CHAPTER III

METHODOLOGY

This chapter presents the methodologies of BD, NNs, and model based controller. The organization of this chapter is as follows: the system of conventional BD, the brief background of NN, and the basic concept of MPC.

3.1 Batch Distillation

3.1.1 Conventional System

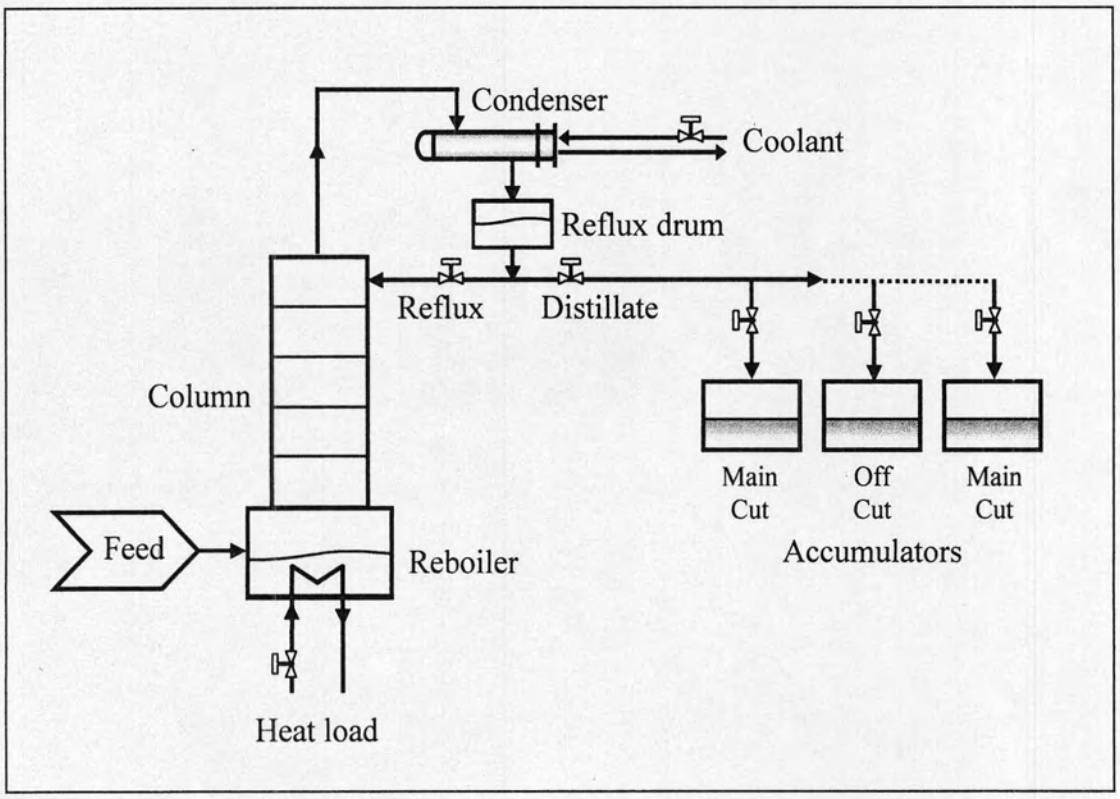

Figure 3.1 A flow schematic for a conventional BD system

A flow schematic of a conventional BD system is shown in Figure 3.1. A *feed* mixture is charged to a batch tank and is heated to a boiling point in the bottom *reboiler*. The purpose of a reboiler is to generate a vapor by vaporization of a mixture. As the mixture boils, vapor is carried up the *column* and is condensed in the top *condenser*. Then, the condensate flows to a *reflux drum*. A part of condensate is returned back to the column as *reflux* while the remainder is withdrawn in the accumulator as *distillate*. The production of a distillate can be considered *main–cut*s (main distillate) and *off–cut*s (off–specification distillate). Each off–cut may be collected and recycled to the reboiler in the next batch.

3.1.2 Operation Periods

The operation of a BD column can be divided into three periods: *start–up*, *production*, and *shutdown* period. Following describes briefly each of these operations for a conventional BD column. (Mujtaba, 2004)

- *Start–up period*

After a feed mixture is charged and heated in the reboiler, a part of the mixture is vaporized and travel through the column. Vapor is condensed in the condenser and then is stored in the reflux drum. During startup period, the system is operated under *total reflux* operation until the desired distillate purity or steady state is achieved.

- *Production period*

The production period starts when the product is withdrawn, which can be operated under the following conditions: a *constant reflux* (with distillate composition varying), *variable reflux* (with constant distillate composition), and *optimal reflux* operation. For operation with constant reflux, the product is collected at some constant finite reflux ratio until the withdrawn product composition reaches its desired purity. Next, variable reflux operation, the product is collected at constant composition by varying the reflux ratio until a specified amount at distillate has been collect. Final operation is a trade off between the previous two operations, called optimal reflux operation. The optimal policy will be to determine the optimal profile

from some objective function (minimum time, maximum product, maximum profit, etc.) and take this profile as the guiding operating principle.

- *Shutdown period*

A BD column can be shutdown by cut–off heat supply to the column. The overall column holdup is collected in the reboiler and condenser holdup may be mixed with the top product or with the reboiler material.

3.1.3 Operation and Control

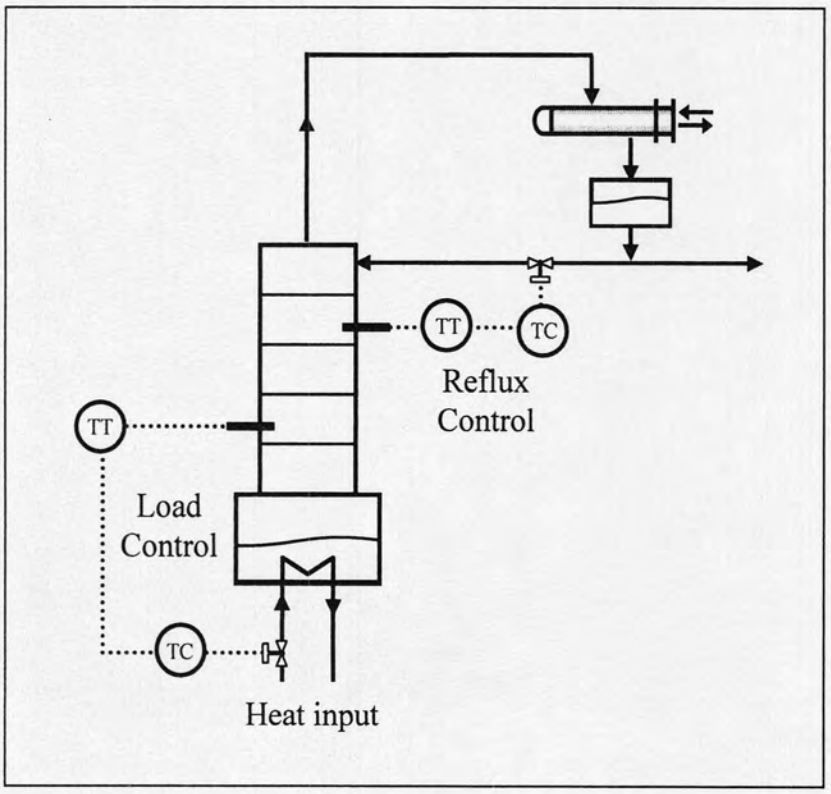

Figure 3.2 A temperature control schematic for a BD system

Typically, the control structure of BD can be classified as follow:

- *Load Control*

The BD column load is represented by the vapor flow. The vapor flow can be manipulated by heat input to the batch still. If the heating medium is steam, a simple method is to maintain constant steam pressure or flow rate to the batch still.

- *Reflux Control*

This is used when operating with constant vapor flow. The reflux rate can be manipulated to control product quality. The ways of operating the BD are maintaining constant reflux ratio, constant distillate composition or optimal reflux policy.

3.1.4 Mathematical Model

The mathematical model of a conventional BD column is based on a dynamic model using a plate–by–plate modeling approach. The model equations consist of differential and algebraic equations (DAEs). Ordinary differential equations (ODEs) represent mass and energy balances. Algebraic equations are composed of equations such as summation equations, vapor liquid equilibrium relationships, physical property estimations, etc. A general differential and algebraic BD model is denoted by

$$f(t, X'(t), X(t), U(t), C) = 0 \tag{3.1}$$

where t is time, $X(t)$ is the set of all state variables, $X'(t)$ is the time derivatives of $X(t)$, $U(t)$ is the set of control variables, and C is the set of constant parameters. For more detail see Appendix A (Distefano, 1968).

3.1.5 Optimization Problem

The optimization problem of a conventional BD column can be defined in terms of an objective function and constraints as show below.

$$\underset{u'(t)}{Minimize(or Maximize)} \quad J \tag{3.2}$$

subject to equality constraints

inequality constraints

where J is the objective function to be optimize (e.g. profit function, amount of product, batch time), $u'(t)$ denotes all the optimization variables (e.g. reflux ratio, switching time, final time). Equality and inequality constraints refer to model equations and bounds of operating variables, respectively.

The optimization problem formulation which is presented above is aimed to achieve optimal operation policies for a single period operation.

3.2 Neural Networks

This section presents the NNs background that is divided into five subsections as follows: biological and artificial neurons, network architecture, backpropagation NNs, and NN applications.

3.2.1 Biological Neurons

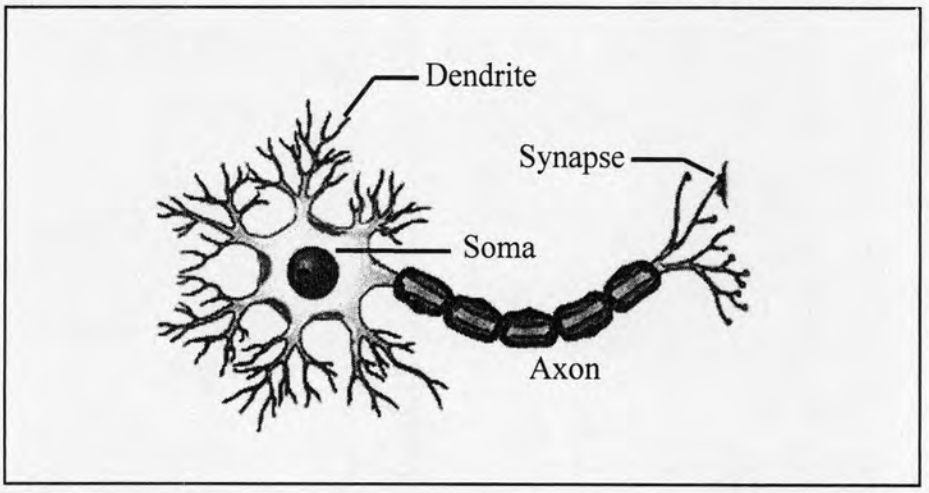

Figure 3.3 An ideal biological neuron

An ideal biological neuron is shown in Figure 3.3. A neural cell consists of a cell body with nucleus, called the *soma*. The soma receives stimuli and performs further processing of the information. In addition, the soma has two extensions: the *dendrites* and the *axon*. The dendrites consist of a branching tree of fibers, which receive the information from other neurons, input signals, and send them to the soma. Inversely, the axon receives the information from the soma, output signal, and transmits them to other neurons. The axon ends in *synapse*, which makes contact with other neurons. The synapse is a storage area for knowledge and memories from past experience. (Zupan and Gasteiger, 1993)

The neural information processing can be concluded into two operations: *synaptic* and *somatic* operations (Gupta et al., 2003). Synaptic operation assigns a

relative weight to each incoming signal according to the past experience stored in the synapse. The somatic operation provides various mathematical operations such as aggregation, thresholding, nonlinear activation and dynamic processing to the synaptic inputs. If the weighted aggregation of the neural inputs exceeds a threshold, the soma will produce an output signal to its axon.

3.2.2 Artificial Neurons

In 1958, Rosenblatt introduced the mechanism of the single artificial neuron called *Perceptron*. Basic findings from the biological neuron operation enabled early researchers to model the operation of simple artificial neurons (Figure 3.4).

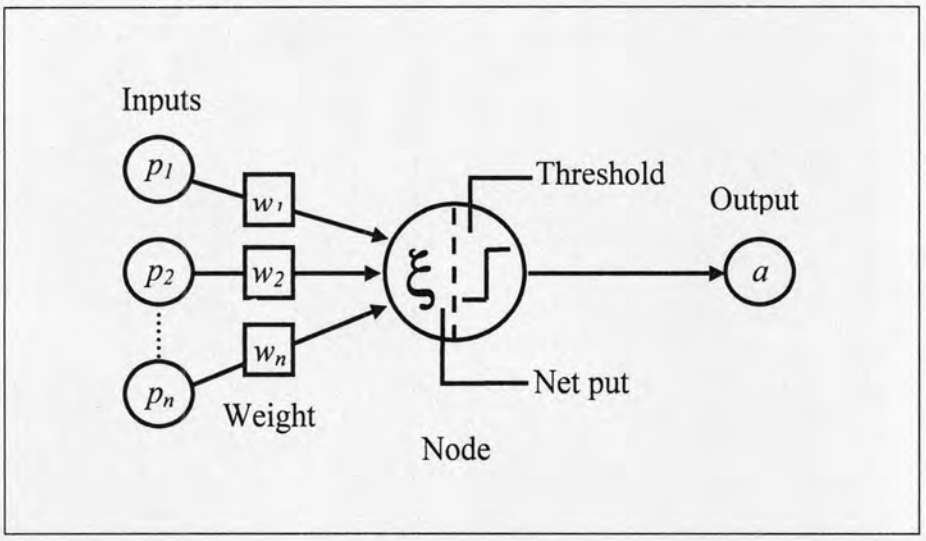

Figure 3.4 An artificial neuron

An artificial processing neuron receives inputs as stimuli from the environment, combines them in a special way to form a *net input* (ξ), passes that over through a linear threshold gate, and transmits the *output* (a) forward to another neuron. Only when a net input exceeds (i.e., is stronger than) the neuron's *threshold* limit (also called *bias*, b), will the neuron fire (i.e., becomes activated). Commonly, linear neuron dynamics are assumed for calculating a net input. The net input is computed as the inner (dot) product of the *input* signals (p) impinging on the neuron and their *strengths* (w). For n signal, the perceptron neuron operation is expressed as

$$y = \begin{cases} 1, & if\ \xi \le b \\ 0, & if\ \xi < b \end{cases} \tag{3.3}$$

where 1 indicating 'on' and *0* indicating 'off', or class A and B, respectively, in solving classification problem.

3.2.3 NN Model

NNs have not standard mathematical notations and architectural representations. To prevent confusion in using them, this thesis will use the notation as the following:

Scalars – small *italic* letters: *a*, *b*, *c*

Vector – small **bold** letters: **a**, **b**, **c**

Matrices – capital **BOLD** letters: **A**, **B**, **C**

3.2.3.1 Single–Input Neuron

A simplified model of an artificial neuron model shows in Figure 3.4. There are four basic components of the model as follow: *weights*(w), *threshold*(b), *summation function*$(\sum)$, and *activation function*(f), which are described in the following. (Zupan and Gasteiger, 1993)

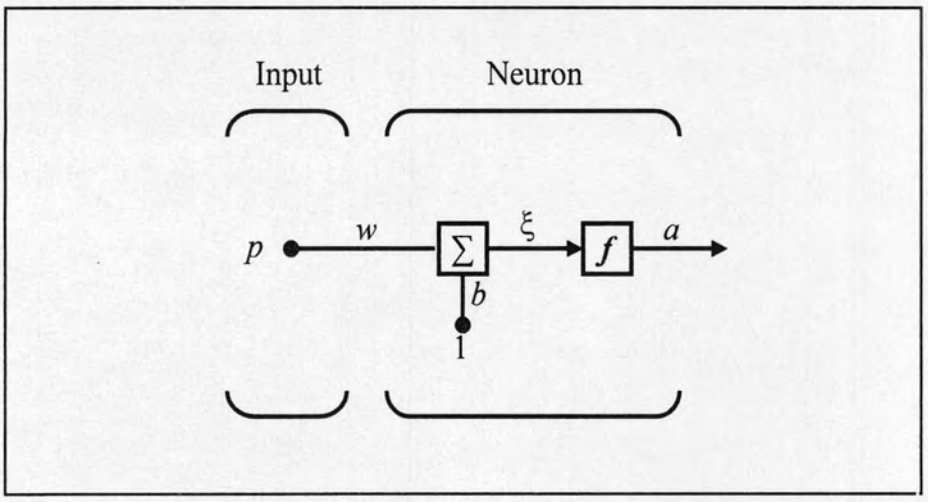

Figure 3.5 An artificial single–input neuron model

- *Weight* (w)

A neuron usually receives the input signals which are labeled p and the corresponding weight w. Weight is adaptive coefficient within the network that determines the intensity of the input signal. Weight may be positive (excitatory) or negative (inhibitory) values which can be different value at given moment.

- *Threshold or bias* (b)

Threshold or bias is a quantity added to (or subtracted from) the weighted sum of inputs into a neuron, which forms the neuron's *collective effect* or *net input*. The net input (or bias) is proportional to the amount that the incoming neural activations must exceed in order for a neuron to fire.

Both the weight (w) and threshold (b) are adjustable scalar parameters of the neuron. Typically, the w and b parameters will be adjusted by some learning rule.

- *Summation function* $(\sum)$

The neural model consists of two steps in obtaining the output signals from the input signals. The first step is to compute the *collective effect* or *net input* (ξ). The simple summation function is a linear-basis function as shows below.

$$\xi = wp + b \tag{3.4}$$

The summation function can be more complex than the simple input and weight sum of product. The input and weight can be combined in many ways such as radial-basis function and elliptic-basis function.

- *Transfer or activation function* (f)

The next step, the net input is transformed to an output (a) through transfer function.

$$a = f(\xi) = f(wp + b) \tag{3.5}$$

The transfer function may be a linear or a nonlinear function. Typically, the

transfer function is nonlinear, linear function is limited because the output is simply proportional to the input. There are several functions to use for transform net input as show in Table 3.1.

Table 3.1 Sample of transfer function.

Name	**Input/Output Relation**	**Icon**	**MATLAB Function**
Hard limit	$a=0 \quad \xi<0$ $a=1 \quad \xi\geq 0$		hardlim
Linear	$a=\xi$		purlin
Positive Linear	$a=0 \quad \xi<0$ $a=\xi \quad \xi\geq 0$		poslin
Log-Sigmoid	$a=\frac{1}{1+e^{-\xi}}$		logsig
Hyperbolic Tangent Sigmoid	$a=\frac{e^{\xi}-e^{-\xi}}{e^{\xi}+e^{-\xi}}$		tansig
Radial Basis	$a=e^{-2\xi}$		radbas

3.2.3.2 Multi–Inputs Neuron

Typically, a neuron has more than one input. The neuron with R inputs is shown in Figure 3.6. The input vector $\mathbf{p}$:

$$\mathbf{p}=\begin{bmatrix} p_1 & p_2 & \cdots & p_R \end{bmatrix} \tag{3.6}$$

and the corresponding weight matrix $\mathbf{W}$:

$$\mathbf{W} = \begin{bmatrix} w_{1,1} \\ w_{1,2} \\ \vdots \\ w_{1,R} \end{bmatrix} \tag{3.7}$$

The net input ξ and out put a is shown below.

$$\xi = w_{1,1}\, p_1 + w_{1,2}\, p_2 + \ldots + w_{1,R}\, p_R + b = \mathbf{Wp} + b \tag{3.8}$$

$$a = f(\mathbf{Wp} + b) \tag{3.9}$$

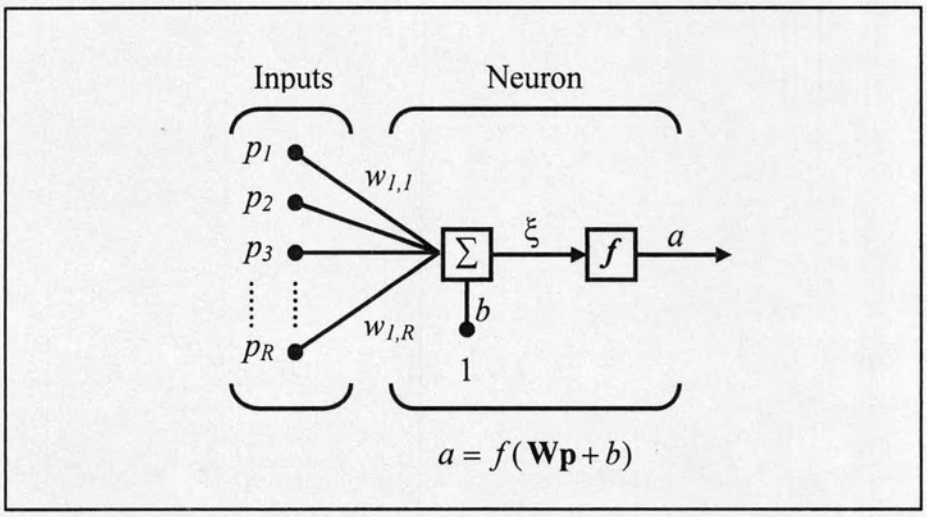

Figure 3.6 A multi–input neuron

Figure 3.7 shows the abbreviated notation of a multi–input neuron. The input vector **p** is represented by the solid vertical bar. These inputs go to the weight matrix **W**. The net input ξ, the sum of the threshold b and the product **Wp**, is transformed by the transfer function f to the output a. The dimensions of variable are displayed below the variable.

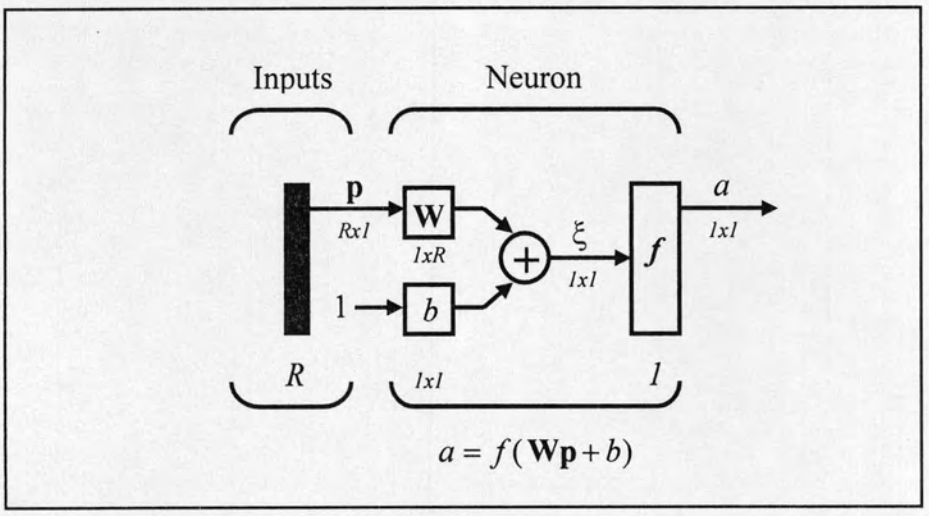

Figure 3.7 The abbreviated notation of a multiple–input neuron

3.2.3.3 Single–Layer Neurons

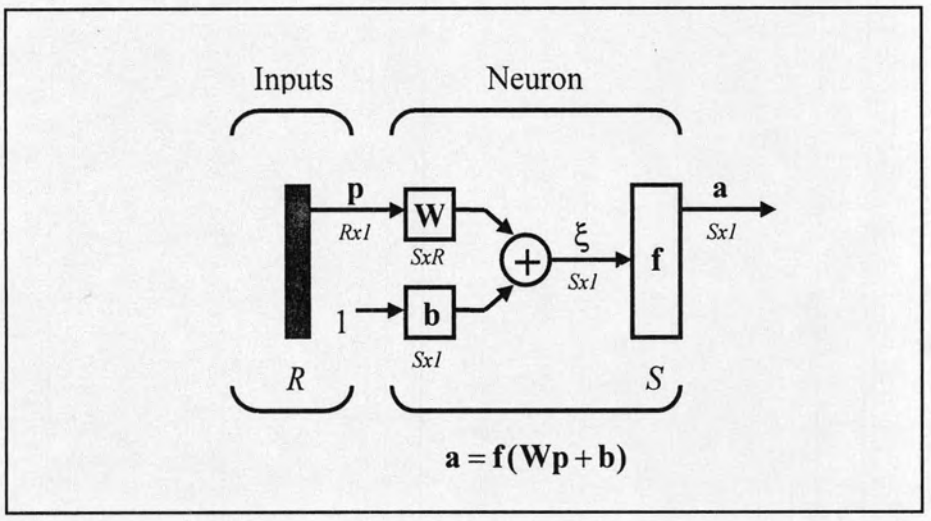

Figure 3.8 A layer of S neurons

A single–layer network of S neurons is shown in Figure 3.8. Each of the R inputs is connected to each of the neurons and **W** has a $S \times R$ matrix as shows below.

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix} \tag{3.10}$$

The neuron has the threshold weight vector **b** which is vectors of length *S*.

$$\mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_S \end{bmatrix} \tag{3.11}$$

The net input ξ and output **a** show below.

$$\xi = \mathbf{Wp} + \mathbf{b} \tag{3.12}$$

$$\mathbf{a} = \mathbf{f}(\mathbf{Wp} + \mathbf{b}) \tag{3.13}$$

3.2.3.4 Multi–Layers Neurons

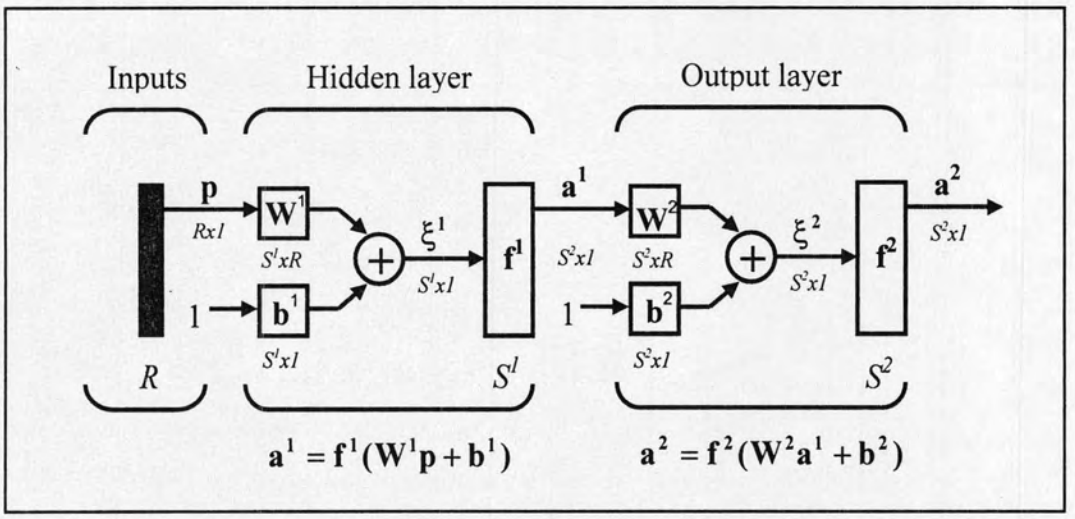

Figure 3.9 Two layers network

Figure 3.9 shows the two layers network. There are R inputs, S^1 neurons in the first layer, S^2 neurons in the second layer; different layers can have different numbers of neurons. A layer whose output is the network output is called an *output layer*. The other layers are called *hidden layer*s. The network shown above has an output layer (2nd layer) and one hidden layer (1st layer).

3.2.4 Multilayer Feedforward NNs

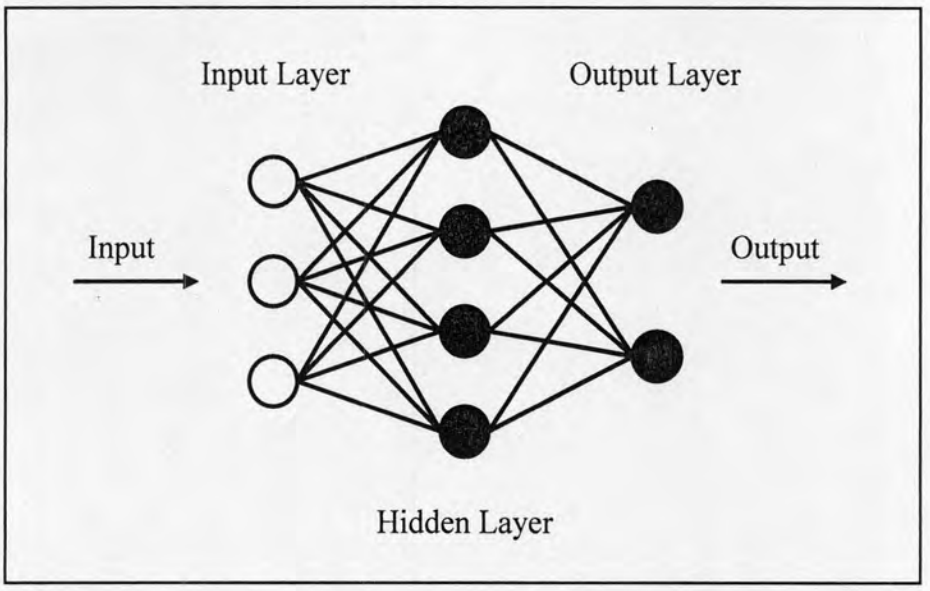

Figure 3.10 A multilayer feedforward NN

The multilayer feedforward NNs are the most popular and most widely used models in many practical applications. The networks consist of three or more layers of nodes: *an input layer* with nodes representing input variables, *an output layer* with nodes representing the dependent variables, and *one or more hidden layer*s containing nodes to help capture the nonlinearity in the data. Every node in a layer is connected with all nodes in the previous layer. These connections may have a different strength or weight. An example of feedforward network as shows in Figure 3.10 is a three–layer network or one–hidden–layer network that comprise of an input layer with three nodes, an output layer with two nodes, and a hidden layer with four nodes.

In addition, the number of layers and the number of hidden nodes in each hidden layer are design parameters. The general rule is to choose these design parameters so that the best possible model with as few parameters as possible is obtained. This is, of course, not a very useful rule, and in practice you have to experiment with different designs and compare the results, to find the most suitable NN model for the problem at hand.

For many practical applications, one or two hidden layers will suffice. The

recommendation is to start with a linear model; that is, NNs with no hidden layers, and then go over to networks with one hidden layer but with no more than five to ten neurons. As a last step you should try two hidden layers.

3.2.5 Backpropagation NNs

The *backpropagation NNs (BPNNs)* are the most widely used network. A BPNN uses a feedforward topology, supervised learning, and the back propagation learning algorithm. The basic backpropagation algorithm consists of three steps as shows in Figure 3.11.

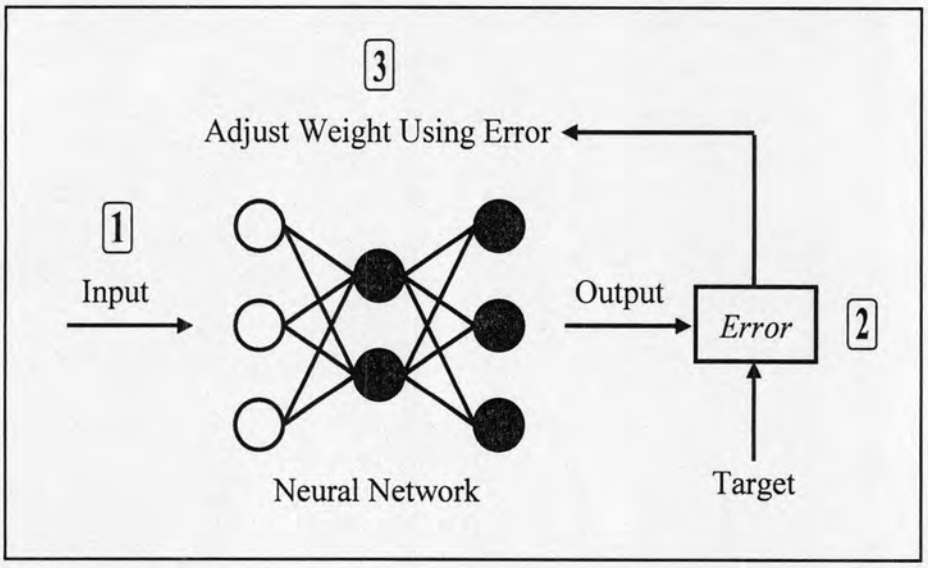

Figure 3.11 A BPNN

The input data are presented to the input layer of the network. These inputs are propagated through the network. This forward pass produces the predicted output data. The predicted outputs are subtracted from the target outputs and an error is produced. This error is then the basis for the backpropagation step, whereby the errors are propagated back through the NN by computing the contribution of each hidden node and deriving the corresponding adjustment needed to produce the correct output. The connection weights are then adjusted and the NN has just learned from an experience. After repeating this process for a sufficiently large number of training cycles the network will converge to some state where the error of the calculations is small. (see Appendix B for more details)

- *Design of BPNNs*

In general, the design parameters of a NN such as data size, hidden layer size, etc. are based on the application. Several design parameters are required with a good selection of values. Especially, five parameters should not be set too high or too low. These parameters should be carefully selected. Table 3.2 shows these parameters and their effect on both learning convergence and overall network performance (Basheer and Hajmeer, 2000).

Table 3.2 Effect of values of design parameters

Design Parameter	Too High	Too Low
Number of hidden nodes	Overfitting NN (No generalization)	Underfitting NN (NN unable to obtain the underlying rules embedded in the data)
Learning rate	Unstable NN (The optimal solution are oscillated)	Slows training
Momentum coefficient	Reduces risk of local minima Speeds up training	Slows training
Number of training cycles	Good recalling NN Bad generalization to untrained data	Produces NN that is incapable of representing the data
Size of training subset	NN with good recalling and generalization	NN with limited or bad generalization

3.2.6 NN Applications in Control System

Typically, there are two steps involved when using NNs for control system: *system identification* and *control design*. (Hagan, 1996)

- *System identification*

The aim of system identification is to build a mathematical model that captures the functions and dynamics of a real-world system. Such models are very important for analysis, simulation, prediction, monitoring, and control system design.

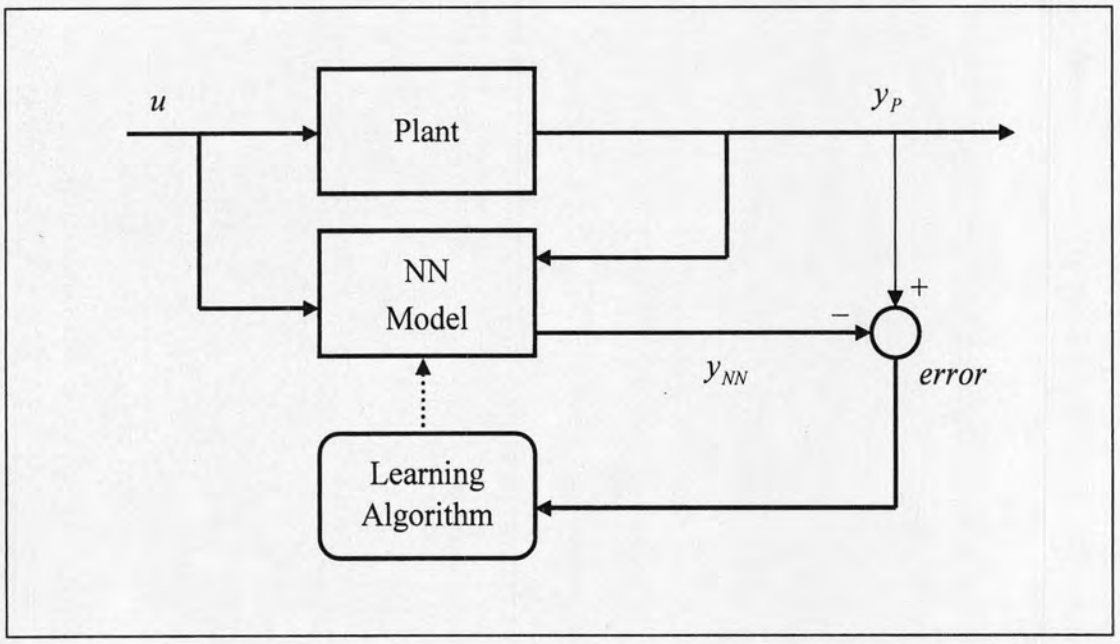

Figure 3.12 Plant identification

The first step of system identification is to train a NN to represent the forward dynamics of the system. The prediction error between the plant output and the NN output is used as the NN training signal. The process of system identification is represented by the block diagram in Figure 3.12.

- *Control design*

In the system identification, a NN model of the system is developed. In the control design, the NN model of the system is used to design the controller. There are several

control architectures, such as model reference adaptive control, MPC, and internal model control. Because of this work focus on MPC which has an effect on this section only to present MPC controller.

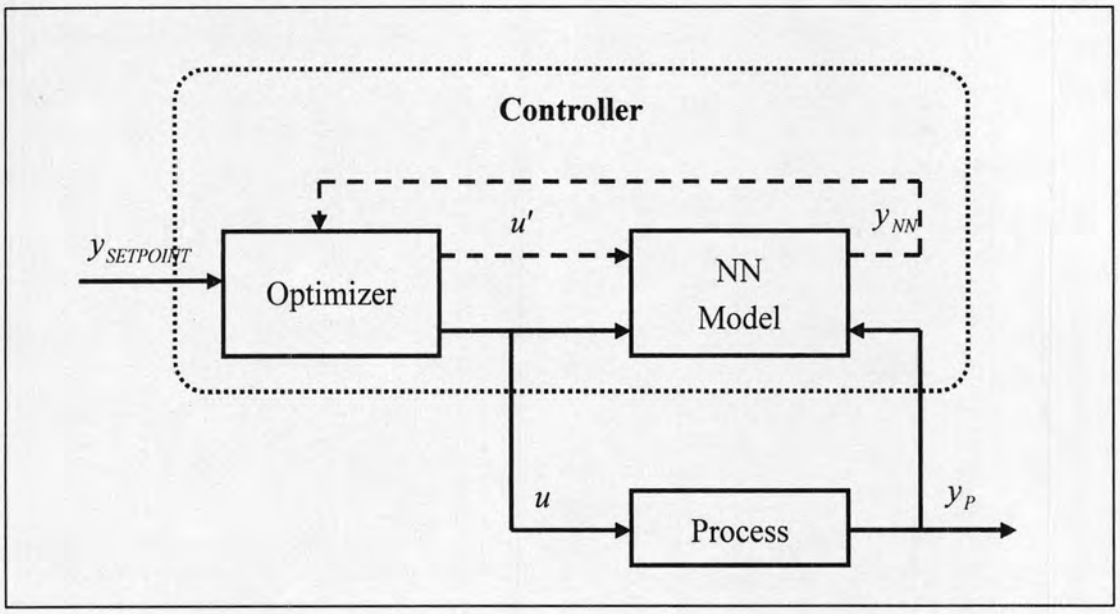

Figure 3.13 NN MPC

The MPC method is based on the receding horizon technique. The NN model predicts the plant response over a specified time horizon. The predictions are used by an optimization program to determine the control signal that maximize or minimizes the objective over the specified horizon.

Figure 3.13 illustrates the NN MPC process. The controller consists of the NN plant model and the optimizer block. The optimizer determines the values of u' that maximize or minimizes the objective, and then the optimal u is input to the plant.

3.3 Model Predictive Control

In general, the MPC problem is formulated as solving on–line a finite horizon open–loop optimal control problem subject to system dynamics and constraints involving states and controls. Figure 3.14 shows the basic concept of MPC.

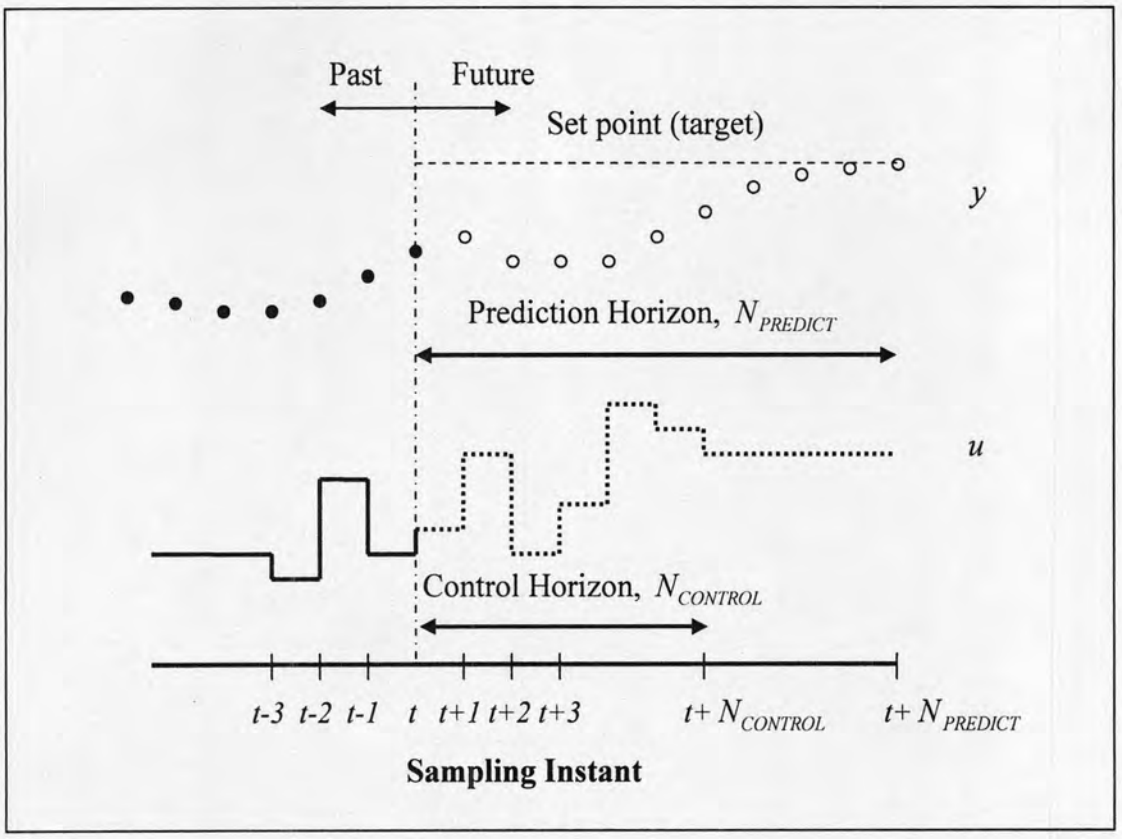

Figure 3.14 Basic concept for MPC

Based on measurements obtained at time t, the controller predicts the future dynamic behavior of the system over a prediction horizon $N_{PREDICT}$ and determines the input such that a predetermined open–loop performance objective functional is optimized. If there were no disturbances and no model-plant mismatch, and if the optimization problem could be solved for infinite horizons, then one could apply the input function found at time $t = 0$ to the system for all times $t \geq 0$. However, this is not possible in general. Due to disturbances and model-plant mismatch, the true system behavior is different from the predicted behavior. In order to incorporate some feedback mechanism, the open-loop manipulated input function obtained will be implemented only until the next measurement becomes available. The time difference between the measurements can vary, however often it is assumed to fix, i.e. the measurement will take place every d sampling time-units. Using the new measurement at time $t + d$, the whole procedure – prediction and optimization – is repeated to find a new input function with the control and prediction horizons moving forward. (Seborg et al., 1989)