4.3 TRAINING WITH BACK PROPAGATION

To begin our discussion of training the network we must first recognize the need for a measure of how close the network has come to an established desired value. This measure is the network error. Since we are dealing with supervised training, the desired value is known to us for the given training set. We will see later that the proper selection of a training set will be a crucial factor in any successful network application. The training set must be of an appropriate size and it must be reasonably well representative of the problem space. For now we assume that such a training set exists so we may interrogate how to use it to train a network.

Therefore we begin by defining an error measure. Typically for the back propagation [Rumelhart et al., 1986] training algorithm, an error measure known as the mean square error is used. This is in fact not a requirement. Any continuously differentiable error function can be used, but the choice of another error function does add additional complexity and should be approached with a certain amount of caution. Remember also that whatever function is chosen for the error function must provide a meaningful measure of the "distance" between desired and actual outputs of the network. The mean square error is defined as follows:

\[
E_p = \frac{1}{2} \sum_{j=1}^{n} (t_{pj} - o_{pj})^2
\]  

(5)

where \(E_p\) is the error for the \(p^{th}\) presentation vector; \(t_{pj}\) is the desired value for the \(j^{th}\) output neuron (i.e., the training set value); and \(o_{pj}\) is the actual output of the \(j^{th}\) output neuron.

Each term in the sum is the error contribution of a single output neuron. Notice that by taking the square of the absolute error, the difference between desired and actual, we cause outputs that are distant from the desired value to contribute most strongly to the total error. Increasing the exponent, if we chose to do that, would augment this effect.

Back propagation is one of the simpler members of a family of training algorithms collectively termed gradient descent. The idea is to minimize the network total error by adjusting the weights. Gradient descent, sometimes known as the method of steepest descent, provides a means of doing this. Each weight may be thought of as a dimension in an \(N\)-dimensional error space. In error space the weights act as independent variables and the shape of the corresponding error surface is determined by the error function in combination with the training set.
The negative gradient of the error function with respect to the weights then points in the direction which will most quickly reduce the error function. If we move along this vector in weight space, we will ultimately reach a minimum at which the gradient becomes zero. Unfortunately this may be a local minimum, but we will have more to say about that a bit later. Figure 4.8 illustrates the operation of the gradient in the context of a two-dimensional cross section of the error space.

**FIGURE 4.8** Surface gradient diagram.

We can express the above observations mathematically as:

$$\Delta_p W_{ji} \propto -\frac{\partial E_p}{\partial w_{ji}}$$

(6)

The term, $\Delta_p W_{ji}$, designates the change in the weight connecting a **source** neuron, $i$, in layer L-1 and a **destination** neuron, $j$, in layer L. This change in the weight results in a step in the weight space (shown in Figure 4.8) toward lower error.
Feed-Forward Networks with Supervised Learning

The objective is to determine how we must adjust each weight to achieve convergence for the network. Equation 6 states that the change in each weight \( w_{ji} \) will be along the negative gradient leading to a steepest descent along the local error surface.

The task now is to convert equation 6 into a difference equation suitable for use in a computer implementation. To accomplish this we must evaluate the partial derivative, \( \frac{\partial E_p}{\partial w_{ji}} \). We begin by applying the chain rule:

\[
\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial \text{net}_{pi}} \frac{\partial \text{net}_{pi}}{\partial w_{ji}}
\]

(7)

However, we know that \( \text{net}_{pi} \) is given by:

\[
\text{net}_{pi} = \sum_j w_{ji} O_{pi}
\]

(8)

where the sum in equation 8 is taken over the output, \( O_{pi} \), of all neurons in the L-1 layer. We may therefore evaluate \( \frac{\partial \text{net}_{pi}}{\partial w_{ji}} \), the second term in equation 7, as follows:

\[
\frac{\partial \text{net}_{pi}}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \sum_j w_{ji} O_{pi}
\]

(9)

By expanding equation 9 we obtain:

\[
\frac{\partial \text{net}_{pi}}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \left( \sum_j w_{ji} O_{pi} + w_{ji} O_{pi} \right) = O_{pi}
\]

(10)

Substituting equation 10 into equation 7 we obtain:

\[
\frac{\partial E_p}{\partial w_{ji}} = O_{pi} \frac{\partial E_p}{\partial \text{net}_{pi}}
\]

(11)

Now we define the error signal \( \delta_{pi} \) as:

\[
\delta_{pi} = -\frac{\partial E_p}{\partial \text{net}_{pi}}
\]

(12)

By combining equations 11 and 12 we have:

\[
\frac{\partial E_p}{\partial w_{ji}} = \delta_{pi} O_{pi}
\]

(13)

We may rewrite equation 6 by substituting equation 13 and supplying a constant of proportionality, \( \eta \).
\[ \Delta_{\rho} w_{ij} = \eta \delta_{ij} O_{ij} \]  \tag{14}

The constant \( \eta \) is known as the learning rate. As its name implies, it governs the distance traveled in the direction of the negative gradient when a step in weight space is taken.

In order to achieve a usable difference equation, the task of evaluation \( \delta_{ij} \) still remains. Once again we must apply the chain rule:

\[ \delta_{ij} = -\frac{\partial E_p}{\partial \text{net}_{ij}} = -\frac{\partial E_p}{\partial O_{ij}} \frac{\partial O_{ij}}{\partial \text{net}_{ij}} \]  \tag{15}

Now recall that the output \( O_{ij} \) is directly a function of \( \text{net}_{ij} \) as follows:

\[ O_{ij} = f(\text{net}_{ij}) \]  \tag{16}

\[ \frac{\partial O_{ij}}{\partial \text{net}_{ij}} = f'(\text{net}_{ij}) \]  \tag{17}

where \( f() \) is the squashing function.

To evaluate \( \partial E_p / \partial O_{ij} \) (the first term of equation 15), we must consider two cases individually:

1. The destination neuron \( j \) is an output neuron.
2. The destination neuron \( j \) is a hidden layer neuron.

For a destination neuron \( j \) in the output layer we have direct access to the error \( E_p \) as a function of \( O_{ij} \). Therefore we write:

\[ \frac{\partial E_p}{\partial O_{ij}} = \frac{\partial}{\partial O_{ij}} \left( \frac{1}{2} \sum_j (t_{ij} - O_{ij})^2 \right) = -(t_{ij} - O_{ij}) \]  \tag{18}

Notice that with equation 18 we have specialized the algorithm to the specific error function. An alternate choice of error function will lead to a different difference equation. Substituting equations 17 and 18 into equation 15 we may now write \( \delta_{ij} \) (for destination neurons in the output layer) as:

\[ \delta_{ij} = (t_{ij} - O_{ij}) f'(\text{net}_{ij}) \]  \tag{19}

For destination neurons that reside in hidden layers we cannot differentiate the error function directly. Therefore we must once again apply the chain rule to obtain:
In equation 20 the sum \( k \) is over all neurons in the \( L + 1 \) layer. Recalling the definition of \( \text{net}_{pk} \), we may evaluate the second factor in equation 20 as follows:

\[
\frac{\partial \text{net}_{pk}}{\partial O_{pj}} = \frac{\partial}{\partial O_{pj}} \left( \sum_i w_{ki} O_{pi} \right) \\
= \frac{\partial}{\partial O_{pj}} \left( \sum_{i \neq j} w_{ki} O_{pi} + w_{kj} O_{pj} \right) \\
= w_{kj}
\]

Substituting equation 21 back into equation 20 yields:

\[
\frac{\partial E_p}{\partial O_{pj}} = \sum_k \frac{\partial E_p}{\partial \text{net}_{pk}} w_{kj}
\]

Now we have it by definition that:

\[
\delta_{pk} = p \frac{\partial E_p}{\partial \text{net}_{pk}}
\]

Substituting equation 23 into equation 22 yields:

\[
\frac{\partial E_p}{\partial O_{pj}} = \sum_k \delta_{pk} w_{kj}
\]

Finally combining equation 15, 17, and 24 we can represent the error signal \( \delta_{pj} \) for hidden layers as:

\[
\delta_{pj} = f'(\text{net}_{pj}) \sum_k \delta_{pk} w_{kj}
\]

To summarize the results so far, equation 14 provides the difference equation in terms of \( \delta_{pj} \). This is valid for both hidden and output layer weights. Equations 19 and 25 specify \( \delta_{pj} \) for the output layer and hidden layer weights, respectively. Equation 18 particularized our solution to the mean square error. Therefore to use an alternative error function equation 18 would require modification. To obtain a
difference equation suitable for use on a digital computer it now only remains to evaluate \( f(\text{net}_p) \). To do this we must again particularize our solution by choosing a specific squashing function \( f'(\text{net}_p) \). We now proceed using the sigmoid function as follows:

\[
O_p = f(\text{net}_p) = \frac{1}{1 + e^{-\text{net}_p}}
\]

(26)

From equations 17 and 26 we may write \( f'(\text{net}_p) \) as:

\[
f'(\text{net}_p) = \frac{\partial}{\partial \text{net}_p} \left( \frac{1}{1 + e^{-\text{net}_p}} \right)
\]

(27)

Evaluating the derivative in equation 27 leads to:

\[
f'(\text{net}_p) = \left( \frac{-1}{(1 + e^{-\text{net}_p})^2} \right) \frac{\partial}{\partial \text{net}_p} \left( 1 + e^{-\text{net}_p} \right)
\]

(28)

We continue the evaluation of \( f'(\text{net}_p) \) as follows:

\[
f'(\text{net}_p) = \left( \frac{-1}{(1 + e^{-\text{net}_p})^2} \right) e^{-\text{net}_p} \frac{\partial}{\partial \text{net}_p} (-\text{net}_p + \theta)
\]

(29)

\[
= \left( \frac{1}{1 + e^{-\text{net}_p}} \right) \left( \frac{1}{1 + e^{-\text{net}_p}} \right) \frac{\partial}{\partial \text{net}_p} (-\text{net}_p + \theta)
\]

(30)

\[
= \left( \frac{1}{1 + e^{-\text{net}_p}} \right) \left( \frac{1}{1 + e^{-\text{net}_p}} \right) \left( \frac{1 + e^{-\text{net}_p}}{1 + e^{-\text{net}_p}} \right)
\]

(31)

\[
= \left( \frac{1}{1 + e^{-\text{net}_p}} \right) \left( 1 - \frac{1}{1 + e^{-\text{net}_p}} \right)
\]

(32)

We may now express \( f'(\text{net}_p) \) in terms of \( O_p \) by substituting equation 26 into equation 32. We then obtain:

\[
f'(\text{net}_p) = O_p \left( 1 - O_p \right)
\]

(33)

Taken together equations 14, 19, 25, and 33 provide all that is necessary to write the difference equation needed to implement training by back propagation on a
digital computer where the error function is the mean square error and the squashing function is the sigmoid. As we have proceeded through this derivation we have taken pains to show the points at which modifications would be required for alternative error or activation functions.

To summarize, the difference equation required for back-propagation training is

$$\Delta w_{ji} = \eta \delta_{ji} O_{ji}$$  \hspace{1cm} (34)

where $\eta$ refers to the learning rate; $\delta_{ji}$ refers to the error signal at neuron $j$ in layer $L$; and $O_{ji}$ refers to the output of neuron $i$ in layer $L-1$.

With the error signal, $\delta_{ji}$, given by:

$$\delta_{ji} = (t_{ji} - O_{ji})O_{ji}(1 - O_{ji})$$ for output neurons \hspace{1cm} (35)

$$\delta_{ji} = O_{ji}(1 - O_{ji}) \sum \delta_{kj} w_{kj}$$ for hidden neurons \hspace{1cm} (36)

where $O_{ji}$ refers to layer $L$; $O_{ji}$ refers to layer $L - 1$; and $\delta_{jk}$ refers to layer $L + 1$.

True gradient descent would proceed in infinitesimal steps along the direction established by the gradient. Since this is obviously impractical for our purposes, the learning rate, $\eta$, is defined (see equation 34). It can be seen that equation 34 results in a finite step size in the direction of the gradient. Here $\eta$ is a constant which acts like a gain to determine the step size. The idea is to choose $\eta$ large enough to cause the network to converge quickly without introducing overshoot and therefore oscillations. Later we will look at the conjugate gradient technique which in effect uses the rate at which the gradient is changing to establish a step size. Understanding the effect of the learning rate can help to choose its value judiciously. Even so, a certain amount of experimental tuning is generally required to optimize this parameter.

For clarity, the application of equations 34, 35, and 36 is shown in Figure 4.9. In particular, this figure is useful to clarify which layers are involved when calculating the various components of the difference equation. The top half of the figure delineates the training of the output layers. The bottom half depicts training in hidden layers. A cautionary reminder is that the above difference equation is valid only for mean square error with a sigmoidal activation function (given in equation 2). To use alternate error or activation functions with back propagation the difference equation must be modified as shown in the derivation.

In practice a momentum term is frequently added to equation 34 as an aid to more rapid convergence in certain problem domains. The momentum takes into account the effect of past weight changes. The momentum constant, $\alpha$, determines the emphasis to place on this term. Momentum has the effect of smoothing the error-surface in weight space by filtering out high-frequency variations. The weights are adjusted in the presence of momentum by:

$$\Delta w_{ji}(n+1) = \eta \delta_{ji} O_{ji} + \alpha \Delta w_{ji}(n)$$ \hspace{1cm} (37)
The momentum term is but the first of several departures from what might be described as pure gradient descent that is intended to augment the algorithm with respect to its ability to converge more rapidly. At this point we remark that there is much in back propagation and its variations that is largely empirical.

The overall process of back-propagation learning including both the forward and backward pass is presented in Figure 4.10 above. To apply the back-propagation algorithm the network weights must first be initialized to small random values. It is important to make the initial weights "small". Choosing initial weights too large will make the network untrainable for reasons we will discuss later. After initialization, training set vectors are then applied to the network. Running the network forward will yield a set of actual values. Back propagation can then be utilized to establish a new set of weights. The total error should decrease over the course of many such iterations. If it does not, an adjustment to the training parameters, $\alpha$ and $\eta$, may be required. (Contradictory data, a training vector duplicated with an oppositely
Feed-Forward Networks with Supervised Learning

**FIGURE 4.10** Back-propagation flow chart.

1. **START**
2. Initialize Small Random Weights
3. Get Next Training Vector
4. Forward Pass (then compare to desired output layer values)
5. Backward Pass (update weights using Delta Propagation algorithm)
6. End of Epoch?
   - Yes: Error Within Tolerance?
     - Yes: Declare Success (save Trained Weight)
     - No: No Exceeded Maximum Iteration?
       - Yes: Declare failure (network did not converge within the iteration limit)
       - No: END