2

Neural Networks

Artificial neural networks derive their origins from biological neural networks. They are systems of parallel and distributed processing that simulate the basic operating principles of the biological brain. Sometimes they are referred to in the literature as *machine learning algorithms*. The biological brain in its basic structure is a network of neural cells (neurons) attached through connections that have the ability to adjust the power of the electrical pulse that runs through them (synapses). The external stimulus in the form of an electrical pulse is transmitted as information through synapses to the neurons, where it is processed, and eventually, an output response of the network is produced. The information is encoded as "knowledge" through continuous updating of the existing synapses between neurons.

In this book we treat neural networks as the eminent expression of nonparametric regression. Nonparametric regression is a very powerful approach, especially for financial applications. Neural networks can approximate any unknown nonlinear function and are generally less sensitive than classical approaches to assumptions about the error term; hence, they can perform well in the presence of noise, chaotic sections, and fat tails of probability distributions.

The basic aspects of neural networks are presented below. More precisely, the usual training algorithm and network structures are presented. In addition, a geometric explanation of the backpropagation learning rule is given.

Wavelet Neural Networks: With Applications in Financial Engineering, Chaos, and Classification, First Edition. Antonios K. Alexandridis and Achilleas D. Zapranis.
© 2014 John Wiley & Sons, Inc. Published 2014 by John Wiley & Sons, Inc.

PARALLEL PROCESSING

Although there are now many types of artificial neural networks, they all have one common characteristic: They are systems of parallel distributed processing (PDP). The processing of information is distributed over several computing units, while its encryption is accomplished by the interactions of all these units. Each PDP system consists of the following components (Rumelhart et al., 1986b):

- A set of processing units.
- A state of activation.
- An output function for each unit.
- An activation rule that combines all the inputs of a unit with the current activation state in order to compute the new activation state.
- A connectivity model between units.
- A signal propagation rule through the connections between units.
- A learning rule according to which the connectivity model alters through training.
- An environment in which the system must operate.

Usually, the output function and the activation rule are the same. A schematic representation of a PDP system is shown in Figure 2.1, and Figure 2.2 illustrates the general features of a processing unit. Below we examine the most important of these features and the assumptions made for them, and the interdependence of the components within a PDP system.

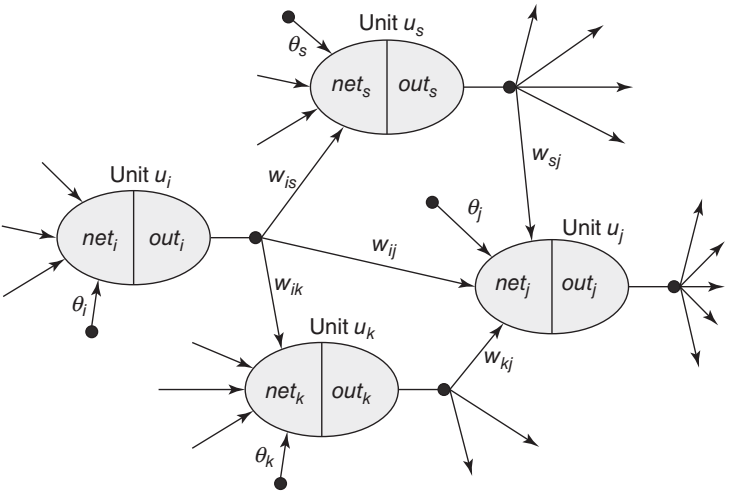

Figure 2.1 *Schematic representation of parallel distributed processing.*

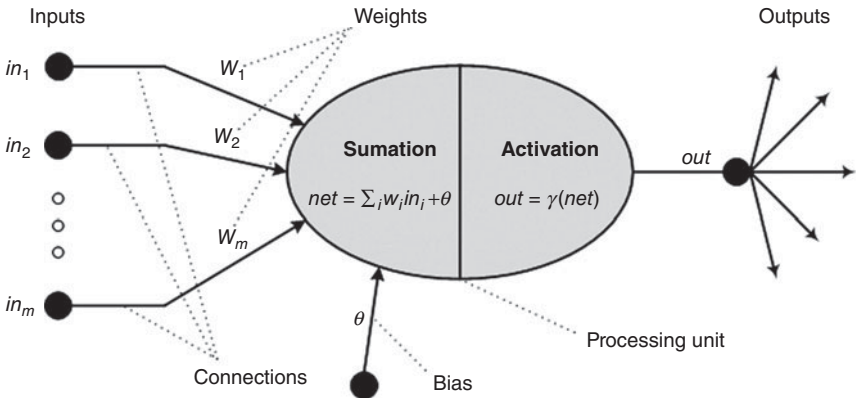

Figure 2.2 *Schematic representation of the general characteristics of a processing unit (an artificial neuron).*

Processing Units

Each processing unit receives inputs from the neighboring units or external sources and uses them to calculate an output signal which is distributed to other units. As illustrated in Figure 2.2, a unit receives some input signals, in_1, in_2, …, in_m, which, unlike electrical pulses of the biological neuron, correspond to continuous variables. A weight value w_i changes any such input signal. The role of the weights is equivalent to the biological neuron's synapses. The value of the weight can be positive or negative. The input signal that is transferred via the bias connection is constant and has a value of 1.

The body of the artificial neuron is divided into two sections. In the first, the net input is estimated by the weighted summation of the inputs, *net*; in the second, the output value is estimated by the activation function γ. We can distinguish among three types of units: input units, which receive the data; output units, which are sending data out of the network; and hidden units, whose inputs and outputs signals are staying inside the network. During operation of the neural network, the units can be adjusted synchronously or asynchronously. In synchronous updating, all units update their activation level simultaneously; in asynchronous updating each unit has some (usually, fixed) probability to adjust its activation level at time t, and usually only one unit will be able to do so at this particular time.

Activation Status and Activation Rules

A rule that defines the effect of the net input to the activation state of a unit is needed. This rule is given by the function γ, which is also called an *activation function* or *transfer function*, and takes as inputs the net input $\text{net}(t)$ and the current activation status $\text{out}(t)$ at time t and returns as output the new value of the activation state $\text{out}(t+1)$ at time $t+1$:

$$\text{out}(t+1) = \gamma(\text{net}(t), \text{out}(t)) \tag{2.1}$$

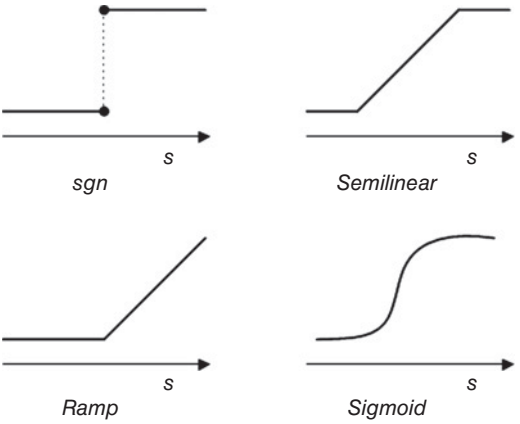

Figure 2.3 *Different types of activation functions.*

Usually, the activation function is a nondecreasing function of the net input only. Also, the activation functions are not strictly limited to nondecreasing forms. That is,

$$\text{out}\,(t+1) = \gamma\,(\text{net}\,(t)) = \gamma\left(\sum_i w_i\,(t)\,\text{in}_i\,(t) + \theta\,(t)\right) \tag{2.2}$$

Generally, some type of threshold function is used. Such examples are the sign function (sgn), the linear or semilinear function, the ramp function, or the sigmoid function. These functions are presented in Figure 2.3.

The sign function is given by

$$\gamma\,(s) = \text{sgn}(s) = \begin{cases} +1 & s > 0 \\ -1 & s \leq 0 \end{cases} \tag{2.3}$$

However, the classic case is the use of the family of sigmoid functions that belong to the class

$$\Gamma = \{\gamma = \gamma\,(s,k,T,c)\,|x,s \in \Re - \{0\}\} \tag{2.4}$$

and is defined as follows:

$$\gamma\,(s) = k + \frac{c}{1+e^{-Ts}} \tag{2.5}$$

where T is a factor regulating the speed of transition to one of two asymptotic values. This type of function is very important because it provides nonlinearity to the neuron, which is essential in modeling nonlinear phenomena. If in equation (2.5) we set

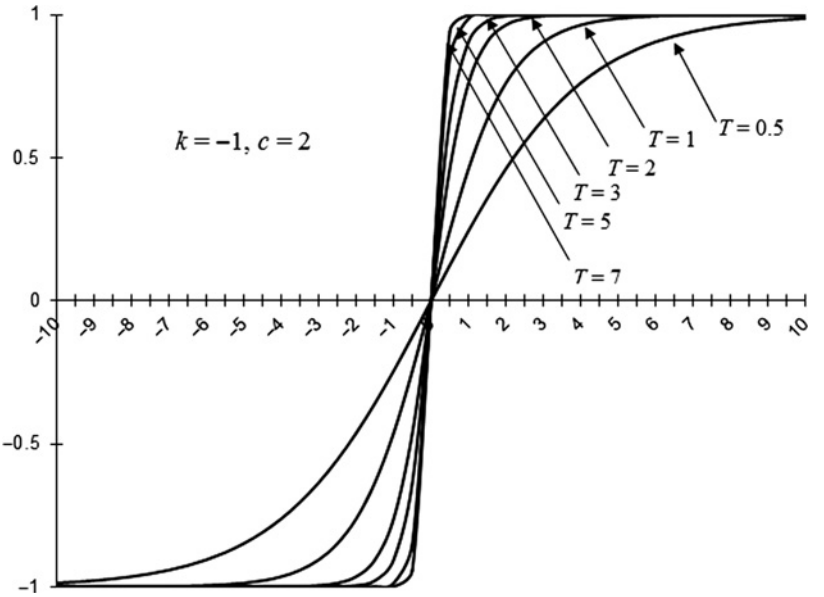

Figure 2.4 *Family of symmetric sigmoid activation functions with values between −1 and 1.*

$k = -1$ and $c = 2$, the symmetric sigmoid activation functions are obtained, which return values between -1 and 1.

As presented in Figure 2.4, if the value of T increases, this family of functions converges to the sign function (2.3). The most commonly used function of this form is obtained for $T = 2$:

$$\gamma(s) = \frac{2}{1 + e^{-2s}} - 1 \tag{2.6}$$

Similarly, if in equation (2.5) we set $k = 0$ and $c = 1$, we obtain the family of asymmetric sigmoid activation functions which return values between 0 and 1 (Figure 2.5). Usually, for this function the value of $T = 1$ is used:

$$\gamma(s) = \frac{1}{1 + e^{-s}} \tag{2.7}$$

Connectivity Model

The connectivity model is related to the organization of the connections between units. It determines what the system knows and how it will respond to some random input.

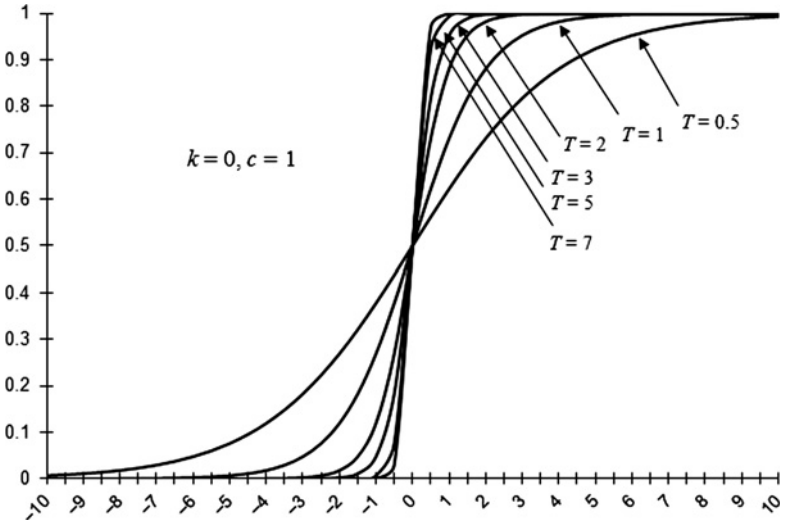

Figure 2.5 *Family of asymmetric sigmoid activation functions with values between 0 and 1.*

Usually, it is assumed that each unit provides an additive contribution to the inputs of the units with which it is connected. Hence, the total (net) input of a unit is simply the weighted sum of inputs from the other units plus a bias term:

$$\text{net} = \sum_i w_i \text{in}_i + \theta \tag{2.8}$$

Propagation Rule The propagation rule refers to the way that data flow through a network. There are two basic categories of neural networks: feedforward networks and recurrent networks.

Feedforward Networks In this case, there is a forward flow of data from the input units to the output units. The processing of the data can be extended to multiple layers of hidden units, but there are no feedback connections. In other words, there are no connections from the outputs of the units to the inputs of the units of the same or previous layers.

Recurrent Network These networks include feedback connections. Unlike in feedforward networks, in this case the dynamic properties of the network are significant.

Learning Rule When a set of inputs is inserted in the neural network, it should return the desired set of outputs. To do so, appropriate values of the weights must be selected. One method is to give values to the weights relying on existing (a priori) knowledge of the problem. However, in most cases this knowledge is not available. Another method is to "train" the network by presenting "training examples" and allow

the network to change the weights of the connections according to some learning rule. The ways in which learning is conducted generally falls into two broad categories: supervised and unsupervised learning.

In *supervised* or *associative learning*, the network is trained by providing training input and their corresponding output examples. The network gradually learns the underlying relationship between the inputs and the output. In *unsupervised learning* or *self-organizing*, the output units are trained to respond in some complexes of input examples and to discover some of their prominent statistical properties. Unlike supervised learning, there are no a priori categories in which the patterns can be classified, but the network should develop its own representation for the input stimuli. Both types of learning result in an adjustment of the weights of the connections between the units, according to some learning rule.

PERCEPTRON

Suppose that a feedforward neural network has no hidden layers. Hence, the output units are connected directly with the input units. Such a network is shown in Figure 2.6. More precisely, the network shown has only one output unit, two input units, and a bias term, θ.

Furthermore, we assume a training sample that consists of the input vector $\mathbf{x} = (x_1, x_2)$ and the corresponding desired output y. In classification problems y usually takes values of -1 and $+1$. The *perceptron learning algorithm* that updates the weights is the following:

1. Initialize the weights to random values.
2. Select an input vector from the training sample, present it to the network, and compute the output of the network, out, for the input vector specified and the values of the weights.
3. If out $\neq y$ (i.e., the response of the perceptron is incorrect), modify all connections w_i by adding the changes $\Delta w_i = yx_i$.
4. Go back to step 1.

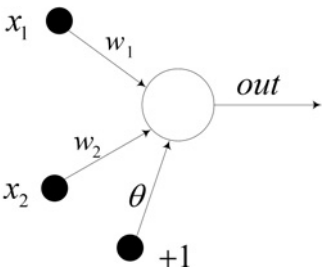

Figure 2.6 *Feedforward neural network without hidden layers with one output and two inputs.*

When the response of the network is correct, the weights are not updated. The weight of the bias term should also be updated. The bias may be seen as the connection w_0 between the output of the network and an entry input that always has the value of 1. Hence, based on the perceptron algorithm, the change of bias is zero if the network response is correct and is equal to y otherwise.

However, in problems where the input–output pairs are not linearly separable, they cannot be modeled with the perceptron rule; the use of neural networks with intermediate hidden layers is required.

The Approximation Theorem

For the perceptron learning rule described above, Rosenblatt (1959) formulated and proved the famous theorem of convergence, which states: If there is a set of connection weights $\mathbf{w}^*$ that is able to perform the transformation out $= y$, the perceptron learning rule will converge to a solution (which may or may not be the same as $\mathbf{w}^*$) after a finite number of steps for any initial selection of weights. For a proof of the theorem the reader is directed to Rosenblatt (1959).

THE DELTA RULE

For a network with no hidden units, one output, and a linear activation function, the output of the network is given by

$$\text{out} = \sum_i w_i x_i + \theta \tag{2.9}$$

Then, for a given input vector output from the training sample, we have that

$$\text{out}_p = \sum_i w_i x_{pi} + \theta \tag{2.10}$$

Such a simple network has the ability to represent a linear relationship between the input and output variables. Using a sign function as an activation function in the output unit, we can construct a classifier, such as the Adaline of Widrow and Hoff (1960). Here we focus on the linear relationship, but we will use the network for a function approximation problem.

We assume that we want to train a network to adapt as best as possible to a hyperplane in a training sample consisting of pairs of the form $(\mathbf{x}_p, y_p)$, where $\mathbf{x}_p = (x_1, x_2, \ldots, x_m)$ and $p = 1, \ldots, n$. For each input vector $\mathbf{x}_p$, the network's output differs from the target value by $y_p - \text{out}_p$. The error function that uses the delta rule is based on the squares of these differences. Specifically, the error for the example p is calculated from the relationship

$$E_p = \tfrac{1}{2}(y_p - \text{out}_p)^2 \tag{2.11}$$

The total error that is minimized by the delta rule is given by the relationship

$$E = \sum_{p=1}^{n} E_p = \tfrac{1}{2} \sum_{p=1}^{n} (y_p - \text{out}_p)^2 \tag{2.12}$$

The minimum mean error finds the values of the weights that minimize the error function (2.12) with the *method of gradient descent*, also called the *method of steepest descent*. This method is based on the change in weight w_i by $\Delta_p w_i$, which is proportional to the negative value of the derivative of the error E_p which has been calculated for the training pattern p with respect to the weight w_i, namely:

$$\Delta_p w_i = -\eta \frac{\partial E_p}{\partial w_i} \tag{2.13}$$

where η is a constant called the *learning rate*. Using the chain rule, the derivative $\partial E_p / \partial w_i$ can be written as

$$\frac{\partial E_p}{\partial w_i} = \frac{\partial E_p}{\partial \text{out}_p} \frac{\partial \text{out}_p}{\partial w_i} \tag{2.14}$$

The first derivative is easily calculated from (2.11) as follows:

$$\frac{\partial E_p}{\partial \text{out}_p} = -(y_p - \text{out}_p) = -\delta_p \tag{2.15}$$

where δ_p is the difference between the output of the network and the target value y for pattern p. Because of the linear output unit (2.10), the partial derivative of the output of the network with respect to the weight w_i is

$$\frac{\partial \text{out}_p}{\partial w_i} = x_{pi} \tag{2.16}$$

Placing equations (2.15) and (2.15) into (2.14), we have that

$$-\frac{\partial E_p}{\partial w_i} = \delta_p x_{pi} \tag{2.17}$$

Substituting into equation (2.13), we finally get

$$\Delta_p w_i = -\eta \delta_p x_{pi} \tag{2.18}$$

which express the delta rule. Combining equation (2.17) with the equation

$$\frac{\partial E}{\partial w_i} = \sum_p \frac{\partial E_p}{\partial w_i} \tag{2.19}$$

we conclude that the change in the weight w_i after a complete cycle of presentation of all the training patterns of the training sample is similar to that of the derivative, and thus the delta rule implements a gradient descent in the space $E - w$. This is true only if the weights change only at the end of the cycle. If the weights change after the presentation of each training example, the method deviates slightly from the true gradient descent. If the learning rate is small enough, this deviation would be negligible and the delta rule would be a very good approximation of the gradient descent to the sum of squared errors. More precisely, if the learning rate is small enough, the delta rule will find a set of weights that minimize the error function.

Although this algorithm is better than the one applied to perceptrons, it cannot be applied to networks that have hidden layers. For each neuron, its output must be known exactly, which is not possible when there are hidden layers.

BACKPROPAGATION NEURAL NETWORKS

As we saw earlier, neural networks without hidden layers are characterized by severe limitations, as the set of problems which they solve is very limited. Minsky and Papert (1969) showed that neural networks with hidden layers can overcome many of these limitations; however, a solution to the problem of adjusting the weights of the connections between the input units and the hidden units was not given. One solution to this problem was given by Rumelhart et al., (1986a).

The idea behind this method is that errors in the hidden units of the hidden layer are determined by the backpropagation of errors in the output units. This is why this training algorithm is called the *backpropagation learning rule*. The backpropagation algorithm can be seen as a generalization of the delta rule for the case of neural networks with hidden layers and nonlinear activation functions.

Multilayer Feedforward Networks

The processing units of feedforward networks are organized in layers, which is why they are called *multilayer networks*. The first of these layers, the *input layer*, is used for data entry. The processing units of this layer do not perform any computations (they do not have any input weights or activation functions).

Next are one or more intermediate hidden layers, then an *output layer*. The units of each layer receive their inputs from the units of the layer immediately below (behind) and send their outputs to the units of the layer lying directly above (front).

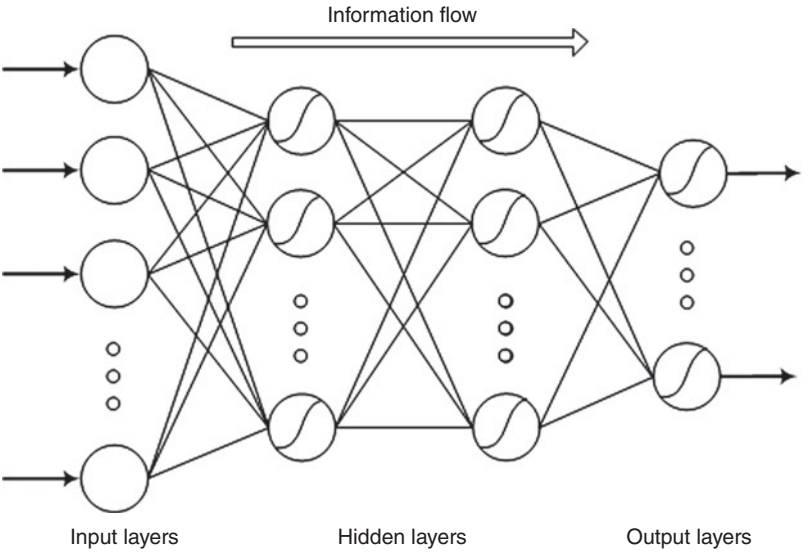

Figure 2.7 *Fully connected feedforward neural network with two hidden layers.*

The network can be connected either fully or partially. In the former case, the processing units of different layers are associated with all the units of the next layer, whereas in the Latter case the connection is with only some of them. There are no feedback connections (i.e., connections that send the output of a unit in the same or a previous layer). Also, there are no connections between units of the same layer. The flow of information is done through the connections and the processing units in one direction only: from the input layer to the output layer. In Figure 2.7, a schematic representation of a fully connected feedforward neural network with two hidden layers is presented.

Although the backpropagation learning rule may be applied to feedforward networks with any number of hidden layers, a series of studies (Funahashi, 1989; Hartman et al., 1990; Hornik et al., 1989) have proved that a single hidden layer is sufficient for the neural network to approximate any function to any random degree of accuracy, with the condition that the activation functions of the network are nonlinear. This is known as the universal approximation theorem. In most cases, neural networks with a single hidden layer and sigmoidal transfer function are used.

THE GENERALIZED DELTA RULE

The backpropagation method is the most widely used method for training multilayer neural networks. The basic idea behind this training algorithm is to determine the percentage of the total error for the weight of each connection. Hence, it is possible

to calculate the correction to the weight of each connection separately, which is quite complex for the hidden layers since their outputs affect many connections simultaneously.

In backpropagation, initially the error of the output unit is estimated in the same way as for the delta rule. This error is used to calculate the errors in the last hidden layer. Then the process is repeated recursively toward the input layer. Based on the propagation of the error backward, it is possible to estimate the contribution of each connection to the total error. Then the errors estimated for each connection of the layer are used to alter the weights of the connections in a manner similar to that of the delta rule. The process is therefore based on a generalization of the delta rule, which is why it is called the generalized delta rule. The procedure is repeated until the value of the total error reaches a predefined value.

As discussed earlier, the delta rule performs a gradient descent on the sum of squared errors for linear transfer functions. In the case of no hidden units, error surface has a convex shape with a unique minimum. It is therefore it is guaranteed that the gradient descent will eventually find the best set of weights. However, when the neural network has hidden units, the error surface does not have a unique minimum. As a result, the alogrithm might reach and be trapped into a local minimum.

In the delta rule algorithm, first a set of training patterns are presented to the network. The training patterns consist of the input and output vectors. First, the network uses the input vector to calculate its own output vector and then it compares it against the desired output vector (the target). If there is no difference, the learning procedure stops. Otherwise, the weights of the network connections are modified to reduce the difference. If we do not have any hidden layers, the network connections change according to the delta rule. More precisely, the change in the weight of the connection between the input and output units is given by (2.18).

In the remainder of the section we present the generalized delta rule for multilayer feedforward neural networks with nonlinear activation functions. The net input of an output unit u_j in Figure 2.8 for the output–input vector p is given by

$$\text{net}_{pj} = \sum_i w_{ij} \text{out}_{pi} \tag{2.20}$$

To calculate the output of the same unit, an upward, continuous, and differentiable function is used:

$$\text{out}_{pj} = \gamma(\text{net}_{pj}) \tag{2.21}$$

The usual function that is used is the sigmoid. In the generalized delta rule the changes in the weights are given by

$$\Delta_p w_i = -\eta \frac{\partial E_p}{\partial w_{ij}} \tag{2.22}$$

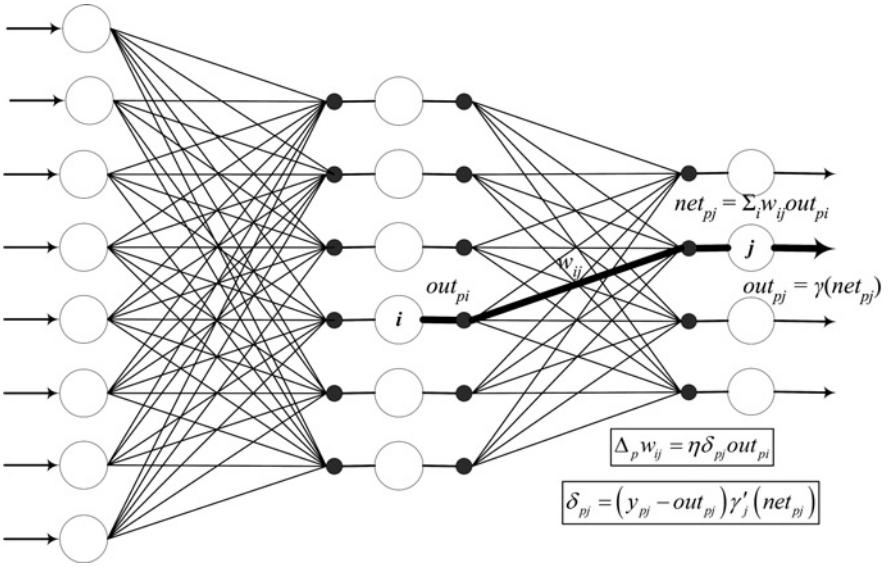

Figure 2.8 *Estimation of the delta in a feedforward neural network for the connections to the output units of the network.*

where E is the same measurement of the square errors used earlier. Using the chain rule, the derivative $\partial E / \partial w_{ij}$ can be written as

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial \text{net}_{pj}} \frac{\partial \text{net}_{pj}}{\partial w_{ij}} \tag{2.23}$$

From equation (2.20) we see that the partial derivative of the total input to unit j based on the weight of one of the connections w_{ij} is

$$\frac{\partial \text{net}_{pj}}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \left(\sum_k w_{kj} \text{out}_{pk} \right) = \text{out}_{pi} \tag{2.24}$$

Next, we define

$$\delta_{pj} = -\frac{\partial E_p}{\partial net_{pj}} \tag{2.25}$$

This definition is consistent with the definition of the delta given by equation (2.18) since for linear transfer functions we have that $\text{net}_{pj} = \text{out}_{pj}$.

From relationships (2.23) to (2.25) we have that

$$-\frac{\partial E_p}{\partial w_{pj}} = \delta_{pj}\text{out}_{pj} \tag{2.26}$$

and by replacing (2.26) with (2.22) we observe that the changes in the weights of the connections can be computed by

$$\Delta_p w_{pj} = \eta\delta_{pj}\text{out}_{pj} \tag{2.27}$$

From equation (2.27) we observe that the change in weight w_{ij} of the connection between the units u_i and u_j (the unit u_j is located in the next layer of unit u_i) is a product of the learning rate and the delta of unit u_j and the output unit u_i. The learning rate is a constant chosen by us, and the output of unit i is easily calculated from equation (2.21). Hence, we are interested in estimation of the delta of each unit. Applying the chain rule in (2.25), we have that

$$\delta_{pj} = -\frac{\partial E_p}{\partial \text{net}_{pj}} = -\frac{\partial E_p}{\partial \text{out}_{pj}}\frac{\partial \text{out}_{pj}}{\partial \text{net}_{pj}} \tag{2.28}$$

From (2.21) we see that the second term of the product in (2.28) is just the derivative of the transfer function of the unit u_j:

$$\frac{\partial \text{out}_{pj}}{\partial net_{pj}} = \gamma'(\text{net}_{pj}) \tag{2.29}$$

To estimate the first term of (2.28) we have to distinguish between two cases. First we consider the case where the unit u_j is located in the output layer, as in Figure 2.8. From the definition of the error criterion we have that

$$\frac{\partial E_p}{\partial \text{out}_{pj}} = -(y_{pj} - \text{out}_{pj}) \tag{2.30}$$

Therefore, substituting relations (2.30) and (2.29) into (2.28) we find that the delta of the output units of the network is calculated as follows:

$$\delta_{pj} = (y_{pj} - \text{out}_{pj})\,\gamma'(\text{net}_{pj}) \tag{2.31}$$

Next, we focus in the case where the unit u_j is located in a hidden layer, as in Figure 2.9. In this case the contribution of the output of the unit to the error E_p cannot be calculated directly. However, measurement of the error is a function of the net input to the units of the output layer:

$$E_p = E_p(\text{net}_{p1}, \text{net}_{p2}, \ldots, \text{net}_{pk}, \ldots) \tag{2.32}$$

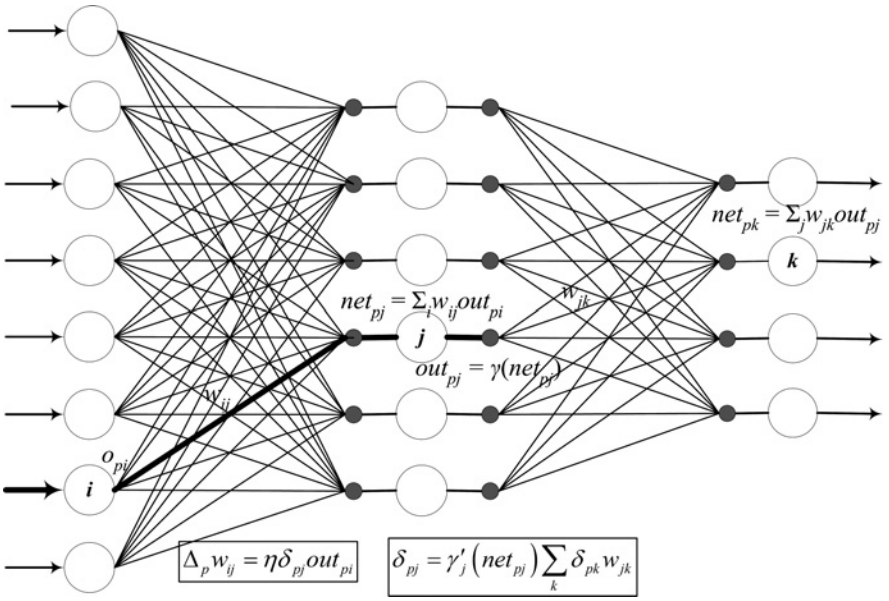

Figure 2.9 *Estimation of the delta of a feedforward network in the hidden layer.*

Applying the chain rule, we have

$$\begin{aligned}\frac{\partial E_p}{\partial \text{out}_p} &= \sum_k \frac{\partial E_p}{\partial \text{net}_{pk}} \frac{\partial \text{net}_{pk}}{\partial \text{out}_p} \\ &= \sum_k \frac{\partial E_p}{\partial \text{net}_{pk}} \frac{\partial}{\partial \text{out}_p} \left(\sum_j w_{jk} \text{out}_{pj} \right) \\ &= \sum_k \frac{\partial E_p}{\partial \text{net}_{pk}} w_{jk} \\ &= -\sum_k \delta_{pk} w_{jk} \end{aligned} \tag{2.33}$$

By replacing equations (2.29) and (2.33) with (2.28), we get

$$\delta_{pj} = \gamma'_j(\text{net}_{pj}) \sum_k \delta_{pk} w_{jk} \tag{2.34}$$

BACKPROPAGATION IN PRACTICE

The generalized delta rule is applied in two steps. During the first step, the input vector **x** is introduced to the network and is propagated from layer to layer toward the

output layer and the network's outputs are computed. The outputs are compared with the target values desired, creating an error signal from each output of the network. At the second step, the error signal propagates backward through the network (i.e., from the output to the input layer). At the same time, the appropriate changes in the weights of the network's connections are estimated.

These two steps are summarized as follows:

- For a given input vector and initial values of the weights of the network connections, the net inputs and outputs of each unit are estimated, moving from the input layer to the output layer. Finally, the approximation error is estimated.
- Then, the derivatives of the transfer functions and the delta of the output units are estimated.

The most commonly used activation function is the asymmetrical sigmoid with asymptotic values 0 and 1:

$$\gamma(\text{net}_{pj}) = \frac{1}{1 + e^{-\text{net}_{pj}}} \tag{2.35}$$

The derivative of this transfer function with respect to net_{pj} is given by

$$\begin{aligned}
\gamma'(\text{net}_{pj}) &= \frac{d}{d\,\text{net}_{pj}} \frac{1}{1 + e^{-\text{net}_{pj}}} \\
&= \frac{e^{-\text{net}_{pj}}}{(1 + e^{-\text{net}_{pj}})^2} \\
&= \frac{1}{(1 + e^{-\text{net}_{pj}})} \frac{e^{-\text{net}_{pj}}}{(1 + e^{-\text{net}_{pj}})} \\
&= \gamma(\text{net}_{pj})[1 - \gamma(\text{net}_{pj})]
\end{aligned} \tag{2.36}$$

As we see the derivative of the activation function is expressed as a function of itself. Substituting the relationship above to the delta, we have that

$$\delta_{pj} = (y_{pj} - \text{out}_{pj})\gamma_j(\text{net}_{pj})[1 - \gamma_j(\text{net}_{pj})] \tag{2.37}$$

- Next, the changes in the weights of connections of the output layer are computed using the generalized delta rule.
- Next, the derivatives of the transfer functions and the delta of the units of the previous hidden layer are computed.

$$\delta_{pj} = \gamma_j(\text{net}_{pj})[1 - \gamma_j(\text{net}_{pj})] \sum_k \delta_{pk} w_{jk} \tag{2.38}$$

- Then the changes of weights of the connections of the previous hidden layer are computed using the generalized delta rule.
- The previous two steps are repeated for all hidden layers moving from the output layer to the input layer.

TRAINING WITH BACKPROPAGATION

The backpropagation algorithm implements a search for the total minimum error of the function $E(\mathbf{w})$, which has as parameters the values of the weights. In each step the weights are corrected by choosing a change that seems to reduce the error locally with the aim of minimizing the error $E(\mathbf{w})$. Given the current weight vector $\mathbf{w}_c$, for each iteration a direction $\mathbf{u}_c$ is calculated and then the weight vector $\mathbf{w}_c$ is updated using the following learning rule:

$$\mathbf{w}_+ = \mathbf{w}_c - \eta \mathbf{u}_c \tag{2.39}$$

where $\mathbf{w}_+$ is the new weight vector and η is the learning rate. Specifically, the backpropagation algorithm calculates the slope $\nabla E(\mathbf{w}_c) = \partial E \big/ \partial \mathbf{w}_c$ and then performs a minimization step toward the direction $\mathbf{u}_c = \nabla E(\mathbf{w}_c)$.

Note that the gradient operator ∇ is meaningless by itself. On the other hand, the gradient vector $\nabla E(\mathbf{w}_c)$ points in the direction of the maximum growth rate of the error function at point $\mathbf{w}_c$ and is equal to this growth rate. Hence, the learning rule is

$$\mathbf{w}_+ = \mathbf{w}_c - \eta \nabla E(\mathbf{w}_c) \tag{2.40}$$

As presented in (2.39), the product between $\mathbf{u}_c$ and the learning rate is subtracted from the current weight vector. When $\nabla E(\mathbf{w}_+)$ is vertical to $\mathbf{u}_c$, the backpropagation algorithm has found a minimum $\mathbf{w}_{\min}$.

In Figure 2.10 a simplistic schematic representation of the gradient descent is shown. In reality, the surface $(E - \mathbf{w})$ cannot be represented graphically when there are two more weights: in other words, for any useful neural network topology.

However, the greater the value of the learning rate η, the greater the risk that the stepwise gradient descent starts oscillating, as presented in Figure 2.11. In this case it is impossible to find the global minimum as the algorithm oscillates between two points (e.g., in points 3 and 4 in Figure 2.11).

On the other hand, very small values of the learning rate drastically reduce the chances that the algorithm will start oscillating. However, the algorithm becomes significantly slower. A schematic representation of the behavior of backpropagation algorithm with a relatively small learning rate is presented in Figure 2.12.

To avoid oscillations and at the same time to speed up the learning process, a momentum term is added to equation (2.27):

$$\Delta_p w_{ij}(t+1) = \eta \delta_{pj} \mathrm{out}_{pi} + m \Delta_p w_{ij}(t) \tag{2.41}$$

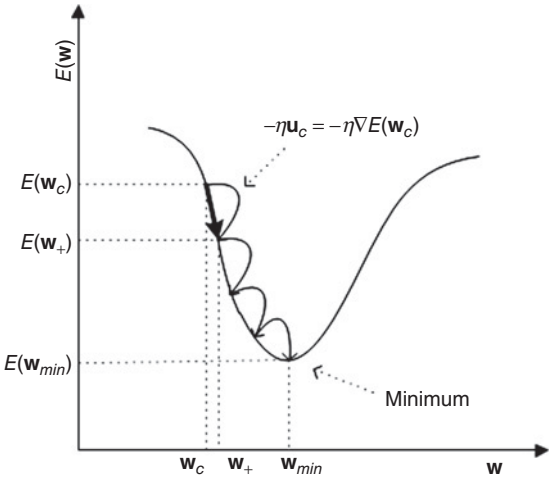

Figure 2.10 *Simplistic representation of the minimum error search through stepwise gradient descent in the weights–error surface* ($E - \mathbf{w}$) *that is implemented by the backpropagation algorithm.*

where m is the momentum term and t refers to the iteration number. From equation (2.41) we observe that the change in the weight estimated in the previous presentation of the training patterns is multiplied by a constant (the momentum) and then is added to the current change of the weight. The momentum determines the effect of the previous change to the next. The addition of the momentum term allows us to use relatively small values for the learning rate without increasing the learning time significantly.

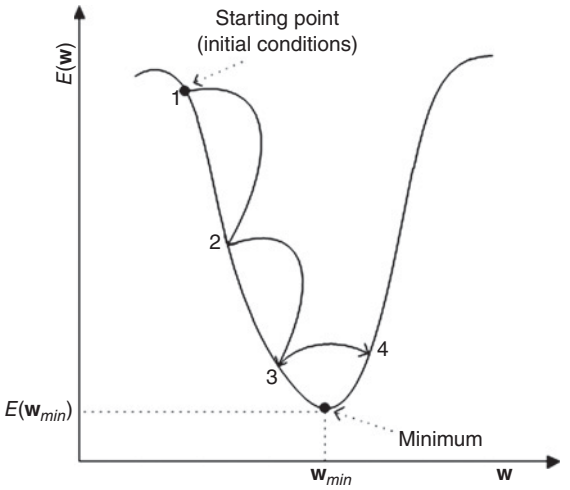

Figure 2.11 *Stepwise descent to the surface* ($E - \mathbf{w}$) *with a very large learning rate. The backpropagation algorithm starts swinging between points 3 and 4, so it is not possible to find the point that minimizes the error function.*

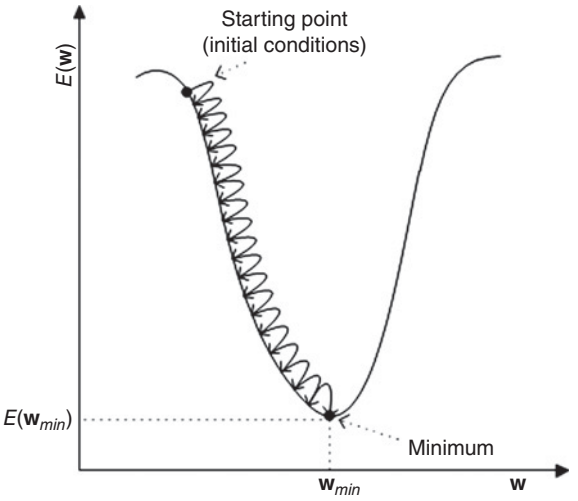

Figure 2.12 *Stepwise descent to the surface* $(E - \mathbf{w})$ *with a very small learning rate, which results in long training times (more iterations).*

Since in the change of weight a proportion of the previous variation is added, when the algorithm enters a region of the surface $(E - \mathbf{w})$ with a high gradient, it begins to perform a continuously accelerated descent until it reaches the minimum (Figure 2.13). Hence, the relationship (2.40) with the momentum term becomes

$$\mathbf{w}_{+} = \mathbf{w}_{c} - \eta \nabla E(\mathbf{w}_{c}) - m \nabla E(\mathbf{w}_{c-1}) \tag{2.42}$$

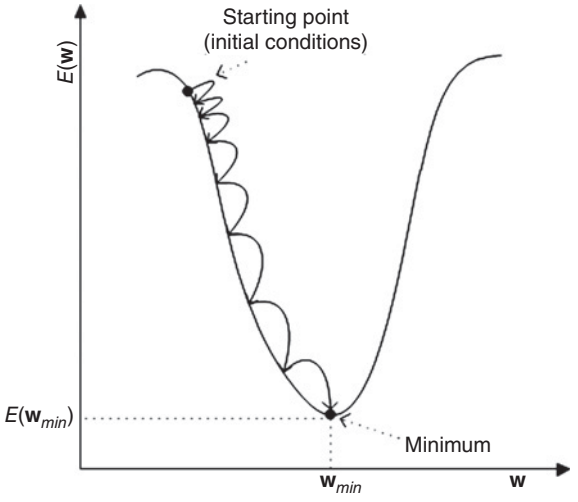

Figure 2.13 *Stepwise descent to the surface* $(E - \mathbf{w})$ *with a relatively small learning rate and momentum term, which accelerates the search for the minimum of the error function.*

In general, the purpose of the addition of the momentum term is to accelerate the network's training process without a disproportionate increase the probability that the algorithm will be trapped in oscillations between two nonoptimal solutions.

Despite the obvious potential of the backpropagation algorithm, under some circumstances various problems can occur. It is possible, for example, for the algorithm to reach a static state known as *network paralysis*, due to the saturation of the activation function. Additional problems arise due to the existence of *local minima* in the weights–error surface and due to the nonuniqueness of the solutions. Finally, a common problem that can occur is long training times, as a result of the suboptimal choice of the learning rate and the momentum values. This can be treated by extending the backpropagation algorithm and using a variable learning rate and momentum. We examine these problems and how to tackle them next.

Network Paralysis

One problem that may possibly arise in the training of the network is for the backpropagation algorithm to reach a stationary state. Network paralysis may occur due to the values that the network's weights may reach. If the values of the weights become too high, this will also lead to very high values of the net input of some hidden units. Furthermore, if a sigmoid activation function is used, the output of the unit will be very close to 1. In this case the delta estimated from relations (2.37) and (2.38) will be very close to zero, as $\gamma_j(\text{net}_{pj})[1 - \gamma_j(\text{net}_{pj})] \approx 0$. Hence, the changes of the weights calculated from relation (2.27) would be practically zero. So, in reality, the learning process stops.

The same problem would arise in the event that some weights had very low values. Then the net input of some units will be very small, and as a result the output of the unit will be very close to zero. Since $\gamma_j(\text{net}_{pj})[1 - \gamma_j(\text{net}_{pj})] \approx 0$, the delta would also be negligible, as would the changes in weights.

Wang et al., (2004) proposed a solution to the problem of network paralysis. For each training sample they suggested using a different activation function. In addition, the activation functions would be adjusted continuously during training to avoid saturation.

Local Minima

Another problem that arises during the training of a network is associated with the morphology of the weight–error surface ($E - \mathbf{w}$), which for feedforward neural networks with hidden layers is generally very complicated. A simplistic two-dimensional representation is shown in Figure 2.14, where we see that there is more than one minimum (points B, C, and D). The backpropagation algorithm can reach one of these solutions, depending on the starting point (points A and E).

The way in which the stepwise gradient descent is implemented does not provide any guarantee that the solution found will be the overall optimal, that is, that it will correspond to the lower overall error level (point C in Figure 2.14). This solution is known as the *global minimum*; other solutions are called *local minima*. The inherent

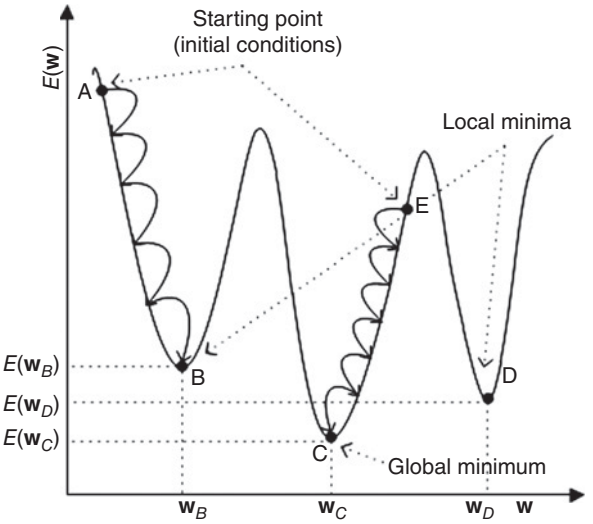

Figure 2.14 *Surface weight–error* ($E - \mathbf{w}$) *with multiple minima. The backpropagation algorithm can be "trapped" in a local minimum (point B) which does not correspond to the total minimum error (point C), depending on the starting point of the gradient descent.*

weakness of the gradient descent to find the global minimum is due to the selection of the architecture (topology) of the neural network in relation to the complexity of the problem. When the complexity of the network is greater than required for a relevant application and for the size of the training sample, the estimated model will suffer from a high level of variance. This means that for the same network architecture, repeated sampling from the training sample, then fitting a model to the new sample, will result in quite different solutions: in other words, to different weight vectors. On the other hand, when the network architecture is simpler than necessary—that is, the number of network parameters is relatively small—the fitted model runs the risk of being underparameterized or *biased*.

A simple method that is used to increase the chances of finding the global minimum is called *weight jogging*. In this method, when an initial solution is found, a small and random change is added to the weight vector and the training process continues until the algorithm converges to a new solution. A simple example of weight jogging is shown in Figure 2.15. The backpropagation algorithm has converged initially at point B, which corresponds to a local minimum. In the weight vector of the solution $\mathbf{w}_B$, a small random change is added. The modified vector $\mathbf{w}_C$ corresponds, as expected, to a greater level of error, but is located such that if network training continues, it will lead to the global minimum D.

Of course, there is no guarantee that this process will lead to a better solution, as it is simply an improvement in the local search carried out by the backpropagation algorithm. Global search algorithms, as the simulated annealing, deal much better with the problem of local minima.

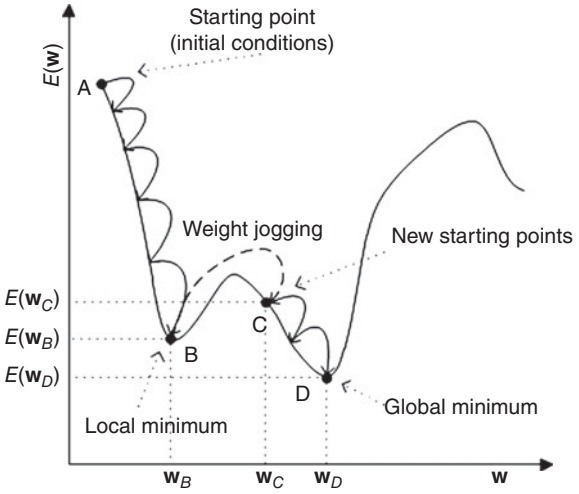

Figure 2.15 *Small and random changes in the original solution (weight jogging) increase the chances of finding the global minimum.*

Nonunique Solutions

Whether the solution corresponds to a local or a global minimum, it is sometimes not unique. This means that there are multiple combinations of weights and/or processing units (i.e., more than one network topology) that correspond to the same value of the error function (Chen and Hecht-Nielsen, 1991; Chen et al., 1993).

In general, we may have different local minima, derived from different weights that correspond to the same error level. This case is presented in Figure 2.16 in solutions Z and G, where $E_2 = E(\mathbf{w}_Z) = E(\mathbf{w}_G)$. Different initial conditions lead to different solutions but are equivalent in the sense of minimizing the error function.

Another case is to have many similar solutions corresponding to the same level of error and to create a flat minimum. This is the plateau that is shown in Figure 2.16, where the adjacent weight vectors $\mathbf{w}_B$, $\mathbf{w}_C$, and $\mathbf{w}_D$ correspond to the same error, $E_1 = E(\mathbf{w}_B) = E(\mathbf{w}_C) = E(\mathbf{w}_D)$. The existence of such plateaus is a characteristic of the overparameterized networks, where the complexity of their architecture exceeds the requirements of the particular application. This problem is addressed by such prunning methods as the ICE (Zapranis and Haramis, 2001), OBS (Hassibi and Stork, 1993), and OBD (LeCun et al., 1989), algorithms.

CONFIGURATION REFERENCE

Neural networks are nonlinear estimators; hence, there are no a priori assumptions regarding the structure of the model. As presented earlier, the stepwise algorithms used for training of the neural networks are known as learning algorithms, due to their iterative nature. White (1989) has demonstrated that the learning algorithms

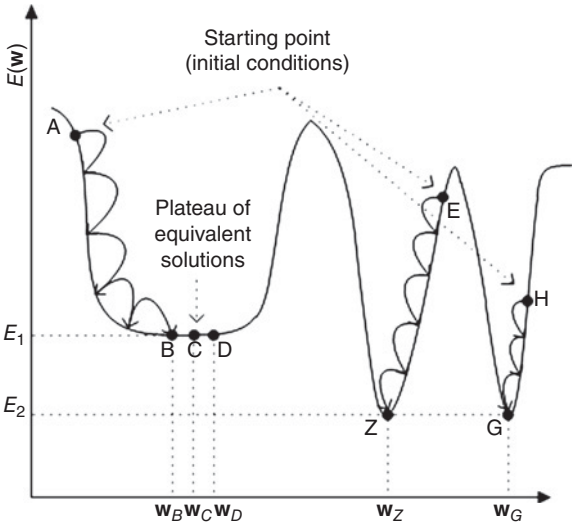

Figure 2.16 *Weight–error surface* $(E - \mathbf{w})$ *with a flat minimum. Various vector weights* $\mathbf{w}_B$, $\mathbf{w}_C$, *and* $\mathbf{w}_D$ *of the solutions* B, C, *and* D, *corresponding to the same level of error,* E_1. *Also, the local minima G and I correspond to the same level error,* E_2.

of neural networks can be formulated as a nonlinear regression problem. We have also demonstrated that the main strength of the neural networks lies in the universal approximation property. Furthermore, 1 hidden layer is sufficient for a backpropagation neural network to approximate to any degree any function and its derivatives.

The usual structure of a backpropagation neural network that is used in the majority of the applications is presented in Figure 2.17. The network has only 1 hidden layer with λ hidden units. The input layer consists of m units, without any activation function. The input signal of these units is identical to the output signal. All other units of the neural network use the asymmetric sigmoidal activation function with values in (0, 1). In addition, in the input and output layers, a bias unit is added. These units do not accept any input, and their output has a constant value of 1. The desired output of the network can be continuous or discrete.

The number m of the input units is determined by the relevant problem. The term *architecture* or *topology* of a network, A_λ, refers to the topological arrangement of the network's connections. We define a class of neural networks S_λ:

$$S_\lambda \equiv \{g_\lambda(\mathbf{x}; \mathbf{w}), \mathbf{x} \in \Re^m, \mathbf{w} \in \mathbf{W}, \mathbf{W} \subseteq \Re^p\} \tag{2.43}$$

where $g_\lambda(\mathbf{x}; \mathbf{w})$ is a nonlinear function, $\mathbf{w} = (w_1, w_2, \ldots, w_p)$ is the vector of parameters (i.e., the weights of network connections), and p is the number of parameters determined by the topology A_λ.

The class of neural networks S_λ is a set of neural networks that have the same architecture and whose members are differentiated and simultaneously defined entirely

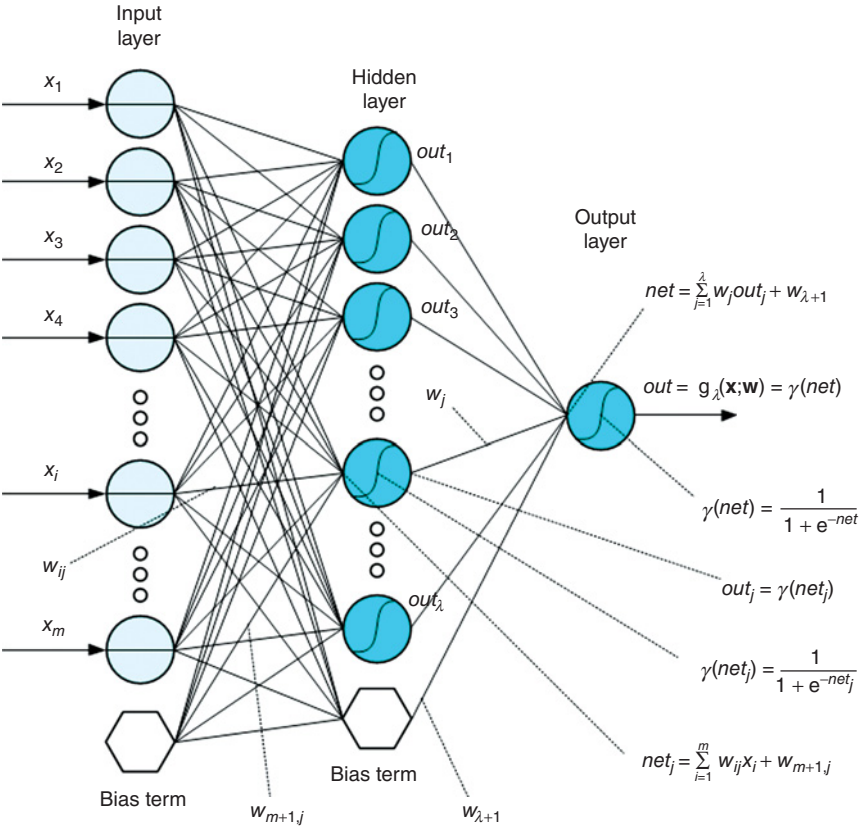

Figure 2.17 *Backpropagation neural network with 1 hidden layer and 1 output unit.*

by the weight vector **w**. In the case of a neural network with a 1-hidden-layer architecture (Figure 2.17), the number of hidden units λ defines the different classes S_λ since it uniquely determines the dimension of the parameter vector p, where $p = (m + 2)\,\lambda + 1$.

For this type of network, given an input vector **x** and a weight vector **w**, the output of the network $g_\lambda(\mathbf{x};\mathbf{w})$ is

$$g_\lambda(\mathbf{x};\mathbf{w}) = \gamma\left(\sum_{j=1}^{\lambda} w_j \gamma\left(\sum_{i=1}^{m} w_{ij}x_i + w_{m+1,j}\right) + w_{\lambda+1}\right) \tag{2.44}$$

where w_{ij} is the weight of the connection between the input i and the hidden unit j, $w_{m+1,j}$ is the weight of the connection between the term bias in the input layer $m + 1$ and the hidden unit j, w_j is the weight of the connection between the hidden unit j and the output unit, and $w_{\lambda+1}$ is the weight of the connection between the bias term of the hidden layer $\lambda + 1$ unit and the output.

CONCLUSIONS

Artificial neural networks (or simply, neural networks) derive their origins from biological neural networks. They are systems of parallel and distributed processing that simulate the basic operating principles of the biological brain.

Feedforward neural networks with at least 1 hidden layer and nonlinear activation functions have the property of universal approximation. In other words, they can approximate any function. The stepwise algorithm that is used for their training is based on the generalization of the delta rule that is used in perceptron networks. Updating the weights of the network is done by propagating the error backward, so this method is known as the backpropagation algorithm and the networks as backpropagation neural networks. The backpropagation algorithm, despite its disadvantages, is the most commonly used training algorithm, as it is simpler and significantly less computationally expensive.

From a statistical perspective, neural networks can be formulated as a nonparametric nonlinear regression model. In this book we treat neural networks as the eminent expression of nonparametric regression, which constitutes a very powerful approach, especially for financial applications. The main characteristic of neural networks is their ability to approximate any nonlinear functions without making a priori assumptions about the nature of the process that created the available observations. This is particularly useful in financial applications, where there are many hypotheses but little is actually known about the nature of the processes that determine the prices of assets. Nevertheless, knowledge of the relative theory of neural learning and the basic models of neural networks are essential for their effective implementation, especially in complex applications such as the ones encountered in finance.

REFERENCES

Chen, A. M., and Hecht-Nielsen, R. (1991). "On the geometry of feedforward neural network weight spaces." *Second International Conference on Artificial Neural Networks*, 1–4.

Chen, A. M., Lu, H.-M., and Hecht-Nielsen, R. (1993). "On the geometry of feedforward neural network error surfaces." *Neural Computation*, 5(6), 910–927.

Funahashi, K.-I. (1989). "On the approximate realization of continuous mappings by neural networks." *Neural Networks*, 2(3), 183–192.

Hartman, E. J., Keeler, J. D., and Kowalski, J. M. (1990). "Layered neural networks with Gaussian hidden units as universal approximations." *Neural Computation*, 2(2), 210–215.

Hassibi, B., and Stork, D. G. (1993). "Second order derivatives for network pruning: optimal brain surgeon." *Advances in Neural Information Processing Systems*, 164–164.

Hornik, K., Stinchcombe, M., and White, H. (1989). "Multilayer feedforward networks are universal approximators." *Neural Networks*, 2(5), 359–366.

LeCun, Y., Denker, J. S., Solla, S. A., Howard, R. E., and Jackel, L. D. (1989). "Optimal brain damage." *NIPS*, 598–605.

Minsky, M., and Papert, S. (1969). *Perceptron: an Introduction to Computational Geometry*, expanded edition. MIT Press, Cambridge, MA, pp. 19, 88.

Rosenblatt, F. (1959). *Principles of Neurodynamics*. Spartan Books, New York.

Rumelhart, D. E., Hintont, G. E., and Williams, R. J. (1986a). "Learning representations by back-propagating errors." *Nature*, 323(6088), 533–536.

Rumelhart, D. E., McClelland, J. L., and University of California–San Diego PDP Research Group. (1986b). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, MIT Press, Cambridge, MA.

Wang, X., Tang, Z., Tamura, H., Ishii, M., and Sun, W. (2004). "An improved backpropagation algorithm to avoid the local minima problem." *Neurocomputing*, 56, 455–460.

White, H. (1989). "Learning in artificial neural networks: a statistical perspective." *Neural Computation*, 1, 425–464.

Widrow, B., and Hoff, M. E. (1960). "Adaptive switching circuits." *IRE WESCON Convention Report*, 96–104.

Zapranis, A. D., and Haramis, G. (2001). "An algorithm for controlling the complexity of neural learning: the irrelevant connection elimination scheme." *Fifth Hellenic European Research on Computer Mathematics and Its Applications*, Athens, Greece.