■ **Theorem 8.6.1**

If a vector w^* for which

$$f(x_i^T \cdot w^*) = t_i \quad , \quad 1 \le i \le m \tag{8.6.6}$$

exists, the perceptron learning rule will converge to a weight vector w^{**} which satisfies

$$f(x_i^T \cdot w^{**}) = t_i \quad , \quad 1 \le i \le m \tag{8.6.7}$$

in a finite number of iterations.

The finite weight vector which provides correct responds, is generally not unique.

Proof.

Define

$$T^+ = \{x_i \mid t_i = 1\} \quad , \quad T^- = \{x_i \mid t_i = -1\} \tag{8.6.8}$$

For the sake of simplicity we assume $\alpha = 1,\ \theta = 0$. Consequently, the existence of w^* guarantees $(\theta = 0)$

$$\begin{aligned} x^T \cdot w^* > 0 \quad , \quad x \in T^+ \\ x^T \cdot w^* < 0 \quad , \quad x \in T^- \end{aligned} \tag{8.6.9}$$

Let $T = T^+ \cup (-T^-)$. Then

$$x^T \cdot w^* > 0 \quad , \quad x \in T \tag{8.6.10}$$

If an arbitrary response is incorrect (i.e., ≤ 0) for the current weight vector w, the updating is performed $(\alpha = 1)$ by

$$w(new) = w(old) + x \tag{8.6.11}$$

where x is the training vector which provided the incorrect response.

Let w_0 denote an initial weight vector and denote by y_0 the first training vector with an incorrect response, i.e., it is the first x_i, $1 \le i \le m$ for which $y_0^T \cdot w_0 \le 0$ (if y_0 does not exist the process terminates and $w** = w_0$). To update w_0 we define

$$w_1 = w_0 + y_0$$

and take y_1 as the first training vector which satisfies $y_1^T \cdot w_1 \le 0$. If y_1 does not exist the process terminates and $w** = w_1$. To update w_1 we choose

$$w_2 = w_1 + y_1 = w_0 + y_0 + y_1$$

At every step of the process we have

$$\boldsymbol{w}_k = \boldsymbol{w}_0 + \sum_{i=0}^{k-1} \boldsymbol{y}_i \tag{8.6.12}$$

and we will show that k cannot increase indefinitely. Let

$$a = \min_{1 \le i \le m} [\boldsymbol{x}_i^T \cdot \boldsymbol{w}^*] \tag{8.6.13}$$

Clearly $a > 0$ and consequently by Eq. (8.6.12)

$$\boldsymbol{w}_k^T \cdot \boldsymbol{w}^* = \boldsymbol{w}_0^T \cdot \boldsymbol{w}^* + \sum_{i=0}^{k-1} \boldsymbol{y}_i^T \cdot \boldsymbol{w}^* \ge \boldsymbol{w}_0^T \cdot \boldsymbol{w}^* + ka \tag{8.6.14}$$

If $\boldsymbol{w}_0^T \cdot \boldsymbol{w}^* + ka$ is always negative, k is bounded and the process for obtaining $\boldsymbol{w}^{**}$ is finite. Assume k in contrast to be such that

$\boldsymbol{w}_0^T \cdot \boldsymbol{w}^* + ka$ is already positive. By combining Eq. (8.6.14) and the Cauchy-Schartz inequality we get

$$(\boldsymbol{w}_0^T \cdot \boldsymbol{w}^* + ka)^2 \leq (\boldsymbol{w}_k^T \cdot \boldsymbol{w}^*)^2 \leq \|\boldsymbol{w}_k\|^2 \|\boldsymbol{w}^*\|^2$$

which leads to

$$\|\boldsymbol{w}_k\|^2 \geq \frac{(\boldsymbol{w}_0^T \cdot \boldsymbol{w}^* + ka)^2}{\|\boldsymbol{w}^*\|^2} \tag{8.6.15}$$

i.e., $\|\boldsymbol{w}_k\|^2 \geq Ak^2$ for some $A > 0$. However, for arbitrary k

$$\boldsymbol{w}_k = \boldsymbol{w}_{k-1} + \boldsymbol{y}_{k-1} \quad , \quad \boldsymbol{y}_{k-1}^T \cdot \boldsymbol{w}_{k-1} \leq 0$$

and therefore,

$$\|\boldsymbol{w}_k\|^2 = \|\boldsymbol{w}_{k-1} + y_{k-1}\|^2 \leq \|\boldsymbol{w}_{k-1}\|^2 + \|y_{k-1}\|^2$$

This implies

$$\|\boldsymbol{w}_j\|^2 \leq \|\boldsymbol{w}_{j-1}\|^2 + \|\boldsymbol{y}_{j-1}\|^2 \quad , \qquad 1 \leq j \leq k \tag{8.6.16}$$

and we finally obtain

$$\|\boldsymbol{w}_k\|^2 \leq \|\boldsymbol{w}_0\|^2 + kb \tag{8.6.17}$$

where

$$b = \max \|\boldsymbol{x}_i\|^2 \quad , \quad 1 \leq i \leq m \tag{8.6.18}$$

Obviously, Eqs. (8.6.15) and (8.6.17) lead to contradiction if k increases indefinitely. They also provide an upper bound for k given by

$$\frac{(w_0^T \cdot w^* + ka)^2}{\|w^*\|^2} \leq \|w_0\|^2 + kb \tag{8.6.19}$$

If we assume (without a real loss of generality) $w_0 = 0$ we get

$$k \leq \frac{b\|w^*\|^2}{a^2} \tag{8.6.20}$$

which concludes the proof. □

Clearly, the bound of Eq. (8.6.20) is not exactly practical since $\boldsymbol{w}^*$ (and therefore, a) is unknown. If $\alpha \neq 1$ we obtain a similar proof and for $\boldsymbol{w}_0 = 0$, Eq. (8.6.20) still holds. The validity of the proof for $\theta > 0$ is also straightforward. The restriction that the number of training vectors m is finite can be lifted if $0 < p \leq \|\boldsymbol{x}\| \leq q < \infty$ for all $\boldsymbol{x} \in X$. If X includes training vectors whose norms are very large or very small, the perceptron learning rule may require an extensive number of iterations to converge.

The perceptron learning rule performs better than the Hebb rule as can be seen from the next example.

■ **Example 8.6.2** Consider the 3-D four training patterns and targets given by

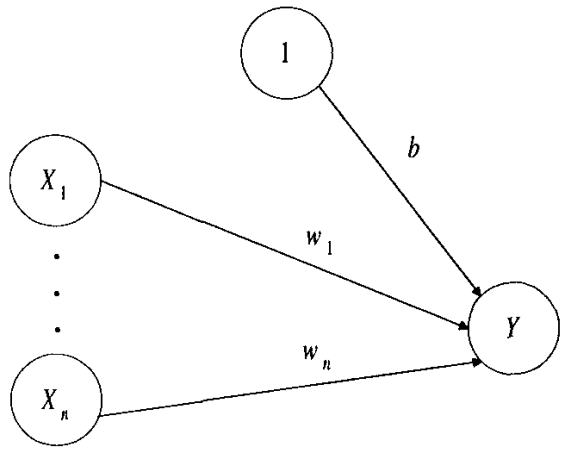
x_1	x_2	x_3	t
-1	-1	1	1
1	-1	-1	-1
1	-1	1	-1
-1	1	1	-1

Here the Hebb rule, starting with $w_i = 0$, $1 \le i \le 3$; $b = 0$, yields the final weights $w_1 = -2$, $w_1 = w_3 = 0$, $b = -2$ which do not provide an appropriate linear separator. If $\boldsymbol{x}^T \cdot \boldsymbol{w} \ge 0$ is considered a positive target and $\boldsymbol{x}^T \cdot \boldsymbol{w} < 0-$ a negative one, the first three patterns are properly classified but not the fourth.

If we treat the same problem using the perceptron learning rule with $\theta = 0.3$, $\alpha = 1$ we obtain after two iterations $w_1 = w_2 = -2$, $w_3 = 0$, $b = -2$.

8.7 ADALINE

The ADALINE (ADAptive Linear NEuron) usually uses bipolar activations and targets. It is a single neuron which receives its input from several input units including one (a bias unit) which provides a constant signal 1. The ADALINE's architecture is shown in Fig. 8.7.1.



■ **Figure 8.7.1** The architecture of a single ADALINE

If several ADALINEs receive their inputs from the same units, they may create a single-layer net. However, if the outputs of several ADALINEs are the inputs for others, we obtain a multilayer net–MADALINE (Many ADAptive Linear NEurons). The training of the ADALINE is done using the *delta rule* and its general design is given next.

Algorithm 8.7.1

(An algorithm for a single ADALINE: ADAL)

Input:

m – the number of training patterns.

n – the number of associate units.

α – the ADALINE learning rate.

$\{x_{ij}\}_{j=1}^{n}$ – the activations of the $i-$th pattern, $1 \le i \le m$

t_i, $1 \le i \le m$ – the correct targets of the training patterns.

w_{j0}, $1 \le j \le n$ – the initial weights.

b_0 – the initial bias.

ε – a given tolerance for determining convergence.

N – maximum number of iterations allowed.

Output: $w_j,\ 1 \le j \le n$ – the final weights.

b – the final bias.

it – number of iterations.

Step 1. Set $it = 0$, $w_{j0}^* = w_{j0}$, $1 \le j \le n$; $b_0^* = b_0$.

Step 2. For $i = 1,\ldots,m$ do Steps 3-4.

Step 3. Compute the net input to the output unit:

$$y\_in = b_0^* + \sum_{j=1}^{n} w_{j0}^* x_{ij}$$

Step 4. Update the weights and bias using the *delta rule*:

$$w_j = w_{j0}^* + \alpha(t_i - y\_in)x_{ij}\,, \quad 1 \le j \le n$$

$$b = b_0^* + \alpha(t_i - y\_\text{in})$$

and set $w_{j0}^* \leftarrow w_j$, $1 \le j \le n$; $b_0^* \leftarrow b$

Step 5. Calculate E : the maximum weight (or bias) change in Steps 2-4; $it = it + 1$

Step 6. If $E < \varepsilon$ output $\{w_j\}_{j=1}^n$, b, it and stop; otherwise if $it = N$ output 'maximum number of iterations exceeded' and stop; otherwise go to Step 2.

If the targets are bipolar we apply in Step 3 an activation function which receives y_in and provides a step function:

$$f(y\_in) = \begin{cases} 1\,, & y\_in \ge 0 \\ -1\,, & y\_in < 0 \end{cases}$$

which replaces y_in in Step 4.

The delta rule applied in Algorithm 8.7.1 is a consequence of trying to reduce the squared error of an arbitrary training pattern. This error is $E = (t - y\_in)^2$ where t is the desired output. To minimize it we apply the method of steepest descent, i.e. following the opposite direction of the error's gradient. The error is obviously a function of the current weights w_j, $1 \le j \le n$ and the bias b, and since we have

$$\frac{\partial E}{\partial w_j} = -2(t - y\_in)x_j \tag{8.7.1}$$

we obtain

$$w_j(new) = w_j(old) + \alpha(t - y\_in)x_j \tag{8.7.2}$$

where α is some prefixed learning rate.

■ **Example 8.7.1** Consider the OR function using bipolar patterns and targets:

x_1	x_2	t
1	1	1
1	−1	1
−1	1	1
−1	−1	−1

The total squared error for given weights and bias w_1, w_2, b is

$$E = (w_1 + w_2 + b - 1)^2 + (w_1 - w_2 + b - 1)^2$$

$$+ (-w_1 + w_2 + b - 1)^2 + (-w_1 - w_2 + b + 1)^2$$

and its minimum is attained by choosing $w_1 = w_2 = b = 0.5$ i.e., the linear separator is $0.5x_1 + 0.5x_2 + 0.5 = 0$

8.8 BACKPROPAGATION NEURAL NET AND ITS APPLICATIONS

The limitations of single-layer neural networks inspired the interest in multilayer neural networks and the discovery of a general method for training such networks — the *backpropagation* method. It consists of applying the steepest descent method to minimize the error produced by the neural net's output.

The architecture of a multilayer neural network with a single hidden layer is illustrated in Fig. 8.8.1. It has input units $\{X_i\}_{i=1}^{n}$ (and a bias); hidden units $\{Z_j\}_{j=1}^{l}$ (and a bias); output units $\{Y_k\}_{k=1}^{m}$. The weights associated with the connections between the input and the hidden units are v_{ij}; $0 \le i \le n$, $1 \le j \le l$ and those between the hidden and the output units are w_{jk} ; $0 \le j \le l$, $1 \le k \le m$.

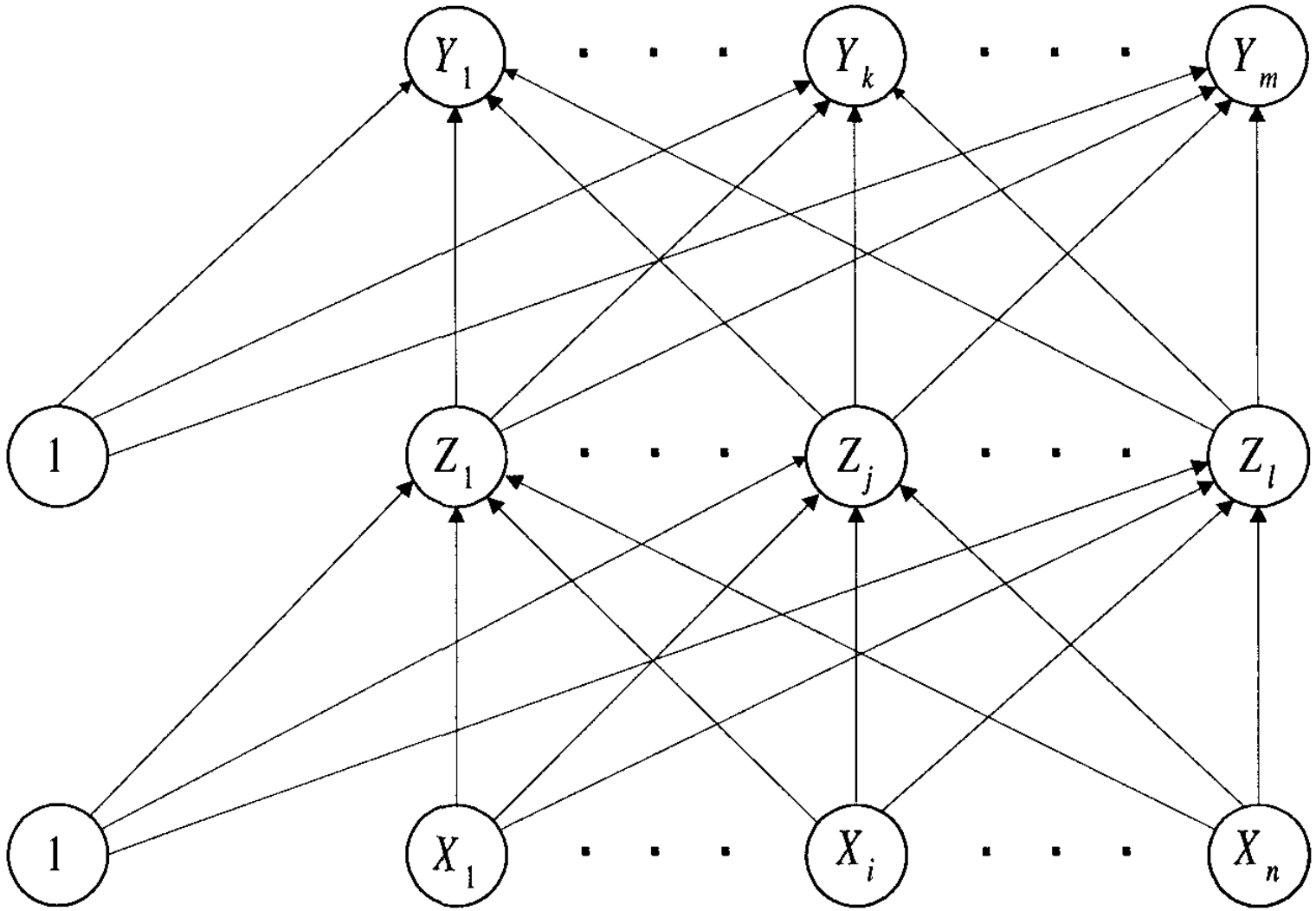


Figure 8.8.1 A multilayer neural net with a single hidden layer.

Backpropagation Method

The training of a network using backpropagation consists of three stages: (a) feedforward the input training pattern throughout the neural network; (b) backpropagation analysis of the error; (c) updating the weights. Without loss of generality, we will consider and discuss the single hidden layer case of Fig. 8.8.1.

The feedforward is performed as follows: each input unit receives a signal and transfers it via its connections and weights to all the hidden units. Each Z_j computes its activation and transfers its own signal to all the output units. Finally, each Y_k computes its activation y_k. The set $\{y_k\}_{k=1}^{m}$ is the network response (or output) of the given input.

The backpropagation analysis of the error is the training stage. Each y_k is compared with its associated target value. This provides the error for that pattern with the unit Y_k. Next, a quantity of δ_k which divides this error back to the units of the previous layer, is computed. In the case of Fig. 8.8.1, these are the hidden units. After obtaining $\{\delta_k\}_{k=1}^{m}$ we compute similar quantities $\{\delta'_j\}_{j=1}^{l}$ which are associated with $\{Z_j\}_{j=1}^{l}$.

Once determined, the numbers $\{\delta_k\}_{k=1}^{m}$, $\{\delta'_j\}_{j=1}^{l}$ are used to adjust *all* the weights *simultaneously*. The weight w_{jk} (from Z_j to Y_k) is adjusted using δ_k and the activation z_j while v_{ij} (from X_i to Z_j) is adjusted by δ'_j and the activation x_i.

The backpropagation procedure is performed on all the training patterns and if the maximum weight adjustment is less than a given tolerance, the process is completed. The standard training algorithm for the backpropagation neural network with a single hidden layer is given next.

Algorithm 8.8.1 (Training by backpropagation).

Input: A set of M training patterns $\boldsymbol{x}^{(p)} = (x_1^{(p)}, \ldots, x_n^{(p)})^T$, $1 \le p \le M$ with targets $\boldsymbol{t}^{(p)} = (t_1^{(p)}, \ldots, t_m^{(p)})^T$, $1 \le p \le M$; initial weights $v_{ij}^{(0)}; 0 \le i \le n, \ 1 \le j \le l$ and

$w_{jk}^{(0)}, \ 0 \le j \le l, \ 1 \le k \le m$; a tolerance $\varepsilon > 0$; a maximum allowed number of iterations N and a learning rate α.

Output: Final weights for the neural network - v_{ij} and w_{jk}.

Step 1. Set $v_{ij} = v_{ij}^{(0)}; \ 0 \le i \le n, \ 1 \le j \le l$ and $w_{jk} = w_{jk}^{(0)}; \ 0 \le j \le l, \ 1 \le k \le m$. Set $it = 0$ (current number of iterations).

Step 2. For $1 \le p \le M$ do Steps 3-4 (feedforward) and Steps 5-7 (backpropagation of the error).

Step 3. For each hidden unit Z_j, $1 \le j \le l$ calculate the total weighted input from input layer:

$$z\_in_j = v_{0j} + \sum_{i=1}^{n} v_{ij}\, x_i^{(p)}, \quad 1 \le j \le l$$

and use the activation function $f(x)$, to get the output signals obtained from the hidden units:

$$z_j^{(p)} = f\,(z\_in_j), \quad 1 \le j \le l$$

Step 4. For each output unit Y_k, $1 \le k \le m$ calculate the total weighted input from the hidden layer:

$$y\_in_k = w_{0k} + \sum_{j=1}^{l} w_{jk} z_j^{(p)}, \quad 1 \le k \le m$$

and use the activation function $f(x)$, to get the output signals

$$y_k^{(p)} = f(y\_in_k), \quad 1 \le k \le m$$

Step 5. Calculate the error terms (steepest descent method).

$$\delta_k^{(p)} = (t_k^{(p)} - y_k^{(p)}) f'(y\_in_k^{(p)}),\ 1 \le k \le m$$

the weight correction terms

$$\Delta w_{jk} = \alpha \delta_k^{(p)} z_j\ ;\ \ 1 \le j \le l\ ,\ 1 \le k \le m$$

and the bias correction terms

$$\Delta w_{0k} = \alpha \delta_k^{(p)}\ ,\ 1 \le k \le m$$

Step 6. For each hidden unit sum its *delta inputs* from the output layer:

$$\delta\_in_j = \sum_{k=1}^{m} \delta_k^{(p)} w_{jk} \ , \ 1 \le j \le l$$

and its backward error term

$$\delta'^{(p)}_j = \delta\_in_j \ f'(z\_in_j) \ , \ 1 \le j \le l$$

Then, obtain the weight correction terms

$$\Delta v_{ij} = \alpha \delta'^{(p)}_j x_i \ ; \ 1 \le i \le n, \ 1 \le j \le l$$

and the bias correction term

$$\Delta v_{0j} = \alpha \delta'^{(p)}_j \ , \ 1 \le j \le l$$

Step 7. Update the weights of the neural net:

$$v_{ij} \leftarrow v_{ij} + \Delta v_{ij} \; ; \; 0 \le i \le n \, , \; 1 \le j \le l$$

$$w_{jk} \leftarrow w_{jk} + \Delta w_{jk} \; ; \; 0 \le j \le l \, , \; 1 \le k \le m$$

Step 8. Set $it \leftarrow it + 1$ and

$$e = \sum_{p=1}^{M} \sum_{k=1}^{m} (t_k^{(p)} - y_k^{(p)})^2$$

If $it \le N$ and $e > \varepsilon$ go to Step 2; otherwise, if $it \le N$ and $e < \varepsilon$, output v_{ij}, w_{jk}, it and stop; else if $it > N$ output 'no convergence' and stop.

Mathematical Background

The backpropagation procedure is based on a popular minimization process—the steepest descent method. The learning rules of algorithm 8.8.1 are obtained as follows. For arbitrary input pattern $\boldsymbol{x} = (x_1, \ldots, x_n)^T$ and target $\boldsymbol{t} = (t_1, \ldots, t_m)^T$, the error to be minimized is (the factor ½ is chosen for convenience)

$$E = \frac{1}{2} \sum_{k=1}^{m} (t_k - y_k)^2 \tag{8.8.1}$$

where

$$y_k = f(y\_in_k) \tag{8.8.2}$$

and

$$y\_in_k = w_{0k} + \sum_{j=1}^{l} w_{jk} z_j \tag{8.8.3}$$

Let *I, J, K* denote fixed values in the sets $\{0, 1, \ldots, n\}, \{0, 1, \ldots, l\}, \{1, 2, \ldots, m\}$ respectively. To apply the steepest descent method we obtain the derivatives

$$\frac{\partial E}{\partial w_{JK}} = -(t_K - y_K)\frac{\partial y_K}{\partial w_{JK}} = -(t_K - y_K) f'(y\_in_K)\frac{\partial (y\_in_K)}{\partial w_{JK}}$$

$$= -(t_K - y_K) f'(y\_in_K) z_J = -\delta_K z_J \tag{8.8.4}$$

and

$$\frac{\partial E}{\partial v_{IJ}} = -\sum_{k=1}^{m} (t_k - y_k) \frac{\partial y_k}{\partial v_{IJ}} = -\sum_{k=1}^{m} (t_k - y_k) f'(y\_in_k)\frac{\partial (y\_in_k)}{\partial v_{IJ}}$$

$$= -\sum_{k=1}^{m} \delta_k \frac{\partial (y\_in_k)}{\partial v_{IJ}} = -\sum_{k=1}^{m} \delta_k w_{Jk} \frac{\partial z_J}{\partial v_{IJ}}$$

$$= -\sum_{k=1}^{m} \delta_k w_{JK} f'(z\_in_J) x_I = -\delta\_in_J f'(z\_in_J) x_I$$

$$= -\delta'_J x_I \tag{8.8.5}$$

and update the weights using the correction terms

$$\Delta w_{jk} = -\alpha \frac{\partial E}{\partial w_{jk}} \;\; ; \;\; 0 \le j \le l \, , \; 1 \le k \le m$$

$$\Delta v_{ij} = -\alpha \frac{\partial E}{\partial v_{ij}} \;\; ; \;\; 0 \le i \le n \;\; , \; 1 \le j \le l \tag{8.8.6}$$

where α is some prefixed learning rate.

Activation Function

Due to its role in the design of a neural network the activation function $f(x)$ is expected to be monotonic nondecreasing and to belong to $C^1(-\infty, \infty)$. Usually, it is also expected to *saturate*, i.e.

$$\lim_{x \to \infty} f(x) = A < \infty$$
$$\lim_{x \to -\infty} f(x) = B > -\infty \tag{8.8.7}$$

Finally, from computational point of view, it is desirable to apply an activation such that $f(x)$ and $f'(x)$ are easily computed. Typical activation functions are the previously defined binary and the bipolar sigmoid functions and $\tanh(x)$.

Initialization

Choosing appropriate initial values for the various weights may determine the speed of the learning process and whether we reach a global or local minimum of the error (a typical problem when applying the steepest descent method). Since updating the weights involve values of activations and their derivatives, it is preferable that these values should not vanish. Consequently it is usually advisable to choose initial weights which are not too large. For example, one could choose the initial weights randomly within the range (– 0.5, 0.5).

A modification to this trivial choice is the Nguyen-Widrow initialization. The initial weights and biases from the hidden layer to the output layer are chosen randomly, say between –0.5 to 0.5. As to the initial weights over the connections between the input and the hidden layers, they are determined using the following procedure. Define a scale factor

$$\beta = 0.7\sqrt[n]{l} \tag{8.8.8}$$

where n and l are the numbers of the units at the input and hidden layers respectively. We first choose v_{ij} randomly between –0.5 and 0.5 and denote them by $v_{ij}^{(0)}$. *Let* $\boldsymbol{v}_j^{(0)}$ denote the vector of the weights from the input units to the hidden unit Z_j, i.e. $\boldsymbol{v}_j^{(0)} = (v_{1j}^{(0)}, \ldots, v_{nj}^{(0)})^{\mathrm{T}}$. We now update $v_{ij}^{(0)}$ and use

$$v_{ij}^{(1)} = \frac{\beta v_{ij}^{(0)}}{\left\| \boldsymbol{v}_j^{(0)} \right\|} \;\; ; \;\; 1 \le i \le n \, , \;\; 1 \le j \le l$$

as the final initial weights between X_i and Z_j. The initial biases v_{0j} are taken randomly between $-\beta$ and β.

The Nguyen-Widrow initialization procedure is based on the activation function $\tanh(x)$ but is also effective for the similarly behaving bipolar sigmoid function.

■ **Example 8.8.1** Consider a neural network with two input units, three hidden units and a single output unit, i.e. $n = 2$, $l = 3$ and $m = 1$. This network is supposed to operate as the 'XOR' function. Four training patterns which are 2-D vectors ($n = 2$) are the inputs. The outputs are four scalars ($m = 1$). If a binary representation is considered, the patterns are $(1,1)^T$, $(1,0)^T, (0,1)^T, (0,0)^T$, with targets 0, 1, 1, 0 respectively and the activation function is the binary sigmoid. For a bipolar representation the input patterns are $(1,1)^T, (1,-1)^T, (-1,1)^T, (-1,-1)^T$ with targets –1, 1, 1, –1 respectively and the activation function is the bipolar sigmoid.

The initial weights chosen randomly between –0.5 and 0.5 are:

$$\begin{aligned} v_{01} &= \ 0.396, & v_{02} &= -0.030, & v_{03} &= -0.401 \\ v_{11} &= -0.066, & v_{12} &= -0.046, & v_{13} &= \ 0.414 \\ v_{21} &= -0.017, & v_{22} &= \ 0.220, & v_{23} &= \ 0.231 \end{aligned}$$

and

$$w_{01} = 0.118,\ w_{11} = -0.173,\ w_{21} = 0.204,\ w_{31} = -0.271$$

The training of the network continues until the total squared error is sufficiently small, i.e.

$$\sum_{p=1}^{M} \sum_{k=1}^{m} \left(y_k^{(p)} - t_k^{(p)}\right)^2 < \varepsilon$$

For a tolerance $\varepsilon = 0.05$ and learning rate $\alpha = 0.1$ we obtain the following results:

(a) Binary representation: 15342 iterations are needed for convergence and the final outputs are:

$$y_1^{(1)} = 0.112\,,\ \ y_1^{(2)} = 0.870\,,\ \ y_1^{(3)} = 0.899\,,\ \ y_1^{(4)} = 0.102$$

(b) Bipolar representation: 823 iterations are needed for convergence and the final outputs are:

$$y_1^{(1)} = -0.883\,,\ \ y_1^{(2)} = 0.910\,,\ \ y_1^{(3)} = 0.895\,,\ \ y_1^{(4)} = -0.869$$

▪ **Example 8.8.2** Consider the previous example with $\varepsilon = 0.01$ and $\alpha = 0.15$. Applying standard (ST) and Nguyen-Widrow (NW) initializations provide the following results.

(a) Binary representation:

Type of Initialization	No. of iterations
ST	12971
NW	6883

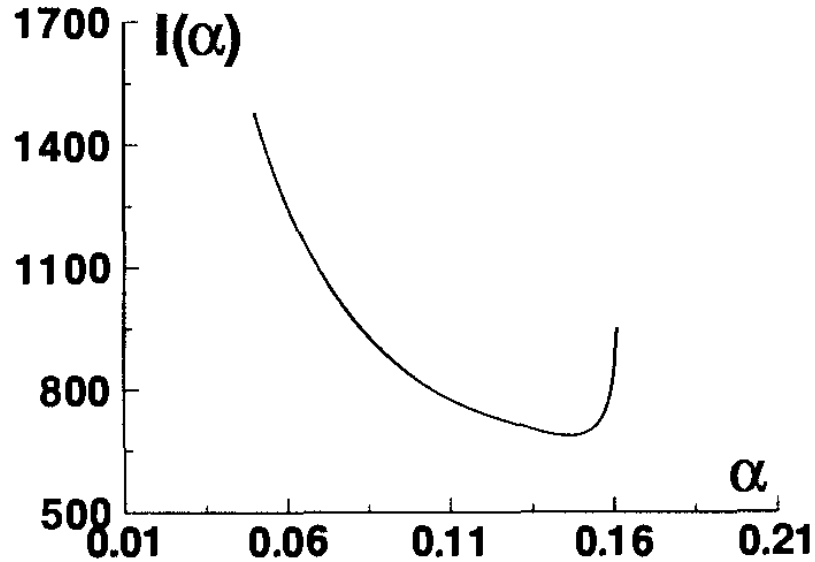
(b) Bipolar representation:

Type of Initialization	No. of iterations
ST	1534
NW	1246

For fixed initial values of the weights and a given tolerance $\varepsilon > 0$, the speed by which the backpropagation network trains itself, depends on the learning-rate α.

In the next example we obtain the number of iterations needed for convergence, $\mathrm{I}(\alpha)$, for the 'XOR' function. The weights are chosen randomly.

▪ **Example 8.8.3** Consider Example 8.8.1 $(\varepsilon = 0.05)$. Fig. 8.8.2 illustrates the speed of convergence as a function of α in the case of bipolar representation.



- **Figure 8.8.2** $I(\alpha)$ for the 'XOR' function-bipolar representation.

A reasonable choice for 'optimal' α is between 0.1 and 0.15.

Updating Weights Using Momentum

In order to speed up the convergence of a backpropagation network, it is sometimes recommended to derive the weights at step $(t+1)$, using not only the weights at step t but also those at step $(t-1)$. The updating is performed by

$$\Delta v_{ij}(t+1) = \alpha \delta_j x_i + \mu \Delta v_{ij}(t)$$
$$\Delta w_{jk}(t+1) = \alpha \delta_k z_j + \mu \Delta w_{jk}(t) \tag{8.8.9}$$

where

$$\Delta v_{ij}(t) = v_{ij}(t) - v_{ij}(t-1)$$
$$\Delta w_{jk}(t) = w_{jk}(t) - w_{jk}(t-1) \tag{8.8.10}$$

and μ, the *momentum* coefficient is between 0 and 1.

This modified updating may allow reasonably large weights adjustments and reduces the likelihood to obtain a local minimum rather than a global one while training the network.

Batch Updating

Sometimes it is more efficient to accumulate the weight adjustments for several patterns and then perform a single adjustment using the average of the various correction terms. A possible disadvantage is that this procedure may have a smoothing effect on the correction terms and consequently may lead to a local minimum.

Updating the Learning Rate

In many applications each weight may have its own learning rate. Moreover, if the weight change is in the same direction for several steps, its associated learning rate should be increased. On the other hand, if the

weight change sign alternates, the associated learning rate should be decreased.

■ **Example 8.8.4** Consider Example 8.8.1 with bipolar representation. Let $\varepsilon = 0.01$ and $\alpha = 0.1$. The standard implementation of Algorithm 8.8.1 converges after 2081 iterations. By using the momentum procedure with $\mu = 1$ the number of iterations is reduced to 30.