Chapter 3

Backpropagation Networks

In Chapter 2, the mathematical details of a neuron at the single cell level and as a network were described. Although single neurons can perform certain simple pattern detection functions, the power of neural computation comes from the neurons connected in a network structure. Larger networks generally offer greater computational capabilities but the converse is not true. Arranging neurons in layers or stages is supposed to mimic the layered structure of certain portions of the brain.

For many years, there was no theoretically sound algorithm for training multilayer artificial neural networks. Since single layer networks proved severely limited in what they could represent (in what they could learn), the entire field went into virtual eclipse. The resurgence of interest in artificial neural network began after the invention of backpropagation algorithm.

Backpropagation is a systematic method of training multilayer artificial neural networks. It is built on high mathematical foundation and has very good application potential. Even though it has its own limitations, it is applied to a wide range of practical problems and has successfully demonstrated its power.

Rumelhart, Hinton and Wilham (1986) presented a clear and concise description of the backpropagation algorithm. Parker (1982) has also shown to have anticipated Rumelhart's work. It was also shown elsewhere that Werbos (1974) had described the method still earlier.

In this chapter, we describe the most popular Artificial Neural Network (ANN) architecture, the Multilayer Feedforward (MLFF) network with backpropagation (BP) learning. This type of network is sometimes called multilayer perceptron because of its similarity to perceptron networks with more than one layer. First, we briefly review the perceptron model to show how this is altered to form (MLFF) networks. We derive the generalized delta (backpropagation) learning rule and see how it is implemented in practice. We will also examine variations in the learning process to improve the efficiency, and ways to avoid some potential problems that can arise during training. We will also discuss optimal parameters' settings and discuss various other training methods. We will also look at the capabilities and limitations and discuss a number of applications in various engineering fields.

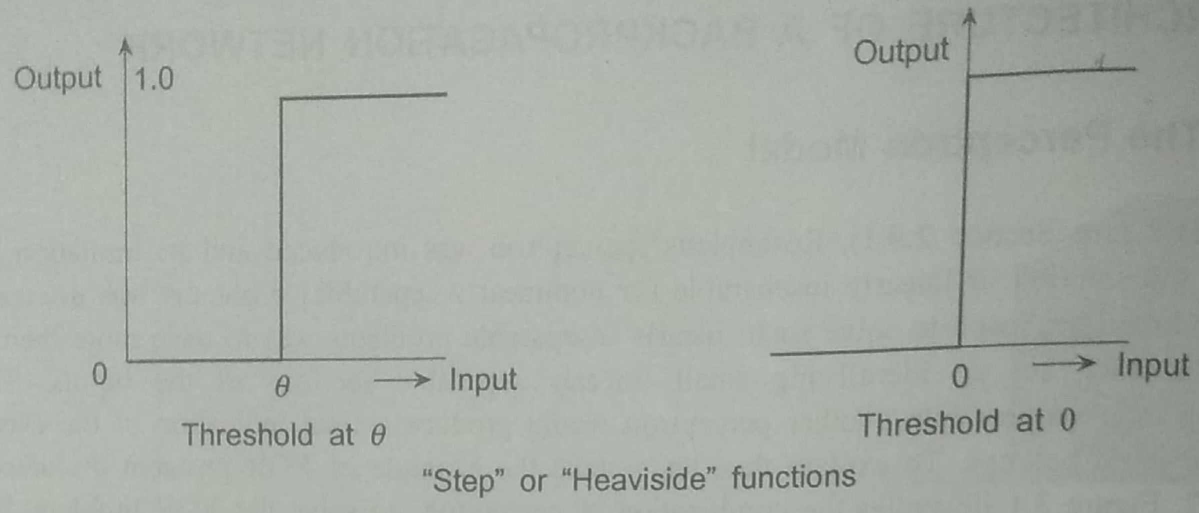


Fig. 3.2 Hard-hitting threshold function.

3.1.2 The Solution

If we smoothen the threshold function so that it more or less turns on or off as before but has a sloping region in the middle that will give us some information on the inputs, we will be able to determine when we need to strengthen or weaken the relevant weights. Now, the network will be able to learn as required. A couple of possibilities for the new thresholding function are given in Table 3.1. Some of the functions have already been introduced in Chapter 2. Even now, the value of the outputs will practically be one if the input exceeds the value of the threshold a lot and will be practically zero if the input is far less than the threshold. However, in cases when input and threshold are almost same, the output of the neuron will have a value between zero and one, meaning that the output to the neurons can be related to input in a more informative way.

As seen in Chapter 2, an artificial neuron is developed to mimic the characteristics and functions of a biological neuron. Analogous to a biological neuron, an artificial neuron receives much input representing the output of the other neurons. Each input is multiplied by the corresponding weights analogous to synaptic strengths. All of these weighed inputs are then summed up and passed though an activation function to determine the neuron output. The artificial neural model (with new notations to suit the discussion in this chapter) is shown in Fig. 3.3.

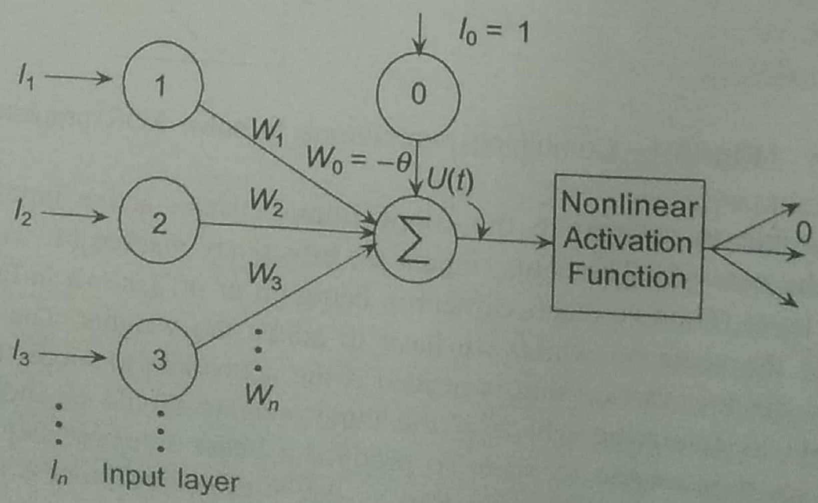


Fig. 3.3 An artificial neuron.

$$u(t) = W_1I_1 + W_2I_2 + \cdots + W_nI_n \tag{3.1a}$$

or

$$u = \langle W \rangle \{I\} \tag{3.1b}$$

Table 3.1 Typical nonlinear activation operators

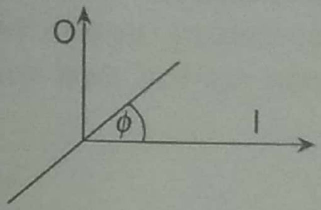
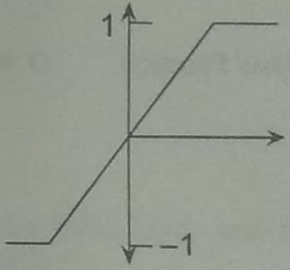
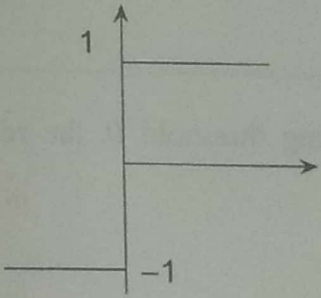
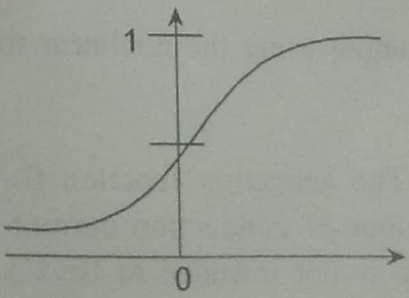
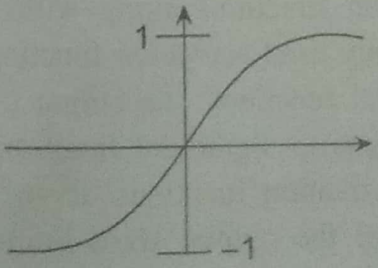
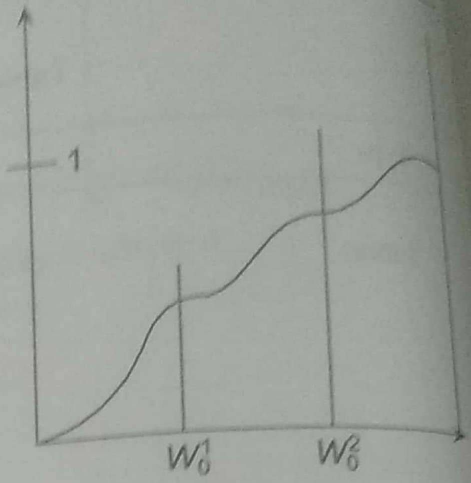
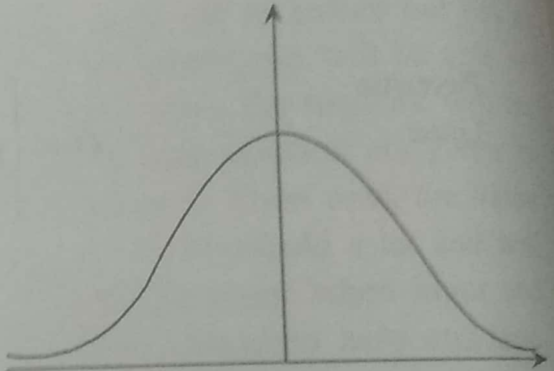
Type	Equation	Functional form
Linear	$O = gI$ $g = \tan\phi$	
Piecewise Linear	$O = \begin{cases} 1 & if \quad mI > 1 \\ gI & if \quad \lvert mI < 1 \rvert \\ -1 & if \quad mI > -1 \end{cases}$	
Hard Limiter	$O = \text{sgn}\ [I]$	
Unipolar Sigmoidal	$O = \dfrac{1}{(1 + \exp(-\lambda I)}$	
Bipolar Sigmoidal	$O = \tanh\ [\lambda I]$	

Table 3.1 Typical nonlinear activation operators (cont.)

Type	Equation	Functional form
Unipolar Multimodal	$O = \frac{1}{2}\left[1 + \frac{1}{M}\sum_{m=1}^{M} \tanh\left(g^{m}(I - W_0^{m})\right)\right]$	
Radial Basis Function (RBF)	$O = \exp(I)$ $I = \left[\frac{-\sum_{i=1}^{N}(W_i(t) - X_i(t))^2}{2\sigma^2}\right]$	

Considering threshold θ, the relative input to the neuron is given by

$$u(t) = W_1 I_1 + W_2 I_2 + \cdots + W_n I_n - \theta \tag{3.2a}$$

$$= \sum_{i=0}^{n} W_i I_i \quad \text{where} \quad W_0 = -\theta;\ I_0 = 1 \tag{3.2b}$$

The output using the nonlinear transfer function 'f' is given by

$$O = f(u) \tag{3.3}$$

The activation function $f(u)$ is chosen as a nonlinear function to emulate the nonlinear behaviour of conduction current mechanism in a biological neuron. However, as the artificial neuron is not intended to be a xerox copy of the biological neuron, many forms of nonlinear functions have been suggested and used in various engineering applications. Some of the activation functions along with their mathematical descriptions given in Table 3.1 are most commonly used activation functions in multilayered static neural networks. It is also seen that for sigmoidal functions, the output of a neuron varies continuously but not linearly with the input. Neurons with sigmoidal functions bear a greater resemblance to biological neurons than with other activation functions. Even if the sigmoidal function is differentiated, it gives continuou values of the output. Hard limiter (see Table 3.1) and radial basis functions are also equall popular.

$$f(I) = \frac{1}{(1 + e^{-\lambda I})} \tag{3.9a}$$

and
$$f'(I) = \lambda f(I)\,(1 - f(I)) \tag{3.9b}$$

3.1.4 Model for Multilayer Perceptron

The adapted perceptrons are arranged in layers and so the model is termed as multilayer perceptron. This model has three layers; an input layer, and output layer, and a layer in between not connected directly to the input or the output and hence, called the hidden layer. For the perceptrons in the input layer, we use linear transfer function, and for the perceptrons in the hidden layer and the output layer, we use sigmoidal or squashed-S functions. The input layer serves to distribute the values they receive to the next layer and so, does not perform a weighted sum or threshold. Because we have modified the single layer perceptron by changing the nonlinearity from a step function to a sigmoidal function and added a hidden layer, we are forced to alter the learning rules as well. So now, we have a network that should be able to learn to recognize more complex things. The input–output mapping of multilayer perceptron is shown in Fig. 3.6(a) and is represented by

$$O = N_3\,[N_2\,[N_1[I]]] \tag{3.10}$$

In Eq. (3.10), N_1, N_2, and N_3 (see Fig. 3.6(b)) represent nonlinear mapping provided by input, hidden and output layers respectively. Multilayer perceptron provides no increase in computational power over a single layer neural network unless there is a nonlinear activation function between layers. Many capabilities of neural networks, such as nonlinear functional approximation, learning, generalization etc. are in fact due to nonlinear activation function of each neuron.

The three-layer network shown in Fig. 3.6(a) and the block diagram shown in Fig. 3.6(b) show that the activity of neurons in the input layers represent the raw information that is fed into the network. The activity of neurons in the hidden layer is determined by the activities of the neurons in the input layer and the connecting weights between input and hidden units. Similarly, the activity of the output units depends on the activity of neurons in the hidden layer and the weight between the hidden and output layers. This structure is interesting because neurons in the hidden layers are free to construct their own representations of the input.

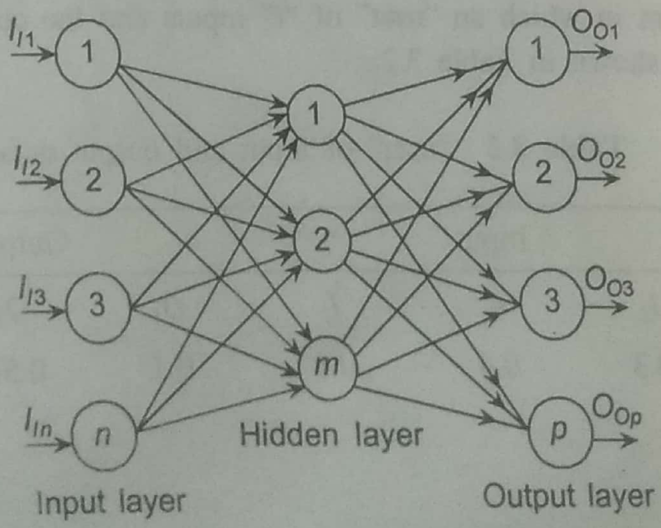


Fig. 3.6(a) Multilayer perceptron.

Scanned by CamScanner

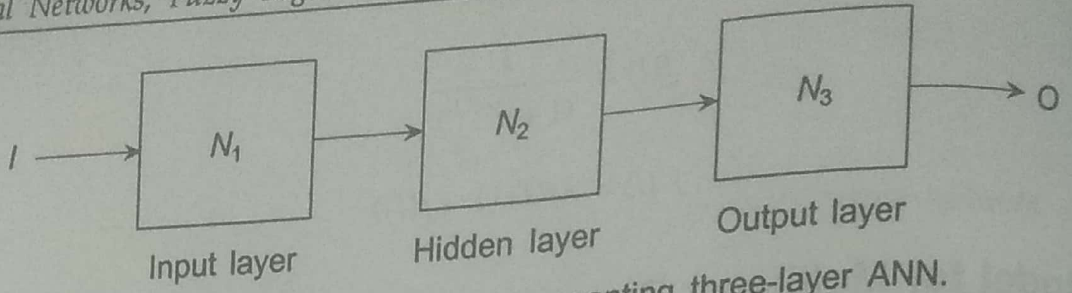


Fig. 3.6(b) Block diagram representing three-layer ANN.

3.2 BACKPROPAGATION LEARNING

Consider the network as shown in Fig. 3.7 where the subscripts *I*, *H*, *O* denote input, hidden an output neurons.

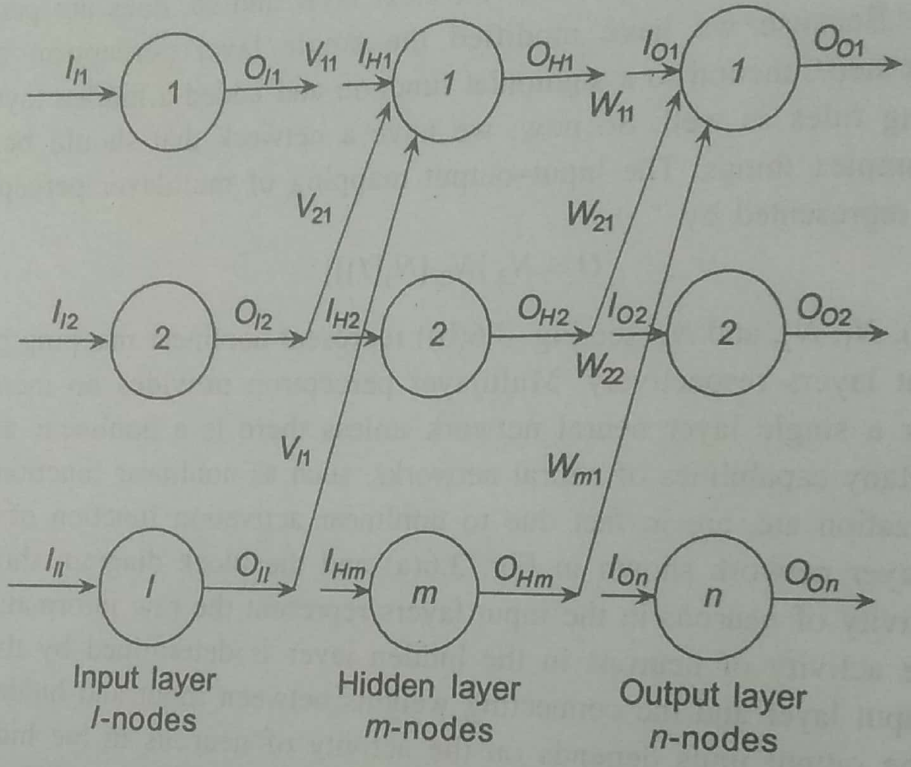


Fig. 3.7 Multilayer feedforward backpropagation network.

Consider a problem in which an "nset" of "*l*" inputs and the corresponding "nset" of "*n*" output data is given as shown in Table 3.2.

Table 3.2 "nset" of input and output data

No.	Input				Output			
	I_1	I_2	$\cdots$	I_l	O_1	O_2	$\cdots$	O_n
1	0.3	0.4	$\cdots$	0.8	0.1	0.56	$\cdots$	0.82
2								
⋮								
nset								

3.2.1 Input Layer Computation

Consider linear activation function the output of the input layer is input of input layer (considering $g = \tan\phi = 1$). Taking one set of data

$$\underset{l\times 1}{\{O\}_I} = \underset{l\times 1}{\{I\}_I} \tag{3.11}$$

The hidden neurons are connected by synapses to input neurons and (let us denote) V_{ij} is the weight of the arc between ith Input neuron to jth hidden neuron. As shown in Eq. (3.1b), the input to the hidden neuron is the weighted sum of the outputs of the input neurons to get I_{Hp} (i.e. Input to the pth hidden neuron) as

$$I_{Hp} = V_{1p}O_{I1} + V_{2p}O_{I2} + \ldots + V_{lp}O_{Il} \tag{3.12}$$

$$(p = 1, 2, 3, \ldots, m)$$

Denoting weight matrix or connectivity matrix between input neurons and hidden neurons as $\underset{l\times m}{[V]}$, we can get an input to the hidden neuron as

$$\underset{m\times 1}{\{I\}_H} = \underset{m\times l}{[V]^T}\ \underset{l\times 1}{\{O\}_I} \tag{3.13}$$

3.2.2 Hidden Layer Computation

Considering sigmoidal function or squashed-S function, the output of the pth hidden neuron is given by

$$O_{Hp} = \frac{1}{\left(1 + e^{-\lambda(I_{Hp} - \theta_{Hp})}\right)} \tag{3.14}$$

where O_{Hp} is the output of the pth hidden neuron, I_{Hp} is the input of the pth hidden neuron, and θ_{Hp} is the threshold of the pth neuron. A non-zero threshold neuron is computationally equivalent to an input that is always held at –1 and the non-zero threshold becomes the connecting weight values as shown in Fig. 3.8.

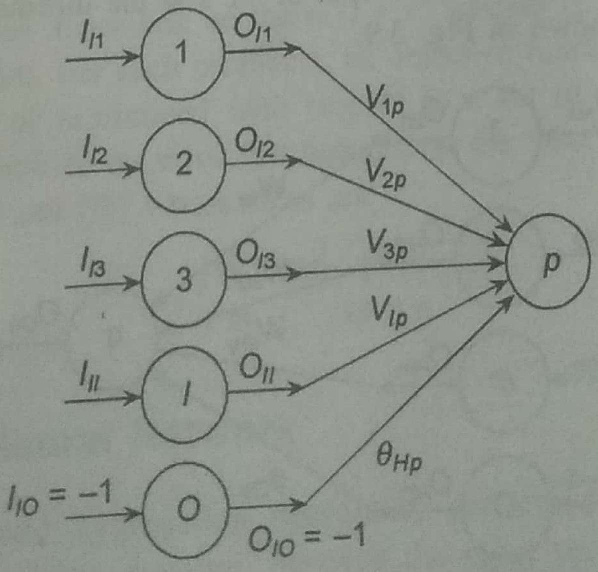


Fig. 3.8 Treating threshold in hidden layer.

But in our derivations we will not treat threshold as shown in Fig. 3.8. Now, output to the hidden neuron is given by

$$\{O\}_H = \begin{Bmatrix} - \\ - \\ \dfrac{1}{(1+e^{-\lambda(I_{Hp}-\theta_{Hp})})} \\ - \\ - \end{Bmatrix} \tag{3.15}$$

Treating each component of the input of the hidden neuron separately, we get the outputs of the hidden neuron as given by Eq. (3.15).

As shown in Fig. 3.1(b) the input to the output neurons is the weighted sum of the outputs of the hidden neurons. To get I_{Oq} (i.e. the input to the qth ouput neuron)

$$I_{Oq} = W_{1q}O_{H1} + W_{2q}O_{H2} + \cdots + W_{mq}O_{Hm} \tag{3.16}$$

$$(q = 1, 2, 3, \ldots, n)$$

Denoting weight matrix or connectivity matrix between hidden neurons and output neurons as $[W]$, we can get input to the output neuron as

$$\underset{n\times 1}{\{I\}_O} = \underset{n\times m}{[W]^T}\ \underset{m\times 1}{\{O\}_H} \tag{3.17}$$

3.2.3 Output Layer Computation

Considering sigmoidal function, the output of the qth output neuron is given by

$$O_{Oq} = \frac{1}{(1+e^{-\lambda(I_{Oq}-\theta_{Oq})})} \tag{3.18}$$

where, O_{Oq} is the output of the qth output neuron, I_{Oq} is the input to the qth output neuron, and θ_{Oq} is the threshold of the qth neuron. This threshold may also be tackled again by considering extra Oth neuron in the hidden layer with output of -1 and the threshold value θ_{Oq} becomes the connecting weight value as shown in Fig. 3.9.

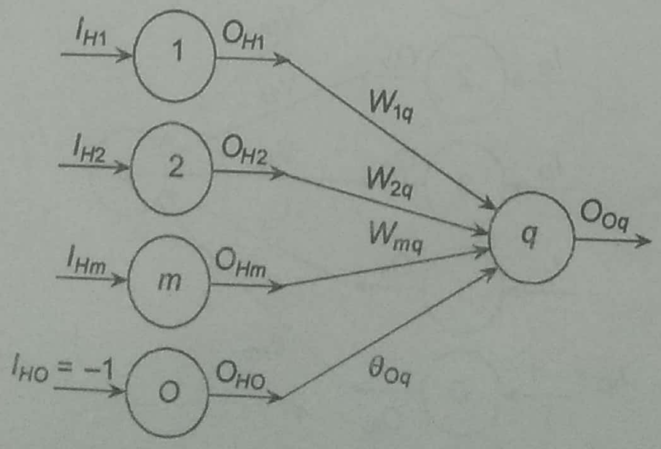


Fig. 3.9 Treating threshold in output layer.

Hence, the outputs of output neurons are given by

$$\{O\}_O = \left\{\begin{array}{c} - \\ - \\ \dfrac{1}{\left(1+e^{-\lambda(I_{O_q}-\theta_{O_q})}\right)} \\ - \\ - \end{array}\right\} \tag{3.19}$$

3.2.4 Calculation of Error

Considering any rth output neuron and for the training example we have calculated the output 'O' for which the target output 'T' is given in Table 3.2.

Hence, the error norm in output for the rth output neuron is given by

$$E_r^1 = \frac{1}{2}e_r^2 = \frac{1}{2}(T-O)^2 \tag{3.20}$$

where E_r^1 = 1/2 second norm of the error in the rth neuron ('e_r') for the given training pattern. The square of the error is considered since irrespective of whether error is positive or negative, we consider only absolute values. The Euclidean norm of error E^1 for the first training pattern is given by

$$E^1 = \frac{1}{2}\sum_{r=1}^{n}(T_{Or}-O_{Or})^2 \tag{3.21}$$

Equation 3.21 gives the error function in one training pattern. If we use the same technique for all the training patterns, we get

$$E(V,W) = \sum_{j=1}^{\text{nset}} E^j(V,W,I) \tag{3.22}$$

where E is the error function depending on the m (1 + n) weights of [W] and [V]. This is a classic type of optimization problem. For such problems, an objective function or cost function is usually defined to be maximized or minimized with respect to a set of parameters. In this case, the network parameters that optimize the error function E over the 'nset' of pattern sets [I^{nset}, t^{nset}] are synaptic weight values [V] and [W] whose sizes are

$$\underset{l\times m}{[V]} \quad \text{and} \quad \underset{m\times n}{[W]} \tag{3.23}$$

3.2.5 Training of Neural Network

The synaptic weighting and aggregation operations performed by the synapses and soma respectively, provide a 'similarity measure' between the input vector I and the synaptic weights [V] and [W] (accumulation knowledge base). When a new input pattern that is significantly