Each hidden or output neuron of a multilayer perceptron is designed to perform two computations:

1. The computation of the function signal appearing at the output of a neuron, which is expressed as a continuous nonlinear function of the input signals and synaptic weights associated with that neuron
2. The computation of an instantaneous estimate of the gradient vector (i.e., the gradients of the error surface with respect to the weights connected to the inputs of a neuron), which is needed for the backward pass through the network

The derivation of the back-propagation algorithm is rather involved. To ease the mathematical burden involved in this derivation, we first present a summary of the notations used in the derivation.

Notation

- The indices i, j, and k refer to different neurons in the network; with signals propagating through the network from left to right, neuron j lies in a layer to the right of neuron i, and neuron k lies in a layer to the right of neuron j when neuron j is a hidden unit.
- The iteration n refers to the nth training pattern (example) presented to the network.
- The symbol $\mathscr{E}(n)$ refers to the instantaneous sum of error squares at iteration n. The average of $\mathscr{E}(n)$ over all values of n (i.e., the entire training set) yields the average squared error $\mathscr{E}_{av}$.
- The symbol $e_j(n)$ refers to the error signal at the output of neuron j for iteration n.
- The symbol $d_j(n)$ refers to the desired response for neuron j and is used to compute $e_j(n)$.
- The symbol $y_j(n)$ refers to the function signal appearing at the output of neuron j at iteration n.
- The symbol $w_{ji}(n)$ denotes the synaptic weight connecting the output of neuron i to the input of neuron j at iteration n. The correction applied to this weight at iteration n is denoted by $\Delta w_{ji}(n)$.
- The net internal activity level of neuron j at iteration n is denoted by $v_j(n)$; it constitutes the signal applied to the nonlinearity associated with neuron j.
- The activation function describing the input–output functional relationship of the nonlinearity associated with neuron j is denoted by $\varphi_j(\cdot)$.
- The threshold applied to neuron j is denoted by θ_j; its effect is represented by a synapse of weight $w_{j0} = \theta_j$ connected to a fixed input equal to -1.
- The ith element of the input vector (pattern) is denoted by $x_i(n)$.
- The kth element of the overall output vector (pattern) is denoted by $o_k(n)$.
- The learning-rate parameter is denoted by η.

6.3 Derivation of the Back-Propagation Algorithm

The error signal at the output of neuron j at iteration n (i.e., presentation of the nth training pattern) is defined by

$$e_j(n) = d_j(n) - y_j(n), \qquad \text{neuron } j \text{ is an output node} \tag{6.1}$$

We define the instantaneous value of the squared error for neuron j as $\frac{1}{2}e_j^2(n)$. Correspondingly, the instantaneous value $\mathscr{E}(n)$ of the sum of squared errors is obtained by summing $\frac{1}{2}e_j^2(n)$ over *all neurons in the output layer;* these are the only "visible" neurons for which error signals can be calculated. The *instantaneous sum of squared errors* of the network is thus written as

$$\mathscr{E}(n) = \frac{1}{2}\sum_{j \in C} e_j^2(n) \tag{6.2}$$

where the set C includes all the neurons in the output layer of the network. Let N denote the total number of patterns (examples) contained in the training set. The *average squared error* is obtained by summing $\mathscr{E}(n)$ over all n and then normalizing with respect to the set size N, as shown by

$$\mathscr{E}_{\text{av}} = \frac{1}{N}\sum_{n=1}^{N} \mathscr{E}(n) \tag{6.3}$$

The instantaneous sum of error squares $\mathscr{E}(n)$, and therefore the average squared error $\mathscr{E}_{\text{av}}$, is a function of all the free parameters (i.e., synaptic weights and thresholds) of the network. For a given training set, $\mathscr{E}_{\text{av}}$ represents the *cost function* as the measure of training set learning performance. The objective of the learning process is to adjust the free parameters of the network so as to minimize $\mathscr{E}_{\text{av}}$. To do this minimization we use an approximation similar in rationale to that we used for the derivation of the LMS algorithm in Chapter 5. Specifically, we consider a simple method of training in which the weights are updated on a *pattern-by-pattern* basis. The adjustments to the weights are made in accordance with the respective errors computed for *each* pattern presented to the network. The arithmetic average of these individual weight changes over the training set is therefore an *estimate* of the true change that would result from modifying the weights based on minimizing the cost function $\mathscr{E}_{\text{av}}$ over the entire training set.

Consider then Fig. 6.3, which depicts neuron j being fed by a set of function signals produced by a layer of neurons to its left. The net internal activity level $v_j(n)$ produced

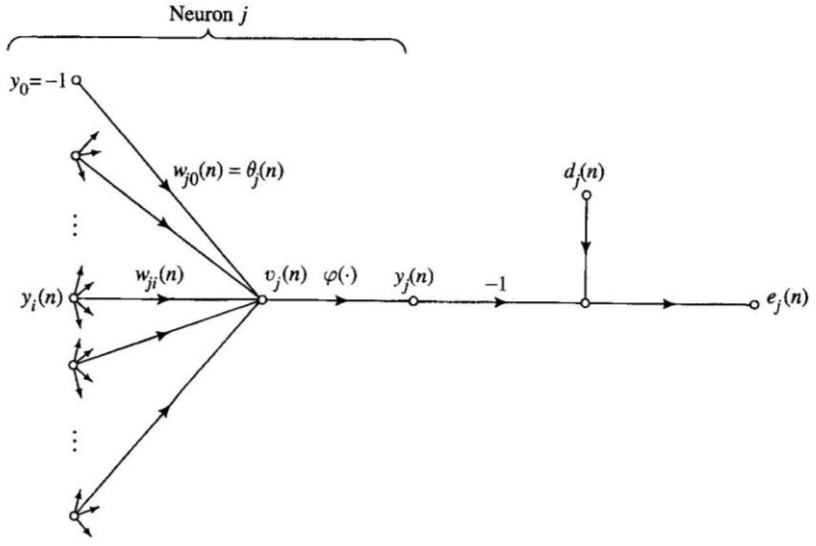

FIGURE 6.3 Signal-flow graph highlighting the details of output neuron j.

at the input of the nonlinearity associated with neuron j is therefore

$$v_j(n) = \sum_{i=0}^{p} w_{ji}(n) y_i(n) \tag{6.4}$$

where p is the total number of inputs (excluding the threshold) applied to neuron j. The synaptic weight w_{j0} (corresponding to the fixed input $y_0 = -1$) equals the threshold θ_j applied to neuron j. Hence the function signal $y_j(n)$ appearing at the output of neuron j at iteration n is

$$y_j(n) = \varphi_j(v_j(n)) \tag{6.5}$$

In a manner similar to the LMS algorithm, the back-propagation algorithm applies a correction $\Delta w_{ji}(n)$ to the synaptic weight $w_{ji}(n)$, which is proportional to the instantaneous gradient $\partial \mathscr{E}(n)/\partial w_{ji}(n)$. According to the chain rule, we may express this gradient as follows:

$$\frac{\partial \mathscr{E}(n)}{\partial w_{ji}(n)} = \frac{\partial \mathscr{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} \tag{6.6}$$

The gradient $\partial \mathscr{E}(n)/\partial w_{ji}(n)$ represents a *sensitivity factor,* determining the direction of search in weight space for the synaptic weight w_{ji}.

Differentiating both sides of Eq. (6.2) with respect to $e_j(n)$, we get

$$\frac{\partial \mathscr{E}(n)}{\partial e_j(n)} = e_j(n) \tag{6.7}$$

Differentiating both sides of Eq. (6.1) with respect to $y_j(n)$, we get

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1 \tag{6.8}$$

Next, differentiating Eq. (6.5) with respect to $v_j(n)$, we get

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j'(v_j(n)) \tag{6.9}$$

where the use of prime (on the right-hand side) signifies differentiation with respect to the argument. Finally, differentiating Eq. (6.4) with respect to $w_{ji}(n)$ yields

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_i(n) \tag{6.10}$$

Hence, the use of Eqs. (6.7) to (6.10) in (6.6) yields

$$\frac{\partial \mathscr{E}(n)}{\partial w_{ji}(n)} = -e_j(n)\varphi_j'(v_j(n))y_i(n) \tag{6.11}$$

The correction $\Delta w_{ji}(n)$ applied to $w_{ji}(n)$ is defined by the *delta rule*

$$\Delta w_{ji}(n) = -\eta \frac{\partial \mathscr{E}(n)}{\partial w_{ji}(n)} \tag{6.12}$$

where η is a constant that determines the rate of learning; it is called the *learning-rate parameter* of the back-propagation algorithm. The use of the minus sign in Eq. (6.12) accounts for *gradient descent* in weight space. Accordingly, the use of Eq. (6.11) in (6.12) yields

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n) \tag{6.13}$$

where the *local gradient* $\delta_j(n)$ is itself defined by

$$\delta_j(n) = -\frac{\partial \mathscr{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)}$$

$$= e_j(n)\varphi_j'(v_j(n)) \tag{6.14}$$

The local gradient points to required changes in synaptic weights. According to Eq. (6.14), the local gradient $\delta_j(n)$ for output neuron j is equal to the product of the corresponding error signal $e_j(n)$ and the derivative $\varphi_j'(v_j(n))$ of the associated activation function.

From Eqs. (6.13) and (6.14) we note that a key factor involved in the calculation of the weight adjustment $\Delta w_{ji}(n)$ is the error signal $e_j(n)$ at the output of neuron j. In this context, we may identify two distinct cases, depending on where in the network neuron j is located. In case I, neuron j is an output node. This case is simple to handle, because each output node of the network is supplied with a desired response of its own, making it a straightforward matter to calculate the associated error signal. In case II, neuron j is a hidden node. Even though hidden neurons are not directly accessible, they share responsibility for any error made at the output of the network. The question, however, is to know how to penalize or reward hidden neurons for their share of the responsibility. This problem is indeed the *credit-assignment problem* considered in Section 2.6. It is solved in an elegant fashion by back-propagating the error signals through the network.

In the sequel, cases I and II are considered in turn.

Case I: Neuron j Is an Output Node

When neuron j is located in the output layer of the network, it would be supplied with a desired response of its own. Hence we may use Eq. (6.1) to compute the error signal $e_j(n)$ associated with this neuron; see Fig. 6.3. Having determined $e_j(n)$, it is a straightforward matter to compute the local gradient $\delta_j(n)$ using Eq. (6.14).

Case II: Neuron j Is a Hidden Node

When neuron j is located in a hidden layer of the network, there is no specified desired response for that neuron. Accordingly, the error signal for a hidden neuron would have to be determined recursively in terms of the error signals of all the neurons to which that hidden neuron is directly connected; this is where the development of the back-propagation algorithm gets complicated. Consider the situation depicted in Fig. 6.4, which depicts neuron j as a hidden node of the network. According to Eq. (6.14), we may redefine the local gradient $\delta_j(n)$ for hidden neuron j as

$$\delta_j(n) = -\frac{\partial \mathscr{E}(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)}$$

$$= -\frac{\partial \mathscr{E}(n)}{\partial y_j(n)} \varphi_j'(v_j(n)), \qquad \text{neuron } j \text{ is hidden} \tag{6.15}$$

where, in the second line, we have made use of Eq. (6.9). To calculate the partial derivative $\partial \mathscr{E}(n)/\partial y_j(n)$, we may proceed as follows. From Fig. 6.4 we see that

$$\mathscr{E}(n) = \frac{1}{2} \sum_{k \in C} e_k^2(n), \qquad \text{neuron } k \text{ is an output node} \tag{6.16}$$

which is a rewrite of Eq. (6.2) except for the use of index k in place of index j. We have done so in order to avoid confusion with the use of index j that refers to a hidden neuron

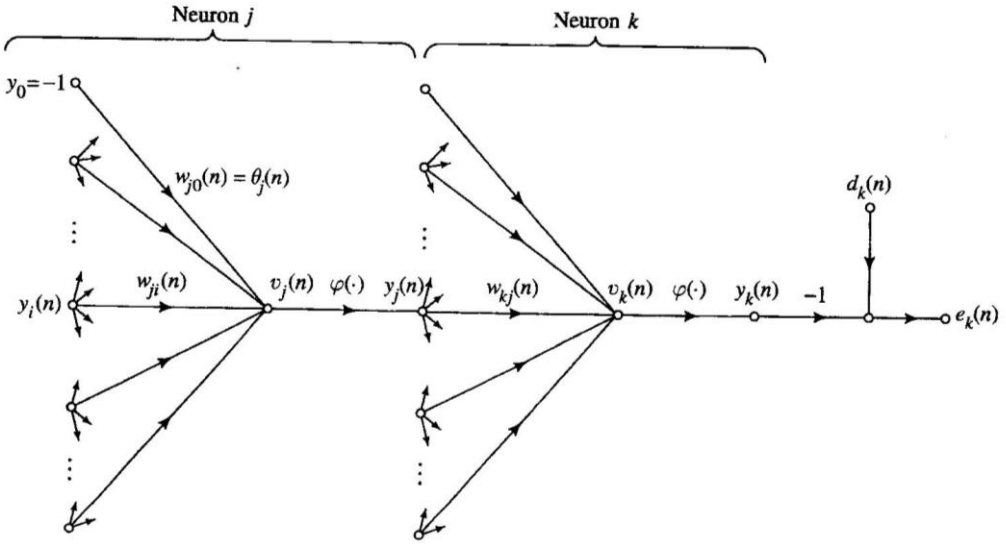

FIGURE 6.4 Signal-flow graph highlighting the details of output neuron k connected to hidden neuron j.

under case II. In any event, differentiating Eq. (6.16) with respect to the function signal $y_j(n)$, we get

$$\frac{\partial \mathscr{E}(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial y_j(n)} \tag{6.17}$$

Next, we use the chain rule for the partial derivative $\partial e_k(n)/\partial y_j(n)$, and thus rewrite Eq. (6.17) in the equivalent form

$$\frac{\partial \mathscr{E}(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)} \tag{6.18}$$

However, from Fig. 6.4, we note that

$$\begin{aligned} e_k(n) &= d_k(n) - y_k(n) \\ &= d_k(n) - \varphi_k(v_k(n)), \qquad \text{neuron } k \text{ is an output node} \end{aligned} \tag{6.19}$$

Hence

$$\frac{\partial e_k(n)}{\partial v_k(n)} = -\varphi_k'(v_k(n)) \tag{6.20}$$

We also note from Fig. 6.4 that for neuron k, the net internal activity level is

$$v_k(n) = \sum_{j=0}^{q} w_{kj}(n) y_j(n) \tag{6.21}$$

where q is the total number of inputs (excluding the threshold) applied to neuron k. Here again, the synaptic weight $w_{k0}(n)$ is equal to the threshold $\theta_k(n)$ applied to neuron k, and the corresponding input y_0 is fixed at the value -1. In any event, differentiating Eq. (6.21) with respect to $y_j(n)$ yields

$$\frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n) \tag{6.22}$$

Thus, using Eqs. (6.20) and (6.22) in (6.18), we get the desired partial derivative:

$$\begin{aligned}\frac{\partial \mathscr{E}(n)}{\partial y_j(n)} &= -\sum_k e_k(n)\varphi_k'(v_k(n))w_{kj}(n) \\ &= -\sum_k \delta_k(n)w_{kj}(n)\end{aligned} \tag{6.23}$$

where, in the second line, we have used the definition of the local gradient $\delta_k(n)$ given in Eq. (6.14) with the index k substituted for j.

Finally, using Eq. (6.23) in (6.15), we get the local gradient $\delta_j(n)$ for hidden neuron j, after rearranging terms, as follows:

$$\delta_j(n) = \varphi_j'(v_j(n))\sum_k \delta_k(n)w_{kj}(n), \qquad \text{neuron } j \text{ is hidden} \tag{6.24}$$

The factor $\varphi_j'(v_j(n))$ involved in the computation of the local gradient $\delta_j(n)$ in Eq. (6.24) depends solely on the activation function associated with hidden neuron j. The remaining factor involved in this computation, namely, the summation over k, depends on two sets of terms. The first set of terms, the $\delta_k(n)$, requires knowledge of the error signals $e_k(n)$, for all those neurons that lie in the layer to the immediate right of hidden neuron j, and that are directly connected to neuron j; see Fig. 6.4. The second set of terms, the $w_{kj}(n)$, consists of the synaptic weights associated with these connections.

We may now summarize the relations that we have derived for the back-propagation algorithm. First, the correction $\Delta w_{ji}(n)$ applied to the synaptic weight connecting neuron i to neuron j is defined by the delta rule:

$$\begin{pmatrix}\text{Weight} \\ \text{correction} \\ \Delta w_{ji}(n)\end{pmatrix} = \begin{pmatrix}\text{learning-} \\ \text{rate parameter} \\ \eta\end{pmatrix} \cdot \begin{pmatrix}\text{local} \\ \text{gradient} \\ \delta_j(n)\end{pmatrix} \cdot \begin{pmatrix}\text{input signal} \\ \text{of neuron } j \\ y_i(n)\end{pmatrix} \tag{6.25}$$

Second, the local gradient $\delta_j(n)$ depends on whether neuron j is an output node or a hidden node:

1. If neuron j is an output node, $\delta_j(n)$ equals the product of the derivative $\varphi_j'(v_j(n))$ and the error signal $e_j(n)$, both of which are associated with neuron j; see Eq. (6.14).
2. If neuron j is a hidden node, $\delta_j(n)$ equals the product of the associated derivative $\varphi_j'(v_j(n))$ and the weighted sum of the δ's computed for the neurons in the next hidden or output layer that are connected to neuron j; see Eq. (6.24).

The Two Passes of Computation

In the application of the back-propagation algorithm, two distinct passes of computation may be distinguished. The first pass is referred to as the forward pass, and the second one as the backward pass.

In the *forward pass* the synaptic weights remain unaltered throughout the network, and the function signals of the network are computed on a neuron-by-neuron basis. Specifically, the function signal appearing at the output of neuron j is computed as

$$y_j(n) = \varphi(v_j(n)) \tag{6.26}$$

where $v_j(n)$ is the net internal activity level of neuron j, defined by

$$v_j(n) = \sum_{i=0}^{p} w_{ji}(n)y_i(n) \tag{6.27}$$

where p is the total number of inputs (excluding the threshold) applied to neuron j, and $w_{ji}(n)$ is the synaptic weight connecting neuron i to neuron j, and $y_i(n)$ is the input signal