ ，转步骤(2)继续计算；如果 $p = P$ ，转步骤(6)。

(6) 按权值修正公式修正各层的权值或阈值。

(7) 按新的权值再计算  $x'_j$  ,  $x''_k$  ,  $y_l$  和  $E_A$  , 若对每个  $p$  和  $l$  都满足  $|d_l^{(p)} - y_l^{(p)}| < \varepsilon$  (或  $E_A < \varepsilon$ ) , 或达到最大学习次数, 则终止学习。否则转步骤(2)继续新一轮的学习。

上述算法实现的一个完整的例子可参考10章灵敏度剪枝算法。

### 3. *BP*算法应用例子

这一节我们给出*BP*算法训练的三个例子，分别对于于单个样本的训练、模式分类、及函数逼近的情况。

## 1) 单个样本训练的例子

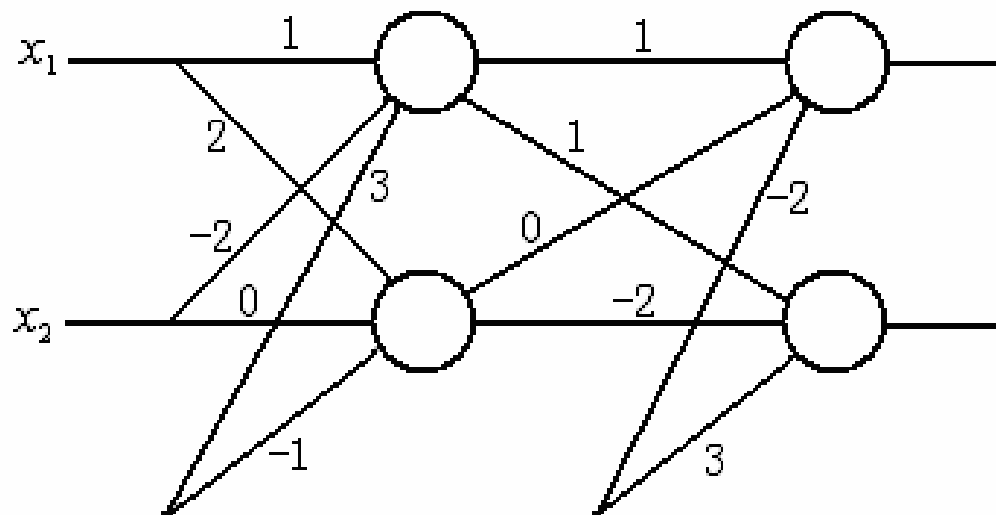


图3.11 三层BP网

图3.11所示的三层BP网，隐层和输出层采用单极性Sigmoid激活函数，即  $f(u) = \frac{1}{1 + e^{-u}}$ ，权系数和偏移的初始值如图所示。模式输入为  $[x_1, x_2]^T = [1, 3]^T$ ，对应的期望输出为  $[d_1, d_2]^T = [0.9, 0.3]^T$ 。学习参数设定如下：

学习率  $\eta = 0.5$ ，目标误差  $\varepsilon = 0.0001$ ，

最大学习次数5000。

于是BP算法的学习过程如下：先正向传播计算网络输出，对第一次叠代可得隐节点输出为  $x' = [0.1192, 0.7311]^T$ ，加偏移增维后为  $x' = [0.1192, 0.7311, 1.0000]^T$ ，于是网络输出为  $[y_1, y_2]^T = [0.1323, 0.8398]^T$ ，此时网络输出与目标输出的误差平方和为  $SSE=0.8808$ 。再反向传播误差调整权值：输出层的反向传播误差为  $\delta_2 = [0.0881, -0.0726]^T$ ，隐层的反向传播误差为  $\delta_1 = [0.0016, 0.0286]^T$ ，于是隐层到输出层的扩展权值调整量为



$dW_2 = \begin{bmatrix} 0.0105, -0.0644, 0.0881 \\ -0.0087, -0.0531, -0.0726 \end{bmatrix}^T$ ，输入层到隐层的扩展权值

调整量  $dW_1 = \begin{bmatrix} 0.0016, 0.0049, 0.0016 \\ 0.0286, 0.0867, 0.0286 \end{bmatrix}^T$ ，调节后的两个扩展

权值分别为  $W_1 Ex = \begin{bmatrix} 1.0008, -1.9976, 3.0008 \\ 2.0143, 0.0428, -0.9857 \end{bmatrix}$ ，

$W_2 Ex = \begin{bmatrix} 1.0053, 0.0322, -1.9599 \\ 0.9957, -2.0265, 2.9637 \end{bmatrix}$ 。第一次叠代后

的  $W_1 = \begin{bmatrix} 1.0008, -1.9976 \\ 2.0143, 0.0428 \end{bmatrix}$ ， $B_1 = \begin{bmatrix} 3.0008 \\ -0.9857 \end{bmatrix}$ ， $W_2 = \begin{bmatrix} 1.0053, 0.0322 \\ 0.9957, -2.0265 \end{bmatrix}$ ，

$B_2 = \begin{bmatrix} -1.9599 \\ 2.9637 \end{bmatrix}$ ，用此组新权值重新计算网络输出与目

标输出的误差平方和，可得  $SSE=0.8510$ ，已经有了

很大的减小。

经过160次叠代后误差平方和 $SSE$ 小于目标值，于是算法停止。

## 2) 三层 $BP$ 网用于三分类

下面是一个两概念（一个三角形和一个矩形）学习的例子，这是一个典型的多分类例子。

我们在 $(-2,2)*(-2,2)$ 范围内随机产生200个均匀分布的样本，如图3.12所示。我们规定三角形内的样本属于 $c_2$ 类（“+”号），矩形内的样本属 $c_3$ 类（“□”号），其余样本属于 $c_1$ 类（“○”号）。

我们可以用一个2输入3输出的三层BP网对三类样本进行学习，网络的隐节点数选为10。令 $c_1$ 类样本的目标输出为  $[1 \ 0 \ 0]^T$ ， $c_2$ 类样本的目标输出为  $[0 \ 1 \ 0]^T$ ， $c_3$ 类样本的目标输出为  $[0 \ 0 \ 1]^T$ 。

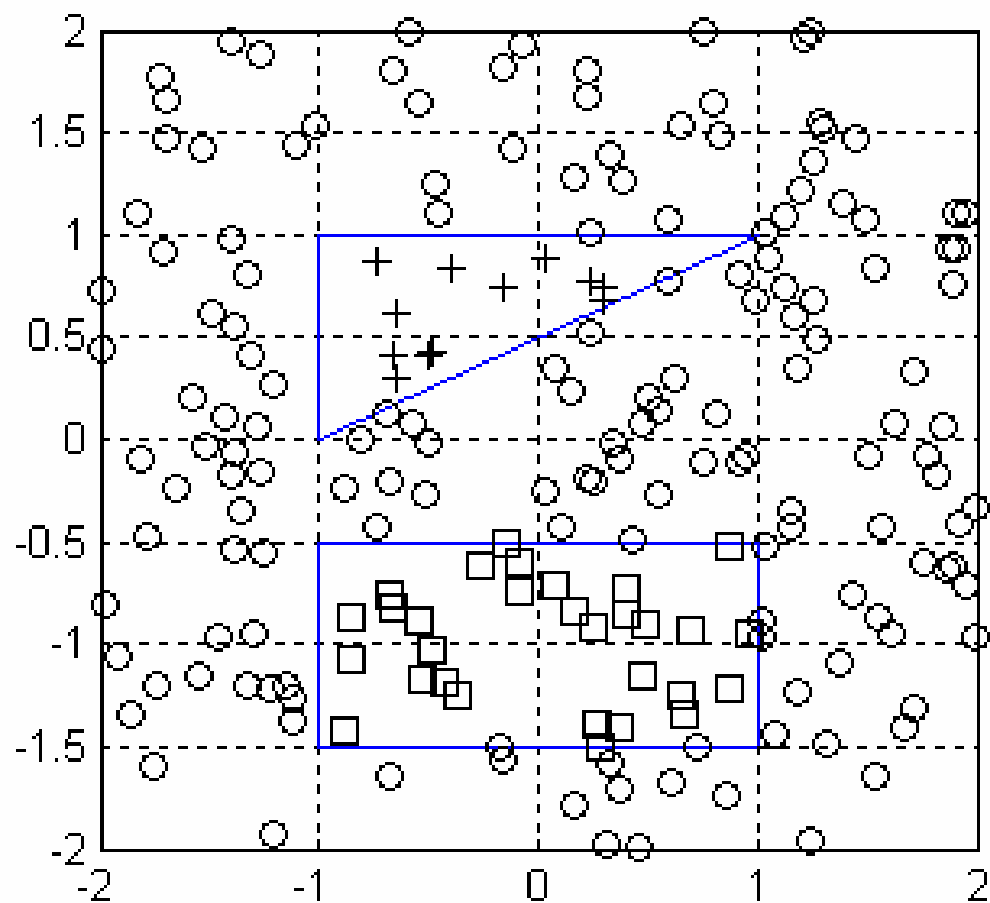


图3.12 概念学习样本

学习参数设定如下：神经网络隐层和输出层采用标准 *Sigmoid* 激活函数，隐节点数取10，学习率  $\eta = 0.1$ ，目标误差  $\varepsilon = 0.01$ ，最大学习次数10000，初始权值和偏移取 $[-0.1, 0.1]$ 内随机值。

经过10000次学习后，我们对神经网络进行测试：用同样方法产生5000个样本，判断分类正确率。

图3.13为某次试验的训练误差—训练次数曲线（学习曲线）。训练结束后对所有200个样本的误差平方和为2.144。

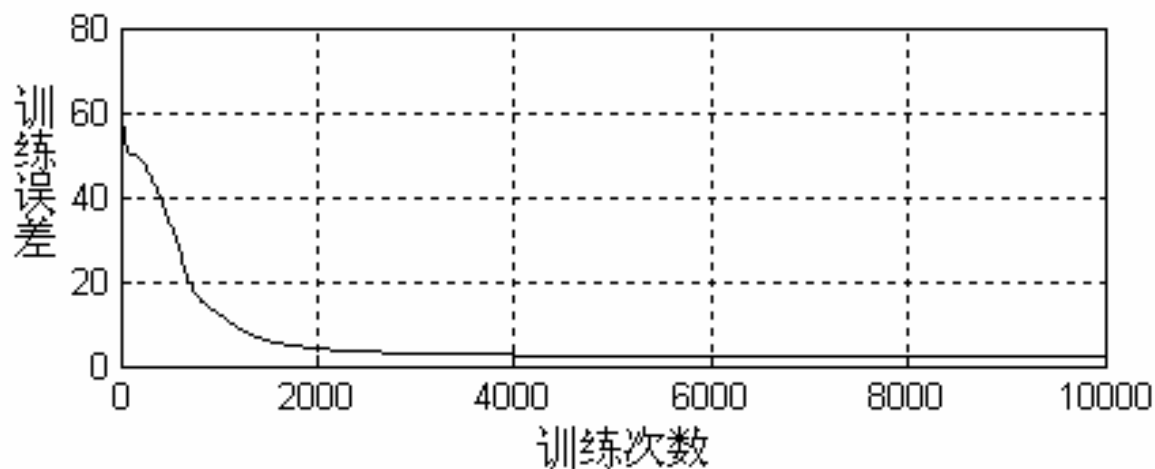


图3.13 神经网络用于分类的学习曲线

表3.1为该次试验对5000个样本的测试结果，表中， $C_i$ 表示测试样本中被分为 $c_i$ 类的样本数， $C_{ij}$ 表示被分为 $c_i$ 、但被学习系统识别为 $c_j$ 的样本数，其中 $i,j=1,2,3$ 。

表3.1 神经网络对5000个测试样本的识别

总测试样本数：5000									正确率
$C_1=4100$			$C_2=242$			$C_3=648$			92.48%
$C_{11}$	$C_{12}$	$C_{13}$	$C_{21}$	$C_{22}$	$C_{23}$	$C_{31}$	$C_{32}$	$C_{33}$	
3909	103	98	45	197	0	130	0	518	

注意，由于神经网络受初始权值的影响较大，如果初始权值的不同，每次训练的结果都可能不同。

### 3) 三层**BP**网用于用于函数逼近

考虑以下*Hermit*多项式的逼近问题：

$$F(x) = 1.1(1 - x + 2x^2) \exp\left(-\frac{x^2}{2}\right) \quad (3.24)$$



其中  $x \in R$ 。训练样本产生方式如下：样本数  $N=100$ ，其中样本输入  $x_i$  服从区间  $[-4,4]$  内的均匀分布，样本输出为  $F(x_i)+e_i$ ， $e_i$  为添加的噪声，服从均值为0，方差为0.1的正态分布。产生的目标函数和一组样本（即图中的“+”号）如下图3.14所示。

对于该函数逼近问题，我们可以用一个1输入1输出的三层BP网对样本进行拟合，网络的隐节点数选为10。

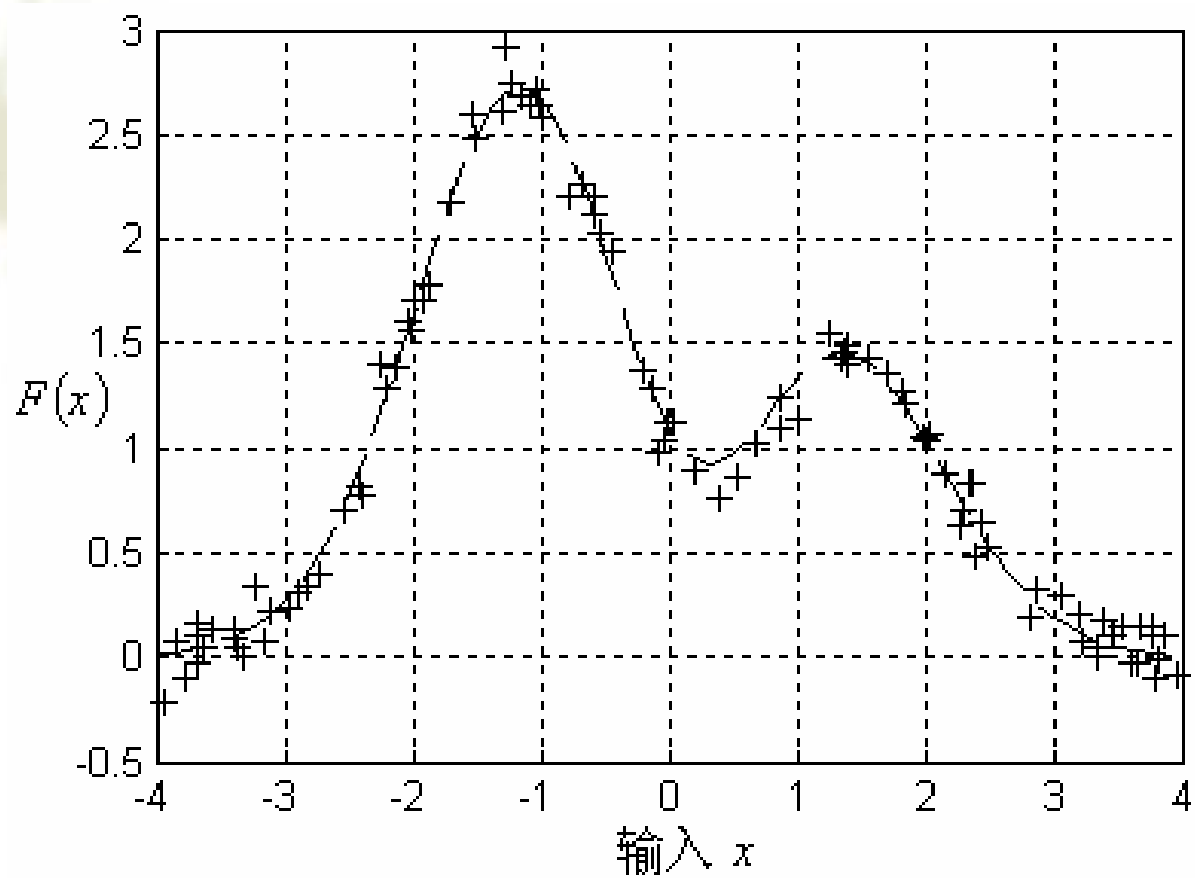


图3.14 目标函数和一组训练样本

其它学习参数设定如下：神经网络隐层采用标准*Sigmoidal*激活函数，输出层采用线性激活函数，即  $f(u) = u$  。

学习率  $\eta = 0.003$  ， 目标误差  $\varepsilon = 0.5$  ， 最大学习次数20000，初始权值和偏移取[-0.1,0.1]内随机值。

图3.15为BP算法对图3.14中训练样本的学习曲线，训练结束后所有100个样本的训练误差平方和为0.081。

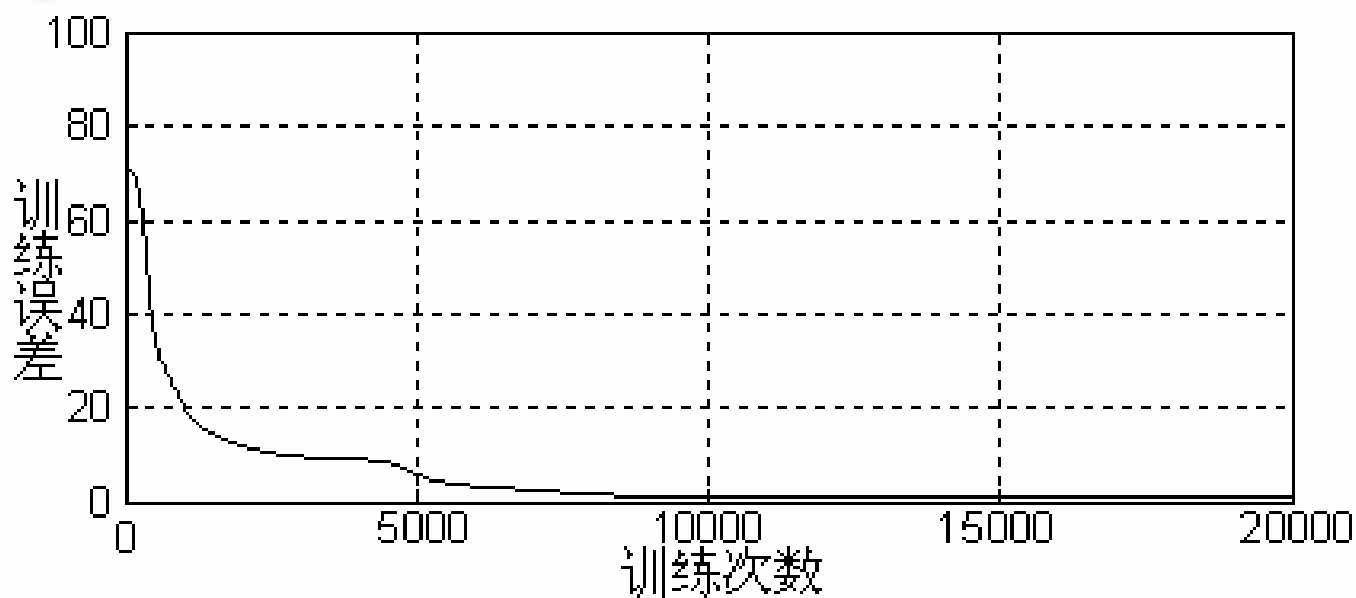


图3.15 神经网络用于函数逼近的学习曲线

对同一批样本，下图3.16同时绘制了训练完成后的神经网络函数曲线与目标曲线。其中虚线部分为目标函数曲线，实线为神经网络的输入输出映射函数，“+”为样本。可见两条曲线是非常接近的

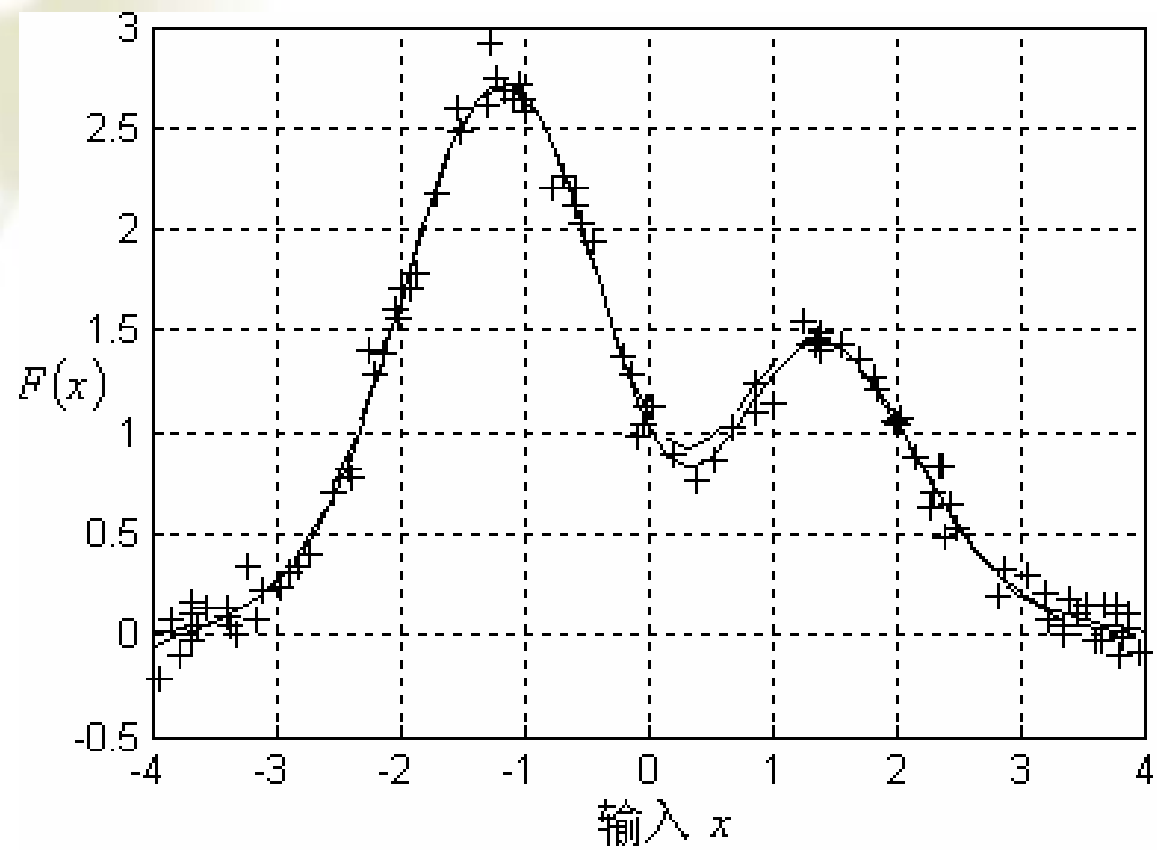


图3.16 训练完成后的神经网络函数曲线

## 4.有关BP网和BP算法的讨论

### 1) BP网用于函数逼近与分类的差别

BP网输出节点的激活函数根据应用的不同而异：如果BP网用于分类，则输出层节点一般用 *Sigmoidal* 函数或硬极限函数；如果BP网用于函数逼近，则输出层节点应该用线性函数，即 $y = f(u) = u$ 。

## 2) 网络的逼近能力

当神经网络的结构和权值确定后，网络从输入到输出就成了一个非线性映射。许多人证明了以下的所谓的**万能逼近定理**：含一个隐层的 $MLP$ 网络（输出层为线性神经元），只要隐节点数足够多，则该网络能以任意精度逼近有界区域上的任意连续函数。

我国陈天平指出，神经元函数的有界性（而不是连续性）是保证任意逼近的充分条件。



尽管万能逼近定理说明了神经网络有一个隐层就能实现任意逼近，这并不是说三层的网络结构就是最合理的。事实上，对同一目标函数，有人发现四层的BP网有时候可能比三层的BP网使用更少的神经元。

### 3) 样本预处理

当网络用于分类时，样本的期望输出值 $d_l = 0$ 为 $d_l = 1$ 或，但由于 $y_l$ 值在 $u = \pm \infty$ 时才为0或1，这有可能将某些网络权值驱向无穷大。为了避免这种饱和现象，期望输

出可适当放宽，如 $y_l > 0.9$ 为1， $y_l < 0.1$ 为0，即每个样本的期望误差定义为  $\varepsilon = (0.1)^2 = 0.01$  。

样本输入也必须进行归一化处理，使归一化后的样本输入均值为零。另外，应使用主成分分析等方法，尽量使各输入变量不相关，各输入变量的协方差也接近相等，以确保各权值的收敛速度大致相同。

#### 4) 网络的结构选择

这里的网络结构选择包括三方面的内容：输入层和输出层节点数选择、网络隐层数的选择，及每个隐层隐节点数的选择。

输入层和输出层节点数选择由应用要求决定。输入节点数一般等于要训练的样本矢量维数，可以是原始数据的维数或提取的特征维数；输出单元数在分类网络中取为类别数  $m$ ，或  $\log_2 m$ ，在逼近网络中取为要逼近的函数输出空间维数。当网络用于工业过程辨识或时间序列预测建模时，输入节点数的选择应由非线性系统定阶的结果确定。

网络的隐层数和隐节点数决定了网络的规模，而网络的规模与其性能密切相关。神经网络的规模越大，网络中的自由参数就越多；反之，网络中的自由参数就越少。如果神经网络用于逼近一个目标函数（分类可以看成函数逼近的特殊情况），则当网络规模过小，神经网络逼近能力不足，容易导致欠拟合；网络规模过大，神经网络逼近能力过剩，则容易导致过拟合。因此，确定网络规模是神经网络设计的一项重要内容。

我们将在以后讨论网络结构对网络性能的影响。

## 5) 增量学习和批学习

批处理时,  $\Delta w = \sum_{p=1}^P \Delta w^{(p)}$ , 不存在输入模式次序问题, 算法稳定性好, 是有平均效应的梯度下降法,  $\eta$  一般取较小值; 增量处理适合于在线处理, 但要求训练模式输入有足够的随机性, 而且增量处理对输入模式的噪声比较敏感, 即对剧烈变化的输入模式, 效果较差。

## 6) 激励函数的形式

上面BP算法推导采用了单极性Sigmoidal函数，即

$$f(u) = 1/(1 + \exp(-u)) \text{ ,此时 } f'(u) = y_l(1 - y_l) \text{ ,}$$

$$\delta_l = (d_l - y_l)y_l(1 - y_l) \text{ 。}$$

我们也可以采用双极Sigmoid函数，此时  $f(u) = 2/(1 + \exp(-u)) - 1$  ,

$$f'(u) = \frac{1}{2}(1 - y_l^2) \text{ , } \delta_l = \frac{1}{2}(d_l - y_l)(1 - y_l^2) \text{ 。}$$

当然，如果我们采用其它连续可微函数，也可用于构造其它类型的前馈网络模型。

另外，神经元函数的斜率由  $f(u) = 1/(1 + \exp(-\lambda u))$  中的  $\lambda$  确定， $\lambda$  值越大，激励函数就越陡峭， $f'(u)$  就越大，权值的调节量  $|\Delta w|$  就越大。用可变  $\lambda$  的可摆脱局部极小点。但大的  $\lambda$  值相当于大  $\eta$  值（学习常数），因此不如固定  $\lambda$ ，只调节  $\eta$  值。



## 7) 误差函数的选择

前面我们采用的误差函数为  $E_A = \frac{1}{2} \sum_{p=1}^P \sum_{l=0}^{m-1} (d_l^{(p)} - y_l^{(p)})^2$ ，其中  $P$  为训练模式数， $m$  为输出层神经元数。显然，如果  $P$ 、 $m$  不同时， $E_A$  值也不同。为客观地比较两种网络地学习性能，可采用归一化的目标函数，即

$$E'_A = \frac{1}{Pm} \sqrt{\sum_{p=1}^P \sum_{l=0}^{m-1} (d_l^{(p)} - y_l^{(p)})^2} \quad (3.25)$$

其中  $\varepsilon$  值的确定与应用有关。



网络用于分类时，还可以用如下地分类误差来判断收敛与否： $E'' = \frac{N_e}{Pm}$ ，其中 $N_e$ 为全部样本输入后，不符合期望输出要求的实际输出分量个数。当然，具体采用哪种目标函数形式，还要考虑具体应用的情况。

## 8) 初始权值的选取

对BP网而言，网络的初始权值不同，每次训练的结果也不同，这是由于误差曲面的局部最小点非常多，BP算法本质上是梯度算法，易陷入局部最小点。一般情况下，网络的初始权值要取小的随机值，既保证各神经元的输入  $u$  值较小，工作在激励函数斜率变化最大的区域，也防止多次连续学习后，某些权值的绝对值不合理的无限增长。

关于初始权值对神经网络泛化能力的影响，以及相关的改进方法，我们将在以后介绍。

## 9) 学习率（步长） $\eta$

在最优的梯度法中  $\eta$  应是可变的，是一个一维搜索的结果。但BP网络复杂， $E_A$ 是非常复杂的非线性函数，这使得求最优  $\eta$  难度很大，计算量也很大。应用实例表明， $\eta$  可取值  $10^{-3} \sim 10$  不等。

一般要求是，当训练到误差曲面得平坦区时，为加快收敛应使  $\eta$  增大；当训练到误差曲面得变化剧烈区时，为防止过学习（使误差增加），应使  $\eta$  减小。为加快收敛，应使  $\eta$  合理化，比如采用变步长算法。

还要注意到，小  $\eta$  值才能保证权值修正是真正沿梯度下降方向。

## 5. *BP*算法的改进

评价一个神经网络学习算法的优劣可以有很多指标：

1) 学习所需的时间；2) 泛化能力。3) 神经网络的结构复杂性。4) 鲁棒性，即算法的学习参数在很大范围变化，算法是否仍能较好地学习。

通常认为，*BP*算法有以下几个缺陷：1) 易陷入局部极小点；2) 收敛速度慢；3) 所设计神经网络地泛化能力不能保证。改进泛化能力的方法我们将在第五章以后讨论，这里主要介绍避免局部最小和提高收敛速度的一些改进方法。

## 1) 加动量算法

为加速算法收敛，可考虑引入动量项，即：

$$w(n_0 + 1) = w(n_0) + \eta(n_0)d(n_0) + \alpha\Delta w(n_0) \quad (3.26)$$

其中： 
$$d(n_0) = -\frac{\partial E_A}{\partial w(n_0)}$$

$$\Delta w(n_0) = w(n_0) - w(n_0 - 1) = \eta(n_0 - 1)d(n_0 - 1)$$

这时权值修正量加上了有关上一时刻权值修改方向的记忆。动量因子  $\alpha$  一般取值0.1~0.8。

动量法对收敛性的改进可用下图3.17作定性说明。

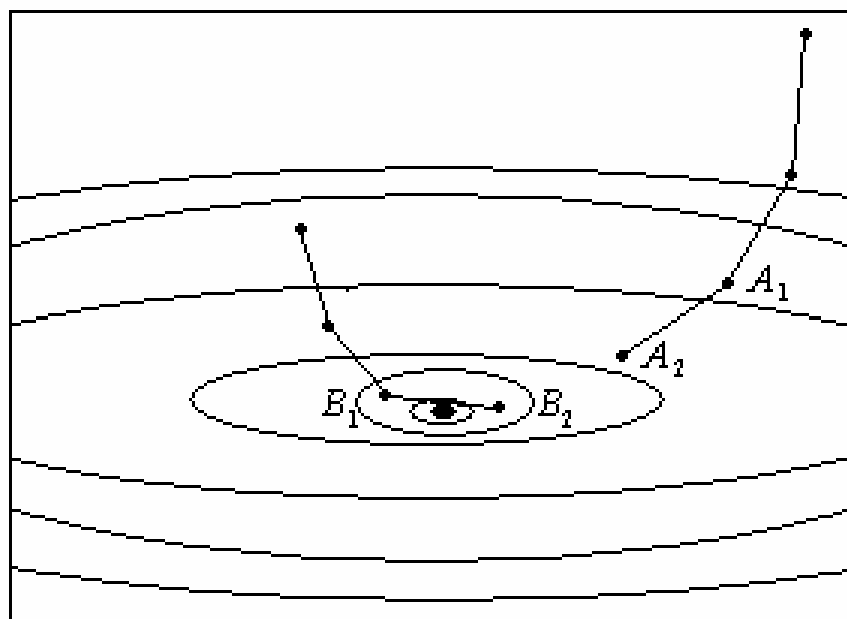


图3.17 动量算法原理示意图

该图是二维误差曲面的俯视图。若 $n_0$ 次迭代在 $A_1$ 点， $n_0+1$ 次迭代在 $A_2$ 点，由于两点处的梯度方向（相邻叠代点之间的连线方向）是一致的，因此动量项可加速 $A_2$ 点的收敛；而若 $n_0$ 和 $n_0+1$ 次迭代分别在 $B_1$ 、 $B_2$ 点，这两点的梯度方向是相反的，这时表明  $-\nabla E(n_0)$  和  $-\nabla E(n_0 + 1)$  都没有准确指向极小点  $M$ ，因此在 $B_2$ 点将修正方向定为  $-\eta \nabla E(n_0 + 1) + \alpha \Delta w(n_0)$ ，使收敛方向更准确地指向 $M$ 点。

当然，对后一种情况应慎重对待。一般建议在碰到  $\Delta E > 0$ ， $\eta(n_0)$  应减小时，应让  $\alpha = 0$  而去掉动量项，因为原  $-\nabla E(n_0 - 1)$  搜索方向很可能完全不合理；而当  $\Delta E < 0$ ， $\eta(n_0)$  应增加时，可让  $\alpha$  再恢复。

## 2) 牛顿法

常规的 **BP** 算法修正权值时只用到了误差函数对权值的梯度，即一阶导数的信息。如果采用二阶导数信息进行权值调整（即牛顿法），则可以加速收敛。



假定神经网络权值修正的目标是极小化误差函数 $E(w)$ ，且网络的当前权值为 $w(t)$ ，权值修正量为 $\Delta w(t)$ ，于是下一时刻的权值  $w(t+1) = w(t) + \Delta w(t)$ ，则对 $E(w(t+1))$ 进行二阶泰勒展开，得：

$$E(w(t+1)) \approx E(w(t)) + g^T(t) \Delta w(t) + \frac{1}{2} \Delta w^T(t) A(t) \Delta w(t) \quad (3.27)$$

其中 $g(t)$ 为 $E(w)$ 的梯度向量，方阵 $A(t)$ 称为的Hessian阵，其元素值为 $E(w)$ 对各权值的二阶导数，即  $A_{ij}(t) = \frac{\partial^2 E(w)}{\partial w_i \partial w_j}$ 。于是权值修正后的误差函数变化量为：

$$\Delta E(t) = g^T(t)\Delta w(t) + \frac{1}{2}\Delta w^T(t)A(t)\Delta w(t) \quad (3.28)$$

我们期望通过变动  $\Delta w(t)$  使上式达最小。显然，当满足

$$\Delta w(t) = -A^{-1}(t)g(t) \quad (3.29)$$

时， $\Delta E(t)$  取得最小值。这就是牛顿法的基本原理。

在实际应用中，*Hessian*阵  $A(t)$  的计算既求逆都比较麻烦，且  $A(t)$  还有可能出现病态的情况，因此还有许多改进的方法。

### 3) 变步长算法

采用一般的固定长梯度下降法求解时，起码可能导致两个主要问题：局部极小解；收敛速度慢。因此可以考虑采用变步长算法。

最简单的变步长算法是先设一初始步长  $\eta_0$ ，若一次迭代后误差  $E$  增大，则将步长乘以  $\beta < 1$ ，沿原方向重新计算下一个迭代点；若一次迭代后  $E$  减小，则将步长乘以  $\alpha > 1$ 。即：

$$\begin{cases} \eta = \alpha \eta_0, \alpha > 1, & \text{当 } \Delta E < 0 \text{ 时} \\ \eta = \beta \eta_0, \beta < 1, & \text{当 } \Delta E > 0 \text{ 时} \end{cases}$$

这种处理方法计算量较小，但有时效果不明显，一般用于批处理 **BP** 算法。

另一种在增量式BP算法中使用的相对复杂但效果较好的方法是每次学习时都寻找最优步长。

$$\text{由于 } E_p = \frac{1}{2} \sum_{l=0}^{m-1} \left( d_l^{(p)} - y_l^{(p)} \right)^2 ,$$

$$m=1 \text{ 时 } E_p = \frac{1}{2} \left( d^{(p)} - f(w^T z) \right)^2 , \text{ 从而:}$$

$$\Delta w^{(p)} = -\eta \frac{\partial E_p}{\partial w^{(p)}} = \eta (d^{(p)} - f(w^{(p)T} z)) f'_w(w^{(p)T} z)$$

权值修正后的 $y^{(p)}$ 输出值  $f(w^{(p+1)T} z)$  可展开为一阶泰勒级数:

$$f(w^{(p+1)T} z) \approx f(w^{(p)T} z) + \left\{ f'_w(w^{(p)T} z) \right\}^T \Delta w^{(p)}$$

对最优步长  $\eta_0$ ，应使权值修正后  $y^{(p)} = f(w^{(p+1)T} z)$  的非常接近  $d^{(p)}$  值，即：

$$d^{(p)} \approx f(w^{(p+1)T} z) \approx f(w^{(p)T} z) + \left\{ f'_w(w^{(p)T} z) \right\}^T \Delta w^{(p)}$$

将  $\Delta w^{(p)}$  的修正公式代入上式，有：

$$d^{(p)} - f(w^{(p)T} z) \approx \eta (d^{(p)} - f(w^{(p)T} z)) \left\| f'_w(w^{(p)T} z) \right\|^2$$

即最优的步长为：

$$\eta_{opt} = \frac{1}{\left\| f'_w(w^{(p)T} z) \right\|^2} = \frac{1}{\left[ f'_u(u) \right]^2 \|z\|^2}$$

一般取为：

$$\eta_{opt} = \frac{\lambda}{[f'_u(u)]^2 \|z\|^2}$$

#### 4) 竞争BP 算法

*J.A.Freeman*基于对生物神经网络的研究，提出在BP网络的隐层采用竞争学习，以避免算法陷入局部极小点，最后获得全局最优解。具体步骤是计算出隐层各单元的  $\delta$  误差信号后，比较一下，具有最大  $\delta$  值的神经元对应的权矢量进行正常修正，而其它神经元的权值矢量都向与最大单元相反的方向修正。即隐层各单元的  $\delta$  误差信号用如下的  $\varepsilon_j$  误差信号取代：

$$\varepsilon_j = \begin{cases} \max(\delta_j) = \delta_j \\ -\frac{1}{4} \max(\delta_j) = -\frac{1}{4} \delta_j \end{cases}$$

从而权值调整公式为：

$$w_{ji}(n_0 + 1) = w_{ji}(n_0) + \eta \varepsilon_j(n_0) x_j(n_0)$$

## 5) 模拟退火方法

使用模拟退火方法主要是为了避免陷入局部极小。当 $E$ 长久不下降且值较大时，就很可能是局部极小，这时就可设置初试高温 $T$ ，从当前状态  $\mathbf{w}$  开始模拟退火。模拟退火步骤如下：



(1) 在所有的权上加一个噪声，获得新状态  $w'$ （在状态  $w$  的邻域内）。

(2) 计算  $\Delta E = E_{w'} - E_w$ 。

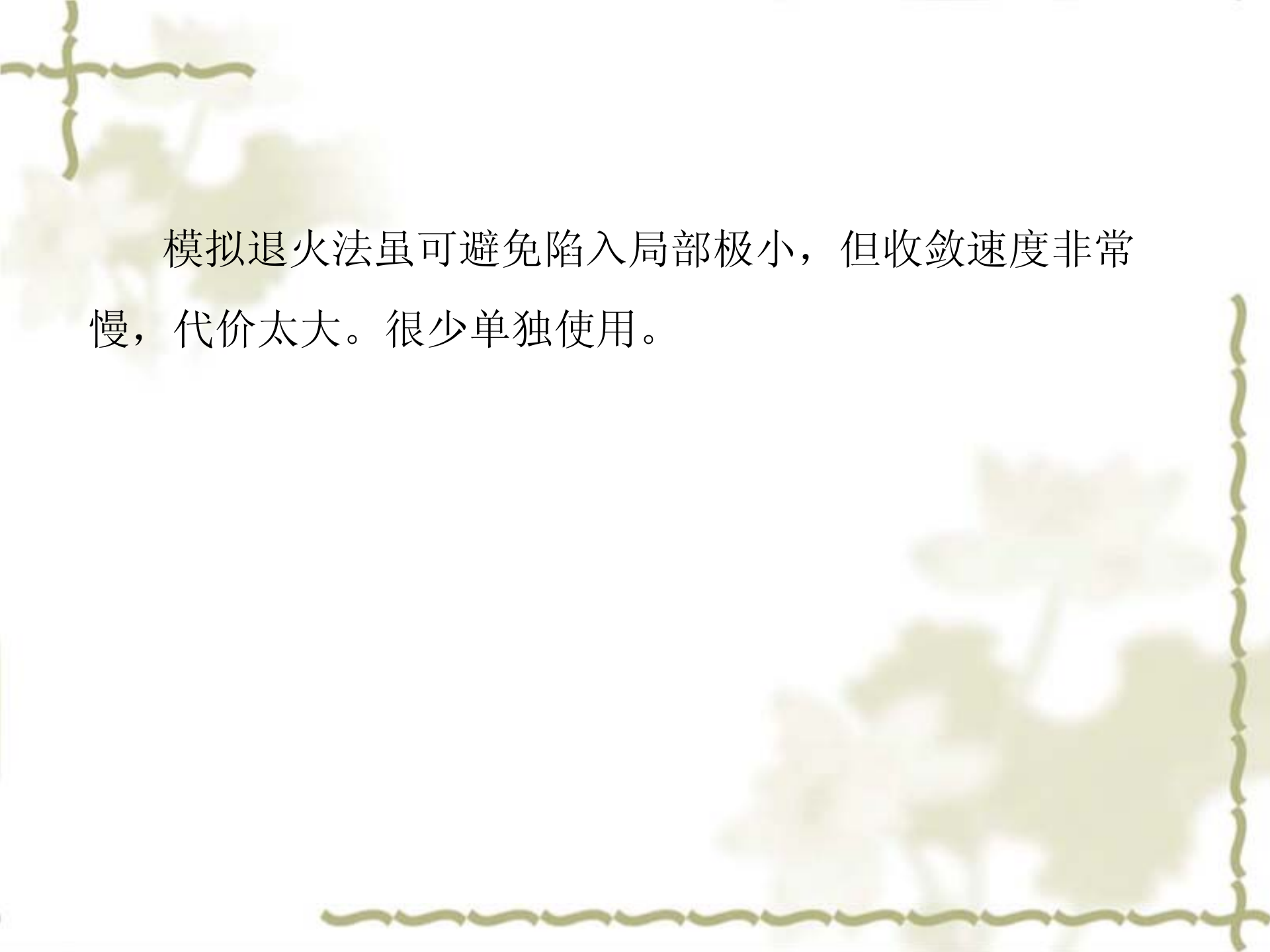
(3)  $\Delta E \leq 0$  时，接受新状态，转步骤(4)； $\Delta E > 0$  时，

若  $e^{-\Delta E/T} > \text{random}[0,1]$ ，接受新状态（爬坡），转步骤(4)；否则转步骤(1)，再随机改变状态。

(4) 从新状态起，继续以前的权值修正，直到稳定平衡状态  $v$ 。

(5) 若  $E_v \geq E_w$ ，搜索失败，转步骤(1)；若  $E_v < E_w$ ，转步骤(6)。

(6) 若  $E_v < \varepsilon$ ，停止；否则  $k=k+1$ ， $T=T/(k+1)$ （降温），转步骤(1)。



模拟退火法虽可避免陷入局部极小，但收敛速度非常慢，代价太大。很少单独使用。