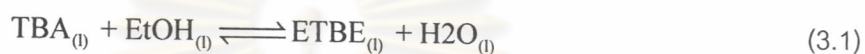


CHAPTER III

THEORY

3.1 The ETBE Hybrid Etherification Process

The reaction system of the pervaporative membrane reactor in this research is an etherification reaction between ethanol (EtOH) and *tert*-butyl alcohol (TBA). The reactions taking place in the reactor can be summarized as follows:



The major side reaction of this system is the production of isobutene (IB) gas. Although, IB can simultaneously react with EtOH to form ETBE, the low operating pressure in this study results in just a small amount of dissolved IB in the liquid mixture. Hence, the reverse reaction in Eq. (3.2) and the reaction in Eq. (3.3) can be neglected.

The rate models for the reactions in Eq. (3.1) and Eq. (3.2) and the kinetic parameters can be expressed as follows:

$$r_1 = k_1 \frac{a_{\text{TBA}} a_{\text{EtOH}} - a_{\text{ETBE}} a_{\text{H}_2\text{O}} / K_1}{1 + K_w a_{\text{H}_2\text{O}}} \quad (3.4)$$

$$r_2 = k_2 \frac{a_{\text{TBA}}}{1 + K_w a_{\text{H}_2\text{O}}} \quad (3.5)$$

where

$$k_1 = \exp\left(3.55 - \frac{2286}{T}\right) \quad (3.6)$$

$$k_2 = \exp\left(36.57 - \frac{13653}{T}\right) \quad (3.7)$$

$$K_w = \exp\left(-16.16 + \frac{6636}{T}\right) \quad (3.8)$$

$$K_1 = \exp\left(1140 - \frac{14580}{T} + 232.9\ln T + 1.087T - 1.114 \times 10^{-3}T^2 + 5.538 \times 10^{-7}T^3\right) \quad (3.9)$$

k_1 and k_2 are the reaction rate constants, while K_w and K_1 are the water inhibition parameter and the equilibrium constant, respectively.

The activities can be calculated from the following relation:

$$a_i = \gamma_i x_i \quad (3.10)$$

Where, x_i are the mole fractions of species i in the liquid mixture and γ_i are the activity coefficients which can be estimated by the UNIFAC method.

The permeation rate of species i through the PVA membrane (PERVAP 2201 from Sulzer Chemtech GmbH-Membrane Systems) can be expressed as follows:

$$N_i = AP_i a_i \quad (3.11)$$

$$P_{H_2O} = \exp\left(2.07 - \frac{2441}{T}\right) \quad (3.12)$$

$$P_{EtOH} = \exp\left(3.25 - \frac{4328}{T}\right) \quad (3.13)$$

$$P_{TBA} = \exp\left(7.67 - \frac{6434}{T}\right) \quad (3.14)$$

It is noted that the permeation of ETBE is negligibly small, thus, its permeation was not included in the models (Assabamrungrat, 2003).

3.2 Neural Networks for Process Control

Artificial neural networks are the mathematical structures having a capability to learn from the demonstrated examples. Neural networks acquired their name from their similarity to the densely connected structure of the human nervous system. The neural network paradigm emerged from attempts to simulate and understand the works of the human brain which consists of the networks of about 10^{10} neurons in the brain and each neurons are randomly connected to approximately 10^4 other neurons. Presently invented neural network models are only the simplified structures and in no way similar to the high complexities of the human brain.

Neural networks typically consist of a number of interconnected processing elements or neurons. The arrangement of the inter-neuron connections and the natures of the connections determine the structure of a network. How the strengths of the connections are adjusted to achieve a desired overall behavior of the network is governed by the learning algorithms.

The applications of neural networks cover a very board area. It would take a long time to discuss all types of neural networks. Therefore, theses sections are focused on the field of process control which is used in this research.

3.2.1 Neuron Model and Network Architectures

This section describes how a neural network calculates its output from the basic units of neural network, to the complex network architectures which are the combination of those simple basic elements.

The basic components of neural networks are called "neurons", and the most basic type is the single-input neuron.

3.2.1.1 Single-Input Neuron

A configuration of single-input neuron is shown in Figure 3.1. The neuron is composed of a summation unit (Σ) and a transfer function (f). The summation output

(n) is the product of the summation of b and the multiplication of w and p which are defined as follows:

$$n = w \cdot p + b \quad (3.15)$$

n is the net input of transfer function unit for producing a transfer function output (a) which is also the net output of neuron.

$$a = f(n) = f(w \cdot p + b) \quad (3.16)$$

where

a is the neuron output

b is the bias

f is the transfer function

n is the transfer function net input

p is the scalar input

w is the scalar weight

Note that the adjustable parameters of the neuron are w and b. The inputs are constants, and the bias input is also the constant "1".

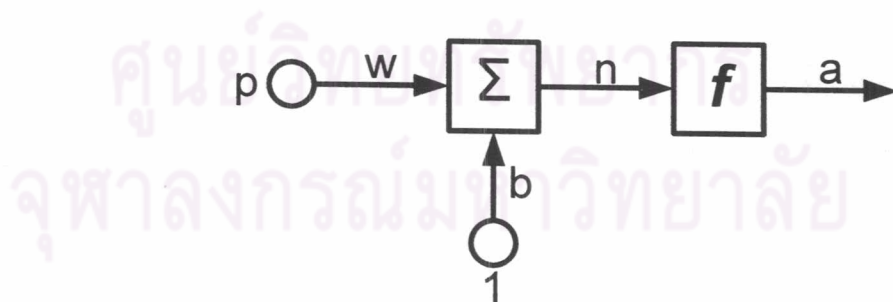


Figure 3.1: Single-Input Neuron

3.2.1.2 Transfer Functions

In this research, the linear and log-sigmoid transfer functions are utilized. The linear transfer function “purelin” is shown below:

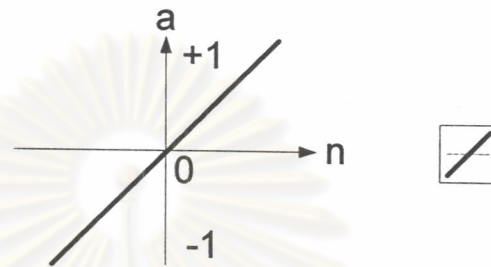


Figure 3.2: Linear Transfer Function

The linear transfer functions are utilized in the output layer for outputs expansion purpose, the calculation can be expressed as follows:

$$a = \text{purelin}(n) = n \quad (3.17)$$

According to the expression above, the output values are the same as the corresponding input values but the expansions can be obtained from the adjustable weights (w).

The log-sigmoid transfer function is shown below:

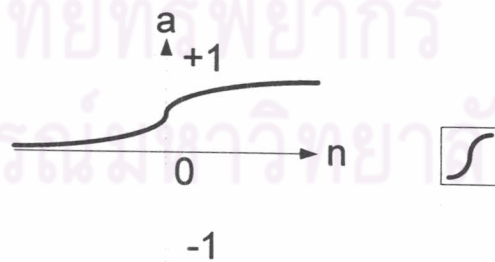


Figure 3.3: Log-Sigmoid Transfer Function

The log-sigmoid transfer functions are utilized in the hidden layers for the nonlinear behavior representation purpose, the calculation can be expressed as follows:

$$a = \text{logsig}(n) = \frac{1}{1 + e^{-n}} \quad (3.18)$$

According to the expression above, the transfer function takes the input and squashes the output into the limited range of 0 to 1; this is the reason why the linear transfer functions are required in the output layer for the outputs expansion purpose.

3.2.1.3 Multiple-Input Neuron

A neuron generally has more than one input. The configuration of a multiple-input neuron with R inputs is shown in Figure 3.4.

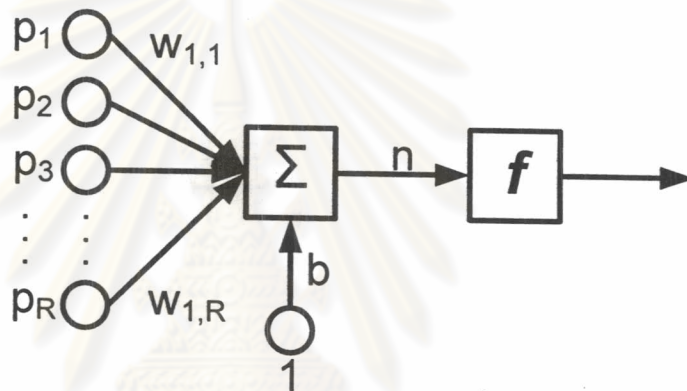


Figure 3.4: Multiple-Input Neuron

The input matrix p is composed of the individual inputs p_1, p_2, \dots, p_R which are weighted by corresponding weights $w_{1,1}, w_{1,2}, \dots, w_{1,R}$ of the weight matrix w respectively.

The net transfer function input n can be expressed as follows:

$$n = w_{1,1}p_1 + w_{1,2}p_2 + \dots + w_{1,R}p_R + b \quad (3.19)$$

Or otherwise,

$$n = w_{1 \times R} * p_{R \times 1} + b \quad (3.20)$$

Where

p is the column vector of R inputs

$$p = [[p_1, p_2, \dots, p_R]^T]_{R \times 1} \quad (3.21)$$

w is the row vector of R weights

$$w = [w_{1,1}, w_{1,2}, \dots, w_{1,R}]_{1 \times R} \quad (3.22)$$

For the indices of the elements in weight matrix, the first index indicates the particular neuron destination for that weight and the second index indicates the source of input signal to the neuron. Thus, the indices in $w_{i,j}$ indicate that this weight is connected to the i^{th} neuron from the j^{th} source.

The neuron outputs can be expressed as:

$$a = f(n) = f(w_{1 \times R} * p_{R \times 1} + b) \quad (3.23)$$

The architectures used in this research have more than one layer of neurons, each layer has more than one neuron and each neuron has more than one input. It would be obscured to introduce the pure applications without the introduction of simple structures. Thus, a single-layer network will be introduced in the next section. And after that, a multiple-layers network will be introduced step-by-step.

3.2.1.4 A Layer of Neurons

A number of neurons operating in parallel are call a "layer". A single-layer network of S neurons with R inputs for each neuron is shown in Figure 3.5. The layer is composed of a weight matrix, the summation units, the bias vector b , the transfer function units and the output vector a . Each element of the input vector p is connected to each neuron through the weight matrix w , each neuron has a bias b_i , a summation unit, a transfer function and an output a_i , thus, every S neurons layers have S outputs.

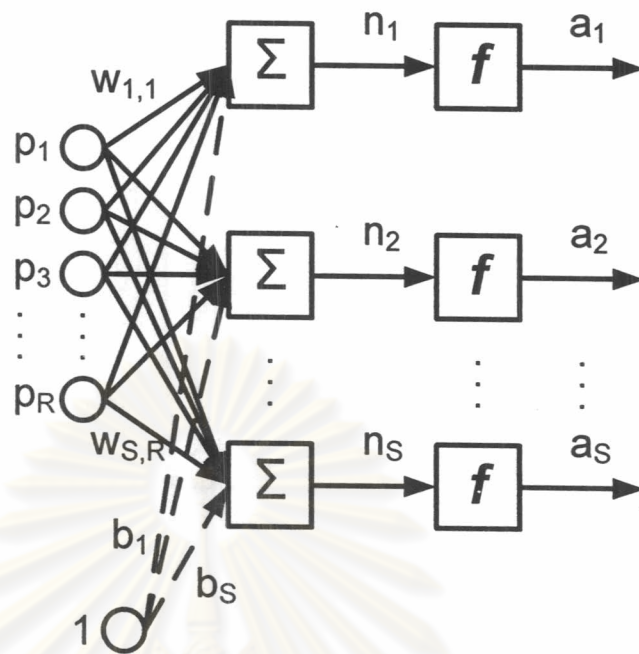


Figure 3.5: A Single-Layer Network

The single-layer network in the figure above contains a lot of details and it would be difficult to introduce a multi-layer network in this manner because it has much more details than this. Thus, the less complicated notation for each individual neuron is introduced as follows:

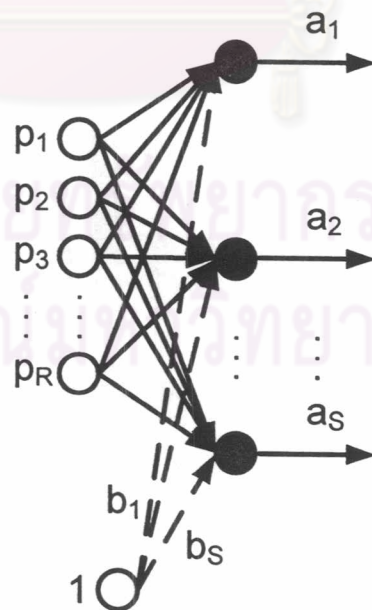

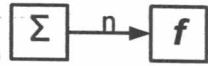


Figure 3.6: Single-Layer Network, the Simpler Configuration

Where  represents each individual neuron and equivalent to .

The input matrix p , the same as previously defined, is composed of the individual inputs p_1, p_2, \dots, p_R which are connected to each neuron through the weight matrix w which is now becoming an $S \times R$ matrix as defined below:

$$w = \begin{pmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{pmatrix}_{S \times R} \quad (3.24)$$

The indices of each element in matrix w indicate as same as previously defined. The bias b and the output a are now becoming the $S \times 1$ column vectors as follows:

$$b = [[b_1, b_2, \dots, b_S]^T]_{S \times 1} \quad (3.25)$$

$$a = [[a_1, a_2, \dots, a_S]^T]_{S \times 1} \quad (3.26)$$

The neuron output, a , can be expressed as follows:

$$a_{S \times 1} = f([w_{S \times R} * p_{R \times 1}] + b_{S \times 1}) \quad (3.27)$$

3.2.1.5 Multiple Layers of Neurons

The multiple-layers networks used in this research have two hidden layers and one output layer for each, but different number of neurons in the hidden layers. For the input layer, some authors did not refer to it as another layer because there is not any calculation in this layer, it just receives the input signal, but some authors did. However, each input is connected to every neurons in the next layer, thus, it would be much clearer to define the input neurons as the white topological nodes for each input and no matter how it is called a layer or not. The neurons in every layers else are defined as the black nodes refer to the black box calculation. The multiple-layer

network shown in Figure 3.7 has R inputs. The first layer (hidden layer) has S_1 neurons, W_1 weight matrix, b_1 bias vector and a_1 output vector which become the input for the second layer. The second layer (hidden layer) has S_2 neurons, W_2 weight matrix, b_2 bias vector and a_2 output vector which become the input for the third layer. The third layer (output layer) has S_3 neurons, W_3 weight matrix, b_3 bias vector and a_3 output vector which is the output of network.

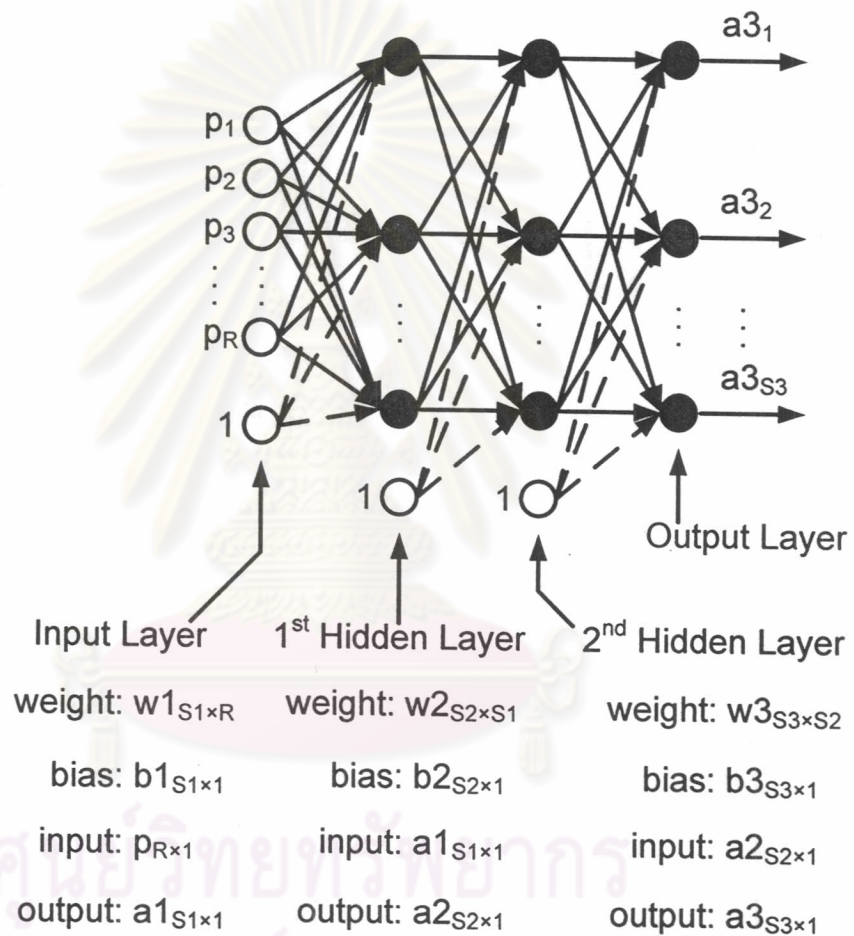


Figure 3.7: Multiple-Layer Network

Each layer can be viewed as a single-layer network as follows:

The first hidden layer,

$$a_{1_{S_1 \times 1}} = f_1 ([W_{1_{S_1 \times R}} * p_{R \times 1}] + b_{1_{S_1 \times 1}}) \quad (3.28)$$

The second hidden layer,

$$a_{2_{S2 \times 1}} = f_2 ([W_{2_{S2 \times S1}} * a_{1_{S1 \times 1}}] + b_{2_{S2 \times 1}}) \quad (3.29)$$

The output layer,

$$a_{3_{S3 \times 1}} = f_3 ([W_{3_{S3 \times S2}} * a_{2_{S2 \times 1}}] + b_{3_{S3 \times 1}}) \quad (3.30)$$

And the net output can be written in term of the net input as follows:

$$a_{3_{S3 \times 1}} = f_3 ([W_{3_{S3 \times S2}} * f_2 ([W_{2_{S2 \times S1}} * f_1 ([W_{1_{S1 \times R}} * p_{R \times 1}}] + b_{1_{S1 \times 1}})] + b_{2_{S2 \times 1}})] + b_{3_{S3 \times 1}}) \quad (3.31)$$

The equation above looks complicated but all parts of it can be replaced by the simple calculation as previously defined. Now, the neural network can be seen clearly through the basic calculation units which are only the simple mathematics. The neural networks are the black boxes but not obscured anymore.

3.2.2 Neural Networks Training

To achieve the goal, the weights and biases of neural networks require an adjustment; the procedure for this is called "training" or "learning". This section will introduce a training procedure called "backpropagation", which is based on error gradient descent, and then, the several more efficient algorithms including the Levenberg-Marquardt algorithm which is used in this research will be introduced. All of these algorithms require the error between network output and plant output to adjust the weights and biases as shown in Figure 3.8 below:

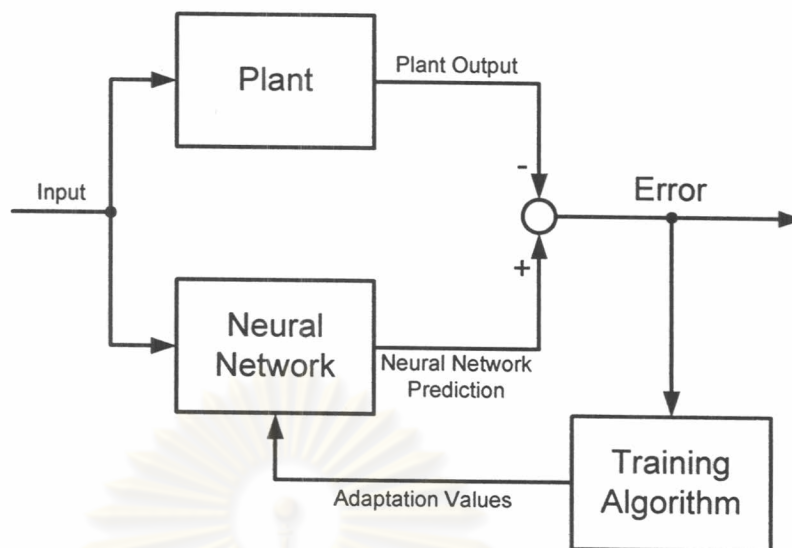


Figure 3.8: The Training Algorithms Configuration

3.2.2.1 Neural Network Preparations

There are several steps required before training as follows:

1. The data set collection, either from the available model simulation or the actual on-line process data, this research uses the available model simulation (white box model). The data are normally divided into various sets as follows:

- The initial training data set, which is utilized to train the network initially and span the operating region of the model.
- The cross validation data set, which is utilized to access the generalization capability of the network. This set is normally the similar quality to the training data set. Training can also be switched between these two sets to improve the identification.
- The testing data set, which is unseen for the network, used for final validation of the trained neural network.

To obtain an adequate model, the training input signal must be able to represent the types of signal anticipated during the normal operation, must be a

particular frequency of excitation and appropriate magnitude of excitation especially for nonlinear identification.

2. Neural network creation, define the structure, number of layers and number of neurons in each layer. In some cases, the users may define the special types of connection. This research uses the standard type which each neuron is connected to every neuron in the next layer.

3. Initialization, the neural network requires the initial values of weights and biases to begin the learning.

4. Simulation, the neural network can not be trained without the first run to generate the first value of error for the training algorithm.

After these steps, the network is ready for training.

3.2.2.2 Backpropagation

The backpropagation is utilized for adjusting the weights and biases of the neural networks in order to minimize the sum-square error. This is done by changing the values of the networks weights and biases in the direction of steepest descent with respect to error.

Derivatives of error (called delta vectors) are calculated for the network output layer, and then propagated back through the network until the delta vectors are available for each hidden layer.

The change to be made in a layer's weight "W" and bias "b" are calculated by that layer's delta vector "D" and the layer's input vector "p", according to the rules:

$$\Delta W = lr * D * p^T \quad (3.32)$$

$$\Delta b = lr * D \quad (3.33)$$

Where, "lr" is the learning rate.

Too large learning rate cause an unstable learning while too small learning rate cause a long training time. The learning rate is constant through out the whole period of training for the normal backpropagation algorithm.

Each pass through all of the training input and target vectors is called an epoch. The backpropagation training algorithm requires the initial weight and bias, input vector, target vector which is the plant output for this research and the training parameters as follows:

- the maximum number of epochs to train
- the desired error goal
- the learning rate

The maximum number of epochs and the error goal are the criteria for stop training if at least one of these is reached.

3.2.2.3 Local and Global Minima

The nonlinear networks may have more than one local minimum. These minima are not equal, just as the bottoms of several valleys in a mountain range may not be all the same height above the sea level.

The network may find the lowest valley of error, the global minimum. But the network also may get stuck by rolling into a higher valley of error, or local minimum, depend on the initial conditions.

3.2.2.4 Learning with Momentum

Momentum allows the network to ignore small features in the error surface. Without momentum, a network may get stuck in a shallow local minimum. With momentum, a network can slide through such a minimum.

Momentum can be added to the backpropagation learning by making weight changes equal to the sum of a fraction of the last weight change and the new change suggested by the backpropagation rule. The magnitude of the effect that a last weight change is allowed to have is mediated by a momentum constant, mc . When the momentum constant is 0, a weight change is based solely on the gradient. When the momentum constant is 1, the new weight change is set to equal the last weight change and the gradient is totally ignored. Typically, the momentum constant is set to 0.95.

The backpropagation with momentum can be expressed as follows:

$$\Delta W_{ij} = mc * \Delta W_{ij} + (1-mc) * lr * D_i * p_j \quad (3.34)$$

The weight change matrix, ΔW , and the bias change vector, Δb , must be initialized to the zero matrices of the same sizes as W and b , respectively. Then weight and bias changes of a layer can be found from the layer's current input p , the delta vector D the learning rate lr and the momentum constant mc . These weight and bias change can then be used to update the weights and biases as follows:

$$W = W + \Delta W \quad (3.35)$$

$$b = b + \Delta b \quad (3.36)$$

The new weights and biases will be rejected if they result in too large increase in error, this stop the momentum from pushing the parameters of a network out of a deep valley in the error surface. The rejection of weights occurs if the ratio of new error to previous error exceeds a maximum error ratio. The maximum error ratio can be any values greater or equal to 1, but it is typically set to 1.04.

3.2.2.5 The Adaptive Learning Rate

The adaptive learning rate algorithm is to increase the learning rate (typically multiply by 1.05) if the new error is less than the previous error and to decrease the learning rate (typically multiply by 0.7) if the new error exceeds the previous error by more than the predefined ratio (typically 1.04).

This procedure increase the learning rate only if a larger rate could result in stable learning. When the learning rate is too high to guarantee a decrease in error, it get decreased until stable learning resumes.

3.2.2.6 Levenberg-Marquardt Algorithm

The gradient descent is simply the technique where parameters, such as weights and biases, are moved in the opposite direction to the error gradient. Each step down, the gradient results in smaller errors until an error minimum is reached.

The network can get a better performance by using an approximation of Newton's method called Levenberg-Marquardt. This technique is more powerful than the gradient descent, but also requires more memory.

The Levenberg-Marquardt update rule is:

$$\Delta W = (J^T * J + \mu * I)^{-1} * J^T * e \quad (3.37)$$

Where J is the Jacobian matrix of derivatives of each error to each weight, μ is a scalar and e is an error vector. If the scalar μ is very large, the above expression approximates the gradient descent, while if it is small; the above expression becomes the Gauss-Newton method. The Gauss-Newton method is faster and more accurate near an error minimum, so the aim is to shift towards the Gauss-Newton method as quickly as possible. Thus, μ is decreased after each successful step, and increased only when a step increases the error.

This algorithm requires the training parameters as follows:

- the maximum number of epochs
- the error goal
- the minimum error gradient
- the initial value of μ

- the multiplier for increasing μ
- the multiplier for decreasing μ
- the maximum value for μ

The training continues until the error goal is reached, the minimum error gradient occurs, the maximum value of μ occurs, or the maximum number of epochs has finished.

3.2.2.7 Underfitting and Overfitting

One of the problems occurred when training the neural network is called overfitting. The error on the training set is driven to a very small value, but when the new data set is presented to the network, the error is large. The network has memorized the training examples, but it has not learned to generalize to the new situations.

The training algorithms such as Backpropagation, Backpropagation with Adaptive Learning Rate and Backpropagation with Levenberg-Marquardt Approximation are sensitive to the number of neurons in hidden layers. While, in general, the more neurons in hidden layers, the better data the network can fit, if far too many neurons are used, overfitting can occur. And as general, if too few neurons are examined, underfitting can occur.

3.2.2.8 Neural Network Performance Improvement

A method used in this research for the network improvement is call "early stopping". In this technique, there are three sets of data used as previously introduced in section 3.2.2.1. The first data set is the training set, used for initially computing the gradient based on the training data set and updating the network weights and biases which are randomly initialized. After the initial training, the trained weights and biases are obtained. If the training is successful, these weights and biases would make the network fit well with the training data set and these become the initial weights and biases for the validation. The trick is in the validation, during the normal training based

on the validation data set, the performance of network on the training data set is monitored. During the validation, the error based on the validation data set will come down from the beginning through the end. When the overfitting begins, the error based on the training data set will begin to rise and this is an additional criterion for stopping the validation, this is the reason why it is called “early stopping”. This technique can improve the network by generalizing for two different data sets, not too fit for the only one set which makes the performance go worse for the generalization.

Other methods are the data preprocessing and postprocessing. The input variables may have various ranges, the most wide range variable will have the largest error, the network will mostly focus on this variable and other variables are tend to be ignored. To prevent this, the data preprocessing is utilized for normalizing the variables into the same range to equalize the errors of every variables. And the data postprocessing is to convert the normalized values back to the actual values for using.

3.2.3 Neural Network Control Strategies

There are many control strategies based on neural network, each of them have their own advantages and disadvantages. For this research, the Nonlinear Internal Model Control (NIMC) was used. The NIMC strategy consists of a Neural Network Controller (NNC), a Neural Network Model (NNM) and a robustness filter (F) as shown in Figure 3.9.

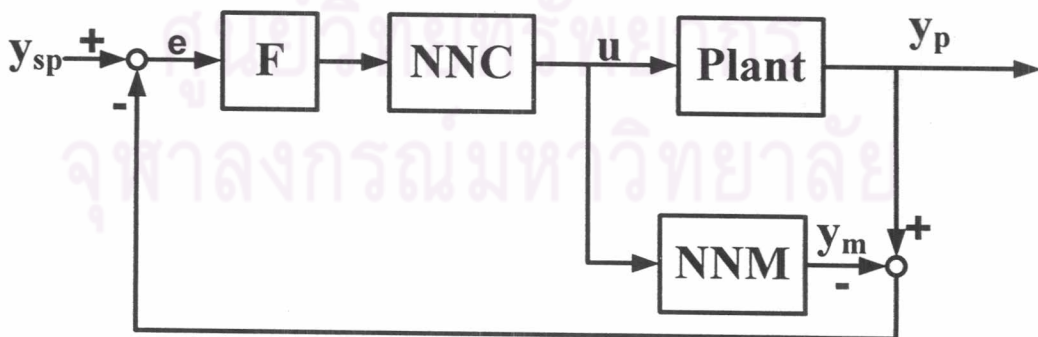


Figure 3.9: NIMC Topology

Where, e is the IMC error

u is the manipulated variable

y_p is the plant output

y_m is the neural model output

and y_{sp} is the controlled variable set point

There are two neural networks, NNC and NNM. NNC represent the inverse model of the plant if the inverse exists and NNM represent the forward model of the plant. The error between the plant output and the NNM output is fed back and subtracted from the controlled variable set point before fed into the robustness filter. The net input for the filter is called "IMC error" as following defined:

$$e = y_{sp} - (y_p - y_m) \quad (3.38)$$

In case of perfect NNM prediction, y_m is equal to y_p and e is consequentially equal to y_{sp} , the controller NNC will behave as a Direct Inverse Controller. Thus the Direct Inverse Control is the subset of the IMC.

In case of error between y_p and y_m , e is not equal to y_{sp} which is typically occurred from the disturbance changes, the plant-model mismatches and the noisy measurement signals. The robustness filter, typically the first order exponential filter, is used for catering these problems to guarantee the closed loop stability.

For other neural network control strategies, they are not suitable for this work. For example, the Model Predictive Control (MPC) requires too many calculation steps ahead for the optimization algorithm and a very accurate plant model which is obtained from the training with all of the possible plant inputs, all these inputs are not available in some cases. The Direct Inverse Control (DIC) is very sensitive to noise and disturbances and it is just a subset in the IMC as described previously. The Model Reference Adaptive Control (MRAC) can not guarantee the stability of the controlled

variables. And the Adaptive Inverse Control requires the existence of a stable plant inverse which is not available in some cases.

More details about the various neural network control strategies can be found in (W. T. Miller, R. S. Sutton and P. J. Werbos, 1990; D. A. White and D. A. Sofge, 1992; K. J. Hunt, D. Sbarbaro, R. Zbikowski and P. J. Gawthrop, 1992; B. Widrow, D. E. Rumelhart and M. A. Lehr, 1994; M. Brown and C. Harris, 1994; A. J. N. Van Breemen and L. P. J. Veelenturf, 1996; B. Widrow and E. Walach, 1996; S. N. Balakrishnan and R. D. Weil, 1996; M. Agarwal, 1997; T. H. Kerr, 1998; C. L. LIN AND H. W. SU, 1999; S. Brückner and S. Rudolph, 2000; A. Fink, S. Töpfer and R. Isermann, 2003; E. K. Juuso, 2005) for example.



ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย