CHAPTER IV

NEURAL NETWORK FUNDAMENTALS

Artificial neural networks are mathematical structures involving learning process. After neural network has learned what it needs to know, the trained network can be used to perform certain tasks depending on the particular application. Neural networks have the ability to learn from their environment and to adapt to it in an interactive manner similar to their biological counterparts. The neural network paradigm emerged from attempts to simulate and understand the working of he human brain. The human brain is composed of networks of neurons. There are about 1010 neurons in the brain and each neurons is randomly connected to approximately 104 other neurons. Today's neural network models are only simplified structures and in no way similar to the complexities of human brain.

4.1 Biological neural networks

The nervous system is a vast and complex neural network. The brain is the central element of the nervous system. It is connected to receptors that shuttle sensory information to it, and it delivers action commands to effectors. The brain itselfs consist of a network of about 10^{11} neurons that are interconnected through subnetworks called nuclei. The subnetworks usually divide up and modify the incoming sensory information before sending the information to other subnetworks. The final form of processed signals is delivered to effectors to initiate an action.

A biological neuron consists of three main components: dendrites, cell body, and axon; see Figure 4.1. Dendrites are branchlike protrusions from neural cell body. The dendrites receive signals from other neurons (dendrites act like input hannels). The receiving zones of impulses, called synapses, are on dendrites and cell body. The number of synaptic connections from other neurons may range from few hundred to 10,000.

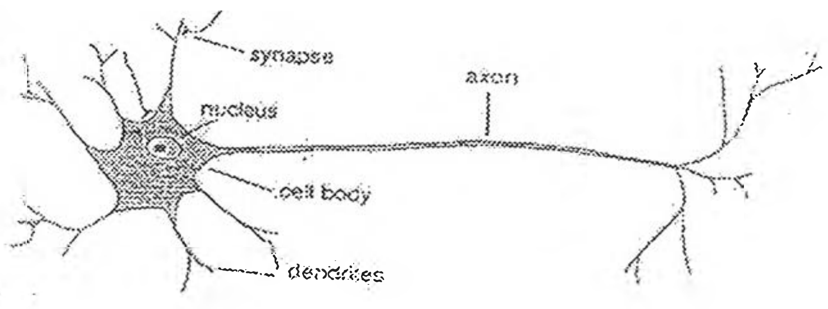


Figure 4.1 Components of neural

The cell body or soma sums the incoming signals from dendrites and sums the signal from the numerous synapses on its surface.

The axons, the transmit channel of impulses, is a long, fiberlike extension of cell body. Each neuron has one axon, which branches or fans out to other neurons.

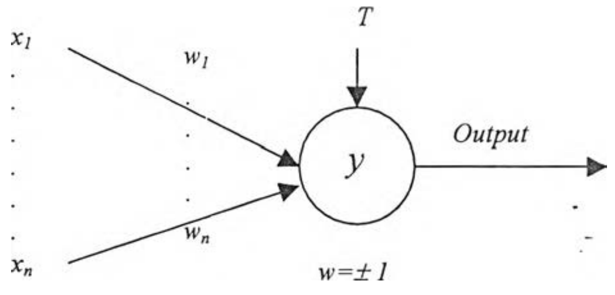
The mentioned basic concept of biological neural networks led to research in the area of the mechanism and model of human brain including develop the model to solve complex problems in science and engineering.

The first artificial neuron was produced in 1943 by the neurophysiologist Warren McCulloch and the logician Walter Pits. They proposed the model of a simple neuron, which seemed appropriate for modeling symbolic logic, perceptron, and behavior. The McCulloch-Pitts neuron is a simple unit having a linear activation function wit threshold value to produce an output. Figure 4.2 shows a simple neuron network model. The signal pass from the neural input (x_1 to x_n) to y. The total input signal received is u and if $u \geq T$, then the neuron output is $y = 1$. If $u < T$, then y = 0, that is,

$$y = \begin{cases} 1 & \text{for } u \geq T \\ 0 & \text{for } u < T \end{cases} \qquad (4.1)$$

where T is threshold.

Later, in 1959, Rosenblatt began work on the Perceptron. The perceptron consisted of neuron-like processing units with linear thresholds, and were arranged in layers similar to biological systems. The perceptron can learn and computes a weighted sum of the inputs, subtracts a threshold, and passes one of two possible values out as the result.



where		
$x_1, x_2, \ldots, x_n$	=	Input
$w_1, w_2, \ldots, w_n$	=	Connection weight
T	=	Threshold

Figure 4.2 Architecture of a McCulloch-Pitts neuron y

In 1959, Widrow and Hoff developed models which is called ADALINE and MADALINE. These models were named for their use of Multiple ADAptive LINear Elements. MADALINE was the first neural network to be applied to a real world problem. It is an adaptive filter which eliminates echoes on phone lines. This neural network is still in commercial use. ADALINE used a different learning method called the delta rule. The ADALINE's method of learning is supervised learning, in which the neuron was given a target value. ADALINE uses this target value to calculate the prediction error and moves the weight values in the direction of the negative gradient of the error. Still the ADALINE is a linear neuron (having a linear transfer function) and is limited to learning linear separable classes.

4.2 Basic artificial neural network

Artificial neural networks consist of many interconnected processing elements (artificial neurons or nodes). That a neuron is an information processing unit that roughly resembles its biological counterpart. Figure 4.3 shows a model of an artificial neural. There are four basic components of model:

(1) There is a set of synapses with associated synaptic or connection weights. As shown in figure 4.3, the continuous-valued input to synapses is a vector signal, with the individual vector components given as x_j for j = 1, 2, …,n, Each vector component x_j is input to the synapses and connected to neuron through a synaptic weight w; that is, each of these inputs are multiplied by weight as shown in equation (4.2).

$$y_{in} = b + w_1x_1 + w_2x_2 + w_3x_3 \text{ ; } b \text{ is bias term.} \qquad (4.2)$$

(2) The summing device acts to add all the signals; that is, each input is multiplied be weight and then summed.

(3) The activation function or transfer function, serves to limit the amplitude of neuron output. When activation function is nonlinear, its finite limits are typically normalized in the rage of either [0,1] or [-1,1]. The activation functions that are commonly supported are sigmoid, sine, hypoberlic tangent, etc. Equation (4.3) shows an example of calculation by using activation function, logistic sigmoid.

$$Output\ y = f(y_{in}) = 1/(1+ exp\ (-y_{in})) \tag{4.3}$$

(4) The threshold or bias is usually externally applied to the activation function or incorporated into weight.

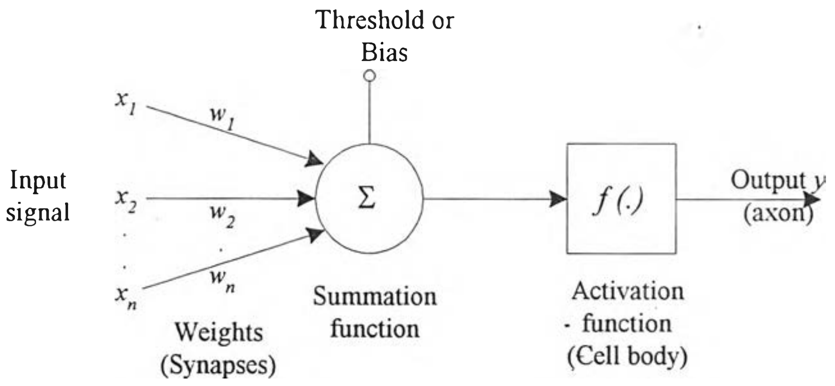


Figure 4.3 Nonlinear model of artificial neuron

4.2.1 Major components of an artificial neuron network

Component 1. Weighting Factors: A neuron usually receives many simultaneous inputs. Each input has its own relative weight which gives the i..put the impact that it needs on the processing element's summation function. These weights perform the same type of function as do the the varying synaptic strengths of biological neurons. In both cases, some inputs are made more important than others so that they have a greater effect on the processing element as they combine to produce a neural response.

Weights are adaptive coefficients within the network that determine the intensity of the input signal as registered by the artificial neuron. They are a measure of an input's connection strength. These strengths can be modified in response to various training sets and according to a network's specific topology or through its learning rules.

Component 2, Summation Function / Basis Function: The first step in a processing element's operation is to compute the weighted sum of all of the inputs. Mathematically, the inputs and the corresponding weights are vectors which can be represented as $(x_1, x_2 \ldots x_n)$ and $(w_1, w_2 \ldots w_n)$. This simplistic summation function is found by muliplying each component of the x vector by the corresponding component of the w vector and then adding up all the products. Input 1 = $x_1 * w_1$, input 2 = $x_2 * w_2$, etc., are added as input 1 + input 2 + . . . + input n.

The summation function can be more complex than just the simple input and weight sum of products. The input and weighting coefficients can be combined in many different ways before passing on to the transfer function.

The basis function has to common forms:

1. Linear-Basis function (LBF) is a hyperplane-type function. This is a first-order basis function. The net value is a linear combination of the inputs,

$$u_i(w,x) = \sum_{j=1}^{n} w_{ij} x_j \tag{4.4}$$

2. Radial-basis function (RBF) is a hypersphere-type function. This involves asecond-order (nonlinear) basis function. The net value represents the distance to a reference pattern,

$$u_i(w,x) = \sqrt{\sum_{j=1}^{n} (x_i - w_{ij})^2} \tag{4.5}$$

Moreover, the second-order function can also be extended to a (more general) elliptic-basis function.

Component 3. Transfer Function or Activation Function: The result of the summation function, almost always the weighted sum, is transformed to a working output through an algorithmic process known as the transfer function. In the transfer function the summation total can be compared with some threshold to determine the neural output. If the sum is greater than the threshold value, the processing element generates a signal. If the sum of the input and weight products is less than the threshold, no signal (or some inhibitory signal) is generated. Both types of response are significant.

The transfer function is generally non-linear. Linear (straight-line functions are limited because the output is simply proportional to the input. Linear functions are not very useful. For example, the most common transfer functions (Figure 4.1) are step function (hard limiter), ramping function, Gaussian function and sigmoid function. sample transfer functions.

The sigmoid functions such as logistic and tanh and the Gaussian function are the most common choices. Sigmoid function or S-shaped curve approaches a minimum and maximum value at the asymptotes. It is common for this curve to be called a sigmoid when it ranges between 0 and 1, and a hyperbolic tangent when it ranges between -1 and 1.

Transfer functions for the hidden units are needed to introduce nonlinearity into the network. Without nonlinearity, hidden units would not make nets more powerful than just plain perceptrons (which do not have any hidden units, just input and output units). The reason is that a composition of linear functions is again a linear function. However, it is the nonlinearity (i.e, the capability to represent nonlinear functions) that makes multilayer networks so powerful. Functions such as tanh that produce both positive and negative values tend to yield faster training than functions that produce only positive values such as logistic, because of better numerical conditioning. For continuous-valued targets with a bounded range, the logistic and tanh functions are useful, provided you either scale the outputs to the range of the targets or scale the targets to the range of the output activation function ("scaling" means multiplying by and adding appropriate constants). But if the target values have no known bounded range, it is better to use an unbounded activation function, most often the identity function (which amounts to no activation function). If the target values are positive but have no known upper bound, an exponential output activation function could be applied.

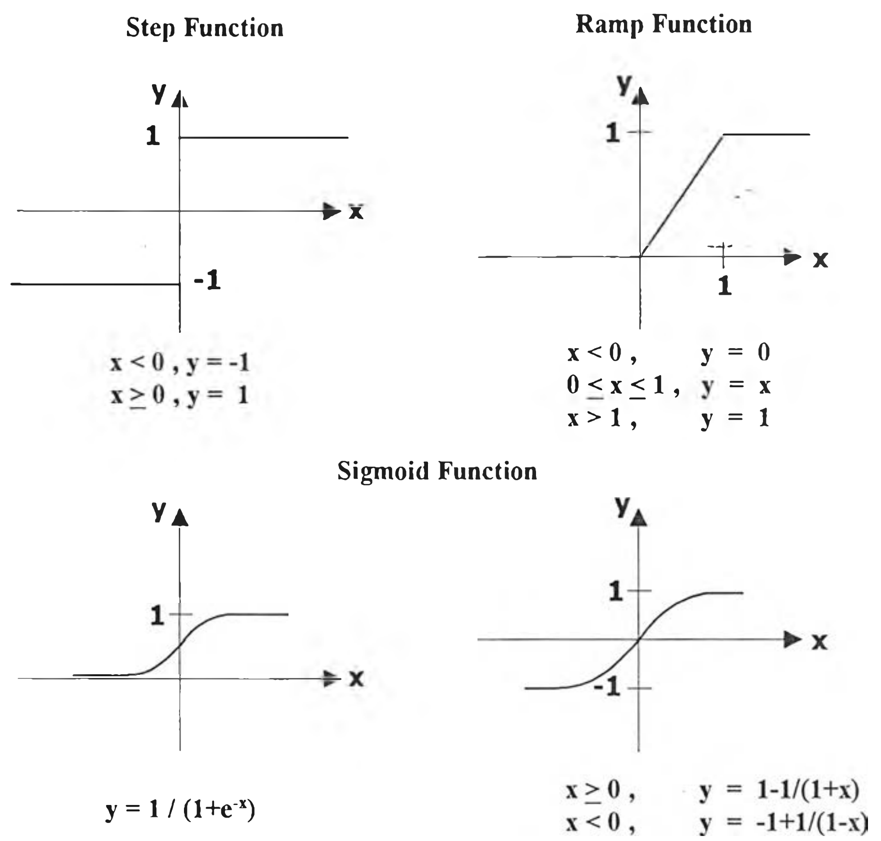


Figure 4.4 Sample transfer functions

<u>**Component 4.**</u> **Scaling** : After the processing element's transfer function, the result can pass through additional processes which scale. This scaling simply add or subtract a constant and then multiply or divide by a constant. That the neural networks training process can be made more efficient if scaling processing steps are carried out on the input pattern and target. For example, in the backpropagation algorithm used to train a feedforward perceptron, if a sigmoid function is used as a nonlinear activation function, the saturation limit are 0 and 1. If the training patterns have large values compared to these limits, the nonlinear activation functions could be operating almost exclusively in a saturated mode and not allow the network to train. Therefore, the training data should be range-scale to avoid this problem.

The training data can be amplitude-scaled in basically two ways: so that the value of the patterns lie between –1 and 1, or so that the values of the patterns lie between 0 and 1. These two types of amplitude scaling are usually referred to as min/max scaling.

Component 5. Output Function : Each processing element is allowed one output signal which it may output to hundreds of other neurons. This is just like the biological neuron, where there are many inputs and only one output action. Normally, the output is directly equivalent to the transfer function's result. Some network topologies, however, modify the transfer result to incorporate competition among neighboring processing elements. Neurons are allowed to compete with each other, inhibiting processing elements unless they have great strength. Competition can occur at one or both of two levels. First, competition determines which artificial neuron will be active, or provides an output. Second, competitive inputs help determine which processing element will participate in the learning or adaptation process.

Component 6. Error Function: Most methods for training supervised networks require a measure of the discrepancy between the networks output value and the target (desired output) value. The difference between the target and output values is called the "error".

Usually, an error function is applied to each case and is defined in terms of the target and output values. Error functions are also called "loss" functions, for example, squared-error, mean or average of squared errors.

a) Square-error

The square error can be calculated by the following equation:

$$E(y,p) = (y-p)^2 \qquad (4.6)$$

Where E = error, y = output value, p = target value

The error function for an entire data set or total error is the sum of squared errors, abbreviated SSE.

b) Mean square error or average square error

The average error is the mean or average of squared errors, abbreviated MSE or ASE The average error is the proportion of misclassified cases.

Let $z_m = (x_i, y)$, $i = 1, \ldots, m$ where is order of training set

W = connection weight

$g(x, W)$ = output of neural network

Mean Square Error can be calculated by the following equation:

$$m\text{-}1((y_1-g(x_1, W))^2+(y_1-g(x_1, W))^2+\ldots+(y_m-g(x_m, W))^2) \qquad (4.7)$$

Using the average error instead of the total error is especially convenient when using batch backprop-type training methods where a learning rate must be supplied to multiply by the negative gradient to compute the change in the weights. If the gradient of the average error is used, the choice of learning rate will be relatively insensitive to the number of training cases. But if the gradient of the total error is used, smaller learning rates would be used for larger training sets.

Component 7. **Learning Function:** The purpose of the learning function is to modify the variable connection weights on the inputs of each processing element according to some neural based algorithm. This process of changing the weights of the input connections to achieve some desired result can also be called the adaption function, as well as the learning mode. There are two types of learning: supervised and unsupervised. Supervised learning requires a teacher. The teacher may be a training set of data or an observer who grades the performance of the network results. Either way, having a teacher is learning by reinforcement. When there is no external teacher, the system must organize itself by some internal criteria designed into the network. This is learning by doing.

4.2.2 Architecture of neural network

4.2.2.1 Neural network structure

Neural network structure can be divided into types: feedforward networks and feedback networks:

1) Feed-forward networks

Feed-forward networks (Figure 4.5) allow signals to travel one way only; from input to output. There is no feedback (loops) i.e. the output of any layer does not affect that same layer. Feed-forward networks tend to be straight forward networks that associate inputs with outputs. They are extensively used in pattern recognition. This type of organisation is also referred to as bottom-up or top-down.

2) Feedback networks

Feedback networks can have signals travelling in both directions by introducing loops in the network. Feedback networks are very powerful and can get extremely complicated. Feedback networks are dynamic; their 'state' is changing continuously until they reach an equilibrium point. They remain at the equilibrium point until the input changes and a new equilibrium needs to be found. Feedback architectures are also referred to as interactive or recurrent, although the latter term is often used to denote feedback connections in single-layer organisations.

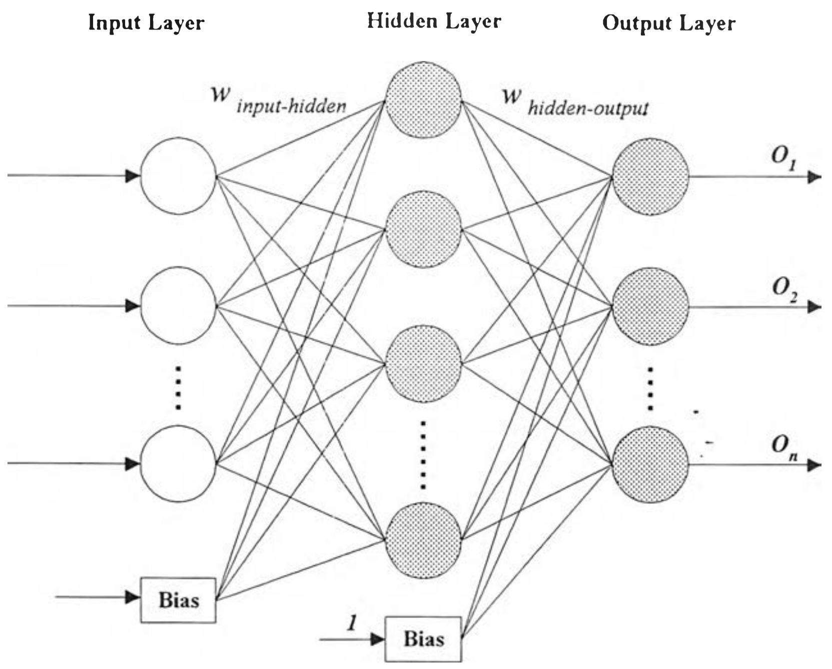


Figure 4.5 General structure of feedforward network with one hidden layer

4.2.2.2 Connection Structures

A neural network comprises the neuron and weight building blocks. The behavior of the network depends largely on the interaction between these building blocks. There are four types of weighted connections: feedforward, feedback, lateral, and time-delayed connections, as shown in Figure 4.6:

1. *Feedforward connections* : For all the neural models, data from neurons of a lower layer are propagated forward to neurons of an upper layer via feedforward connections networks.

2. *Feedback Connections*: Feedback networks bring data from neurons of an upper layer back to neurons of a lower layer.

3. *Lateral Connections*: . In the feature map example, by allowing neurons to interact via the lateral network, a certain topological ordering relationship can be preserved. Another example is the *lateral orthogonalization network* which forces the network to extract orthogonal components.

4. *Time-delayed Connections*: Delay elements may be incorporated into the connections to yield temporal dynamics models. They are more suitable for temporal pattern recognitions.

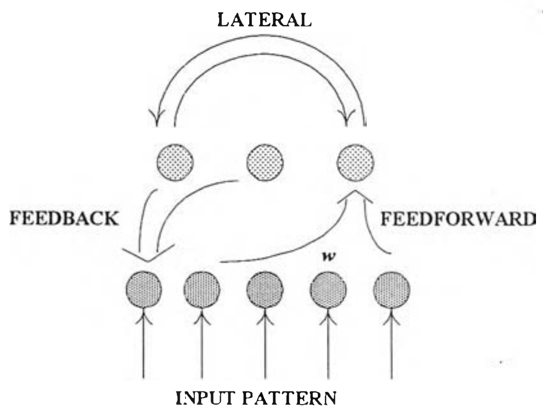


Figure 4.6 Basic structure of neural network weighted connection

4.2.2.3 Network layers

The commonest type of artificial neural network consists of three groups, or layers, of units: a layer of "**input**" units is connected to a layer of "**hidden**" units, which is connected to a layer of "**output**" units.

1) The activity of the input units represents the raw information that is fed into the network.

2) The activity of each hidden unit is determined by the activities of the input units and the weights on the connections between the input and the hidden units.

3) The behavior of the output units depends on the activity of the hidden units and the weights between the hidden and output units.

This simple type of network is interesting because the hidden units are free to construct their own representations of the input. The weights between the input and hidden units determine when each hidden unit is active, and so by modifying these weights, a hidden unit can choose what it represents.

The single-layer organisation, in which all units are connected to one another, constitutes the most general case and is of more potential computational power than hierarchically structured multi-layer organisations. In multi-layer networks, units are often numbered by layer, instead of following a global numbering.

4.2.3 Learning algorithm

Neural networks are trained rather than programmed. "Training" or "Learning" means modifying the values of the weights in the interconnections to achieve some target criteria for the output layer or nodes. Information is stored and distributed throughout the network via the interconnection weights.

Many learning rules have been developed, but there is a common feature in those learning rules. Therefore, the learning methods can be grouped into two types : supervised and unsupervised learning algorithm (Figure 4.7) .

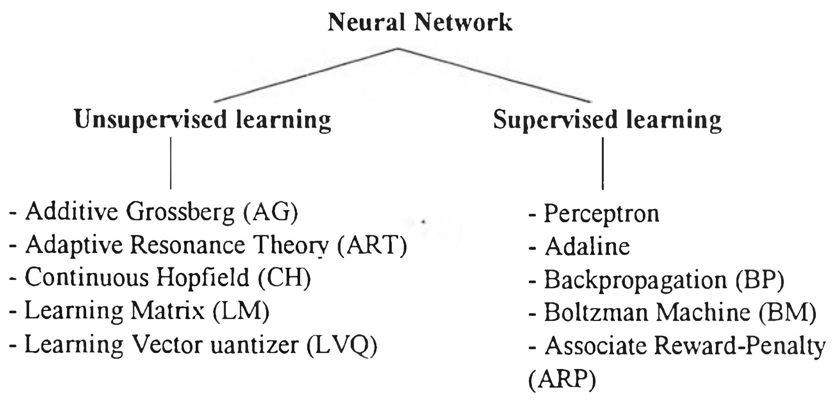

Neural Network
Unsupervised learning
Supervised learning
- Additive Grossberg (AG)
- Adaptive Resonance Theory (ART)
- Continuous Hopfield (CH)
- Learning Matrix (LM)
- Learning Vector uantizer (LVQ)
- Perceptron
- Adaline
- Backpropagation (BP)
- Boltzman Machine (BM)
- Associate Reward-Penalty (ARP)

Figure 4.7 Learning method; unsupervised and supervised learning

a) Supervised learning

The vast majority of artificial neural network solutions have been trained with supervision. In this mode, the actual output of a neural network is compared to the desired output. Weights, which are usually randomly set to begin with, are then adjusted by the network so that the next iteration, or cycle, will produce a closer match between the desired and the actual output. The learning method tries to minimize the current errors of all processing elements. This global error reduction is created over time by continuously modifying the input weights until an acceptable network accuracy is reached.

With supervised learning, the artificial neural network must be trained before it becomes useful (Figure 4.8 a). Training consists of presenting input and output data to the network. This data is often referred to as the training set. That is, for each input set provided to the system, the corresponding desired output set is provided as well. Training sets need to be fairly large to contain all the needed information if the network is to learn the features and relationships that are important. Not only do the sets have to be large but the training sessions must include a wide variety of data.

After a supervised network performs well on the training data, then it is important to see what it can do with data it has not seen before. If a system does not give reasonable outputs for this test set, the training period is not over. Indeed, this testing is critical to insure that the network has not simply memorized a given set of data but has learned the general patterns involved within an application.

b) Unsupervised Learning

Unsupervised learning is sometimes called self-supervised learning (Figure 4.8 b). Unsupervised learning is limited to networks known as self-organizing maps. These kinds of networks are not in widespread use. These networks use no external influences to adjust their weights. Instead, they internally monitor their performance. These networks look for regularities or trends in the input signals, and makes adaptations according to the function of the network. Even without being told whether it's right or wrong, the network still must have some information about how to organize itself. This information is built into the network topology and learning rules.

An unsupervised learning algorithm might emphasize cooperation among clusters of processing elements. In such a scheme, the clusters would work together. If some external input activated any node in the cluster, the cluster's activity as a whole could be increased. Likewise, if external input to nodes in the cluster was decreased, that could have an inhibitory to effect on the entire cluster.

Competition between processing elements could also form a basis for learning. Training of competitive clusters could amplify the responses of specific groups to specific stimuli. As such, it would associate those groups with each other and with a specific appropriate response. Normally, when competition for learning is in effect, only the weights belonging to the winning processing element will be updated.

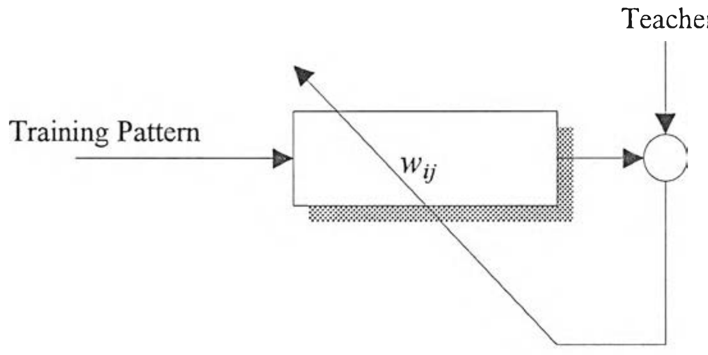


Figure 4.8 a) Supervised learning

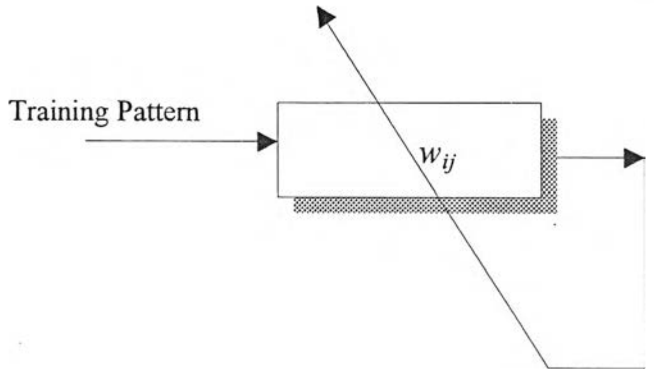


Figure 4.8 b) Unspervised learning

4.2.3.1 Learning rate

The rate at which neural networks learn depends upon several controllable factors. In selecting the approach there are many trade-offs to consider. Obviously, a slower rate means a lot more time is spent in accomplishing the off-line learning to produce an adequately trained system. With the faster learning rates, however, the network may not be able to make the fine discriminations possible with a system that learns more slowly.

Generally, several factors besides time have to be considered such as network complexity, size, paradigm selection, architecture, type of learning rule or rules employed, and desired accuracy must all be considered. These factors play a significant role in determining how long it will take to train a network. Changing any one of these factors may either extend the training time to an unreasonable length or even result in an unacceptable accuracy.

Most learning functions have some provision for a learning rate, or learning constant. Usually this term is positive and between zero and one. If the learning rate is greater than one, it is easy for the learning algorithm to overshoot in correcting the weights, and the network will oscillate. Small values of the learning rate will not correct the current error as quickly, but if small steps are taken in correcting errors, there is a good chance of arriving at the best minimum convergence.

4.2.3.2 Learning Rules

Many learning rules are in common use. Most of these rules are some sort of variation of the best known and oldest learning rules, Hebb's Rule.

Let W weight and bias

X	input signal
r	Learning signal
d	Teachering Signal
O	Output signal
α	Learning Constant
k	Time Step

The general equation of learning signal is shown in equation (4.8)

$$r = r(W,X,d) \tag{4.8}$$

The weight of network is adjusted after learning according to equation (4.9)

$$\Delta W(k) = \alpha.r\ (\ W(k),X(k),d(k)\ )\ X(k) \tag{4.9}$$

Therefore, the weight is adjusted and updated in each iteration of learning process according to equation (4.10)

$$\Delta W(k+1)\ = W(k) + \Delta W(k) \tag{4.10}$$

A few of the major rules are presented as examples.

a) Hebb's Rule: The first, and undoubtedly the best known, learning rule was introduced by Donald Hebb. The basic rule is: If a neuron receives an input from another neuron, and if both are highly active (mathematically have the same sign), the weight between the neurons should be strengthened. The weight is adjusted in the following equation:

$$w1(new) = w1(old) + X_it \tag{4.11}$$

where $wi =$ Connection weights

t = Target value

b) Hopfield Rule: It is similar to Hebb's rule with the exception that it specifies the magnitude of the strengthening or weakening. It states, "if the desired output and the input are both active or both inactive, increment the connection weight by the learning rate, otherwise decrement the weight by the learning rate."

c) The Delta Rule: This rule is a further variation of Hebb's Rule. It is one of the most commonly used. This rule is based on the simple idea of continuously modifying the strengths of the input connections to reduce the difference (the delta) between the desired output value and the actual output of a processing element. This rule changes the synaptic weights in the way that minimizes the mean squared error of the network. This rule is also referred to as the Widrow-Hoff Learning Rule and the Least Mean Square (LMS) Learning Rule. Figure 4.9 shows the process of delta rule.

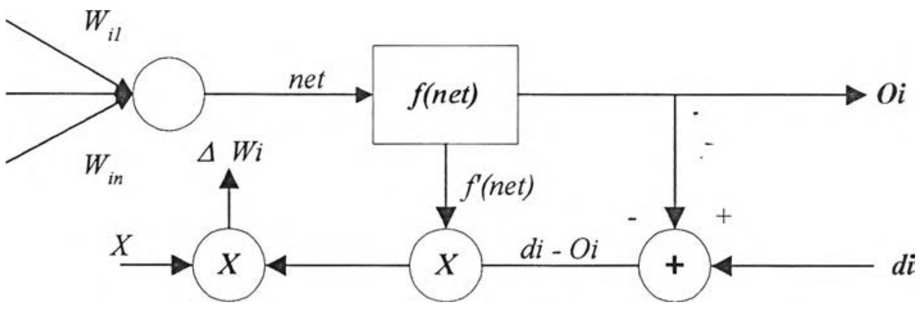


Figure 4.9 Delta learning rule

The way that the Delta Rule works is that the delta error in the output layer is transformed by the derivative of the transfer function and is then used in the previous neural layer to adjust input connection weights. In other words, this error is back-propagated into previous layers one layer at a time. The process of back-propagating the network errors continues until the first layer is reached. The network type called

Feedforward, Back-propagation derives its name from this method of computing the error term.

When using the delta rule, it is important to ensure that the input data set is well randomized. Well ordered or structured presentation of the training set can lead to a network which can not converge to the desired accuracy.

d) The Gradient Descent Rule: This rule is similar to the Delta Rule in that the derivative of the transfer function is still used to modify the delta error before it is applied to the connection weights. Here, however, an additional proportional constant tied to the learning rate is appended to the final modifying factor acting upon the weight. This rule is commonly used, even though it converges to a point of stability very slowly.

It has been shown that different learning rates for different layers of a network help the learning process converge faster. In these tests, the learning rates for those layers close to the output were set lower than those layers near the input. This is especially important for applications where the input data is not derived from a strong underlying model.

e) Kohonen's Learning Rule: This procedure, developed by Teuvo Kohonen, was inspired by learning in biological systems. In this procedure, the processing elements compete for the opportunity to learn, or update their weights. The processing element with the largest output is declared the winner and has the capability of inhibiting its competitors as well as exciting its neighbors. Only the winner is permitted an output, and only the winner plus its neighbors are allowed to adjust their connection weights. Weights are adjusted as the following equation:

$$w_{ij}(new) = w_{ij}(old) + \alpha[w_i - w_{ij}(old)] \qquad (4.12)$$

Further, the size of the neighborhood can vary during the training period. The usual paradigm is to start with a larger definition of the neighborhood, and narrow in as the training process proceeds. Because the winning element is defined as the one that has the closest match to the input pattern, Kohonen networks model the distribution of the inputs.

4.2.4 Backpropagation neural network

Backpropagation is the most widely used learning process in neural networks, and it was first developed by Werbos in 1974. This method has been recovered several times, in 1986 by Rumelhart, Hinton, and William. Backpropagation is a gradient descent learning rule, also called the generalized delta rule. In this method, the network predicted output is compared with the actual output (target), and the weights are changed in the negative direction of error to minimize the prediction error. This type of learning is known as "supervised learning".

4.2.4.1 Backpropagation learning algorithm for feedforward multilyer neural network

Training multilayer feedforward neural network with backpropagation algorithms results in a nonlinear mapping. Thus, given two sets of data, that is, input/output pairs, the multilayer feedforward neural network can have its connection weights adjusted by backpropagation algorithm to develop a specific nonlinear mapping. During training phase, the connection weights are adjusted to minimize the disparity between actual and desired output.

multilayer feedforward neural networks have three layers of weight, namely, one output layer and two hidden layers. An example of this type of neural network is shown in Figure 4.10.

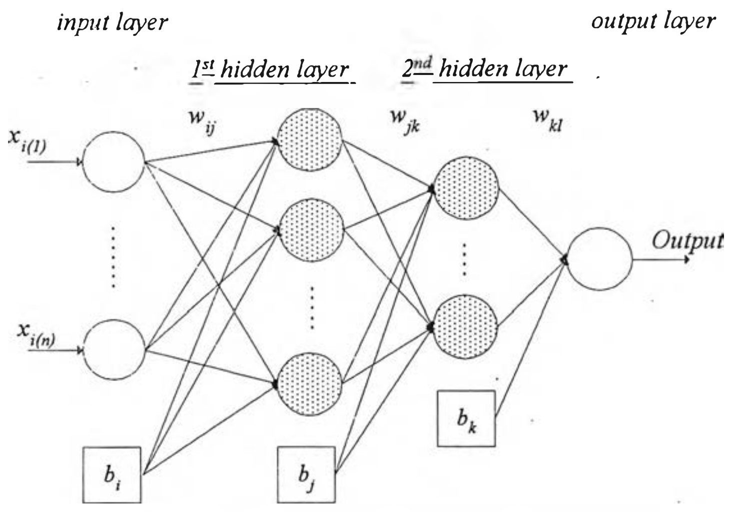


Figure 4.10 Multilayer feedforward network

Backpropagation algorithm is approach applied to minimizing the prediction error. The error function to be minimized is defined as proportional to the square of the difference between the actual and desired output, for all the data patterns to be learned. Let E_p be th e prediction error for pattern p. Then,

$$E_p = \frac{1}{2} \sum_j (d_{pj} - O_{pj})^2 \tag{4.13}$$

where d_{pj} and O_{pj} represent the desired target value and actual output, respectively. The overall error is then given by:

$$E = \Sigma E_p \tag{4.14}$$

Using a gradient descent, the learning rule for a network weight in any one of network layers is given by:

$$\frac{\partial E_p}{\partial w_{ij}} = \frac{\partial E_p}{\partial S_{pj}} \frac{\partial S_{pj}}{\partial w_{ij}} \tag{4.15}$$

where w_{ij} is the weight from node i to node j and w_{ij} is the change in w_{ij}, due to prediction error in pattern p.

The computations in unity for a pattern p can be represented by,

$$S_{pj} = \Sigma w_{ij} O_{pj} \tag{4.16}$$

$$O_{pj} = f_i(S_{pj}) \tag{4.17}$$

where O_{pj} is the output of the neuron j for pattern p . The error gradient with respect to the weights can then be determined as follows:

$$\frac{\partial E_p}{\partial w_{ij}} = \frac{\partial E_p}{\partial S_{pj}} \frac{\partial S_{pj}}{\partial w_{ij}} \tag{4.18}$$

The second term of equation (4.18) can be obtained from equation (4.16)

$$\begin{aligned} \frac{\partial S_{pj}}{\partial w_{ij}} &= \frac{\partial \sum_k w_{kj} O_{pk}}{\partial w_{ij}} \\ &= \sum_k \frac{\partial w_{jk}}{\partial w_{ij}} O_{pk} \\ &= O_{pi} \end{aligned} \tag{4.19}$$

Defining

$$\frac{\partial E_p}{\partial S_{pj}} = \delta_{pj} \tag{4.20}$$

equation (4.18) can be written as

$$-\frac{\partial E_p}{\partial w_{ij}} = \delta_{pj} O_{pi} \tag{4.21}$$

The weight change is proportional to the error gradient with respect to the weights. Therefore,

$$\Delta w_{ij} = \eta \delta_{pj} O_{pi} \qquad (4.22)$$

where η is learning rate. The values of δ_{pj} need to be determined for each neuron j. Then, the weights of the network can be updated such that the prediction error decreases. Using the chain rule on equation (4.21),

$$\delta_{pj} = -\frac{\partial E_p}{\partial S_{pj}} = -\frac{\partial E_p}{\partial O_{pj}} \frac{\partial O_{pj}}{\partial S_{pj}} \qquad (4.23)$$

From equation (4.13) and (4.17), equation (4.23) can be written as,

$$\delta_{pj} = -(d_{pj} - O_{pj}) f_j'(S_{pj}) \qquad (4.24)$$

For sigmoidal functions, $f'_{pj}\ (S_{pj})$ can easily be obtained to be

$$f_j'(S_{pj}) = O_{pj}(1 - O_{pj}) \qquad (4.25)$$

The first derivative can easily be calculated for sigmoid function from the output values only as given above.

Equation (4.24) useful for calculating δ's for the neurons in the output layer. However, this equation cannot be used for the hidden layer neurons, because the "target value" similar to d_{pj}, is not available to define the error for the hidden layer neurons. Therefore, when j refers to a hidden layer neuron, the term $\frac{\partial E_p}{\partial w_{ij}}$ can be obtained as follows:

$$\frac{\partial E_p}{\partial O_{pj}} = \sum \frac{\partial E_p}{S_{pk}} \frac{\partial S_{pk}}{\partial O_{pj}}$$

$$= \sum_k \frac{\partial E_p}{\partial S_{pk}} \frac{\partial}{\partial O_{pj}} \sum_j w_{jk} O_{pj}$$

$$= -\sum_k \delta_{pk} w_{jk} \qquad (4.25)$$

where k is the output layer. Therefore, δ for a hidden layer neuron is given by:

$$\delta_{pj} = \sum_k \delta_{pk} w_{jk} f'_j(S_{pj}) \tag{4.26}$$

The weight update rule is given by equation (4.22) and the δ_{pj} is given by equation (4.24) for neuron in the output layer, and by equation (4.26) for neurons in the hidden layer. The values of $f'_{pj}(S_{pj})$, for sigmoidal functions, required in these equations. The δ values of the output neurons are calculated first and then "propagated back" to the hidden layer, giving the name "backpropagation" to this generalized delta rule.

Implementing the error backpropagation rule as described is very simple. But this method converge slowly to the optimal values. The training algorithm can be significantly improved by using an acceleration method called "momentum algorithm" which is a conventional optimization tool. The idea of the algorithm is to update the weights in the direction with a linear combination of current gradient of the instantaneous error. The weights are updated according to:

$$\Delta w_{ij}(t) = \eta\delta_{pj}(t)\, O_{pj}(t) + \alpha\Delta w_{ij}(t-1) \tag{4.27}$$

where t refers to an epoch, or an iteration, incremented by 1 for each sweep through the whole set of input-output values. The term α is the momentum parameter which can take a value between 0 and I, determining the relative contribution of the earlier gradients to the current weight change. This procedure produces a large change in the weights if the changes are currently large, and will decrease as the changes become less. Therefore the training speed increases, and the network is less likely to get stuck in local optima early on, since the momentum term will push the network out of local optima following the overall general trend in weight movement. Figure 11 shows the forward flow of the data to the feedforward network and backward flow of the error in such network trained with error backpropagation algorithm. The summary of backpropagation algorithm step is presented in Appendix B.

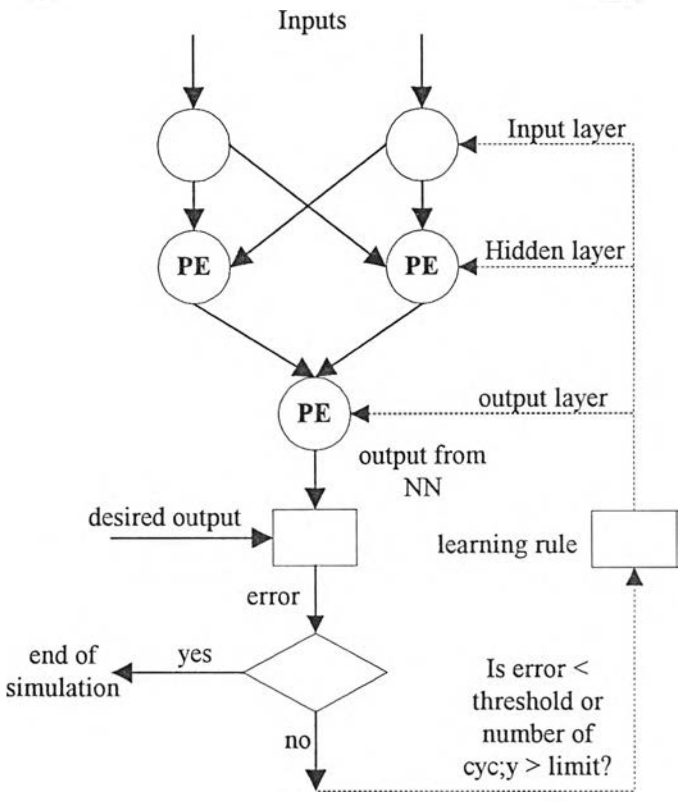


Figure 11 Forward flow of information or data (arrows) and backward flow of error (dash line) in a backpropagation type of neural network

4.2.4.2 Levenberg-Marquardt algorithm

Levenberg- Marquardt backpropagation algorithm represents a simplified version of Newton's method applied to the problem of training multilayer neural networks. To apply the Levenberg- Marquardt backpropagation algorithm, the problem of training has to be formulated as nonlinear optimization problem. Consider multilayer feedforward neural network, the task of neural network training can be viewed as finding a set of neural network output for all patterns in the training set. The algorithm can be described below:

$$e = d - x_{out} \tag{4.28}$$

$$E(w) = e^2/2 \tag{4.29}$$

$$\Delta w = -(\nabla^2 E(w))^{-1} \nabla E(w) \tag{4.30}$$

where $\nabla^2 E(w)$ is the Hessian matrix (***H***).

$\nabla E(w)$ represents the gradient of the error function ($J^T e$)

d is the desired output and *x* is an actual output.

J is the Jacabian matrix defined by,

$$J = \begin{bmatrix} \frac{\partial e_1}{\partial w_1} & \cdots & \frac{\partial e_1}{\partial w_B} \\ \vdots & \ddots & \vdots \\ \frac{\partial e_L}{\partial w_1} & \cdots & \frac{\partial e_L}{\partial w_B} \end{bmatrix} \tag{4.31}$$

By using the expression for Jacobian matrix, the Hessian can be expressed as

$$\nabla^2 E(w) = \boldsymbol{J^T J} + \boldsymbol{S} \tag{4.32}$$

where matrix S is the matrix of second order derivatives given by

$$\boldsymbol{S} = \Sigma e \nabla^2 e \tag{4.33}$$

When approaching the minimum of error function, the elemens of matrix S become small, and Hessian can be closely approximated by

$$\boldsymbol{H} \approx \boldsymbol{J^T J} \tag{4.34}$$

Therefore, we obtain

$$\Delta w = (J^T J + \eta I)^{-1} J^T e \tag{4.35}$$

where η is a small number of learning rate and I is the identity matrix.

The biggest problem in implementing the Levenberg-Marquardt is the calculation of Jacobian matrix (*J(w)*). Each tern in matrix has the form

$$J_{i,j} = \frac{\partial e_i}{\partial w_j} \approx \frac{\Delta e_i}{\Delta w_j} \tag{4.36}$$

The Levenberg-Marquardt algorithm can be summarized in the following step.

Step 1: Initialize the network weights to small random values. Set the learning rate parameter.

Step 2: Present the input pattern, and calculate the output of the network.

Step 3: Use (4.36) to calculate the elements of the Jacobia matrix associated with the input/output pair.

Step 4: When the last input/output pair is present, use (4.35) to perform the update of the weights.

Step 5: Stop if the network has converged; else, go back to step 2.

4.3 Neural network design

a) Model structure and size

No standard method has been known to determine the structure and the number of nodes of a network required for any particular application. Although there are some guidelines and heuristics suggested in the literature the actual choice still remains on a case-to-case basis. The normal procedure for selecting the hidden nodes is to fix an initial size and then check if this model satisfies the error requirement when the training process is stopped. If not, the size is revised and the whole procedure repeated until it satisfies the tolerance for the prediction error. Although the choice of the number of hidden nodes here is done by trial-and-error, normally within a few trials it becomes quite easy to constrain it in an optimum range (within some upper and lower limit) required for achieving acceptable training. The choice is also made keeping in view of one of the objective of this work which is to select parsimonious models i.e. models which contain the smallest number of free parameters such as the connection weights, required to represent the time system adequately.

b) Data collection (input/output data)

Data collection involves a number of important tasks. In utilizing neural networks, the data set collection is normally split into various sets. One is the initial training set, which is the data used to train the network weights and normally span the operating region of the model. Next is the testing data set, which is used for final validation of the trained neural network.

The choice of input data fed into the network is an important consideration in the utilization of neural networks for any particular application. For steady state application, the choice of inputs to the network basically depends on the relevant variables likely to have an effect on the predicted output variable. For modeling the dynamic behavior of a system, it would not only depend on these relevant variables but also the time history of these variables as well as the time history of the output variables. The knowledge of the system such as the model order is use as the initial guide to decide on the time history.

c) Data processing

After data collection, all data should be pre-processed using statistical procedure. Data in the training sets are pre-processed to have zero mean and unit variance. This is necessary to prevent input with large average values in certain dimension.

d) Weight Initialization

The initial weight specification has a pronounced effect on the speed and quality of neural network training. It is best to initialize the weights with small, random numbers e.g. in the range -0.5 to 0.5 (Bhat andMcAvoy, 1990) so that each connection responds slightly differently during training and has the effect of breaking the symmetry and promotes faster convergence to the global minimum. If the final prediction does not satisfy the error tolerance during training, other than reconfiguring the network, the weights are also re-initialized and the identification process repeated. This has been found to improve the performance of the neural network training.

e) Training Methodology

Training is a procedure to determine the optimal values of the connection weights and bias weights. It begins by initially assigning arbitrary small random values (both positive and negative) to the weights. Training proceeds iteratively until a satisfactory model is obtained. In each iteration, called an epoch, the actual outputs corresponding to all the sets of inputs in the training set are predicted, and the weights are adjusted in the direction in which the output prediction error decreases. For training to be complete many iterations are necessary. The weights are incrementally adjusted for every pattern in every iteration and they gradually converge on the optimal values. If n represents the total number of data patterns in the X, Y data set, where X is the matrix of inputs and Y is the matrix of targets, then one iteration corresponds to feeding all the n, patterns once. Actual outputs are not available for the hidden units. Therefore, to adjust the hidden layer weights, error from the output layer is propagated back to the hidden layer, and their weights adjusted to decrease the prediction error.

Different network architectures require different training algorithms and training times can be significantly reduced by the use of suitable algorithms. However backpropagation with its variants remain the mainstay of performing neural network (multilayered feedforward) learning. Hence training or optimization of the weights to achieve the required prediction, is performed in this work by the backpropagation technique with a momentum term. Although there are many other variants of this method to improve the speed of training, this approach is deemed sufficient to produce the required results and accuracy for our application in this work.

f) Model Validation

Overleaming, which occurs when the model starts to learn the presented pattern in a pointwise fashion instead of learning the functionality, is a potential problem that can easily occur in process identification. During overlearning the performance of the network training continues to improve on the learning data set but starts to degrade on the testing set i.e. poor generalization capability at this instance. It can however be dealt with by proper training and validation.

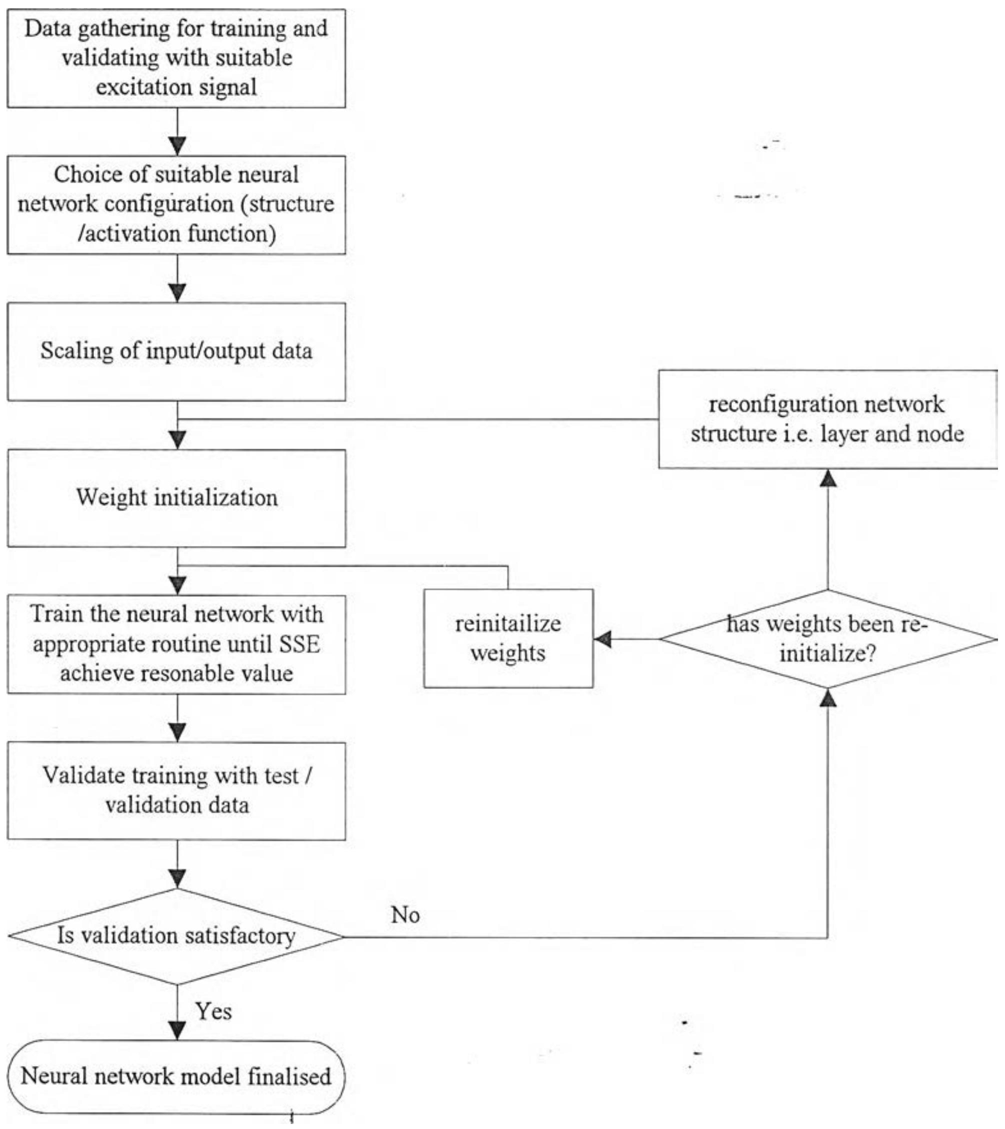

Data gathering for training and validating with suitable excitation signal
Choice of suitable neural network configuration (structure /activation function)
Scaling of input/output data
reconfiguration network structure i.e. layer and node
Weight initialization
Train the neural network with appropriate routine until SSE achieve resonable value
reinitailize weights
has weights been re-initialize?
Validate training with test / validation data
Is validation satisfactory
No
Yes
Neural network model finalised

Figure 12 Basic steps-neural network design

Most of the quantitative validation tests available are based on correlation approaches, for linear systems, intended to check whether the residuals are correlated to the input signal or among themselves (autocorrelated). Other information criteria methods such as Akaike Information Criteria, Final Prediction Error and Bayesian Information Criteria etc., (Ljung, 1987; Cherkassky el al., 1995) attempt to measure how well a model fits the data set provided as well as penalizing complex models by accounting for the number of parameters in the model. However the model can also be validated, as in mainly done in this study, by predicting the output in data sets not used in the identification procedure and the quality of the fit can be observed in terms of its sum-squared error.

The major steps required to be followed in performing neural network designed are outlined in the chart of Figure 12.

4.4 Application of artificial neural network

4.4.1 Identification

The general problem of nonlinear system identification is presented in Figure 4.13. Ideally, the identification process should be capable of producing an accurate model of nonlinear system without any prior knowledge of system dynamics. The Identification being composed of forward modeling, which the forward model of the system is identified and inverse modeling.

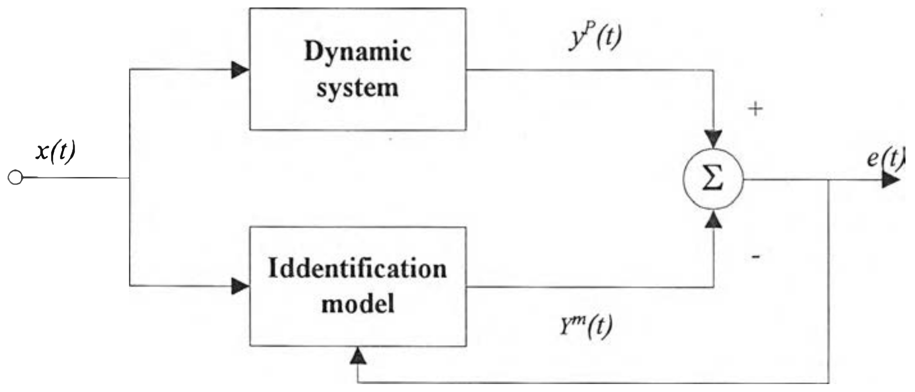


Figure 4.13 general model of the system identification process

4.4.1.1 Forward modeling

The procedure of training a neural network to represent the forward dynamics (i.e. obtain outputs given the inputs) of a system is referred to as *forward modeling*. A structure for achieving this is shown in Figure 4.14. The neural network model is placed in parallel with system and the error between the system and the network outputs (the prediction error) as the neural network training signal. A multilayered feedforward network is used in order to apply a backpropagation training algorithm. Assume that the plant is governed by the following nonlinear discrete time difference equation:

$$y^P(t+1) = f(y^P(t),\dots,y^P(t-n+1);x(t),\dots,x(t-m+1)) \qquad (37)$$

Thus, the plant output y^P at time $t+1$ depends on the past n output values and on the past m values of the input x. We concentrated here is only on the dynamical part of the plant response; the model does not explicitly represent plant disturbances (for a method of including the disturbance see, e.g., Chen et al. (1990)). Special cases of the model (4.2) have been considered by Narendra and Parthasarathy (1990).

An obvious approach for system modeling is to choose the input-output structure of the neural network to be the same as that of the system. Denoting the output of the network by y^m then it is obtained that

$$y^m(t+1) = \hat{F}(y^p(t),\dots, y^p(t-n+1); x(t),\dots, x(t-m+1)) \qquad (38)$$

In the above, the mapping $\hat{F}(.)$ represents the nonlinear input-output map of the network which approximates the plant mapping $F(.)$. Note that the input to the network includes the past values of the plant output but not the past values of the network output (the network has no feedback). The learning statistical backpropagation algorithm is used to find the optimal values of the network weights. The structure of the model equation (38) is called series-parallel.

If it is assumed that after a suitable training period the network gives a good representation of the plant (i.e. $y^m \approx y^p$), then for subsequent post-training purposes the network output itself and its delayed values can be fed back and used as part of the network input. In this way the network can be used independently of the plant. Such a network is described as

$$y^m(t+1) = \hat{F}(y^m(t),\dots,\ y^m(t-n+1);\ x(t),\dots,\ x(t-m+1)) \quad (39)$$

This structure may also be used from beginning that is during the whole process of learning. The structure of equation (39) is called parallel. The series-parallel is supported by stability results. Moreover, the parallel model requires a dynamical backpropagation training algorithm.

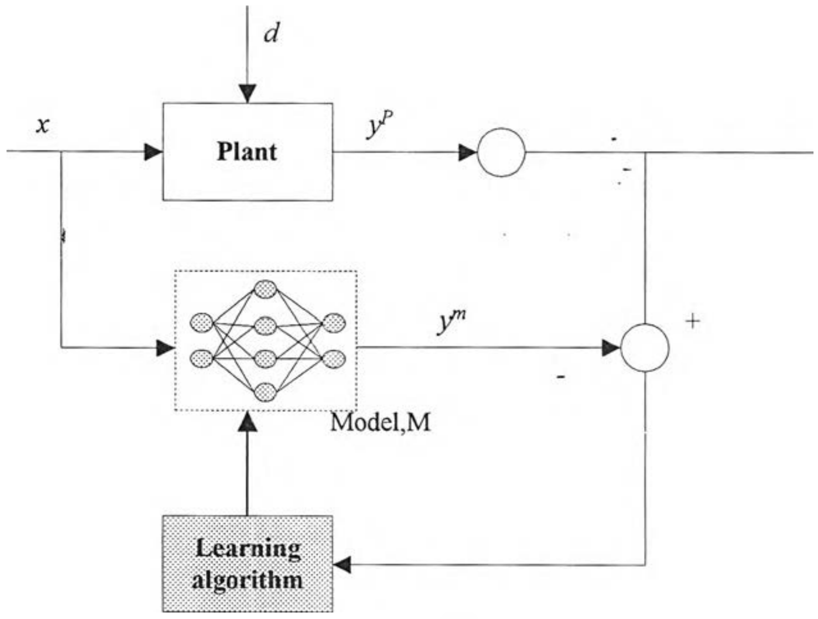


Figure 4.14 Identification

4.4.1.2 Inverse Modeling

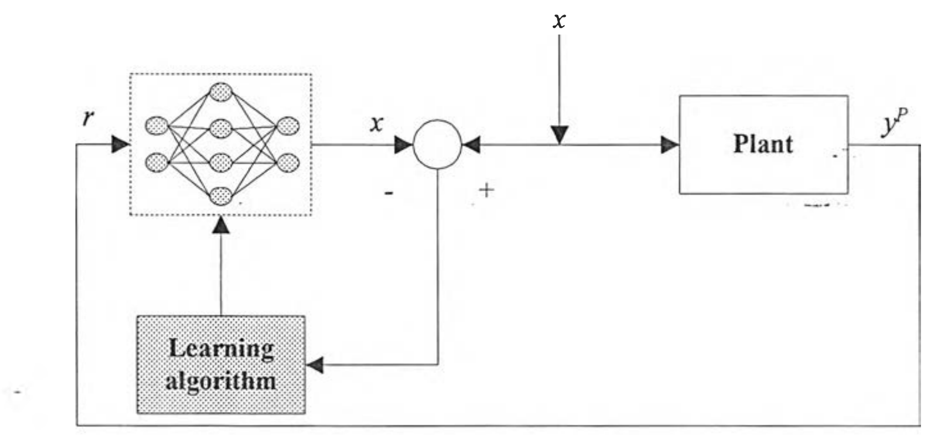
The inverse model of a dynamical system yields input for given output. The models play a crucial role in a range of control structures. However, obtaining inverse models raises several important issues. Conceptually the simplest approach is *direct inverse modeling* as shown in Figure 4.15a. Here, a synthetic training signal (the plant input) is introduced to the system. The plant output is then used as input to the network. The network output is compared with the training signal (the system input) and this error is used to train the network. This structure will clearly force the network to represent the inverse of the plant. However, there are some drawbacks:

- The learning procedure is not "goal directed"; the training signal must be chosen to sample over a wide range of system inputs, and the actual operational inputs may be hard to define a *prior*. The actual goal in the control context is to make the system *output* behave in a desired way, and thus the training signal in direct inverse modeling does not correspond to the explicit goal;
- Second, if the nonlinear system is not one-to-one, then an incorrect inverse can be obtained.

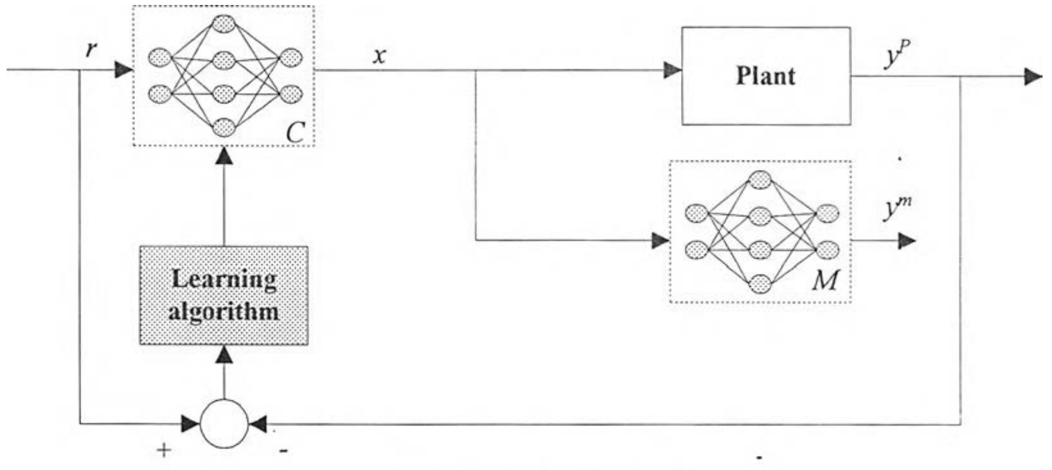
The first point is strongly related with the general concept of persistent excitation. A second approach to inverse modeling which aims to overcome these problems is known as *specialized inverse learning* (Psaltis, Sideris and Yarnamura, 1988). The specialized inverse learning structure is shown in Figure 4.15b. In this approach the network inverse model precedes the system and receives as input a training signal which spans the desired operational output space of the controlled system (i.e. it corresponds to the system reference signal). This learning structure also contains a train forward model of the system placed in parallel with the plant. The error signal for the training algorithm in this case is the difference between the training signal and the system output (it may also be the difference between the training signal and the forward model output if the system is noisy). It can be shown that using the plant output an exact inverse even when the forward model is not exact can be produced; this is not the case when the forward model output is used. The error may then be propagated back through the forward model and the inverse model; only the inverse network model weights are adjusted during this procedure. Thus, the procedure is effective at learning and identifies mapping across the inverse model and the forward model; the inverse model is learned as a side effect. In comparison with direct inverse modeling, the specialized inverse learning approach possesses the following features:

- The procedure is goal directed since it is based on the error between desired system outputs and actual outputs. In other word, the system receives inputs during training which correspond to the actual operational inputs it will subsequently receive.
- In case in which the system forward mapping is not one-to-one a particular inverse (pseudo-inverse) will be found. The problem of bias can also be handled.

Next, the input-output structure of network modeling the system inverse is considered. From equation (40) the inverse F^{-1} leading to the generation of $x(t)$would require knowledge of the future value $y^P(t+1)$. To overcome this problem we replace this future value with the value $r\ (t + 1)$ which is assumed to be available at time t. This seems to be a reasonable assumption since r is typically related to the reference



a) Direct method



b) Specialized method

Figure 4.15 Structure for inverse identification

signal which is normally known one step ahead. Thus, the nonlinear input-output mapping relation of the network modeling the plant inverse is

$$x(t)=F^{-1}(y^p(t),....,y^p(t-n+1);\ r(t+1);\ x(t-1),....,x(t-m+1)) \quad (40)$$

that is the inverse model network receives as inputs the current and past system outputs, t!:e training (reference) signal, and the past values of the system outputs. Where it is desirable to train the inverse without the plant the values of y^p the above relation are simply replaced by the forward model outputs y^m .

4.4.2 Neural network based control

Model of dynamical systems and their inverse have immediate utility for control. This topic presents two direct approaches to control : supervised control and direct inverse control.

4.4.2.1 Supervised control

There are many control situations where a human provides the feedback control actions for a particular task and where it has proven difficult to design an automatic controller using standard control techniques. In some situations it may be desirable to design an automatic controller which mimics the action of the human

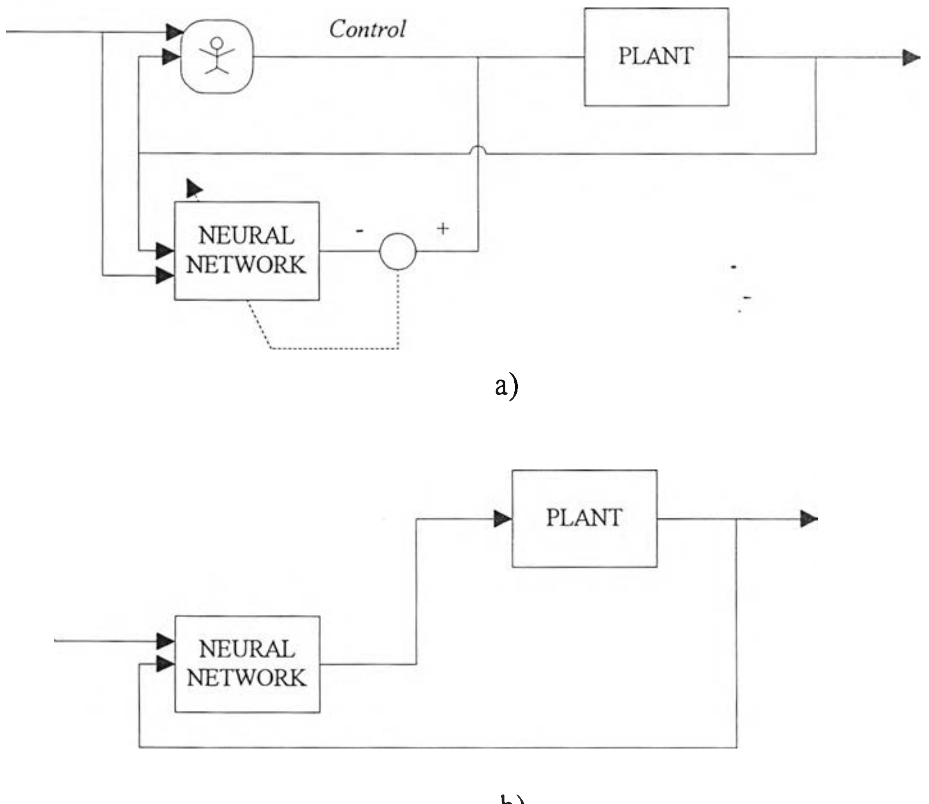


Figure 4.16 Supervised control (a) training the network (b) Operating

(this has been called supervised control).

An artificial neural network provides one possibility for this. Training the neural network is similar in principle to learning a system forward model. The network input corresponds to the sensory input information received by the human. The network target outputs used for training correspond to human control input to the system. The supervised control structure is shown in Figure 4.16.

4.4.2.2 Direct inverse control

Direct inverse control utilizes an inverse system model. The inverse model is simply cascaded with the controlled system in order that the composed system results in an identity mapping between desired response (i.e. the network inputs) and the controlled system output. Thus, the network acts directly as the controller in such a configuration. The direct inverse control structure is shown in Figure 4.17.

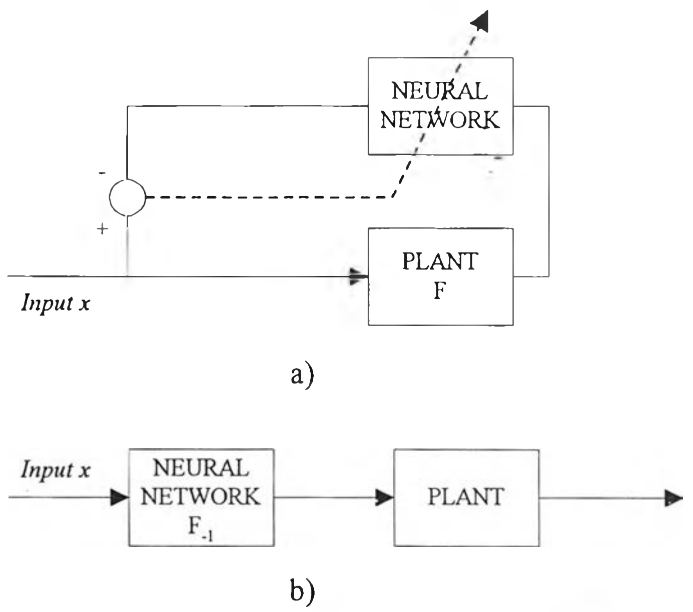


Figure 4.17 Direct inverse control (a) learning for inverse function (b) controlling by inverse function

4.4.2.3 Model reference control

Here, the desired performance of closed-loop system is specified through a stsble reference model, which is defined by its input-output pair. The control system

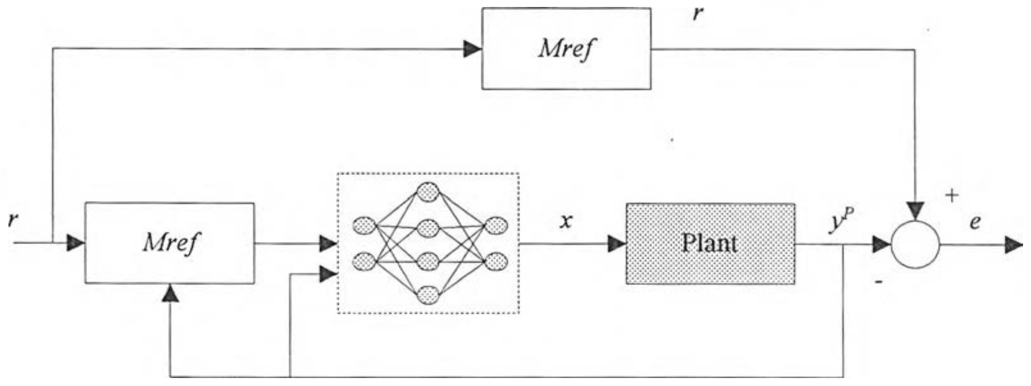


Figure 4.18 Structure for model reference

attempts to make the plant output match reference model output asymptotically. The model reference control structure for nonlinear system utilizing connectionist model is shown in Figure 4.18. In this structure the error defined above is used to train the network acting as controller. Clearly, this approach is related to training of inverse plant models as outline above. In general, the training procedure will force the controller to be detuned inverse, in a sense defined by the reference model.

4.4.2.4 Internal model control

In internal model control (IMC) the role of system forward and inverse model is emphasized. In this structure a system forward and inverse model are used directly as elements within the feedback loop. IMC has been thoroughly examined and shown to yield transparently to robustness and stability analysis. Moreover, IMC extends readily to nonlinear systems control.

In internal model control a system model is placed in parallel with the real system. The difference between the system and model outputs is used for feedback purpose. This feedback signal is then processed by a controller subsystem in the forward path; the properties of IMC dictate that this part of the controller should be related to the system inverse (the nonlinear realization of IMC illustrate in Figure 4.19.

Given network models for the system forward and inverse dynamics the realisation of internal model control using neural networks is straightforward; the system model M and controller C (the inverse model) are realised using the neural network models as shown in Figure 4.19.

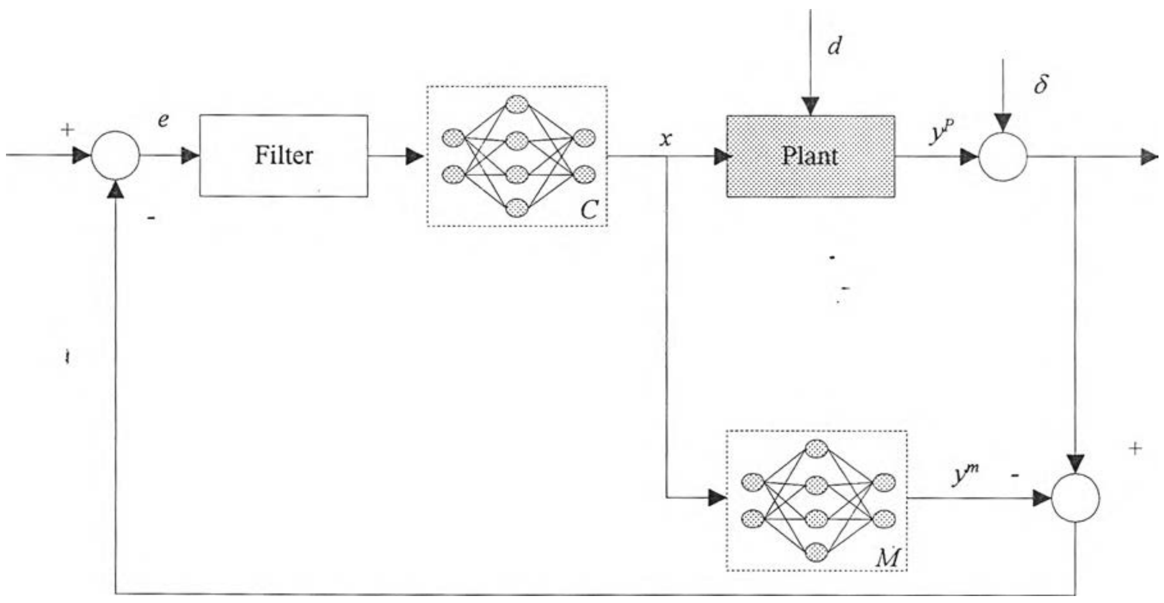


Figure 4.19 Structure for internal model control