CHAPTER

2 Neural Networks

Artificial neural networks (ANN) have played a major role in the development of intelligent control schemes. These networks are used exclusively for system identification as well as for controller parametrization. In this chapter, multi-layered networks, radial basis function, recurrent networks, and self-organizing map networks will be discussed from a function approximation perspective. Multi-layered networks and radial basis function have feed-forward connection structures. When these networks are represented as controllers or as system models, it becomes relatively easy to perform a closed-loop analysis of a control system in terms of stability and convergence. Thus, extensive use of feed-forward networks in intelligent control system development in the existing literature is not surprising.

2.1 FEED-FORWARD NETWORKS

The simplest form of an artificial neural network is called *a perceptron*. The single layer perceptron model was first developed by Rosenblatt in 1958 [20]. The basic model consists of a single neuron with adjustable synaptic weights and bias. However, in 1969, Marvin Minsky and Seymour Papert wrote a book called 'Perceptrons' [21] in which they proved mathematically that single-layer perceptrons could only classify linearly separable patterns. This book made a turning point in the field of artificial neural networks by showing the limitations of single-layer perceptrons. Though the idea of multi-layer perceptrons was there in the minds of people, but they did not know how to train multi-layer perceptrons. As a result, researchers almost lost their interests in neural networks until about the mid-eighties, when the back-propagation algorithm for training multi-layer perceptrons was discovered. The development of back-propagation algorithm, along with its use in machine learning, was reported by Rumelhart, Hinton, and Williams [22] in 1986. Werbos [23] and Parker [24] have also independently proposed back-propagation algorithm in 1974 and 1985. In 1988, Broomhead and Lowe [25] described the design of a layered feed-forward networks using radial basis functions which came as an alternative to multi-layer perceptrons. Basically a feed-forward network consists of multiple layers of neurons where the neurons in one layer are forward-connected to the neurons of the next layer. Although, there are many variants of a feed-forward networks, multi-layered networks, and radial basis function networks have been used in many applications. These two networks can act as universal approximators, i.e., any non-linear function can be approximated with an arbitrary accuracy using a properly tuned multi-layered network or a radial basis function network.

2.2 MULTI-LAYERED NEURAL NETWORKS

A multi-layered neural network (MNN) as shown in Figure 2.1 usually consists of an input layer of a set of sensory units or source nodes, $L - 1$ hidden layers of neurons, and an output layer of neurons. The input signal propagates through the network on a layer-to-layer basis in the forward direction. Each neuron actuates a response using sigmoidal activation function. As mentioned earlier, MNNs have been successfully applied in solving some difficult problems by training them in a supervised manner with the error back-propagation algorithm. The back-propagation algorithm uses the principle of gradient descent to train the network parameters. The synopsis of the notations used to represent an MLN

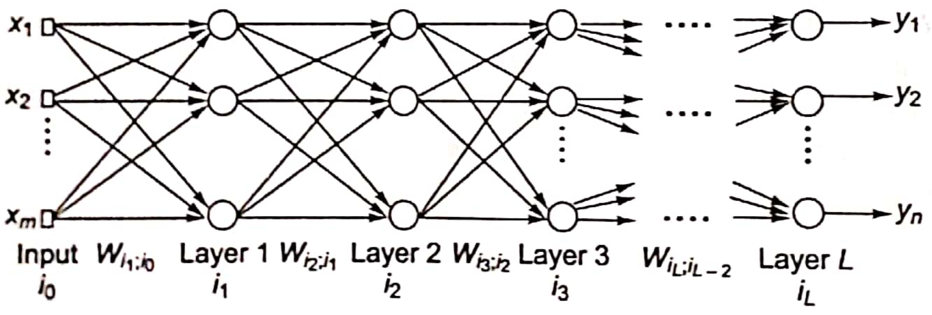

Figure 2.1 An L-layered network

with L layers is as follows:

$\boldsymbol{x}$: $m \times 1$ input vector

$\boldsymbol{y}$: $n \times 1$ output vector

i_k; $k = 1, \ldots, L$: index for representing a neuron in the kth layer

i_0: index for representing a neuron in the input layer

h_{i_k}: Weighted sum of the input stimuli to i_k^{th} neuron in the kth layer

v_{i_k}: response of the i_k^{th} neuron in the kth layer

$W_{i_k i_{k-1}}$: Weight connecting i_k^{th} neuron of the kth layer and i_{k-1}^{th} neuron of $k - 1$th layer

$W_{i_2 i_1}$: Weight connecting the i_2^{th} neuron of the 2nd layer and the i_1^{th} unit of the 1st layer

$W_{i_1 i_0}$: Weight connecting the i_1^{th} unit of the 1st layer and the i_0^{th} unit of the input layer

2.2.1 Principle of Gradient Descent

Suppose that a function $y = f(x)$ has to be approximated by a neural network. If the number of training patterns available is N, then the cost function to be minimized to learn this mapping is defined as follows:

$$E = \sum_{p=1}^{p=N} E^p \tag{2.1}$$

where $E^p = \frac{1}{2}(y_p^d - y_p)^2$. y_p and y_p^d denote the pth actual and desired pattern. Since y_p is a function of the network weight vector w, E can also be expressed as a function of w. Consider a typical relationship between the cost function E and the weight vector w, as shown in Figure 2.2, where the cost function has only one global minimum. The principle of gradient descent tells us that to minimize

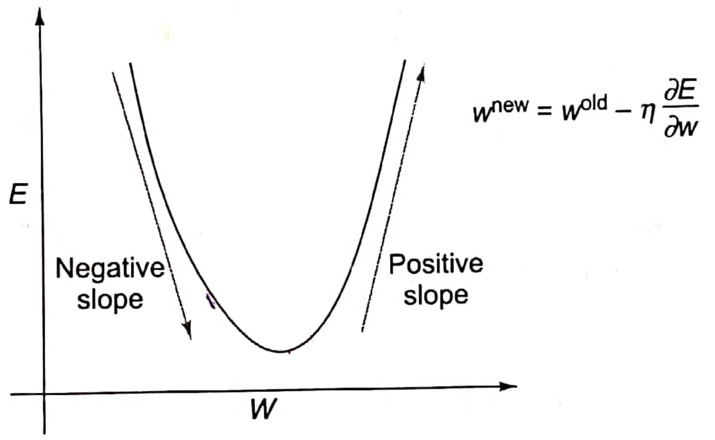

Figure 2.2 Weight update rule in the gradient descent algorithm

the cost function E, the weight vector w should be updated using the following rule:

$$\boxed{w^{new} = w^{old} - \eta \frac{\partial E}{\partial w}}$$

where η is the learning rate. Figure 2.2 shows that E attains its minimum value at $w = w_{\min}$. One should notice that when w is less than $w_{\min}$, the slope $\frac{\partial E}{\partial w}$ is negative thus the change in w is positive which will move w towards $w_{\min}$. Similarly when w is greater than $w_{\min}$, the change in w is negative which makes w to move in the left direction, i.e., towards the direction of $w_{\min}$. In both the cases, w will tend to $w_{\min}$ in a number of steps depending on the learning rate η.

Batch Update: When the weight vector w is updated such that $\Delta w = w^{\text{new}} - w^{\text{old}}$ is a function of the overall cost function E, i.e., $\Delta w = -\eta \frac{\partial E}{\partial w}$, the update rule is termed as a *batch update* which is an offline technique of weight update.

Instantaneous Update: When the weight vector w is updated such that $\Delta w = w^{\text{new}} - w^{\text{old}}$ is a function of the instantaneous cost function E^p, i.e., $\Delta w = -\eta \frac{\partial E^p}{\partial w}$, the update rule is termed as an *instantaneous update*.

2.2.2 Derivation of Back Propagation Algorithm

The back propagation (BP) algorithm [26, 27] offers an effective approach to the computation of the gradients. This can be applied to any optimization formulation, i.e., any type of energy function, either maximization or minimization problem.

Let us consider a two-layered network as shown in Figure 2.3 where i_2, i_1, and i_0 refer to the output layer, hidden layer, and input layer, respectively.

The objective of the back propagation algorithm is to adjust the weights $W_{i_2 i_1}$ and $W_{i_1 i_0}$ so as to minimize a cost function E which will train the network mapping of the required function.

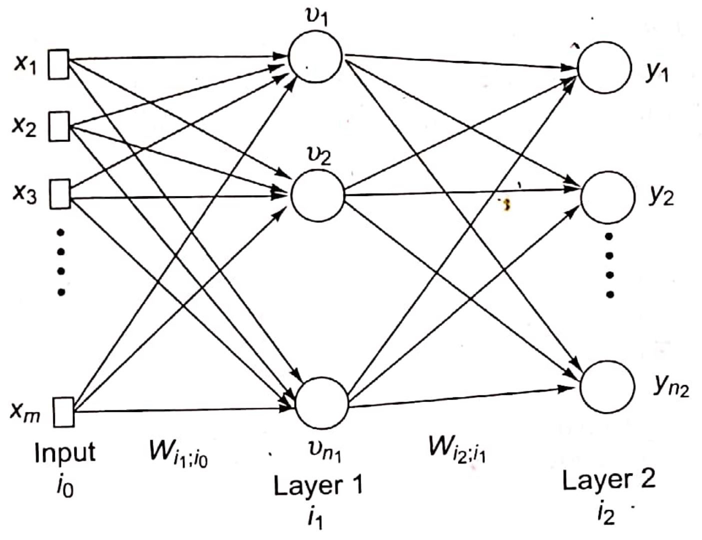

Figure 2.3 Two-layered network

Forward Phase

As shown in Figure 2.3, the input to the i_1^{th} neuron of the hidden layer is given by

$$h_{i_1} = \sum_{i_0=1}^{m} W_{i_1 i_0} x_{i_0} \tag{2.2}$$

where p is the dimension of the input vector $\boldsymbol{x}$.

Output of the i_1^{th} neuron of the hidden layer is given as

$$v_{i_1} = \Psi(h_{i_1}) = \frac{1}{1 + e^{-h_{i_1}}} \tag{2.3}$$

where $\Psi(.)$ is the sigmoidal activation function. The final response (actual output) of the i_2^{th} neuron in the output layer is given as

$$y_{i_2} = v_{i_2} = \Psi(h_{i_2}) = \frac{1}{1 + e^{-h_{i_2}}} \tag{2.4}$$

where

$$h_{i_2} = \sum_{i_1=1}^{n_1} W_{i_2 i_1} v_{i_1} \tag{2.5}$$

Back Propagation of Error

The weights of a multi-layered network can be updated using either batch mode or instaneneous mode. However, for most applications including control applications that need real-time implementations, it is beneficial to perform instanteneous update. Thus, the instanteneous back propagation algorithm will be derived in this section. The instantaneous cost function in error can be expressed as

$$E(t) = \sum_{i_2=1}^{n_2} E_{i_2}(t) = \frac{1}{2}\sum_{i_2=1}^{n_2}(y_{i_2}^d(t) - y_{i_2}(t))^2 \tag{2.6}$$

where

$$E_{i_2}(t) = \frac{1}{2}(y_{i_2}^d(t) - y_{i_2}(t))^2 \tag{2.7}$$

and $y_{i_2}^d$ is the desired response of the i_2^{th} unit of the output layer.

Update of the weights connecting the output layer and the hidden layer
The update of the weight using the gradient descent algorithm looks as

$$W_{i_2i_1}(t+1) = W_{i_2i_1}(t) - \eta\frac{\partial E(t)}{\partial W_{i_2i_1}(t)} \tag{2.8}$$

where η is the learning rate.

The derivative $\frac{\partial E}{\partial W_{i_2i_1}}$ is computed as follows:

$$\frac{\partial E(t)}{\partial W_{i_2i_1}(t)} = \sum_{i_2=1}^{n_2}\frac{\partial E_{i_2}(t)}{\partial y_{i_2}} \times \frac{\partial y_{i_2}}{\partial W_{i_2i_1}(t)} \tag{2.9}$$

where the first term can be computed using Eqn (2.7)

$$\frac{\partial E_{i_2}(t)}{\partial y_{i_2}} = -\frac{1}{2} \times 2(y_{i_2}^d - y_{i_2}) \tag{2.10}$$

and the second term is computed as

$$\frac{\partial y_{i_2}}{\partial W_{i_2i_1}} = \frac{\partial y_{i_2}}{\partial h_{i_2}} \times \frac{\partial h_{i_2}}{\partial W_{i_2i_1}} \tag{2.11}$$

Following relations are obtained using Eqns (2.4) and (2.5), respectively

$$\frac{\partial y_{i_2}}{\partial h_{i_2}} = \frac{\partial}{\partial h_{i_2}}\left[\frac{1}{1+e^{-h_{i_2}}}\right] = y_{i_2}(1 - y_{i_2}) \tag{2.12}$$

$$\text{and } \frac{\partial h_{i_2}}{\partial W_{i_2i_1}} = v_{i_1} \tag{2.13}$$

The computable form of the gradient (2.9) is obtained usings Eqns (2.10), (2.12), and (2.13) as

$$\frac{\partial E(t)}{\partial W_{i_2i_1}(t)} = -(y_{i_2}^d - y_{i_2})y_{i_2}(1 - y_{i_2})v_{i_1} \tag{2.14}$$

Thus, the update law turns out to be

$$\begin{aligned} W_{i_2i_1}(t+1) &= W_{i_2i_1}(t) + \eta(y_{i_2}^d - y_{i_2})y_{i_2}(1 - y_{i_2})v_{i_1} \\ &= W_{i_2i_1}(t) + \eta\delta_{i_2}v_{i_1} \end{aligned} \tag{2.15}$$

where $\delta_{i_2} = y_{i_2}(1 - y_{i_2})(y_{i_2}^d - y_{i_2})$ is the error back propagated from the output layer.

Update of the weights connecting the hidden layer and the input layer

Unlike a neuron in the output layer, a neuron in the hidden layer has no specified desired response. Thus, the output error has to be back-propagated so that weights connecting to the hidden layer from the input layer can be updated.

Update of the weight using the gradient descent algorithm looks as

$$W_{i_1 i_0}(t+1) = W_{i_1 i_0}(t) - \eta \frac{\partial E(t)}{\partial W_{i_1 i_0}(t)} \tag{2.16}$$

The derivative term $\frac{\partial E}{\partial W_{i_1 i_0}}$ is computed as

$$\frac{\partial E(t)}{\partial W_{i_1 i_0}(t)} = \sum_{i_2=1}^{n_2} \frac{\partial E_{i_2}(t)}{\partial y_{i_2}} \times \frac{\partial y_{i_2}}{\partial W_{i_1 i_0}(t)} \tag{2.17}$$

where the first term is computed using Eqn (2.10) and the second term is computed as follows:

$$\frac{\partial y_{i_2}}{\partial W_{i_1 i_0}} = \frac{\partial y_{i_2}}{\partial v_{i_1}} \times \frac{\partial v_{i_1}}{\partial W_{i_1 i_0}} \tag{2.18}$$

In this equation the first term is computed using Eqns (2.4) and (2.5)

$$\frac{\partial y_{i_2}}{\partial v_{i_1}} = \frac{\partial y_{i_2}}{\partial h_{i_2}} \times \frac{\partial h_{i_2}}{\partial v_{i_1}} = y_{i_2}(1 - y_{i_2}) W_{i_2 i_1} \tag{2.19}$$

while the second term is computed using Eqns (2.2) and (2.3)

$$\frac{\partial v_{i_1}}{\partial W_{i_1 i_0}} = \frac{\partial v_{i_1}}{\partial h_{i_1}} \times \frac{\partial h_{i_1}}{\partial W_{i_1 i_0}} = v_{i_1}(1 - v_{i_1}) x_{i_0} \tag{2.20}$$

Thus, the final computable expression for the gradient term (2.17) is obtained using Eqns (2.10), (2.19), and (2.20):

$$\frac{\partial E}{\partial W_{i_1 i_0}} = -\sum_{i_2=1}^{n_2} (y_{i_2}^d - y_{i_2}) y_{i_2}(1 - y_{i_2}) W_{i_2 i_1} v_{i_1}(1 - v_{i_1}) x_{i_0}$$

$$= -v_{i_1}(1 - v_{i_1}) x_{i_0} \sum_{i_2=1}^{n_2} \delta_{i_2} W_{i_2 i_1}$$

Thus the update law (2.16) has the final computable form

$$W_{i_1 i_0}(t+1) = W_{i_1 i_0}(t) + \eta \delta_{i_1} x_{i_0} \tag{2.21}$$

where $\delta_{i_1} = v_{i_1}(1 - v_{i_1}) \sum_{i_2=1}^{n_2} \delta_{i_2} W_{i_2 i_1}$

The above recursive formula is the key to back propagation learning. It allows error signal of a lower layer to be computed as a linear combination of the error signal of the upper layer. In this manner, the error signals are back propagated through all the layers from the top down. This implies that the influence from