CHAPTER 6

Backpropagation Neural Net

6.1 STANDARD BACKPROPAGATION

The demonstration of the limitations of single-layer neural networks was a significant factor in the decline of interest in neural networks in the 1970s. The discovery (by several researchers independently) and widespread dissemination of an effective general method of training a multilayer neural network [Rumelhart, Hinton, & Williams, 1986a, 1986b; McClelland & Rumelhart, 1988] played a major role in the reemergence of neural networks as a tool for solving a wide variety of problems. In this chapter, we shall discuss this training method, known as *backpropagation* (of errors) or the *generalized delta rule*. It is simply a gradient descent method to minimize the total squared error of the output computed by the net.

The very general nature of the backpropagation training method means that a backpropagation net (a multilayer, feedforward net trained by backpropagation) can be used to solve problems in many areas. Several of the applications mentioned in Chapter 1—for example, NETtalk, which learned to read English aloud—were based on some variation of the backpropagation nets we shall describe in the sections that follow. Applications using such nets can be found in virtually every field that uses neural nets for problems that involve mapping a given set of inputs to a specified set of target outputs (that is, nets that use supervised training). As is the case with most neural networks, the aim is to train the net to achieve a balance between the ability to respond correctly to the input patterns that are used for training (memorization) and the ability to give reasonable

(good) responses to input that is similar, but not identical, to that used in training (generalization).

The training of a network by backpropagation involves three stages: the feedforward of the input training pattern, the calculation and backpropagation of the associated error, and the adjustment of the weights. After training, application of the net involves only the computations of the feedforward phase. Even if training is slow, a trained net can produce its output very rapidly. Numerous variations of backpropagation have been developed to improve the speed of the training process.

Although a single-layer net is severely limited in the mappings it can learn, a multilayer net (with one or more hidden layers) can learn any continuous mapping to an arbitrary accuracy. More than one hidden layer may be beneficial for some applications, but one hidden layer is sufficient.

In Section 6.1, we shall describe standard backpropagation, including a few of the choices that must be made in designing a net with this feature. In the next section, we mention a few of the many variations of backpropagation that have been developed. Finally, the mathematical derivation of the training algorithm and a brief summary of some of the theorems dealing with the ability of multilayer nets to approximate arbitrary (continuous) functions are given.

6.1.1 Architecture

A multilayer neural network with one layer of hidden units (the Z units) is shown in Figure 6.1. The output units (the Y units) and the hidden units also may have biases (as shown). The bias on a typical output unit Y_k is denoted by w_{0k}; the bias on a typical hidden unit Z_j is denoted v_{0j}. These bias terms act like weights on connections from units whose output is always 1. (These units are shown in Figure 6.1 but are usually not displayed explicitly.) Only the direction of information flow for the feedforward phase of operation is shown. During the backpropagation phase of learning, signals are sent in the reverse direction.

The algorithm in Section 6.1.2 is presented for one hidden layer, which is adequate for a large number of applications. The architecture and algorithm for two hidden layers are given in Section 6.2.4.

6.1.2 Algorithm

As mentioned earlier, training a network by backpropagation involves three stages: the feedforward of the input training pattern, the backpropagation of the associated error, and the adjustment of the weights.

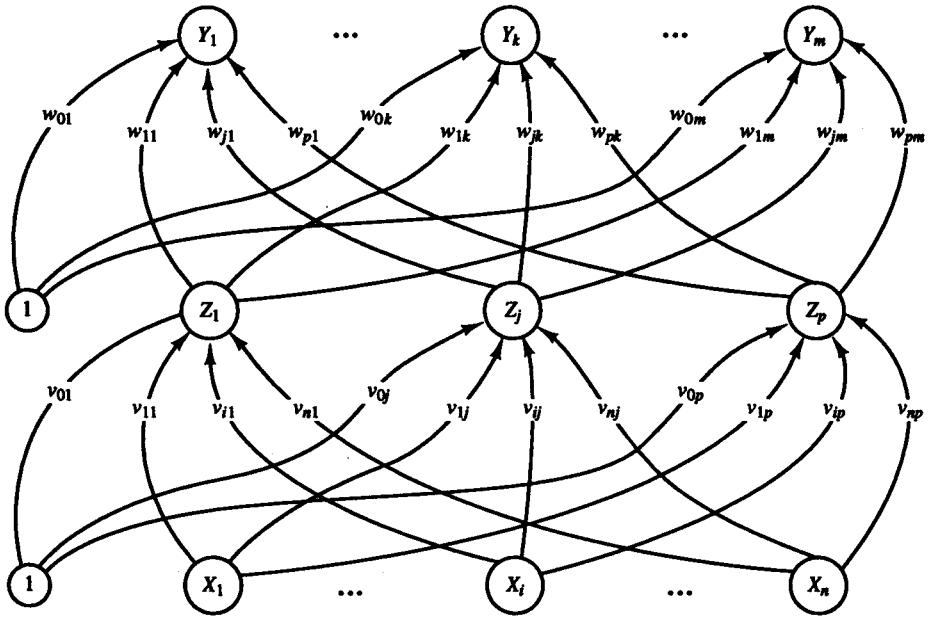

Figure 6.1 Backpropagation neural network with one hidden layer.

During feedforward, each input unit (X_i) receives an input signal and broadcasts this signal to the each of the hidden units $Z_1, \ldots, Z_p$. Each hidden unit then computes its activation and sends its signal (z_j) to each output unit. Each output unit (Y_k) computes its activation (y_k) to form the response of the net for the given input pattern.

During training, each output unit compares its computed activation y_k with its target value t_k to determine the associated error for that pattern with that unit. Based on this error, the factor δ_k ($k = 1, \ldots, m$) is computed. δ_k is used to distribute the error at output unit Y_k back to all units in the previous layer (the hidden units that are connected to Y_k). It is also used (later) to update the weights between the output and the hidden layer. In a similar manner, the factor δ_j ($j = 1, \ldots, p$) is computed for each hidden unit Z_j. It is not necessary to propagate the error back to the input layer, but δ_j is used to update the weights between the hidden layer and the input layer.

After all of the δ factors have been determined, the weights for all layers are adjusted simultaneously. The adjustment to the weight w_{jk} (from hidden unit Z_j to output unit Y_k) is based on the factor δ_k and the activation z_j of the hidden unit Z_j. The adjustment to the weight v_{ij} (from input unit X_i to hidden unit Z_j) is based on the factor δ_j and the activation x_i of the input unit.

Nomenclature

The nomenclature we use in the training algorithm for the backpropagation net is as follows:

x Input training vector:

$$\mathbf{x} = (x_1, \ldots, x_i, \ldots, x_n).$$

t Output target vector:

$$\mathbf{t} = (t_1, \ldots, t_k, \ldots, t_m).$$

δ_k Portion of error correction weight adjustment for w_{jk} that is due to an error at output unit Y_k; also, the information about the error at unit Y_k that is propagated back to the hidden units that feed into unit Y_k.

δ_j Portion of error correction weight adjustment for v_{ij} that is due to the backpropagation of error information from the output layer to the hidden unit Z_j.

α Learning rate.

X_i Input unit i:
For an input unit, the input signal and output signal are the same, namely, x_i.

v_{0j} Bias on hidden unit j.

Z_j Hidden unit j:
The net input to Z_j is denoted z_in_j:

$$z\_in_j = v_{0j} + \sum_i x_i v_{ij}.$$

The output signal (activation) of Z_j is denoted z_j:

$$z_j = f(z\_in_j).$$

w_{0k} Bias on output unit k.

Y_k Output unit k:
The net input to Y_k is denoted y_in_k:

$$y\_in_k = w_{0k} + \sum_j z_j w_{jk}.$$

The output signal (activation) of Y_k is denoted y_k:

$$y_k = f(y\_in_k).$$

Activation function

An activation function for a backpropagation net should have several important characteristics: It should be continuous, differentiable, and monotonically nondecreasing. Furthermore, for computational efficiency, it is desirable that its de-

rivative be easy to compute. For the most commonly used activation functions, the value of the derivative (at a particular value of the independent variable) can be expressed in terms of the value of the function (at that value of the independent variable). Usually, the function is expected to *saturate*, i.e., approach finite maximum and minimum values asymptotically.

One of the most typical activation functions is the binary sigmoid function, which has range of (0, 1) and is defined as

$$f_1(x) = \frac{1}{1 + \exp(-x)},$$

with

$$f_1'(x) = f_1(x)[1 - f_1(x)].$$

This function is illustrated in Figure 6.2.

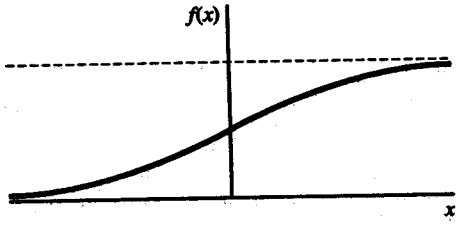

Figure 6.2 Binary sigmoid, range (0, 1).

Another common activation function is bipolar sigmoid, which has range of $(-1, 1)$ and is defined as

$$f_2(x) = \frac{2}{1 + \exp(-x)} - 1,$$

with

$$f_2'(x) = \frac{1}{2}[1 + f_2(x)][1 - f_2(x)].$$

This function is illustrated in Figure 6.3. Note that the bipolar sigmoid function is closely related to the function

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

(See Section 1.4.3.)

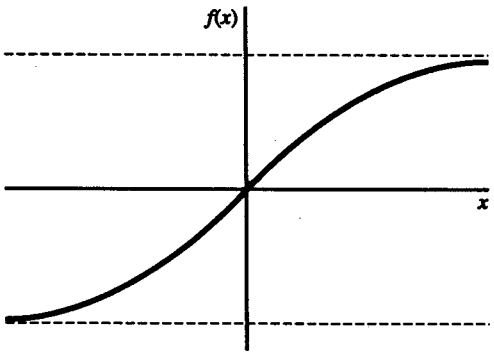

Figure 6.3 Bipolar sigmoid, range $(-1, 1)$.

Training algorithm

Either of the activation functions defined in the previous section can be used in the standard backpropagation algorithm given here. The form of the data (especially the target values) is an important factor in choosing the appropriate function. The relevant considerations are discussed further in the next section. Other suitable activation functions are considered in Section 6.2.2. Note that because of the simple relationship between the value of the function and its derivative, no additional evaluations of the exponential are required to compute the derivatives needed during the backpropagation phase of the algorithm.

The algorithm is as follows:

Step 0. Initialize weights.
(Set to small random values).

Step 1. While stopping condition is false, do Steps 2–9.

Step 2. For each training pair, do Steps 3–8.
Feedforward:

Step 3. Each input unit (X_i, $i = 1, \ldots, n$) receives input signal x_i and broadcasts this signal to all units in the layer above (the hidden units).

Step 4. Each hidden unit ($Z_j, j = 1, \ldots, p$) sums its weighted input signals,

$$z\_in_j = v_{0j} + \sum_{i=1}^{n} x_i v_{ij},$$

applies its activation function to compute its output signal,

$$z_j = f(z\_in_j),$$

and sends this signal to all units in the layer above (output units).

Step 5. Each output unit (Y_k, $k = 1, \ldots, m$) sums its weighted input signals,

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

and applies its activation function to compute its output signal,

$$y_k = f(y\_in_k).$$

Backpropagation of error:

Step 6. Each output unit ($Y_k, k = 1, \ldots, m$) receives a target pattern corresponding to the input training pattern, computes its error information term,

$$\delta_k = (t_k - y_k) f'(y\_in_k),$$

calculates its weight correction term (used to update w_{jk} later),

$$\Delta w_{jk} = \alpha \delta_k z_j,$$

calculates its bias correction term (used to update w_{0k} later),

$$\Delta w_{0k} = \alpha \delta_k,$$

and sends δ_k to units in the layer below.

Step 7. Each hidden unit ($Z_j, j = 1, \ldots, p$) sums its delta inputs (from units in the layer above),

$$\delta\_in_j = \sum_{k=1}^{m} \delta_k w_{jk},$$

multiplies by the derivative of its activation function to calculate its error information term,

$$\delta_j = \delta\_in_j f'(z\_in_j),$$

calculates its weight correction term (used to update v_{ij} later),

$$\Delta v_{ij} = \alpha \delta_j x_i,$$

and calculates its bias correction term (used to update v_{0j} later),

$$\Delta v_{0j} = \alpha \delta_j.$$

Update weights and biases:

Step 8. Each output unit ($Y_k, k = 1, \ldots, m$) updates its bias and weights ($j = 0, \ldots, p$):

$$w_{jk}(\text{new}) = w_{jk}(\text{old}) + \Delta w_{jk}.$$

Each hidden unit ($Z_j, j = 1, \ldots, p$) updates its bias and weights ($i = 0, \ldots, n$):

$$v_{ij}(\text{new}) = v_{ij}(\text{old}) + \Delta v_{ij}.$$

Step 9. Test stopping condition.

Note that in implementing this algorithm, separate arrays should be used for the deltas for the output units (Step 6, δ_k) and the deltas for the hidden units (Step 7, δ_j).

An epoch is one cycle through the entire set of training vectors. Typically, many epochs are required for training a backpropagation neural net. The foregoing algorithm updates the weights after each training pattern is presented. A common variation is batch updating, in which weight updates are accumulated over an entire epoch (or some other number of presentations of patterns) before being applied.

Note that $f'(y\_in_k)$ and $f'(z\_in_j)$ can be expressed in terms of y_k and z_j, respectively, using the appropriate formulas on page 293 (depending on the choice of activation function).

The mathematical basis for the backpropagation algorithm is the optimization technique known as *gradient descent*. The gradient of a function (in this case, the function is the error and the variables are the weights of the net) gives the direction in which the function increases more rapidly; the negative of the gradient gives the direction in which the function decreases most rapidly. A derivation of the weight update rules is given in Section 6.3.1. The derivation clarifies the reason why the weight updates should be done after all of the δ_k and δ_j expressions have been calculated, rather than during backpropagation.

Choices

Choice of initial weights and biases.

Random Initialization. The choice of initial weights will influence whether the net reaches a global (or only a local) minimum of the error and, if so, how quickly it converges. The update of the weight between two units depends on both the derivative of the upper unit's activation function and the activation of the lower unit. For this reason, it is important to avoid choices of initial weights that would make it likely that either activations or derivatives of activations are zero. The values for the initial weights must not be too large, or the initial input signals to each hidden or output unit will be likely to fall in the region where the