UL

On Implementation and Applications of the Adaptive-Network-Based Fuzzy Inference System

By Ong Kai Hin George

A Research Project submitted in partial fulfillment of the requirement of the Degree of Master of Science

Department of Information Engineering
The Chinese University of Hong Kong

June 1994

Acknowledgment

It is my pleasure to acknowledge my debt to the many people who have contributed to this research project. Their suggestions have been a great help in correcting and shaping the project.

The most important acknowledgment is to my supervisor, Dr. H. S. Wang, for his continuous provision of support, help, guidance, knowledge and care. Apart from the actual knowledge and experience gained through the project development, his strong desire in pursuing knowledge has also inspired me and drive me to try my best in this piece of work.

June1994

George Ong

1 Introduction

Conventionally, hard computing is concerning the precision and certainty. By contrast, the point of departure in soft computing is the fact that precision and certainty can be obtained with a high cost. In most cases, computation, reasoning and decision making can exploit the tolerance for imprecision and uncertainty.

For instances, considering the case of parking an automobile. Most people can park an automobile because its final position and orientation have not been specified clearly. If they are specified clearly, parking an automobile will be a very difficult task and not possible to be achieved by humans. The most important point is that parking an automobile is an easy task to humans if imprecision is allowed, while it is a difficult task to traditional methods (based on mathematical model) because such methods do not exploit the tolerance for imprecision[24].

The exploitation of the tolerance for imprecision and uncertainty accounts for the remarkable human ability to understand distorted speech, decipher sloppy handwriting, comprehend nuances of natural language, summarize text, recognize and classify images, drive a vehicle in dense traffic and, more generally, make rational decisions in an environment of uncertainty and imprecision. In order to exploit the tolerance for imprecision and uncertainty, soft computing uses the human mind as a role model which aims at a formalization of the cognitive processes of humans.

In the current consumer market, more and more electronic consumer products are designed based on the concept of fuzzy logic, such as cameras, microwave oven and washing machines. Compare with conventional electronic products, these new products can figure on their own what settings to use to perform their tasks optimally, these products can manifest an impressive capability to reason, make intelligent decisions and learn from experience. These products are referred to as Machine Intelligence Quotient (MIQ) products. The first MIQ product was announced by Matsushita in 1987. This was followed by the first fuzzy-logic-based washing machine which was also designed by Matsushita in 1989.

In 1990, the number of high-MIQ consumer products employing fuzzy logic increases drastically. By the way, neural network techniques were combined with fuzzy logic to be employed in a wide variety of consumer products. The main objective of applying neural network is to endorse the products with learning capability. Such neurofuzzy products are likely to become ubiquitous in the years ahead. Underlying this evolution results in an acceleration of employing the soft computing, especially fuzzy logic and neural network. The objective is to design the intelligence system which can exploit the tolerance for imprecision and uncertainty, learn from experience, and adapt to changes in the operating conditions.

At this juncture, the principal constituents of soft computing are fuzzy logic (FL), neural network (NN), probabilistic reasoning (PR). In the triumvirate of FL, NN and PR, FL is primarily concerned with imprecision, NN is concerned with learning and PR is concerned with uncertainty. The most important point is that there are substantial areas of overlap between FL, NN, and PR. Instead of competition, FL, NN and PR are complementary in general. For this reason, it will be advantageous to employ FL, NN and PR in combination rather exclusively[24].

A case in point is the growing number of so-called neurofuzzy (NF) which employed a combination of fuzzy logic and neural network techniques. Most NF products are fuzzy rule-based systems in which NN techniques are used for the purposes of learning and adaptation.

1.1 Objective

During the design of NF products, it will be more effective if a prototype can be implemented in digital computers for simulation. Such prototype can be used to investigate the performance and feasibility of the end products in advance.

The objective of the project is to build a C/C++ library for the prototype of applications. Based on the library, a number of applications have been implemented for the illustration of the usability of the library.

Among a number of neural-fuzzy models, the Adaptive Network-Based Fuzzy Inference System (ANFIS) is selected for the implementation of the library. This is because of the rich information and remarkable results of ANFIS.

2 Background

2.1 Neural Networks

About a century ago, an American psychologist, William James, published a number of facts which are related to the structures and functions of the brains. Since then, many researches had been conducted on the human neurons and neural networks. However, most of the researches were biological based.

Until 1958, Frank Rosenblatt published his famous paper which defined a neural network structure called perceptron. In the paper, the perceptron was simulated on an IBM 704 computer at the Cornell Aeronautical Laboratory. This aroused the interests and imagination of scientists in the implementation of neural networks on digital computers.

Nowadays, neural networks is a very hot topic in artificial intelligence as well as soft computing, there are many kinds of neural networks that have been implemented in computers. The implementation results show that neural networks can solve problems which are difficult to be solved by conventional computer systems. Because of this capability, neural networks have been applied in various areas, such as image recognition, digital filter, functional mapping and interpolation.

2.1.1 Topology

Basically, the components of neural network are nodes and links. The links are used to connect the nodes and each of them is associated with a weight value. The nodes are processing elements (PE), they process the incoming signals from the links and generate an output signal which will be sent to other nodes. Figure 2.1 shows a simple neural network with links and nodes.

Figure 2.2 shows a node and its operation on the incoming signals. During the transmission of a signal through a link, the signal strength is multiplied by the weight value of the link. The node sums up the signals of all the connected links, and applies a transfer function (F) on the sum. This is not the unique operation of nodes, here, we just describe one of the operations of nodes.

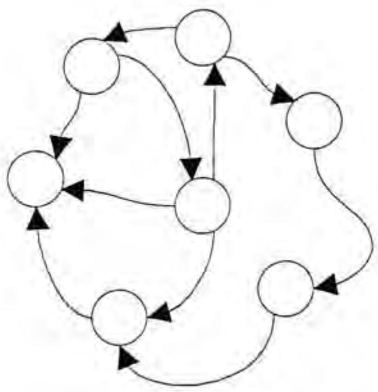


Figure 2.1 : A simple neural networks model, each node is connected to another node through a link. The arrow on the links show the directon of the signal flow.

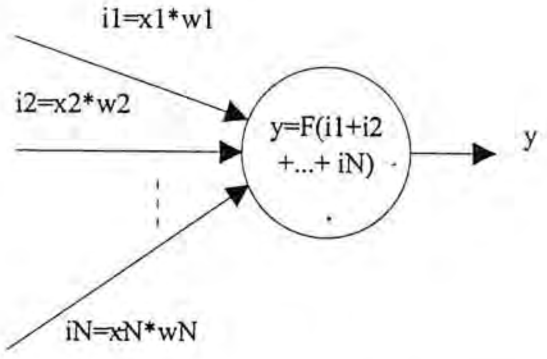


Figure 2.2 : One of the operations of a node (PE). wN is the weight associated to the link, xN is the incoming signal.

There are two types of topology in neural networks, they are feed forward networks and counter propagation networks. Both of them will be discussed subsequently.

2.1.1.1 Feed forward networks

In the feed forward networks, signals are propagated from input to output of the neural networks. They are transmitted in one direction and will not loop back to the previous nodes. Figure 2.3 shows an example of feed forward networks.

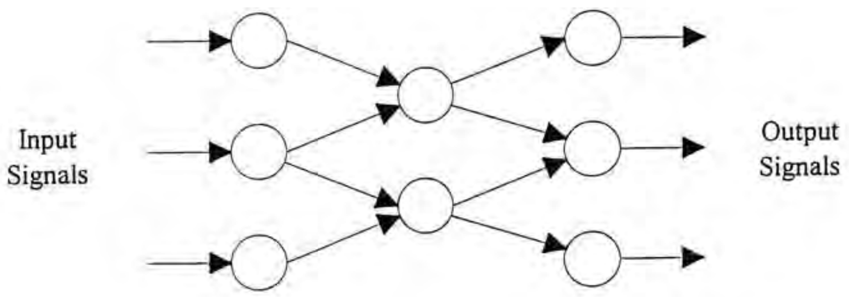


Figure 2.3 : An example of feed forward networks. There are 3 layers in the network, the layer receiving the input signals is input layer, the middle layer is hidden layer, the last layer is output layer. Signals propagate from input layer to output layer.

2.1.1.2 Counter propagation networks (recurrent networks)

In the counter propagation networks, some signals may be transmitted back to the previous nodes or itself. Figure 2.4 shows an example of counter propagation networks.

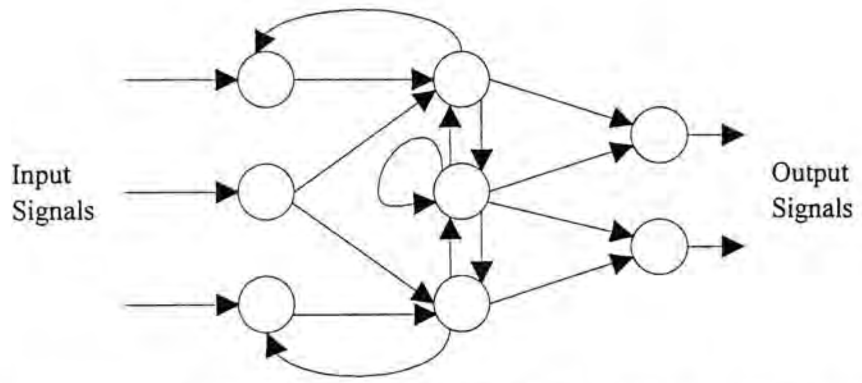


Figure 2.4 : An example of couter propagation networks. In the middle layer, some signals propagate to input layer and some signal recurrent to the node itself.

2.1.2 Neural Network Learning

One of the attractive features of neural networks is their learning capability. After training with the training data, neural networks are able to dynamically change their behaviors to produce desired outputs. There are two types of learning method in neural networks, they are supervised learning and unsupervised learning.

Supervised learning requires a teacher to teach the neural networks, the teacher should present a training sample to the neural networks, and tells the neural networks the performance error. Then the neural networks will based on the error to adjust the weight values of its links in order to minimize the error. Currently, most of the neural networks applications are based on supervised learning.

On the other hand, unsupervised learning does not need a teacher. Neural networks with unsupervised learning have self-organization capability, their learning are based on the local information, and internal control of the signals propagation. They can organize the training samples by itself, and discover the properties of the samples. Currently, unsupervised learning is still in the early stage and many researches are being conducted on the unsupervised learning algorithms.

2.2 Fuzzy Logic

One of the problems in systems design is that most of the systems are in the realm of "humanistic system", such as linguistic, social sciences and control systems. In these systems, hard mathematics does not seem to be very effective. In 1965, Prof. Lotfi Zadeh of University of California at Berkeley published his famous paper - "Fuzzy Sets".

Since then, many researches have been conducted on fuzzy logic. Nowadays, fuzzy logic has a wide range of applications. Especially in control systems, fuzzy logic plays an important role in providing robust systems which are difficult to be achieved by conventional control systems.

2.2.1 The Calculus of Fuzzy If/Then Rules

One of the important parts of fuzzy logic is the operations of fuzzy if/then rules. Since the fuzzy if/then rules constitutes a collection of concepts and methods for handling the knowledge which is in the form of if/then rules, this part of fuzzy logic is referred to as the calculus of fuzzy if/then rules (CFR)[26]. The antecedents and consequences of the if/then rules can be either fuzzy or crisp. Here is an example of fuzzy if/then rules:

If temperature is high, then pressure is low.

Instead of comparing the pressure and temperature with the crisp values, such as 10Pa or 20K, they are compared with the linguistic terms. Each linguistic term represents a fuzzy set which is defined by a membership function. Figure 2.5 shows an example of three membership functions of the variable "temperature".

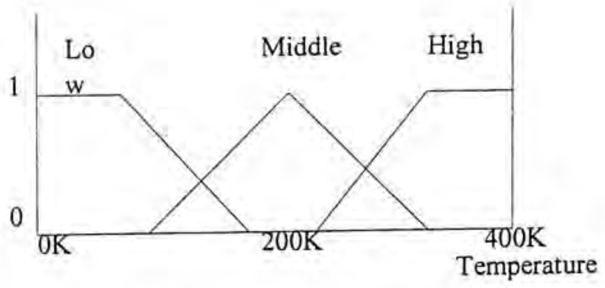


Figure 2.5 : An example of three membership functions (Low, Middle, High) of the variable temperture. In fuzzy if/then rules, the membership functions can have overlapped area.

Since the antecedent part of fuzzy if/then rules composes a fuzzy comparison, the result of the antecedent part is in the sense of degree of match which is used to determine the firing strength of the fuzzy if/then rules. Figure 2.6 shows an example of the firing strength of a fuzzy if/then rule : *if temperature is high, then pressure is* *low*. In this example, the temperature matches the membership function of "high" with a degree of 0.7, so the firing strength of the rule is 0.7.

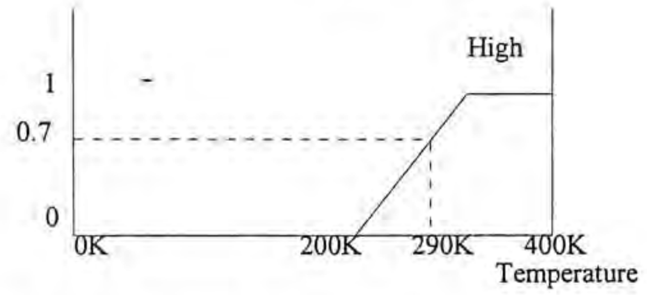


Figure 2.6 :An example of the firing strength of a fuzzy if/then rule. When the reading of temperature is 290K, based on the membership function of "high", the degree of matching of "temperature is high" is 0.7.

The matching process of the antecedent part is an imprecise matching. However, this imprecise matching form a basic of interpolation which can minimize the number of rules to describe the input and output relationship.

2.2.2 Fuzzy Inference System

Fuzzy inference system makes use of the fuzzy if/then rules for inference. Basically, it contains 4 parts[13] which are shown in figure 2.7.

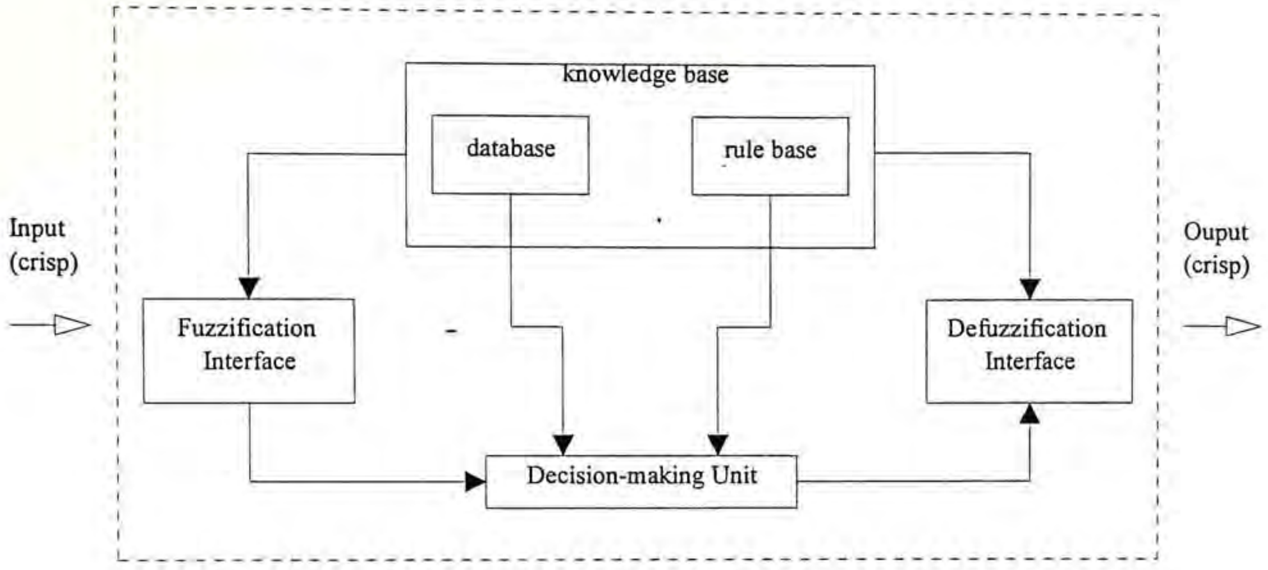


Figure 2.7 : The block diagram of Fuzzy Inference System.

1. *Knowledge Base* : Composing of a rule base and database. The rule base contains the fuzzy if/then rules. The database contains the parameters of the membership functions of the linguistic term (fuzzy sets) in the fuzzy if/then rules.
2. *Decision-making unit* : Performing the inference process on the fuzzy if/then rules.
3. *Fuzzification interface* : Transforming the crisp inputs into the degree of matching of the linguistic values.
4. *Defuzzification interface* : Transforming the fuzzy inference results into the crisp outputs.

2.2.2.1 Inference steps

There are 4 inference steps in the fuzzy inference system. In order to understand each step clearly, an example is used for illustration.[1] Consider the system with two rules as follows;

Rule 1 : if pressure is high and temperature is low, then volume is small.

Rule 2 : if pressure is middle, then volume is middle.

The membership functions of pressure, temperature and volume are defined in figure 2.8. Suppose the current reading of pressure is 70Pa, temperature is 100K.

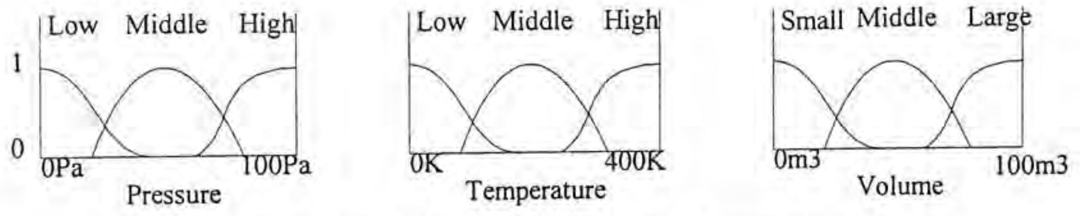


Figure 2.8 : The membership functions of Pressure, Temperature and Volume.

Step 1 : According to the premise part of the fuzzy rules, compare the input variable with the membership function to obtain the membership value of each linguistic term. This step is called Fuzzification. (Figure 2.9)

[1] *The discussed fuzzy inference system is based on the model of Takagi and Sugeno's fuzzy if/then rules. In this model, the output of each rule is a linear combination of the input variables plus a constant term. The final output is the weighted average of each rule's output.*

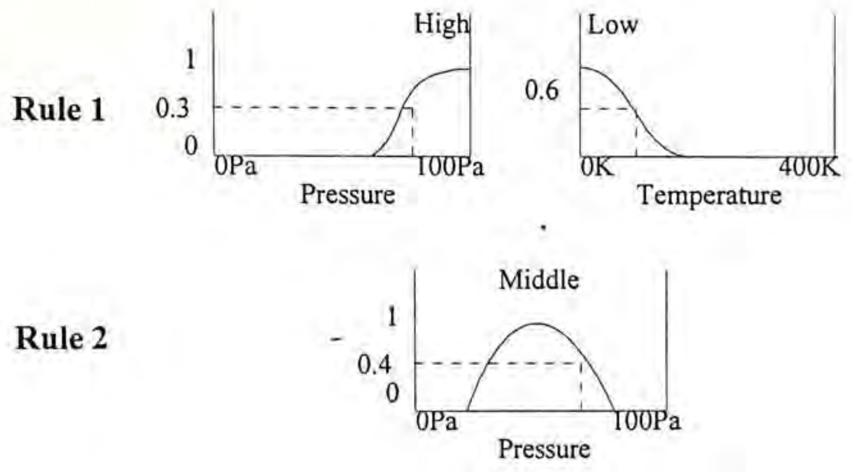


Figure 2.9 : According to each rule, transform the input crisp values into the membership values (Fuzzification).

Step 2 : To each rule, combine the membership values of the premise part to obtain the firing strength (weight) of the rule. Usually, the firing strength is the product of all the membership values in the premise part of the rule. Therefore, the firing strengths are 0.18 and 0.4 for rule 1 and 2 respectively.

Step 3 : Based on the linear combination of the input variables, generate the qualified consequent of each rule. The following are the supposed linear equations of each rule and the qualified consequent outputs.

$$Rule\ 1 : V_1 = A_1P + B_1T + C_1 \Rightarrow V_1 = A_1(70) + B_1(100) + C_1$$

$$Rule\ 2 : V_2 = A_2P + C_2 \Rightarrow V_2 = A_2(70) + C_2$$

Step 4 : Based on the firing strength of each rule to calculate the weight average of the consequent output of each rule. This step is called Defuzzification, the output of Defuzzification is a crisp value, it is the final output of the fuzzy inference system. Hence, the final result of the volume is :

$$V = \frac{0.18 \times V_1 + 0.4 \times V_2}{0.18 + 0.4}$$

2.3 Integration of Neural Networks and Fuzzy Logic

Both of neural networks and fuzzy logic are important concepts in soft computing. They both give inexact result and work in the domain space where the boundaries are not sharply defined. Many researchers tried to integrate these two concepts to obtain a new model which contains the strengths of both of these two concepts.

Currently, there are two research directions of the integration of neural networks and fuzzy logic. They are Fuzzy Neurons[7][39] and Neural Networks based Fuzzy Inference Systems[13][16][23][37], each of them will be discussed below.

2.3.1 Fuzzy Neurons

In conventional neural networks, the data is crisp value, the operations of neurons (nodes or processing elements) are crisp based. Consider the transfer function $y = f(x)$, both of variables x and y are crisp values.

Instead of processing the crisp values, fuzzy neurons process fuzzy values. The transmitted signals and the weight values are fuzzy values. The fuzzy neurons collect and process the incoming signals. One kind of the processing method is applying the fuzzy operations., such as fuzzy "AND" or fuzzy "OR", on the incoming signals. The other kind is generating the linear combination of the incoming signals and mapping the result with the membership function to get the crisp output.

Although, it was reported that the neural networks with fuzzy neurons have fast training rate and more robustness, however, the test cases are all toy problems, such as XOR, etc. The research of fuzzy neurons is in the embryonic state, it is possible to improve the learning algorithms in order to increase the expressive power of fuzzy neural networks. On the other hand, more and more researchers put their eyes on another stream of the researches - Neural Networks based Fuzzy Inference Systems.

2.3.2 Neural Networks based Fuzzy Inference Systems

During the design of fuzzy logic systems, most engineers find that it is very difficult to define the parameters of the membership function of linguistic term. Figure 2.10 shows two membership functions of "Middle" temperature. Both of the membership functions can be used to express the sentence "temperature is middle", however they have different widths, different means and hence different parameters. Which one should be used is depending on the personal judgment of the engineers.

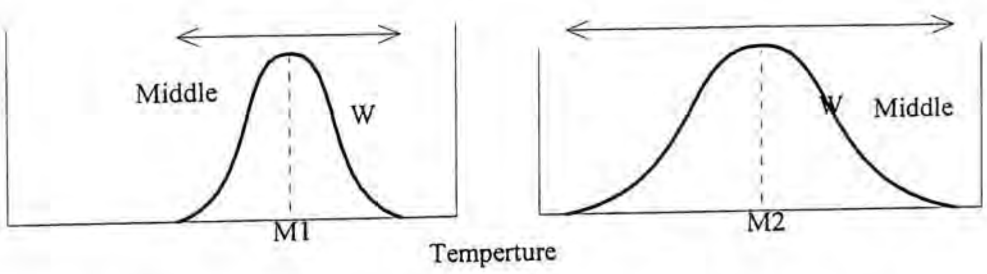


Figure 2.10 : Two membership functions of "Middle" temperture, however, two functions have different mean (M1 & M2) and different width (W1 & W2).

2.3.2.1 Statistical Method

One of the methods is using the statistical data[5] to assist the decision making of engineers. For instances, if most people have the perception that 20°C is middle temperature and the standard deviation is 3°C, then the membership function of "middle" has the mean of 20°C, width of 6°C and range of 17°C to 23°C.

However, this method is based on the perception of human being, it can only give a dirty and quick result. Moreover, the parameters cannot be fine tuned by statistical data. Hence, this method is only suitable to the fuzzy expert systems. For the control systems, the systems involve the highly nonlinear control surfaces which are very difficult to be interpreted by human being. Statistical method cannot give an optimal solution and hence is not suitable for such problems.

2.3.2.2 Neural Networks method

In order to solve the problems of highly nonlinear input-output mapping, neural networks can be made use. Neural networks have the capability of knowledge extraction and generalization. During supervised training, neural networks can extract the relationship between the input and output training patterns. Then, it can generalize the knowledge, such that it still give accurate results for those data which have not been trained, this is called function interpolation. Because of these capabilities, neural networks can be used to tune the parameters of fuzzy logic system, so that the optimal result can be obtained. Such kind of systems are refereed to as Neural Networks based Fuzzy Inference Systems or Neural Fuzzy Systems. The main idea of Neural Fuzzy Systems will be discussed subsequently.

The first step is using the feed forward neural networks to model architecture of fuzzy inference systems. The parameters are represented by the weight values of the links. Figure 2.11 shows an example of incorporating the fuzzy inference system into neural networks. The example represents the following two fuzzy rules in the neural networks.

1. *If Temperature is High and Pressure is High, then Valve Open is Large.*
2. *If Temperature is Low and Pressure is High, then Valve Open is Small*

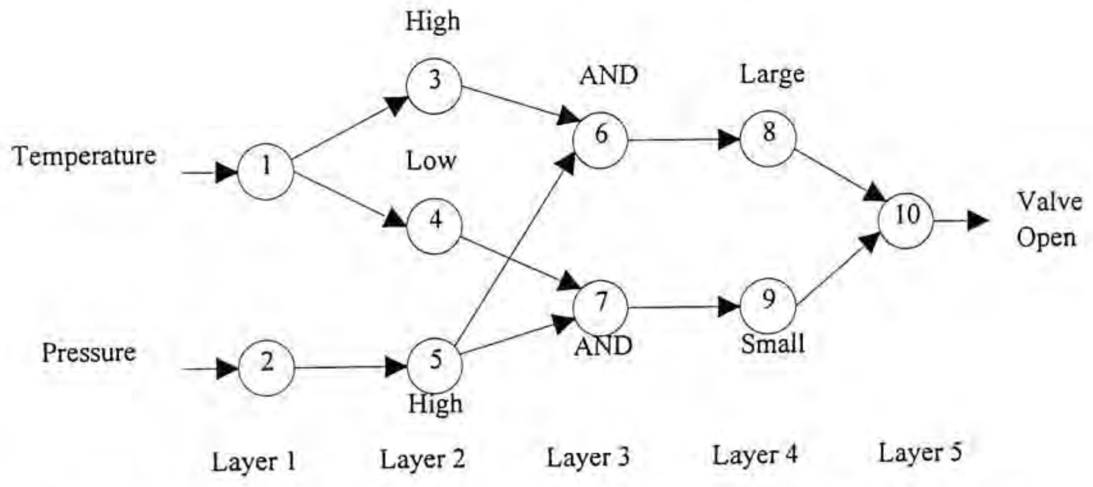


Figure 2.11 : An example of embedding two fuzzy rules into a feed forward neural network.There are two membership functions for temperture (High & Low) and Valve Open (Large & Samll), one membership function for pressure (Low). Node 6 and 7 act as a fuzzy AND operator, they also act as a connection point of permise and consequence part of the fuzzy rules.

By tracing the connection of nodes, we may find that rule 1 is represented by 1, 3, 2, 5, 6, 8, 10. On the other hand, rule 2 is represented by 1, 4, 2, 5, 7, 9, 10. The parameters of the membership functions are represented by the weight values of the links between layer 2 and layer 3.

The second step is using the training data to train the neural networks. The objective of the training is to adjust the parameters in order to minimize the error. Various training algorithm can be used, such as backpropagation learning algorithm, hybrid learning algorithm[2] .

In laboratory, we used to apply the neural fuzzy systems in control systems and nonlinear functions modeling in order to test the performance of the models. Many test results show the neural fuzzy systems can give better result than neural networks and fuzzy logic systems. Indeed, it is testified that neural network and fuzzy inference system in their integrated form have a very tight bonding, each of them overcome the weakness of the other. Neural network can add to the fuzzy inference system the ability to self tune the membership functions, on the other hand, the inputs to the whole system are fuzzified by the membership functions, thus performing a noise filtering function for the neural network component.

[2] *Hybrid Learning Algorithm is unsupervised learning algorithm + supervised learning algorithm.*

3 ANFIS Model

This section introduces the architecture and learning rule of Adaptive-Network-Based Fuzzy Inference System (ANFIS)[12][13][14]. ANFIS was proposed by Jang from the University of California at Berkeley in 1992. ANFIS is one of the neural networks based fuzzy inference systems. It was shown that ANFIS is functionally equivalent to the radial basis function networks. A number of simulations have been performed on ANFIS and all of the simulations yield remarkable results.

3.1 Adaptive Networks Architecture

As the name implied, the architecture of ANFIS is based on the adaptive network which is the superset of all kinds of feed forward neural networks with supervised learning. Figure 3.1 shows an example of Adaptive Networks. The example shows that the adaptive network consisting of nodes and directional links.

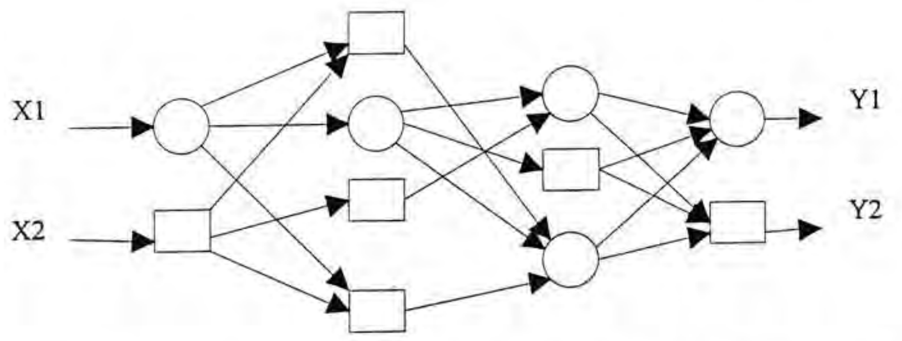


Figure 3.1 : An example of adaptive network. The sequare and cicle nodes represent the nodes have different transfer function. X1 and X2 are input variables. Y1 and Y2 are output variables. Signals are flowed as the direction of the links, i.e. one way.

In adaptive networks, all nodes are adaptive which means that the outputs of the nodes are depending on the parameters associated to the nodes. As long as the parameters are changed, the outputs of the nodes are also changed.

In order to generate the outputs, the nodes receive all input signals and perform a particular function on the signals. The function is called the transfer function or activation function. It can vary from nodes to nodes.

Basically, there is no constrain on the transfer function. However, if the gradient descent learning algorithm is used on the networks, the transfer function must be differentiable everywhere. In fact, gradient descent is the most popular learning algorithm in adaptive networks, you may find that most transfer functions are piecewise differentiable[1][13][15][17].

3.2 ANFIS Architecture

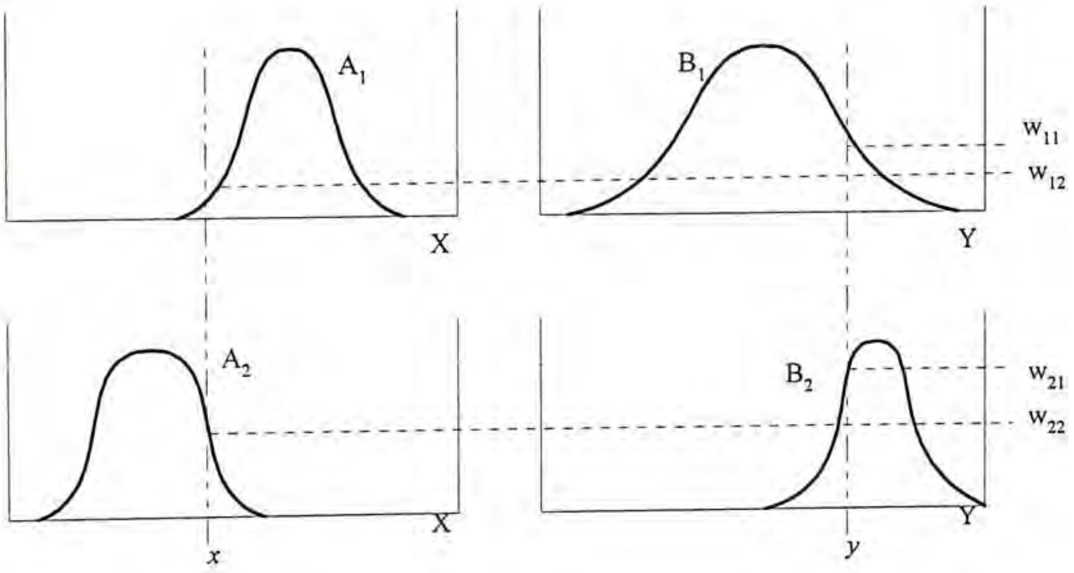
Similar to most neural networks based fuzzy inference systems, ANFIS embeds the fuzzy rules into the adaptive networks, so that the adaptive networks are functionally equivalent to the fuzzy inference systems. However, there is no weight value associated to the links, all the parameters (weight values) are contained in the nodes.

In order to illustrate the architecture of ANFIS, we assume the ANFIS contains 2 fuzzy rules which are shown in the following.

Rule 1 : *If x is A_1 and y is B_1, then* $f_1 = p_1x + q_1y + r_1$,

Rule 2 : *If x is A_2 and y is B_2, then* $f_2 = p_2x + q_2y + r_2$.

The fuzzy rules of ANFIS are based on the fuzzy if/then rules which was proposed by Takagi and Sugeno's in 1983. For this type of fuzzy if/then rules, the output of each rule is a linear combination of input variables plus a constant. The overall output is the weight average of the output of each rule. The weight (firing strength) of each fuzzy rule is the product of the membership values of input variables. Figure 3.2 shows the membership.



$$w_1 = w_{11} \times w_{12} \qquad w_2 = w_{21} \times w_{22}$$

$$\overline{w}_1 = \frac{w_1}{w_1 + w_2} \qquad \overline{w}_2 = \frac{w_2}{w_1 + w_2}$$

$$f = \overline{w}_1 f_1 + \overline{w}_2 f_2$$

$$f_1 = p_1x + q_1y + r_1 \qquad f_2 = p_2x + q_2y + r_2$$

Figure 3.2 : The upper graphs show the membership functions of input variables X and Y. The lower part shows the procedures of inference.

Figure 3.3 shows the corresponding architecture of ANFIS. It shows that the premise and consequent parameters are associated with the nodes in first and fourth layer respectively. The nodes in the first and fourth layer are called adaptive nodes which reflect their adaptive nature. For the other nodes, there is no parameter associated with them and these nodes are called fixed nodes. In the following section, each layer will be discussed in details.

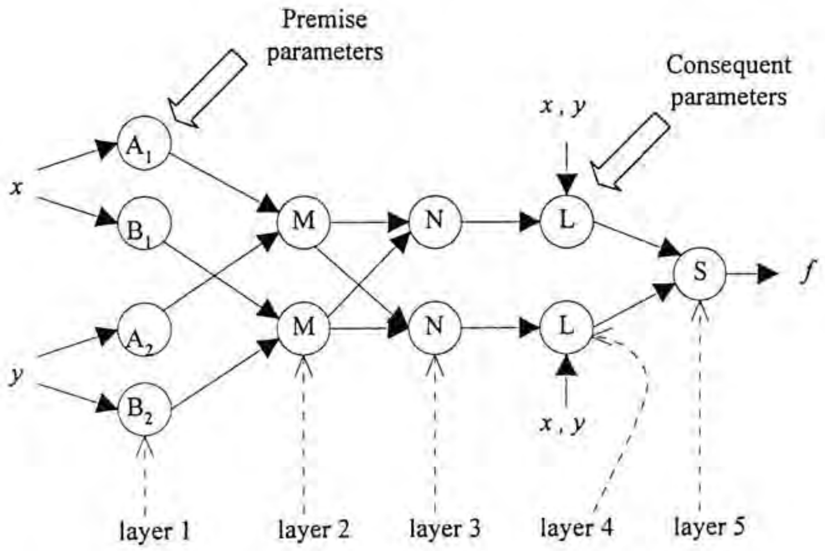


Figure 3.3 : The ANFIS model with rule 1 and rule 2. First layer nodes output the corresponding membership value of inoming signals. M nodes output the product of incoming signals. N nodes output the normalization of the incoming signals. L nodes output the linear combination of the incoming signals.

3.2.1 Layer One

Nodes of layer 1 are corresponding to the membership functions of the input variables. The inputs of the nodes are the values of the input variables, the outputs are the degree of membership of the input variables of the corresponding linguistic terms. Each node is associated with a transfer function (node function) $\mu_{A_i}(x)$ where

A_i is the linguistic term. Therefore, $\mu_{A_i}(x)$ is the membership function of A_i. The output of node j is denoted by O_j^1 where 1 stands for the first layer[1]. Hence, for $i \in \{1, 2\}$ and $j \in \{1, 2, 3, 4\}$.

$$O_j^1 = \mu_{A_i}(x) \tag{3.1}$$

or

$$O_j^1 = \mu_{B_i}(x) \tag{3.2}$$

O_j^1 is the membership value of the input variable to the corresponding linguistic term. Usually, the membership functions are bell-shaped with maximum and minimum are 1 and 0 respectively. The following functions are used to represent the bell-shaped membership functions.

$$\mu_{A_i}(x) = \frac{1}{1+[(\frac{x-c_i}{a_i})^2]^{b_i}} \tag{3.3}$$

or

$$\mu_{A_i}(x) = e^{\{-[(\frac{x-c_i}{a_i})^2]^{b_i}\}} \tag{3.4}$$

where $\{a_i, b_i, c_i\}$ is the parameter set of the membership functions, they are referred to as the premise parameters. These functions are always used as membership function because they reflect the perception of human being. The changes are smooth and continuous, the gain is low at two ends and the middle, the gain is close to linear elsewhere. Beside these functions, other piecewise differentiable functions can also be used, such as trapezoidal or triangular-shaped function.

---

[1] The output of all nodes is denoted by O_j^L, where L is the layer number and j is the node number in layer L.

As long as the parameters change, the shape of the membership functions also change. This results in various forms of membership function. By adjusting the shape of membership functions, we can change the membership values of input variables in order to minimize the error of the outputs.

3.2.2 Layer Two

Nodes of layer 2 act as the fuzzy "AND" operator in the premise part of the fuzzy if/then rules. The fuzzy "AND" operator performs the multiplication of the membership values of the input variables. The output of a node in layer 2 is

$$O_j^2 = \mu_{A_i}(x) \times \mu_{B_k}(x) \tag{3.5}$$

where i, j and $k \in \{1,2\}$. The output O_j^2 stands for the weight (firing strength) of the corresponding rule, and it is refereed to as w_j (i.e. $w_j = O_j^2$).

Each node of layer 2 is corresponding to 1 fuzzy if/then rule, hence, by considering the connection of these nodes, the fuzzy rules can be interpreted. For instance, consider the first node in layer 2, it is connected by the nodes A_1 and B_1 of layer 1. Therefore, the first node of layer 2 stands for the Rule 1.

3.2.3 Layer Three

The nodes of layer three calculate the normalized weight values (firing strength) of the fuzzy rules. The outputs of the nodes are

$$O_j^3 = \frac{w_i}{w_1 + w_2}, \tag{3.6}$$

where j and $i \in \{1, 2\}$. It is used to refer the normalized weight as $\overline{w}$. Hence,

$$\overline{w}_j = O_j^3. \tag{3.7}$$

3.2.4 Layer Four

The nodes of layer 4 calculate the crisp outputs of the fuzzy if/then rules, then multiply the outputs by the normalized weight of the rules. Following is the equation of the nodes outputs.

$$O_i^4 = \overline{w}_i f_i = \overline{w}_i (p_i x + q_i y + r_i), \tag{3.8}$$

where $i \in \{1, 2\}$. $\{p_i, q_i, r_i\}$ is the parameter set of the consequent parameters.

3.2.5 Layer Five

There is only one node in layer 5, the node calculates the sum of all input signals. The sum is the overall output of the fuzzy inference system. The equation of the output of the node is

$$O_1^5 = \sum_i \overline{w}_i f_i = \frac{\sum_i w_i f_i}{\sum_i w_i} = \text{overall output.} \tag{3.9}$$

We have already discussed the architecture of ANFIS. The illustrated ANFIS model is over simple because 2 input variables with each has 2 membership functions can yield 4 fuzzy if/then rules, while only two of them are used for the illustration.

However, based on the input - output patterns, it is very difficult to determine which rules should be adopted or detached. Therefore, in practical implementation, all possible rules will be embedded into the ANFIS model. When all rules are embedded into the ANFIS model, the number of fuzzy rules is equal to the product of the number of membership function of each input variable. Figure 3.4 shows the ANFIS with all possible rules.

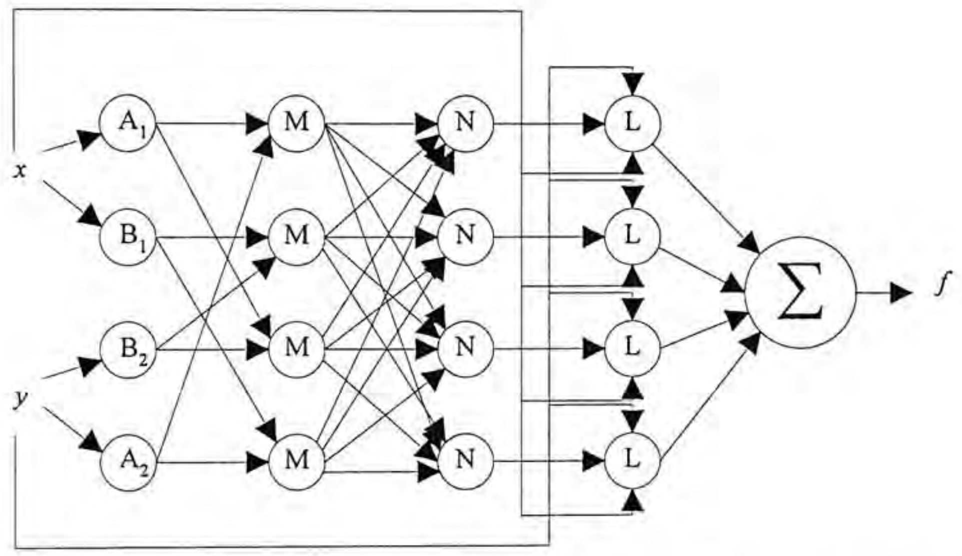


Figure 3.4 : The figure shows all possible rules for ANFIS model which has 2 membership functions of each input variable. In fact, the number of rules is the product of membership functions of all input variables.

Figure 3.5 shows the input space which is partitioned into 4 fuzzy subspaces, each subspace is governed by two membership functions from each input variable. The shared area represents the fuzzy region.

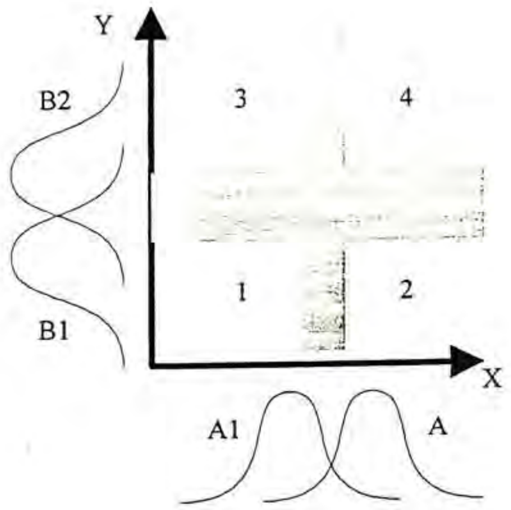


Figure 3.5 : The fuzzy subspaces of the fuzzy rules 1 & 2. 2 rules with 2 membership functions partition the space into 4 subspaces. The shaped area is the fuzzy region.

3.2.6 Multiple Outputs ANFIS

Up till now, the discussed ANFIS model has only one output, this is not very practical for real applications. In many applications, the training data (input/output pairs) are in multi-dimensions and ANFIS will not be applicable.

One of the solutions is to construct multiple ANFIS, each corresponds to one of the dimensions of the output vector. Figure 3.6 shows an example of this solution, assuming that the output vector has 2 dimensions, hence, 2 ANFIS models are constructed for each dimension. Nevertheless, the error propagation in this solution is very serious.

$$Prob(overall\ output\ is\ correct) = \prod_i Prob(ANFIS\ model\ i\ gives\ correct\ output) \qquad (3.10)$$

If the output vector has 10 dimensions, then, 10 ANFIS models should be created. If the accuracy of each ANFIS model is 90%, according to (3.10), the accuracy of the whole system is 35% (i.e. $0.9^{10} \approx 0.35$). Because of the serious error propagation, this solution is not adopted in the project.

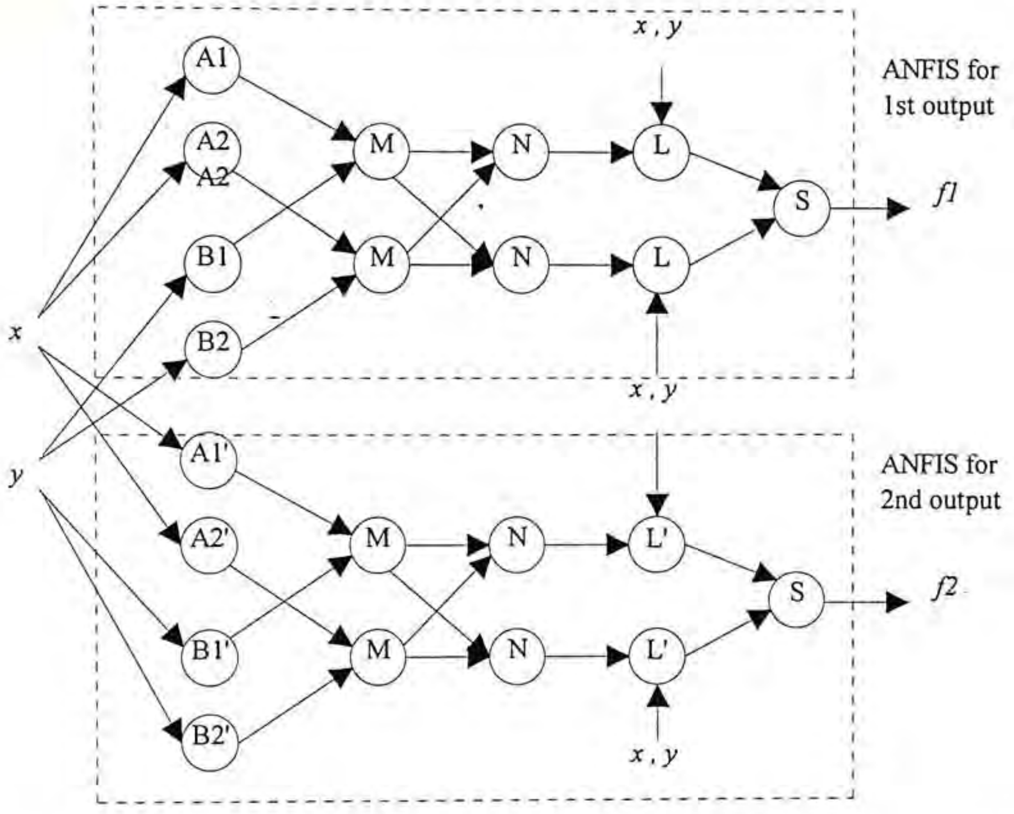


Figure 3.6 : A solution for the problem of 2 dimensions output vector. 2 ANFIS models are built for each output. Note that the parameter sets are different for each ANFIS model.

Instead, another solution is adopted. Figure 3.7 shows the enhanced ANFIS which can support multi-dimensions output vectors. In the enhanced ANFIS, adding one more output dimension means adding one more set of nodes in layer 4 and connecting all nodes in layer 3 to this set. For this enhanced model, the error propagation will not be so serious as the first solution. This is because of the update of the premise part is based on the error of all output nodes.

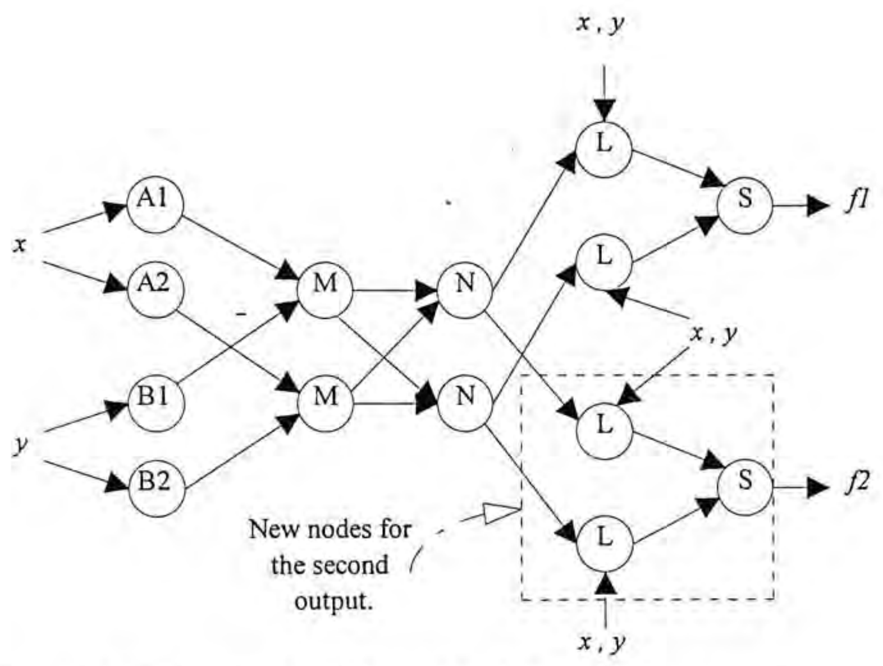


Figure 3.7 : The enhanced ANFIS model for 2 dimensional output vectors.

3.3 ANFIS Learning Algorithms

This section discusses the learning algorithms of ANFIS. In Jang's paper[13], two learning algorithms have been proposed for ANFIS. One is gradient descent learning and the other is hybrid learning. Each of them will be discussed in details. Besides, an algorithm of least square estimate (LSE) will be introduced for hybrid learning. LSE is a technique for approximating the solutions of matrix equations.

3.3.1 Gradient Descent Learning

Gradient descent learning algorithm is a basic and common learning algorithm in feed forward neural networks. It can be used to update the parameters of neural networks in order to achieve the correct input-output mapping of the networks.

3.3.1.1 Gradient Descent Learning for Adaptive Networks

Consider an adaptive network with both adaptive nodes and fixed nodes, the parameter set of the network is the union of all the parameters of adaptive nodes. The following are the definitions of the variables of the network.

L	-	Number of layers in the network.
$\#(k)$	-	Number of nodes in k-th layer.
(k,i)	-	The node at i-th position of k-th layer.
O_i^k	-	The node function and output of (k,i) .
P	-	Number of training data.
$T_{m,p}$	-	Target output of m-th output node for p-th training data.
E_p	-	Error of the network on p-th training data.

Since the output of a node depends on the incoming signals as well as the parameters of the nodes function, we have

$$O_i^k = O_i^k\,(O_1^k,\ldots,O_{\#(k-1)}^{k-1}, a, b, c,\ldots)\,, \tag{3.11}$$

where a, b, c ... etc. are the parameters of the node function.

The error of p-th training data is the sum of error of each output node. Therefore,

$$E_p = \sum_{m=1}^{\#(L)} (T_{m,p} - O_{m,p}^L)^2\,, \tag{3.12}$$

the overall error is sum of error of each training data.

$$E = \sum_{p} E_p . \tag{3.13}$$

The error rate of the output node *(L,i)* can be found by (3.12), it is

$$\frac{\partial E_p}{\partial O_{i,p}^L} = -2(T_{i,p} - O_{i,p}^L) \tag{3.14}$$

For the internal node *(k,i)*, the error rate can be derived by the chain rule and it is the linear combination of the error rates of the nodes in the next layer.

$$\frac{\partial E_p}{\partial O_{i,p}^k} = \sum_{m=1}^{\#(k+1)} \frac{\partial E_p}{\partial O_{m,p}^{k+1}} \frac{\partial O_{m,p}^{k+1}}{\partial O_{i,p}^k}, \tag{3.15}$$

where $1 \le k \le L - 1$. Base on (3.14) and (3.15), we can find the error rate of all nodes.

The following variables are used for finding the update rule of the parameters.

α	-	a parameter of the adaptive network.
S_α	-	Set of nodes where output depending on α.

Based on the chain rule, we have

$$\frac{\partial E_p}{\partial \alpha} = \sum_{O^* \in S_\alpha} \frac{\partial E_p}{\partial O^*} \frac{\partial O^*}{\partial \alpha} . \tag{3.16}$$

The derivative of the overall error measure *E* with respect to α is

$$\frac{\partial E}{\partial \alpha} = \sum_{p=1}^{P} \frac{\partial E_p}{\partial \alpha} . \tag{3.17}$$

The update formula for the generic parameter α is

$$\Delta \alpha = -\eta \frac{\partial E}{\partial \alpha}, \tag{3.18}$$

where η is the learning rate which can be expressed as

$$\eta = \frac{k}{\sqrt{\sum_{\alpha}(\frac{\partial E}{\partial \alpha})^2}}, \tag{3.19}$$

k is called the step size, it controls the transition length of the parameters (movement of parameters in the parameter space) during the gradient decent learning. If k is small, the transition of the parameters will be close to the gradient of the error surface, the learning time and the convergence time will be longer. If k is large, the transition of the parameters will be far from the gradient of the error surface, at the beginning of training, the convergence rate is high. Nevertheless, as long as the parameters are close to the optimum, they will oscillate around the optimum. Figure 3.8 shows the learning path of large k and small k. In order to make the learning rate to be faster and maintain the stability of the networks, an adaptive step size will be used. The adaptive step size will be discussed later.

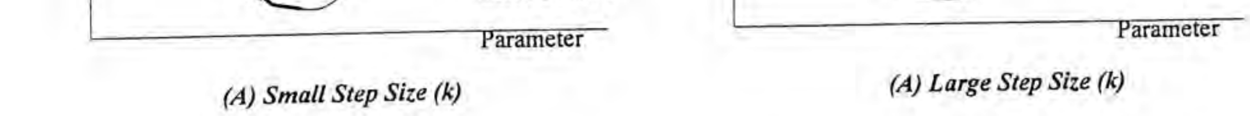


Figure 3.8 : The figure shows the transition paths of small step size (A), and large step size (B). In this example, the error is depending on one parameter only.

There are two learning paradigms for gradient descent learning. One is the batch learning in which the update of parameters is performed after all training data are presented to the network. For the batch learning, the update is according to the equation (3.17). The other is the pattern learning in which the update of parameters is performed after each training data is presented to the network. For the pattern learning, the update is according to the equation (3.16).

3.3.1.2 Gradient Descent Learning for ANFIS

In order to show how to update the parameters in ANFIS, consider the following generic model.

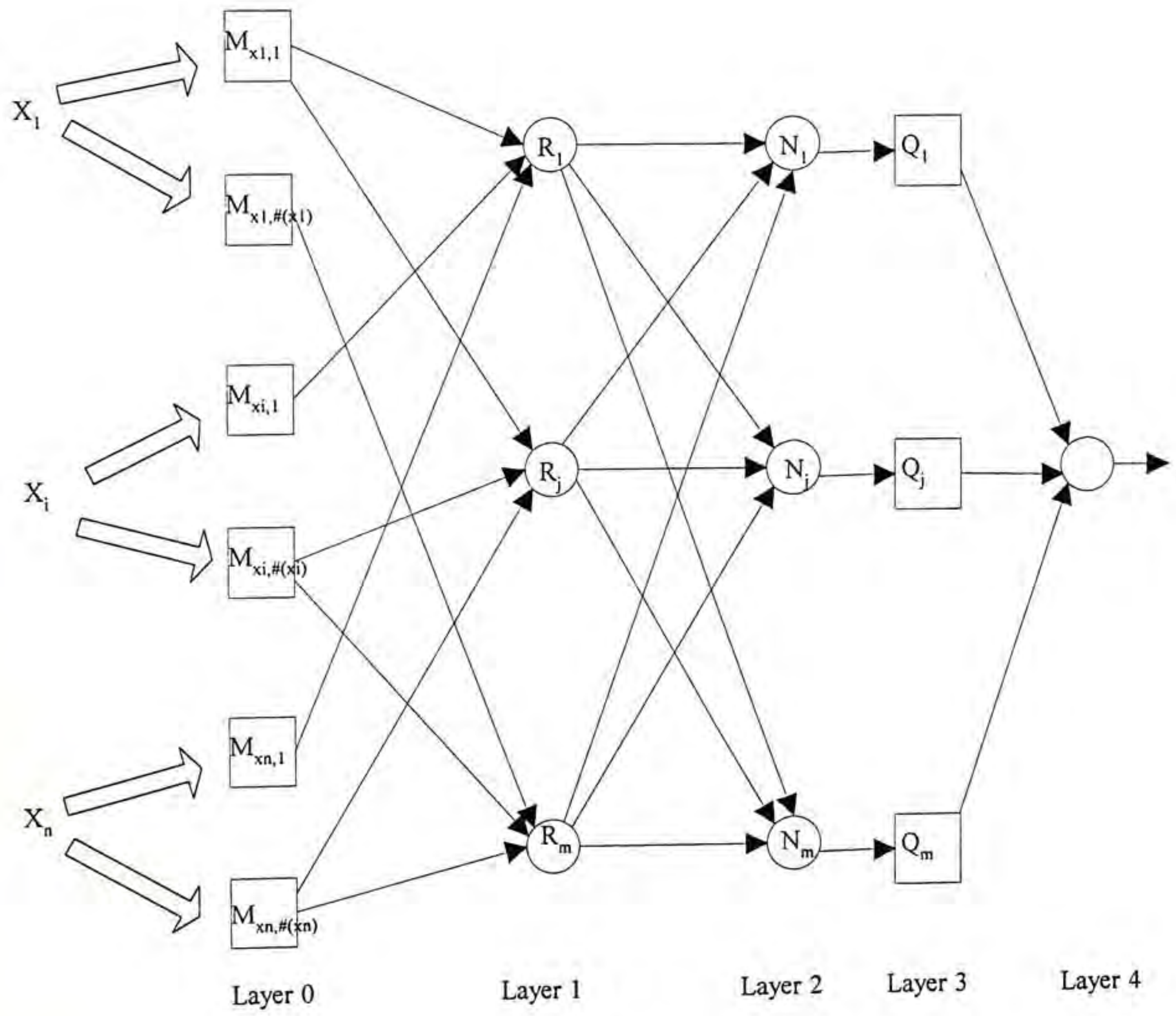


Figure 3.9 : Generic ANFIS Model.

The following are the definitions of variables for the generic model:

x_i	-	i-th input variable.
n	-	Total number of inputs.
$\Im$	-	Number of training patterns.
m	-	Total number of rules.
S_j	-	Set of membership functions associating to rule j.
O	-	Target output value.

The following are the equations of the outputs from each layer.

Layer 0 : Suppose $M^p_{x_i,k}$ is the output of the node as well as the node function, then we have $M^p_{x_i,k} = M^p_{x_i,k}(a_i, b_i, c_i, p)$, where p is the p-th training data.

Layer 1 : Output is $R^p_j = \prod_{M^p_{x_i,k} \in S_j} M^p_{x_i,k}$.

Layer 2 : Output is $N^p_j = \dfrac{R^p_j}{\sum_{h=1}^{m} R^p_h}$.

Layer 3 : Output is $Q^p_j = N^p_j(c_0 + c_1 x^p_1 + c_2 x^p_2 + \ldots + c_n x^p_n)$, where c_i are fixed in forward pass.

Layer 4 : Output is $y^p = \sum_{j=1}^{m} Q^p_j$

Suppose the error of training data p is E^p where $E^p = (O^p - y^p)^2$. Then the total error for the whole training data set is $E = \sum_{p=1}^{3} E^p = \sum_{p=1}^{3} (O^p - y^p)^2$.

Consider the parameter at the k-th membership function of i-th input variable : $a_{i,k}$.

The error rate of the parameter is $\Delta a_{i,k} = \dfrac{\partial E}{\partial a_{i,k}}$.

The close form of $\Delta a_{i,k}$ is shown in the following :

$$\Delta a_{i,k} = \frac{\partial E}{\partial a_{i,k}} = \sum_{p=1}^{3} \frac{\partial E^p}{\partial a_{i,k}} = \sum_{p=1}^{3} \frac{\partial (O^p - y^p)^2}{\partial a_{i,k}} = \sum_{p=1}^{3} -2(O^p - y^p) \frac{\partial y^p}{\partial a_{i,k}}. \tag{3.20}$$

$$\frac{\partial y^p}{\partial a_{i,k}} = \sum_{j=1}^{m} \frac{\partial Q_j^p}{\partial a_{i,k}}. \tag{3.21}$$

$$\frac{\partial Q_j^p}{\partial a_{i,k}} = C_j X^p \frac{\partial N_j^p}{\partial a_{i,k}}, \tag{3.22}$$

where $C_j = \left[c_{o,j}\ c_{1,j} \dots c_{n,j}\right]^T$ and $X^p = \left[1\ x_1^p \dots x_n^p\right]$.

$$\frac{\partial N_j^p}{\partial a_{i,k}} = \frac{\partial}{\partial R_j^p} \left(\frac{R_j^p}{\sum_{h=1}^{m} R_h^p} \right) \frac{\partial R_j^p}{\partial a_{i,k}} = \left(\sum_{h=1}^{m} R_h^p \right)^{-2} \left(\sum_{h=1}^{m} R_h^p - R_j^p \right) \frac{\partial R_j^p}{\partial a_{i,k}} \tag{3.23}$$

$$\frac{\partial R_j^p}{\partial a_{i,k}} = \frac{\partial \prod_{M_{x^*,k^*} \in S_j} M_{x^*,k^*}^p}{\partial a_{i,k}} = \left(\frac{\prod_{M_{x^*,k^*} \in S_j} M_{x^*,k^*}^p}{M_{x_i,k}^p} \right) \frac{\partial M_{x_i,k}^p}{\partial a_{i,k}} \tag{3.24}$$

Let $\Phi_{j,x_i,k}^p = \left(\dfrac{\prod_{M_{x^*,k^*} \in S_j} M_{x^*,k^*}^p}{M_{x_i,k}^p} \right) \dfrac{\partial M_{x_i,k}^p}{\partial a_{i,k}}$ and putting this into (3.23), we have

$$\frac{\partial N_j^p}{\partial a_{i,k}} = \left(\sum_{h=1}^{m} R_h^p\right)^{-2}\left(\sum_{h=1}^{m} R_h^p - R_j^p\right)\delta^p_{j,x_i,k}\,, \tag{3.25}$$

substitute (3.25) into (3.22), we have

$$\frac{\partial Q_j^p}{\partial a_{i,k}} = C_j X^p \left(\sum_{h=1}^{m} R_h^p\right)^{-2}\left(\sum_{h=1}^{m} R_h^p - R_j^p\right)\delta^p_{j,x_i,k}\,, \tag{3.26}$$

substitute (3.26) into (3.21), we have

$$\frac{\partial y^p}{\partial a_{i,k}} = \sum_{j=1}^{m} C_j X^p \left(\sum_{h=1}^{m} R_h^p\right)^{-2}\left(\sum_{h=1}^{m} R_h^p - R_j^p\right)\delta^p_{j,x_i,k}\,, \tag{3.27}$$

substitute (3.27) into (3.20), we have

$$\Delta a_{i,k} = \sum_{p=1}^{3} -2(O^p - y^p)\sum_{j=1}^{m} C_j X^p \left(\sum_{h=1}^{m} R_h^p\right)^{-2}\left(\sum_{h=1}^{m} R_h^p - R_j^p\right)\delta^p_{j,x_i,k}\,. \tag{3.28}$$

Base on section 3.3.1.1 and (3.28), the update rule of parameter $a_{i,k}$ is $a_{i,k}(t+1) = a_{i,k}(t) - \eta\Delta a_{i,k}(t)$, where η is the learning rate.

3.3.2 Least Squares Estimate (LSE)

Before the hybrid learning algorithm is discussed, we discuss the Least Squares Estimate (LSE) first. LSE is a technique which can approximate the solution of a matrix equation.

Assume that the adaptive network has one output node, the output of the output node is

$$output = F(\vec{I}, S), \tag{3.29}$$

where $\overset{\varpi}{I}$ is the set of input variables and S is the set of parameters. If there is a function H, such that the composite function $(H \circ F)$ is linear in S_2, where S_2 is the set of some of the parameters in S. Suppose S can be decomposed into 2 sets, we have

$$S = S_1 \oplus S_2. \tag{3.30}$$

Suppose the parameters in S_1 are fixed, then we have

$$H(output) = H \circ F(\overset{\varpi}{I}, S_2). \tag{3.31}$$

If we have $\Im$ sets of training data, we can plug the training data into (3.31) and get the following matrix equation

$$AX=B. \tag{3.32}$$

Suppose $|S_2| = M$, then the dimension of A, X and B are $\Im \times M$, $M \times 1$ and $\Im \times 1$ respectively. For each training data A_l, $A_l = [a_1^l\, a_2^l \dots a_M^l]$ where a_j^l is the instant value of one of the input variables (suppose it is x_i) at l-th training data, where $x_i^p \in \overset{\varpi}{I}$. For all training data, $A = [A_1\, A_2 \dots A_\Im]^t$, where t stands for the transpose. B is the column vector of the target output. Suppose b_l is the target output of training data l, then $B = [b_1\, b_2 \dots b_\Im]^t$. X is also a column vector such that $X = [p_1\, p_2 \dots p_M]^t$ where $p_i \in S_2$.

Since the number of training data ($\Im$) is usually greater than the number of linear parameters (M), equation (3.32) is overdetermined problem. In general, there is no exact solution for (3.32).

Instead, Least Squares Estimate (LSE) is used to find the approximate solution of X, the main idea of LSE is to minimize the square error $\|AX-B\|^2$. The approximate solution is denoted by X^* and the most popular formula for X^* uses the pseudo-inverse of X:

$$X^* = (A^t A)^{-1} A^t B, \tag{3.33}$$

where $(A^t A)^{-1} A^t$ is the pseudo-inverse of A if A^t is non-singular.

In equation (3.33), X^* is expressed in a close form equation, however, it is computational expensive for determining the inverse of matrices. Moreover, when A^t is singular, X^* becomes ill-defined. Therefore, instead of solving X^* directly, we use the sequential formulas to solve X^*. Let i-th row vector of matrix A in (3.32) be a_i^t and the i-th element of B be b_i^t, then X can be calculated using the iteratively sequential formulas :

$$X_{i+1} = X_i + S_{i+1} a_{i+1} (b_{i+1}^t - a_{i+1}^t X_i) \tag{3.34}$$

$$S_{i+1} = S_i - \frac{S_i a_{i+1} a_{i+1}^t S_i}{1 + a_{i+1}^t S_i a_{i+1}}, \qquad i = 0, 1, \dots, \Im- \tag{3.35}$$

S_i is used to be called as covariance matrix and the least squares estimate X^* is equal to $X_{\Im}$. The initial conditions for equation (3.34) and (3.35) are $X_0 = 0$ and $S_0 = \gamma I$, where γ is a positive large number and I is the identity matrix of dimension $M \times M$.

When dealing with the multiple output networks (multiple output ANFIS in section 3.2.6), the LSE is still applied, but b_i^t is a row vector which is the i-th row of matrix B.

Up till now, our discussion is based on the batch learning. For pattern learning, the parameters have to be updated by E_p instead of E. Actually, this is not the appropriate procedure to minimize E, however, the result can be approximated by setting a small learning rate.

In order to account for the time-varying characteristic of the incoming data for the sequential least formulas, we need to reduce the effect of old training data when new training data is presented to the network. The solution is to modify the original sequential least formulas to its weighted version such that the recent data have larger weight than the past data. For the weighted version, we introduce a forgetting factor (λ) into the original equations (3.34) and (3.35) which become

$$X_{i+1} = X_i + S_{i+1} a_{i+1} (b_{i+1}^t - a_{i+1}^t X_i) \tag{3.36}$$

$$S_{i+1} = \frac{1}{\lambda}\left(S_i - \frac{S_i a_{i+1} a_{i+1}^t S_i}{1 + a_{i+1}^t S_i a_{i+1}} \right), \quad i = 0, 1, \dots, \Im - 1 \tag{3.37}$$

where λ is a value between 0 and 1. For smaller λ, the effect of the old data will be reduced rapidly. Nevertheless, if λ is too small, it may cause the instability of the system.

3.3.3 Hybrid Learning Algorithm

3.3.3.1 Hybrid Learning Algorithm in Adaptive Networks

The hybrid learning algorithm is based on both the gradient descent learning and least squares estimate. The reason of introducing the LSE is that the gradient descent learning is generally slow and easy to be trapped by local minima.

For hybrid learning, each training epoch is composed of forward pass and backward pass. In the forward pass, training data are supplied in order to construct the matrices A and B in equation (3.32). Then, matrix X is calculated by using the sequential least square formulas in (3.34) (3.35) or (3.36) (3.37). Once X can be found, the parameters in S_2 are found. In the backward pass, the signals are propagated to the output node based on the parameters of S_2 which were found in the forward pass. The output error can be found by subtracting the output value by the target value. The error can be used to update the parameters of S_1 by using the gradient descent learning.

By fixing the parameters in S_1, LSE is able to find the parameters in S_2 such that they are guaranteed to be the global minimum in S_2 parameter space. Therefore, hybrid learning makes use of LSE to reduce the parameters searching space firstly, then applies the gradient descent learning. This approach can substantially reduce the convergence time of the network training.

3.3.3.2 Hybrid Learning Algorithm in ANFIS

Clearly, ANFIS is an adaptive networks. Therefore, the hybrid learning algorithm can be applied to ANFIS. This section mainly discuss how to define the parameters in S_1 and S_2.

From figure 3.2, it is observed that given the values of the premise parameters, the overall output can be expressed as a linear combinations of the consequent parameters. The output in figure 3.2 can be expressed as

$$f = (\overline{w}_1 x) p_1 + (\overline{w}_2 u) q_1 + (\overline{w}_1) r_1 + (\overline{w}_2 x) p_2 + (\overline{w}_2 y) q_2 + (\overline{w}_2) r_2 . \tag{3.38}$$

Hence, the linear parameters are the consequent parameters and we have

S = set of total parameters,

S_1 = set of premise parameters,

S_2 = set of consequent parameters.

The function H is the identity function and F is the function of fuzzy inference system of the ANFIS.

In the ANFIS with hybrid learning algorithm, during the forward pass, signals go forward till layer 4 and the consequent parameters are identified by LSE. In the backward pass, the errors propagate backward and the premise parameters are updated by the gradient descent algorithm. Table 3.1 shows the activities in each pass.

	Forward pass	*Backward pass*
Premise parameters	fixed	gradient descent
Consequent parameters	least squares estimate	fixed
Signals	node outputs	error rates

Table 3.1 : Hybrid learning activities of ANFIS for each pass.

However, it should be noted that the computational complexity of LSE is very high. In some situations, such as the large number of training data, LSE may not be applicable. Jang has introduced 4 training modes which are shown in Table 3.2.

Training modes	*Description*
Gradient descent only	All parameters are trained by gradient descent learning algorithm.
Gradient descent and one pass LSE	LSE is applied at the first training epoch to give the initial values of the consequent parameters, then, the gradient descent is applied to all the parameters.
Gradient descent and LSE	Hybrid learning algorithm.
Sequential LSE only	Applying LSE to update all the parameters of ANFIS.

Table 3.2 : Four training modes of ANFIS.

The selection of using which of the training modes is depending on the computation complexity of the problem and the resulting performance.

4 ANFIS Library

As mentioned before, the objective of the project is to construct an ANFIS library. This section discusses the structure and use of the library.

4.1 Library Structure

The library is built by using C++ in order to support the new programming approach - Object-Oriented Programming (OOP). In general, all OOP languages compose of three parts in common : objects, polymorphism and inheritance[10].

An object is a logical entity that contains both data and code that manipulates that data. Within an object, some of the code and data may be private to the object and inaccessible to any thing outside the object. In this way, an object provides a significant level of protection against some other, unrelated part of the program accidentally modifying or incorrectly using the private parts of the object. This linkage of code and data is often referred to as encapsulation.

Polymorphism is characterized by the phase "one interface, multiple methods". This means that one name can be used for several related but slightly different purposes. In essence, polymorphism allows one interface to be used with a general class of actions. The specific action selected is determined by the type of data involved.

Inheritance is the process by which one object can acquire the properties of another object. This is important because it supports the concept of classification. If you think about it, most knowledge is made manageable by hierarchical classifications. Through the feature of inheritance, an object need only define those qualities that make it unique within its class and it is possible for one object to be a specific instance of other objects.

Based on the OOP, large scale systems can be constructed more easily, moreover, the maintenance cost of the systems can be reduced.

4.1.1 System Modules

The ANFIS library contains the following system modules.

Module Name	*Description*
`anfis.h`	Contains the header information of the ANFIS library.
`anfis.cc`	Contains the constructors and destructors of the object "`anfis_model`" which is the main body ANFIS model. it also contains the procedures to establish the architecture of the adaptive network.
`backward.cc`	Contains the procedures of the backward pass of the training phase.
`build.cc`	Contains the procedures of building each layer and its nodes of the ANFIS model.
`fileio.cc`	Contains the procedures of reading the input training data file and writing the output file.
`forward.cc`	Contains the procedures of the forward pass of the training

	phase.
`kalman.cc`	Contains the procedures of calculating the LSE of the network parameters.
`layers.cc`	Contains the destructors of the object "`layer`".
`matrics.cc`	Contains the procedures of matrices and vectors operations.
`node.cc`	Contains the constructors and destructors of the object "`node`".
`parm.cc`	Contains the procedures of initialization and update of the network parameters.
`train.cc`	Contains the training procedures of the ANFIS model.

4.1.2 Class Objects

The ANFIS library contains the following class objects.

Class Object	*Description*
`anfis_model`	This is the main body of the ANFIS library. Programmers only need to declare a class of `anfis_model`, they can train and evaluate the network.
`matrix`	This is the class of matrices, it is used to declare the input parameters of training and evaluating procedures of `anfis_model`. Each element of `matrix` class has the data type "`double`".
`vec`	The is the class of vectors, it is used to declare the input parameters of the procedures for setting the parameters of

	membership functions of anfis_model. Each element of vec class has the data type "double"

.1.3 Class Functions

The prototypes of anfis_model, matrix and vec class declaration are as ollows:

```
anfis_model var_name(int input_nodes, int output_nodes);
matrix var_name(int num_of_columns, int num_of_rows);
vec var_name(int num_of_elements);
```

The value of the elements of matrix and vec can be specified directly as follows :

```
matrix_variable[col][row] = data_item;
vec_variable[idx] = data_item;
```

Since matrix and vec are used to declare the input parameters of the procedures, there is no associated class functions for programmers. In this section, we only list the class functions of anfis_model as follows :

Class functions of anfis_model	*Description*
int is_model_create();	Return 1 is the ANFIS model is created, otherwise, return 0.
int get_input_var_num();	Return the number of input variables (input nodes).

5 Applications

This section presents the performance of ANFIS in various applications. In all applications, ANFIS is trained by the input-output training pair and no expert is consulted for the fuzzy rules. Since no expert is consulted, we do not know which fuzzy rule should be adopted and all possible rules are embedded in ANFIS for all applications.

For the input range and the number of membership functions of each input variable is fixed, the initial values of the premise parameters are set so that the memberships functions are equally spaced along the input range. Moreover, the membership functions have to satisfy the ε-completeness[13] with ε=0.5. "ε-completeness with ε =0.5" means that for any input x within the input range, there exist a membership function such that $\mu_A(x) \geq 0.5$. In this manner, the fuzzy inference system can provide smooth transition and sufficient overlapping from one linguistic label to another. Figure 5.1 shows an initial setting of membership functions for the number of membership functions is 4 and input range is [0,12].

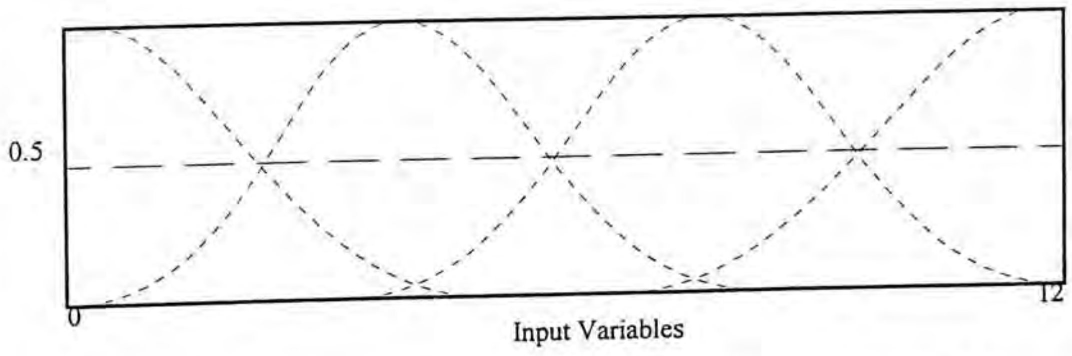


Figure 5.1 : The initial membership functions with ε-completeness=0.5.

As discussed before, the step size should be changed adaptively, the heuristic rules[13] are as follows :

1. If the error measure undergoes 4 consecutive reductions, increase *k* by 10%.
2. If the error measure undergoes 2 consecutive combinations of 1 increase and 1 reduction, decrease *k* by 10%.

Although the selection of 10% is rather arbitrarily, the results are quite satisfactory. Moreover, the initial value of k is not very critical as long as it is not too big. Besides that, other adaptive learning rate can also be used as the step size to control the learning speed. Figure 5.2 shows the error measure w.r.t. the training epochs.

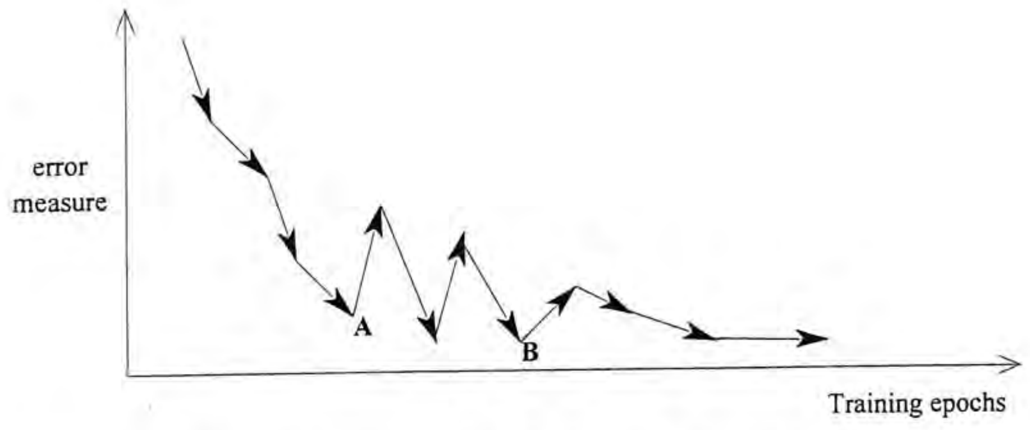


Figure 5.2 : The figure shows the action of the heuristic rules of step size. At point A, rule 1 is used to increase the step size after 4 downs. At point B, rule 2 is used to decrease step size after 2 combinations of 1 up and 1 down.

5.1 Logical Operators

The first application is using ANFIS to model the logical operators : XOR, OR, AND.

5.1.1 Architecture

In this application, the ANFIS contains 2 input variables, 1 output variable and 2 membership functions for each input variable. Based on this architecture, 4 possible rules can be defined, they are

1. *If x is A_1 and y is B_1 then* $z = p_1x + q_1y + r_1$,
2. *If x is A_1 and y is B_2 then* $z = p_2x + q_2y + r_2$,
3. *If x is A_2 and y is B_1 then* $z = p_3x + q_3y + r_3$,
4. *If x is A_2 and y is B_2 then* $z = p_4x + q_4y + r_4$.

A_1, A_2, B_1, B_2 are defined by the membership functions $\mu_{A_1}(x), \mu_{A_2}(x), \mu_{B_1}(y), \mu_{B_2}(y)$

where

$$\mu_{A_1}(x) = \frac{1}{1 + [(\frac{x - c_1}{a_1})^2]^{b_1}} \tag{5.1}$$

$$\mu_{A_2}(x) = \frac{1}{1 + [(\frac{x - c_2}{a_2})^2]^{b_2}} \tag{5.2}$$

$$\mu_{B_1}(x) = \frac{1}{1 + [(\frac{x - c_3}{a_3})^2]^{b_3}} \tag{5.3}$$

$$\mu_{B_2}(x) = \frac{1}{1 + [(\frac{x - c_4}{a_4})^2]^{b_4}} \tag{5.4}$$

Table 5.1 shows the consequence and premise parameters of each rule.

Rule	*Consequence Parameters*	*Premise Parameters*
1	a1, b1, c1, a3, b3, c3	p1, q1, r1
2	a1, b1, c1, a4, b4, c4	p2, q2, r2
3	a2, b2, c2, a3, b3, c3	p3, q3, r3

4	a2, b2, c2, a4, b4, c4	p4, q4, r4

Table 5.1 : Consequence and Premise parameters of ANFIS as logical operators.

After the ANFIS is trained, the values of the parameters are shown in table 5.2.

Parameters	*Values*	*Parameters*	*Values*	*Parameters*	*Values*
a1	0.433637	*b1*	2.01019	*c1*	-0.0448426
a2	0.43755	*b2*	2.01062	*c2*	1.0338
a3	0.433637	*b3*	2.01019	*c3*	-0.0448426
a4	0.43755	*b4*	2.01062	*c4*	1.0338
p1	0.0349429	*q1*	0.0349429	*r1*	-0.0323391
p2	-0.0100705	*q2*	0.530827	*r2*	0.529259
p3	0.530827	*q3*	-0.0100705	*q3*	0.529259
p4	-0.0307884	*q4*	-0.0307884	*r4*	0.00209854

Table 5.2 : Values of consequence and premise parameters.

Figure 5.3 shows the output functions of each rule and the membership functions of A_1, A_2, B_1, B_2 . We found that there is almost no change on the membership functions with their initial setting. This can be explained by the few training epochs of ANFIS, the update of the premise parameters is very small within the few training epochs.

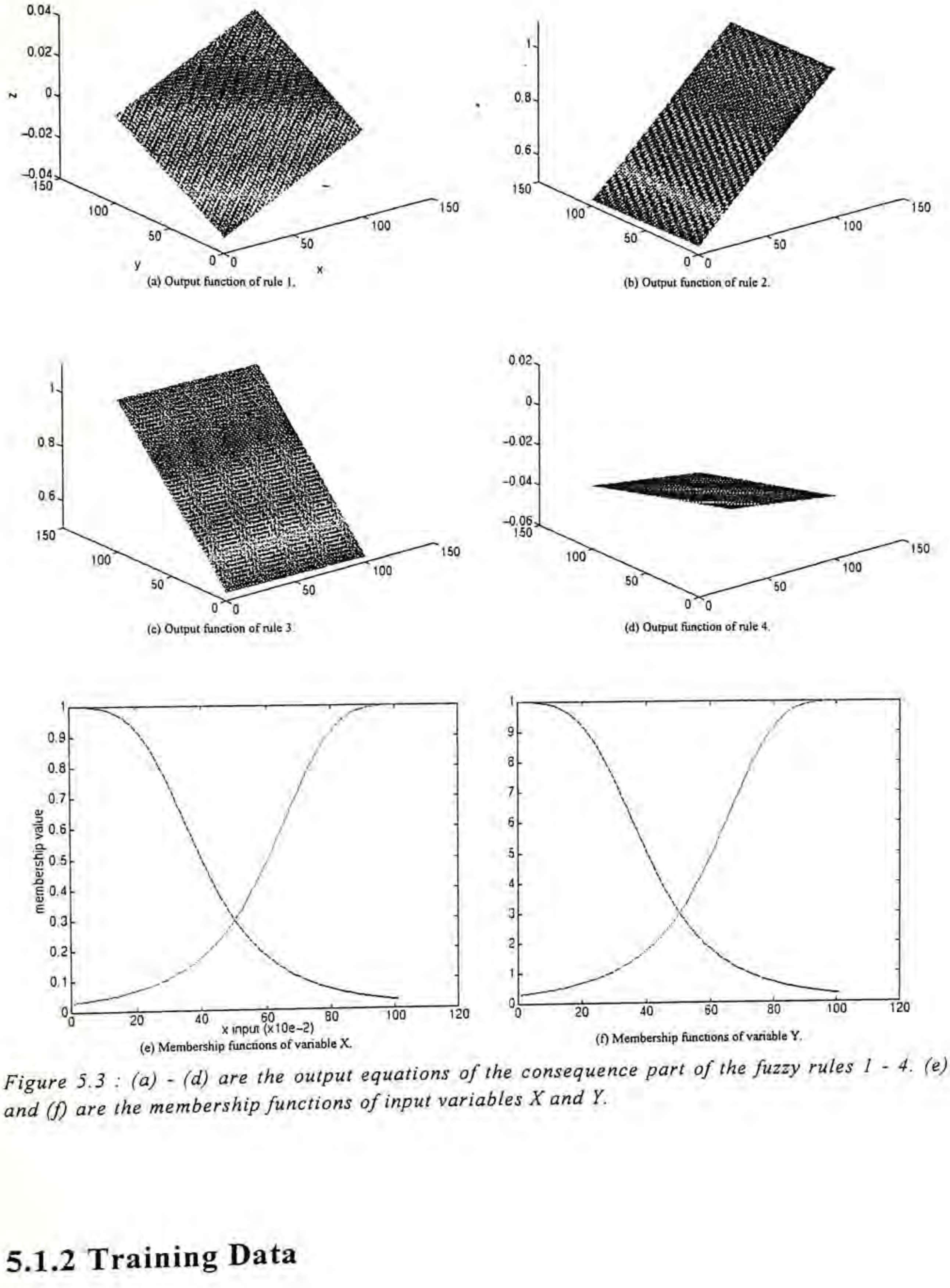


Figure 5.3 : (a) - (d) are the output equations of the consequence part of the fuzzy rules 1 - 4. (e) and (f) are the membership functions of input variables X and Y.

5.1.2 Training Data

The training data are the corresponding input/output values of the logical operators. Table 5.3, 5.4, 5.5 show the training data of the XOR, OR, AND respectively.

Training set	*Input 1*	*Input 2*	*Output*
1	1	1	0
2	1	0	1
3	0	1	1
4	0	0	0

Table 5.3 : Training data of XOR.

Training set	*Input 1*	*Input 2*	*Output*
1	1	1	1
2	1	0	0
3	0	1	0
4	0	0	0

Table 5.4 : Training data of AND.

Training set	*Input 1*	*Input 2*	*Output*
1	1	1	1
2	1	0	1
3	0	1	1
4	0	0	0

Table 5.5 : Training data of OR.

5.1.3 Results and Discussions

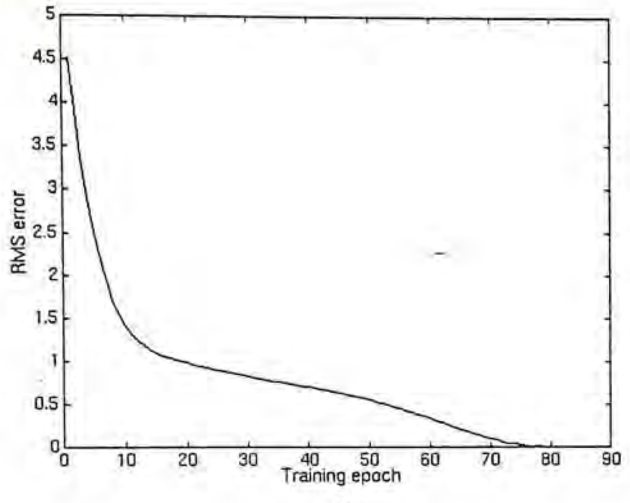
For the modeling of logical operators, ANFIS gives the remarkable results. Only one training epoch is needed to delivery nearly zero error. Moreover, ANFIS still give a correct output under the noise environment. The maximum noise level is 30% deviation from the original values of 0 or 1.

The membership functions of ANFIS can account for the noise toleration capability. Since the membership functions transfer the crisp inputs to the values of degree of matching, within certain range of deviation, the inputs still fall into the same fuzzy subspace, hence, ANFIS still give a correct output.

In fact, neural networks are usually used to model the logical operators. In this application, we found that ANFIS can also work as neural networks. Moreover, it gives a better result in the aspects of training time and noise toleration.

The LSE can account for this better result. When ANFIS is trained by the hybrid learning, it only need 1 training pass to achieve the RMS error less than 10^{-5} for the cases of XOR, AND and OR. However, when ANFIS is trained by gradient descent with adaptive step size and momentum. It requires 110 training epochs for XOR case, 98 training epochs for OR case and 102 training epochs for AND case.

Base on the gradient descent learning, we construct a neural network to simulate the logical operator AND, XOR and OR. The neural network has 3 layers, 2 nodes in input layer, 5 nodes in the hidden layer, 1 node in output layer. The neural network is trained by backpropagation algorithm with adaptive learning rate and momentum. The neural network needs 86, 129, 91 training epochs for AND, OR, XOR respectively. This result shows that with the gradient descent learning for modeling of logical operators, ANFIS is not necessary better than neural network and the remarkable result from ANFIS can be explained by the LSE in hybrid learning algorithm. The error curve of both ANFIS and neural network in the XOR case is shown in figure 5.4.



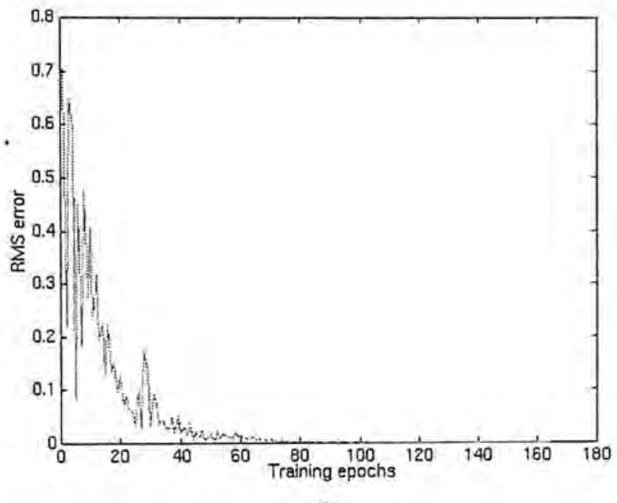


Figure 5.4 : (a) is the error curve for neural network training, (b) is the error curve for ANFIS with gradient descent training.

We have also try to increase the number of fuzzy terms (membership functions) for each input variables. When the number of fuzzy terms is 4, ANFIS still need 108 training epochs for XOR. The result shows that further increase the number of membership functions does not improve the performance. This can be explained by the fact that the optimal number of membership function is reached, further increase the number of membership only increase the computations, but not improve the performance.

5.2 Modeling of Nonlinear Function

The second application is the modeling the *sinc* function which is highly nonlinear in three-dimensional space. The *sinc* function is

$$z = sinc(x, y) = \frac{sin(x)}{x} \times \frac{sin(y)}{y}. \tag{5.5}$$

5.2.1 Architecture

Based on (5.5), the ANFIS has 2 input variables and 1 output variable. Since the *sinc* function is highly nonlinear, more membership functions are needed to give an accuracy result, provided the optimal number of membership functions is not reached. This can be explained by the functional equivalence between radial basis function networks and fuzzy inference systems. In radial basis function networks, the number of basis functions reflects the complexity of the mapping to be learned, the more the basis functions, the more complex the mapping[8][30]. Therefore, instead of using 2 membership functions for each input variable, we use 4 membership functions for each input variable.

5.2.2 Training Data

The range of the input variables is -10 to 10, the even values of the input variables are selected to be the training data. Based on this selection method, the training data can be evenly distributed in the input space. The output is calculated by the *sinc* function directly. Table 5.6 shows part of the training data.

Training Set	*x*	*y*	*z*
1	-10	-10	0.00296
2	-8	-10	-0.00673
3	-6	-10	0.00253
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
120	8	10	-0.00673

121	10	10	0.00296

Table 5.6 : Part of training data for sinc function modeling.

With this architecture, there are maximum 16 possible rules and 72 parameters which are composed of 24 premise parameters and 48 consequence parameters.

5.2.3 Results and Discussions

With the hybrid learning algorithm, ANFIS can simulate the *sinc* function within 150 training epochs, this remarkable result is achieved by adding the momentum to the gradient descent pass, so that ANFIS can escape from the local minimum. In this application, we can visualize how ANFIS perform its update on membership functions in its training, and also the power of using momentum in such kind of complicated problems. The training of ANFIS in this application can be classified in the following stages :

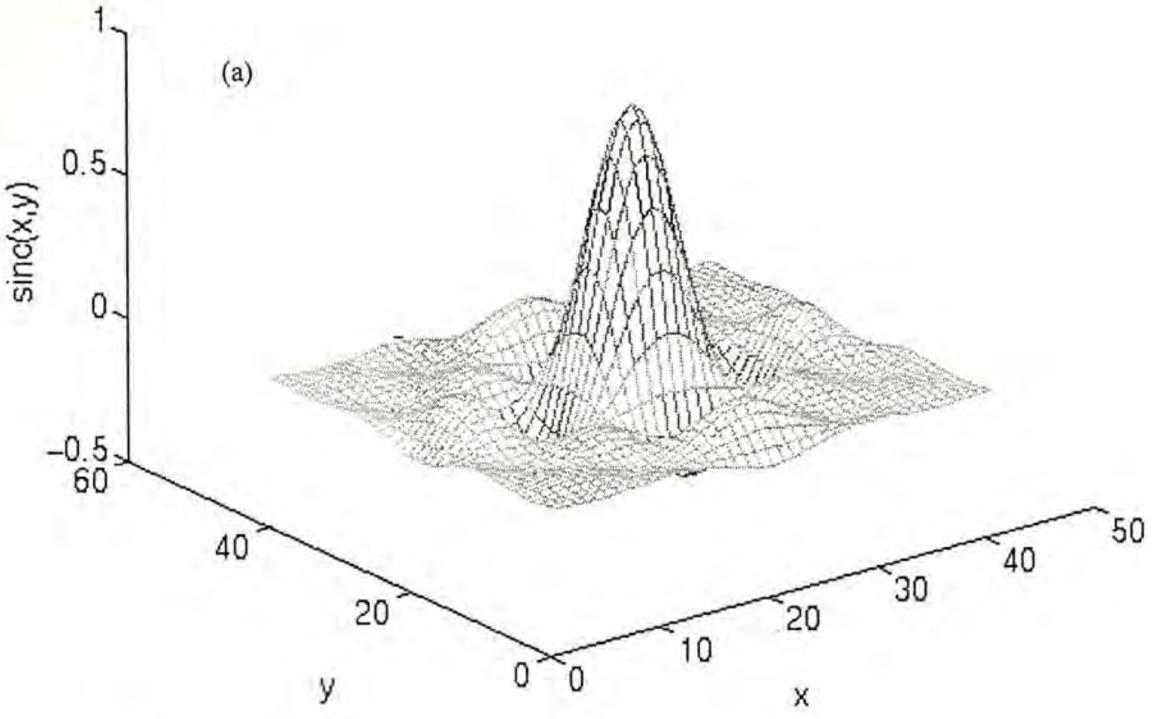
Stage 1 : At the beginning of the training, because of the higher amplitude of the sinc function around the origin, the RMS error is mainly contributed by that region (figure 5.6a). ANFIS shifts the centers of two membership functions to the origin to reduce the RMS error (figure 5.6b). As long as the centers of two membership functions shifting to origin, the firing strengths of the fuzzy rules with input variables around the origin will be higher. Base on (3.9), we found that if the firing strengths of the rules are higher, the effect of the corresponding outputs will be greater to the whole system. Hence, the

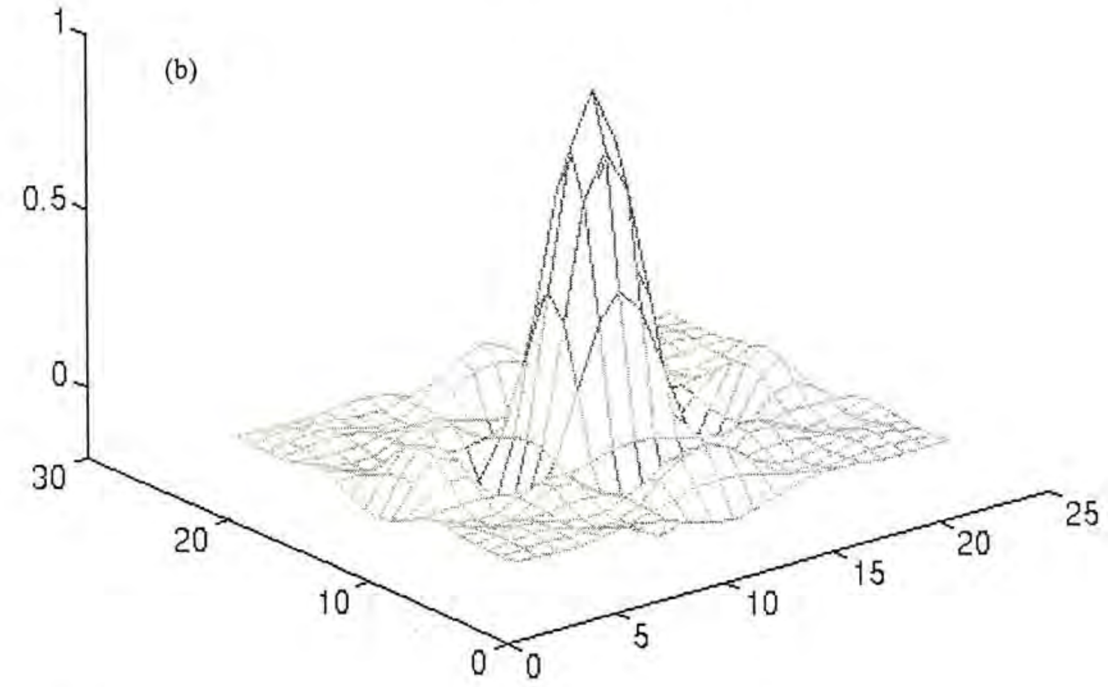
RMS error from the region of origin can be reduced, this also accounts for the rapid drop of RMS error at the beginning of training.

Stage 2 : After two membership functions overlap each other and are centered at origin. The RMS error is mainly due to the error from the side regions. Start from training epoch 15.5 on (figure 5.6b), ANFIS spends all of its effort to approximate regions that contributes most of the error and forgets other regions.

Stage 3 : As the training continue, the error will be mainly contributed by the side regions, and ANFIS will be trapped by a local minimum. With the momentum added, ANFIS can escape from the local minimum with the tradeoff of an increased RMS error (figure 5.6c).

Stage 4 : After escape from the local minimum, the distortion of the central region will be recovered in the subsequent training epochs (figure 5.6d). After 40 training epochs, the approximated surface is already very similar to that of the original one (figure 5.6e). The model will keep on refining the surface and after 150 training epochs, ANFIS gives the result of RMS error less than 0.0005 (figure 5.6f). The result is difficult to be achieved by neural networks even the networks are trained for a number of hours. Figure 5.5 shows the surface of the original continuous *sinc* function.


(a)
sinc(x,y)
1
0.5
0
-0.5
60
40
20
0
y
0
10
20
30
40
50
x


(b)
1
0.5
0
30
20
10
0
0
5
10
15
20
25

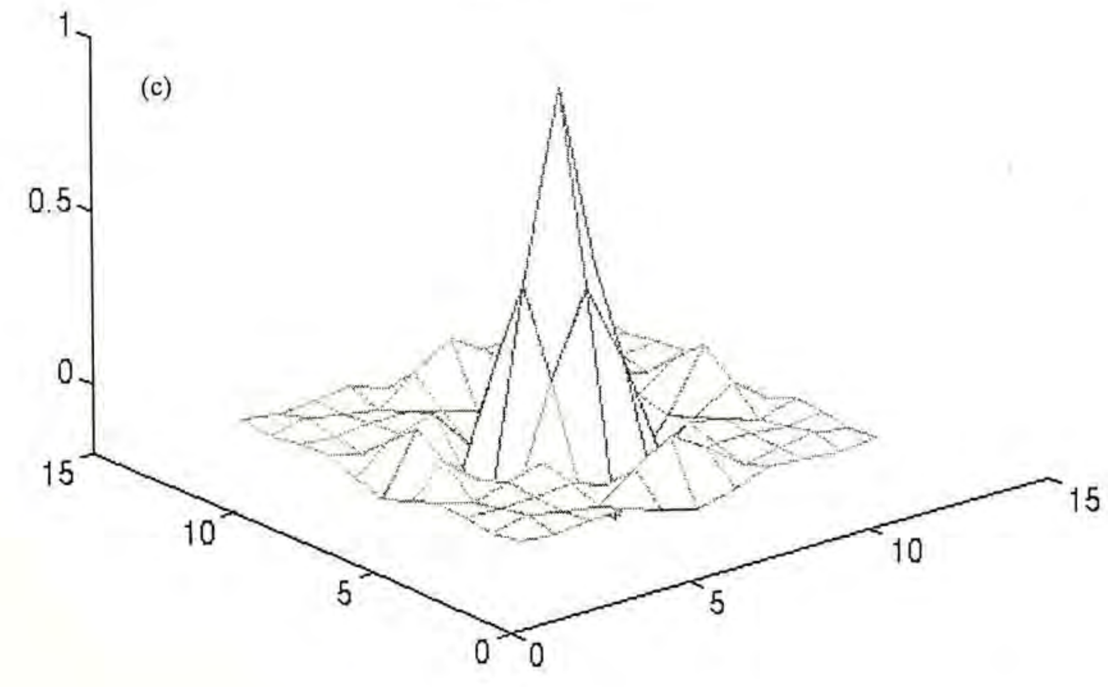
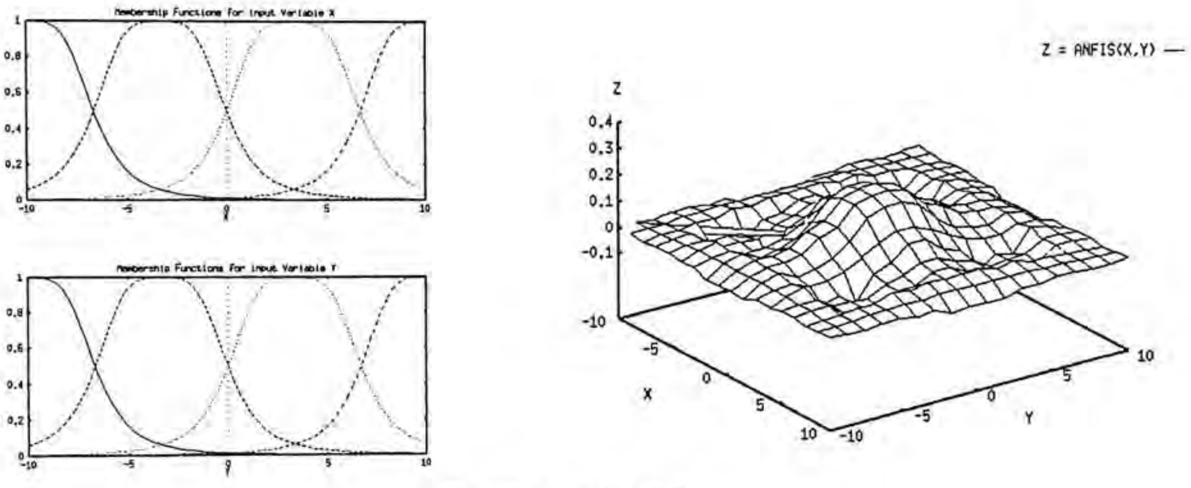
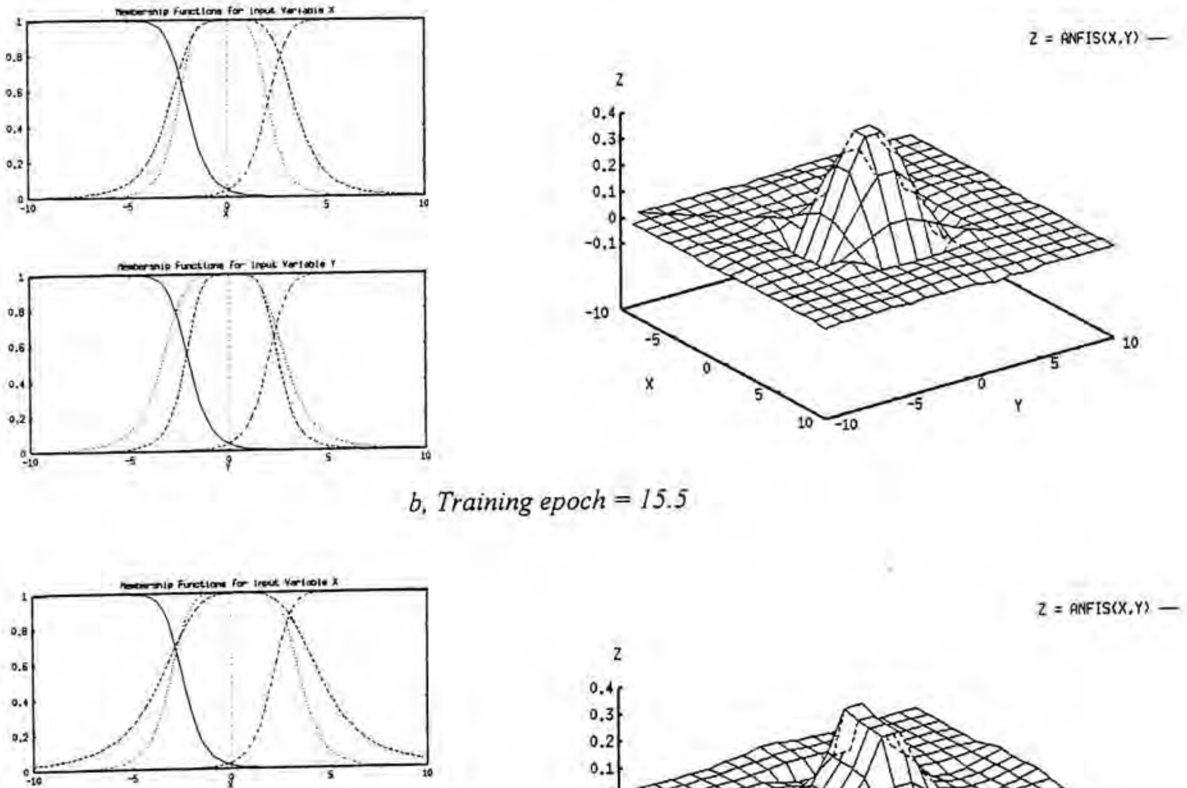

(c)
1
0.5
0
15
10
5
0
0
5
10
15

Figure 5.5 : The figure shows the surface of sinc function at different sampling step (ss). (a) ss = 0.5, (b) = 1, (c) = 2.

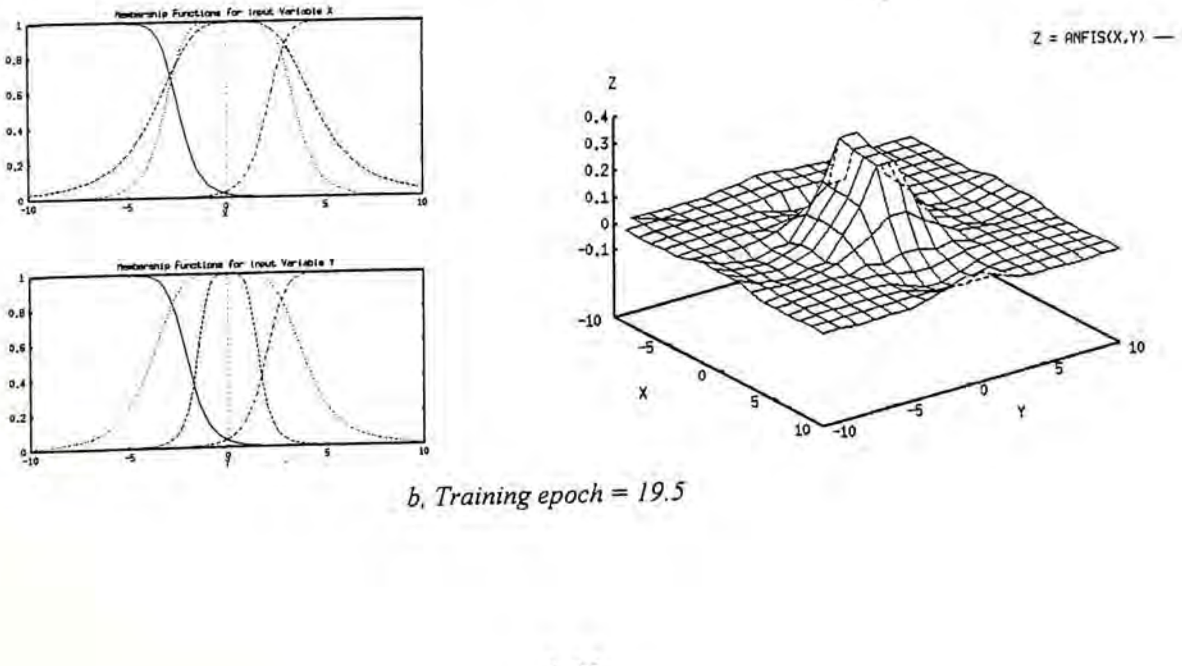
Figure 5.6 shows the memberships functions and reconstructed surface of ANFIS at different training epochs and the RMS error w.r.t training epochs.



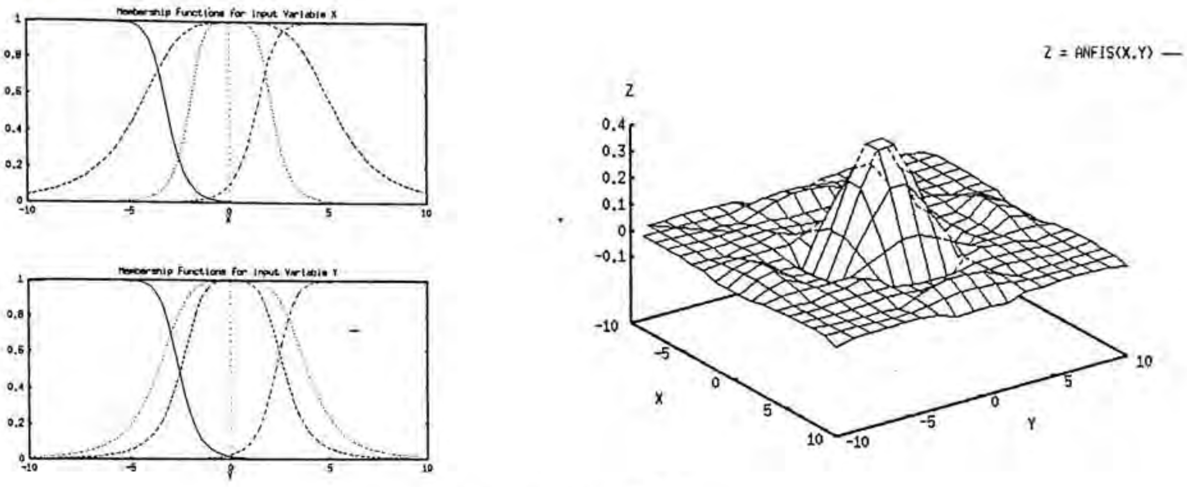
a, Training epoch = 0.5



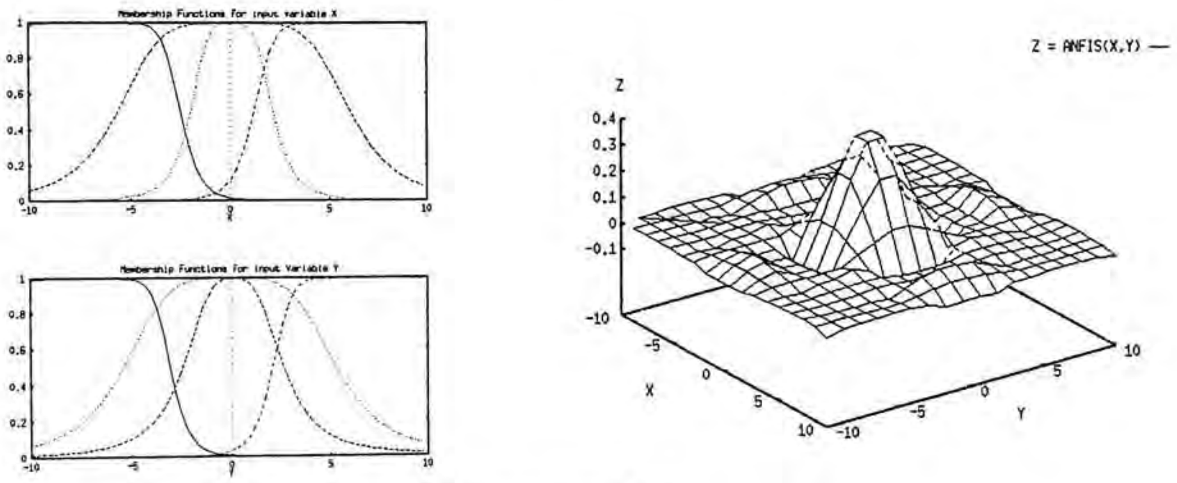
b, Training epoch = 15.5



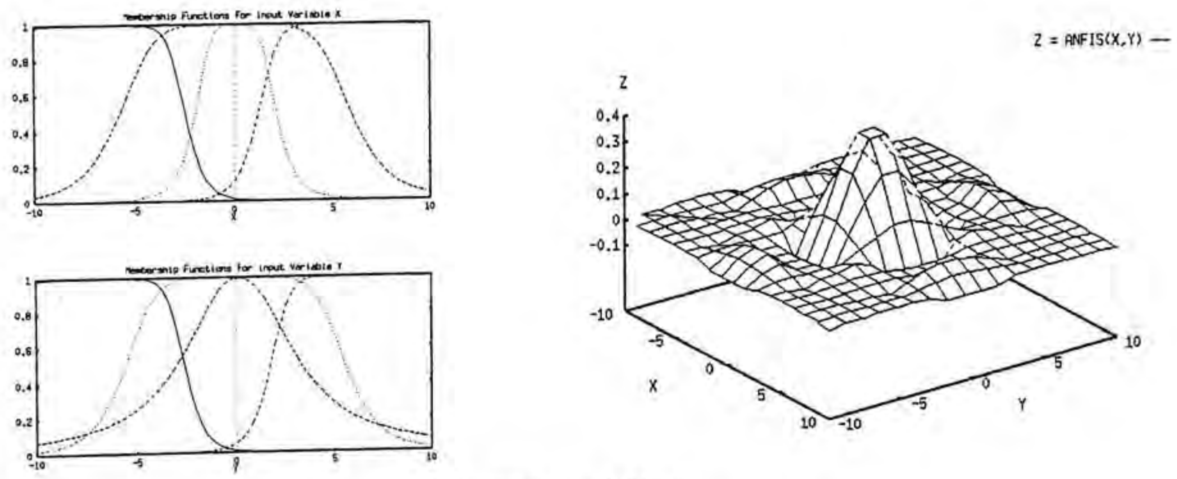
b, Training epoch = 19.5



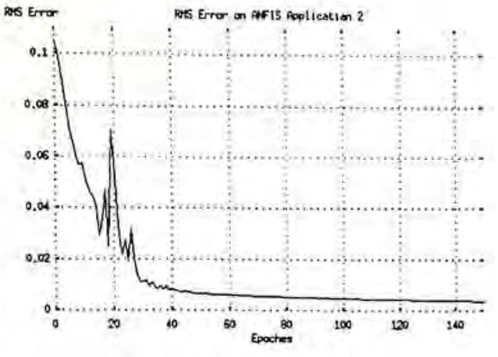
d, Training epoch 23.5



e, Training epoch 40.5



f, Training epoch 150.5



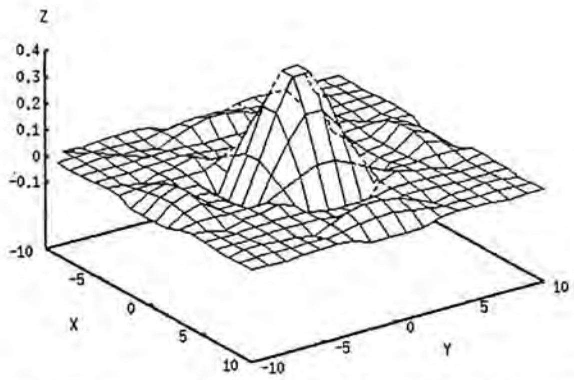


Figure 5.6 : The figure shows the membership functions of each variables and the reconstructed surface of ANFIS at different training epochs. The last two graphs are the error curve and original surface of sinc function for training.

When the momentum is removed from the hybrid learning, we found that ANFIS is not able to escape from the deepen local minimum and yields in a worse result. Figure 5.7 shows the error rate and reconstructed surface of ANFIS (without momentum) after 150 training epochs. We can see that the reconstructed surface is worse than that in figure 5.6. Thus, we can conclude that momentum can be used to assist the ANFIS training to escape from local minima.

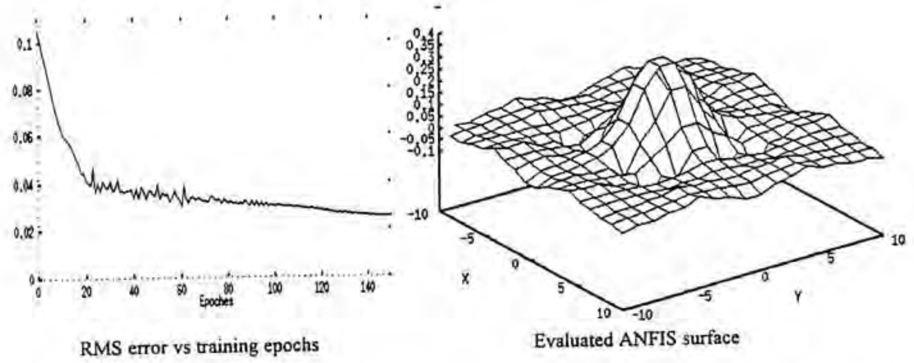


Figure 5.7 : The error curve and evaluated surface of ANFIS training without momentum.

When ANFIS is trained with gradient descent, training takes longer time. This can be explained by the size of searching space. In gradient descent learning, the consequent parameters are also tuned by gradient descent, hence, the searching space is very large and more time (training epochs) is needed. It should be noted that, the searching space increased exponentially with respect to the parameters increasing. Although, ANFIS takes more time in gradient descent training scheme, it still give a better result than neural network. Figure 5.8 shows the final reconstructed surface of ANFIS and neural network after 10000 training epochs. The RMS error of ANFIS is 0.0015 and that of neural network is 0.0022. Similar as the ANFIS, the neural network is trained with momentum and adaptive learning rate.

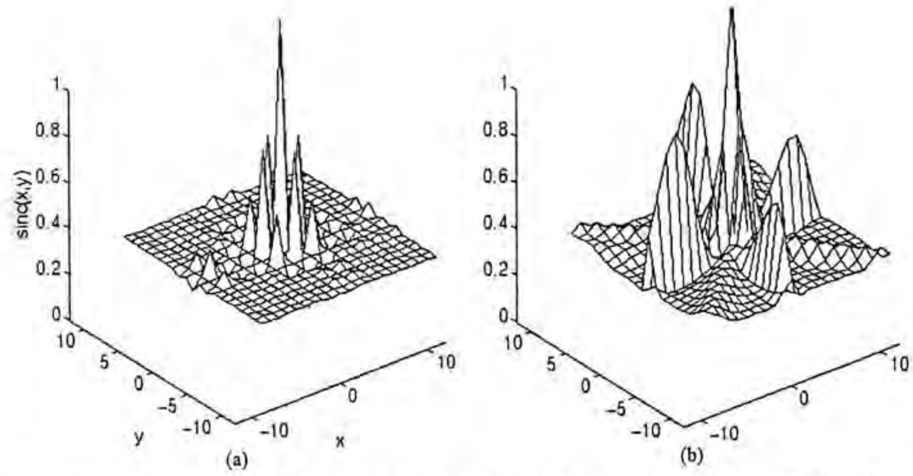


Figure 5.8 : Reconstructed surface of sinc function. (a) From ANFIS trained with gradient descent only. (b) From Backpropagation Neural Networks

5.3 Chaotic Time Series

In the above application, we have shown that ANFIS is able to model a highly non-linear function. This application aims to demonstrate the predictive capability of ANFIS which was used to predict the future values of chaotic time series.

The chaotic time series is a benchmark problem[13] because it's period is non-fixed, moreover, the change of initial parameters can modify the behaviors of the series. In this application, the time-series is based on the chaotic Mackey-Glass differential delay equation defined below :

$$\dot{x}(t) = \frac{0.2x(t-r)}{1+x^{10}(t-r)} - 0.1x(t). \tag{5.6}$$

To find the solution of equation (5.6), we apply the Runge-Kutta method to find the numerical solution to the equation[13].

The objective of this application is using the past *N* data values up to the point *t* to predict the future value at point *t+D*. In this application, we use 4 past data values to predict to future value. We will discuss the training data in the subsequent section.

5.3.1 Architecture

As mentioned before, we use 4 past data values for prediction, that means we have to use 4 input variables. Each input variable was assigned with 2 membership functions arbitrarily. There are totally 16 rules in the system and 104 training parameters, of which 24 are premise parameters and 80 are consequent parameters.

5.3.2 Training Data

Suppose the current time stamp is t, the four input data are *x(t-6)*, *x(t-12)*, *x(t-18)* and *x(t – 24)*. The predictive value is *x(t)*. Each data value is separated by 6.

The initial conditions are *x(0)=1.2* and *r=17*. From the Mackey-Glass time series, 1000 training data are extracted. The time stamp of the training data is from *t=124* to *t=1123*. It is supposed that the data values before *t=124* are known and are used to train the data from *t=124*.

The first Ψ (from $t=124$ to $t=124+\Psi$) data values are used to train the ANFIS, the rest are used to test the accuracy of the prediction. For different test cases, the values of Ψ are different.

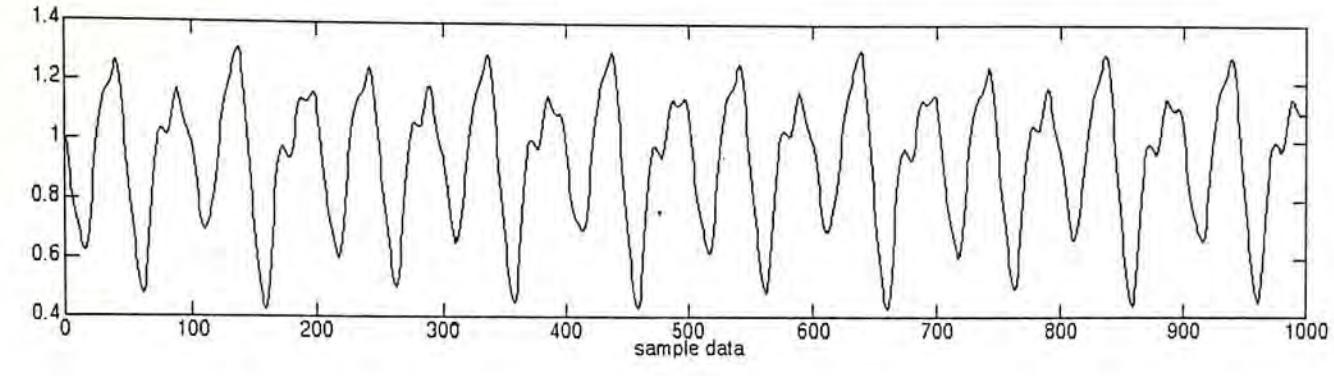
5.3.3 Results and Discussions

A number of cases have been tested to evaluate the predictive power of ANFIS. For each test case, the number of training data is different. The parameter of each test case are shown in the table 5.7.

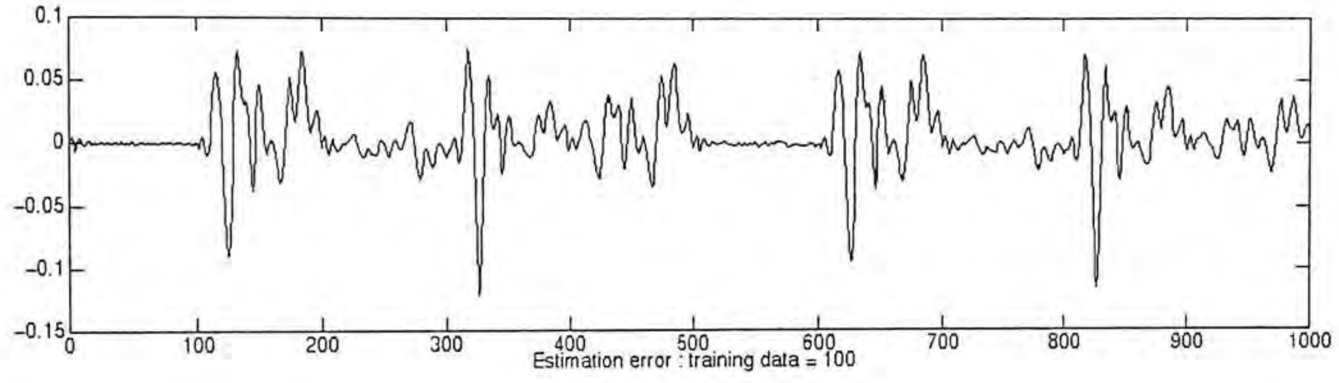
Test cases	Number of training data
1	100
2	200
3	300
4	400
5	500

Table 5.7 : The number of training data for each test case.

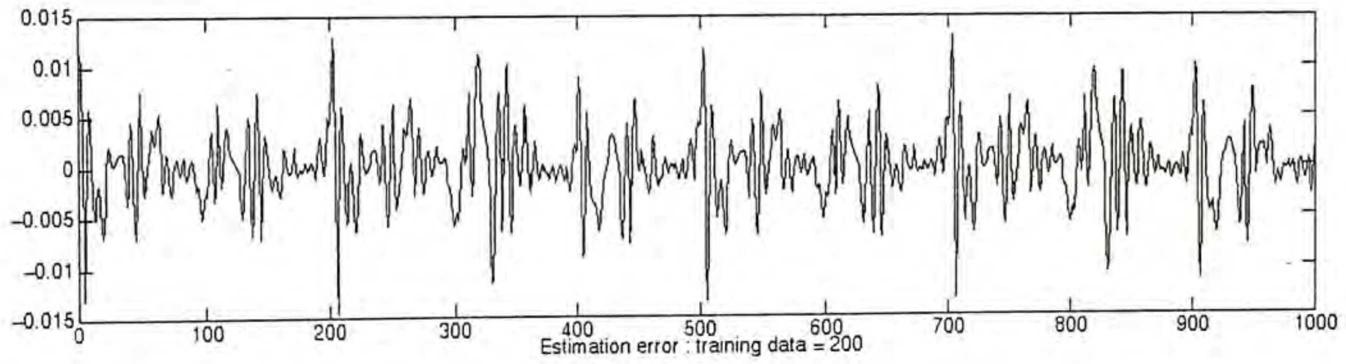
Figure 5.9 shows the chaotic time series and the estimation error each test case.



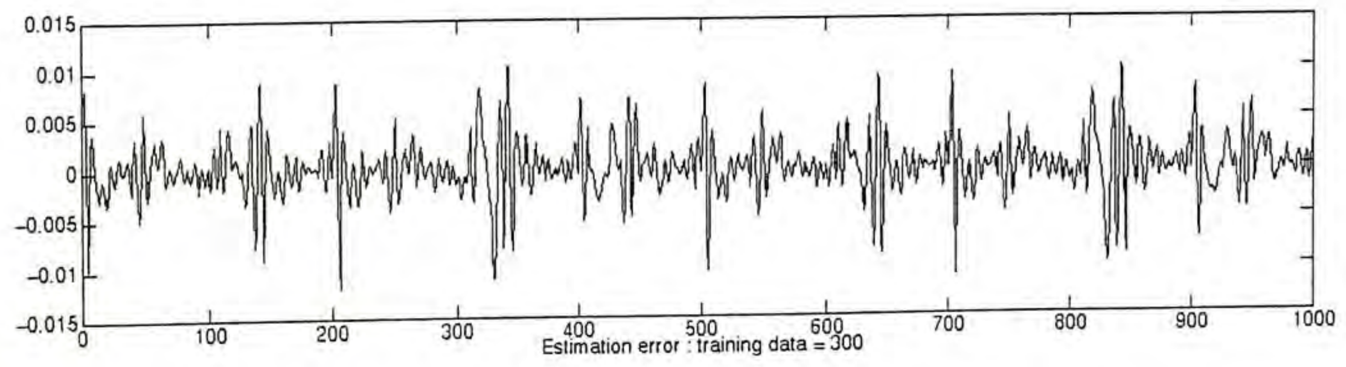
sample data



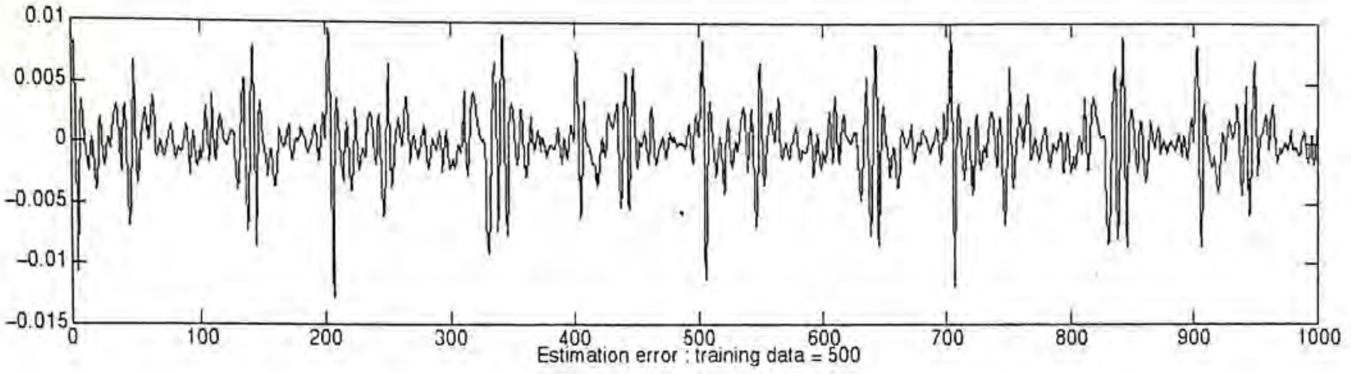
Estimation error : training data = 100



Estimation error : training data = 200



Estimation error : training data = 300



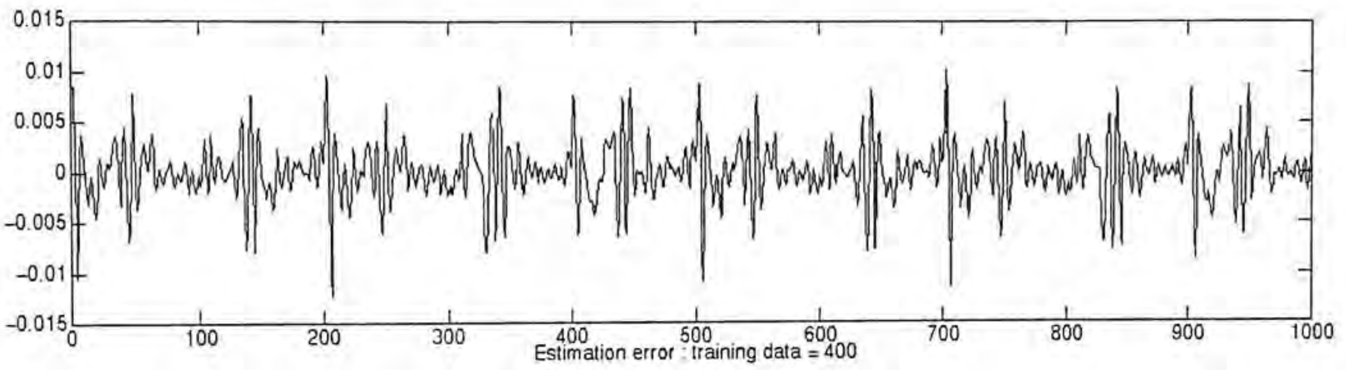


Figure 5.9 : Chaotic time series and the estimation error in each test case.

Note that the series is approximated very accurately and there is visually no difference between the two. Also, the RMS error can only be shown on very fine scale. The application shows that ANFIS give outstanding performance in predicting the chaotic time series even though series highly complex. We can conclude that ANFIS can be used to model highly complexity system.

5.4 Inverted Pendulum System

In this application, ANFIS is used to construct a simple fuzzy controller through the use of temporal back propagation which means to apply the back-propagation-type gradient descent method to propagate the error signals through different time stages[14]. The controller has to keep the state variables of the system to follow a given desired trajectory as close as possible. The whole system contains a fuzzy controller which based on the current values of the state variables to determine the action to be taken, and also a plant calculates the values of the state variables based on the action taken by the controller. The basic idea of our application is to implement both the fuzzy controller and the plant at each time stage as a stage adaptive network, and cascade these stage adaptive networks into a trajectory adaptive network to facilitate the temporal back propagation learning process.

The inverted pendulum system (figure 5.10) is composed of a rigid pole and a cart on which the pole is hinged. The cart moves on the rail tracks to its right or left, depending on the force exerted on the cart. The pole is hinged to the cart through a frictionless free joint such that it has only one degree of freedom. The control goal is to balance the pole starting from non-zero conditions by supplying appropriate force to the cart.

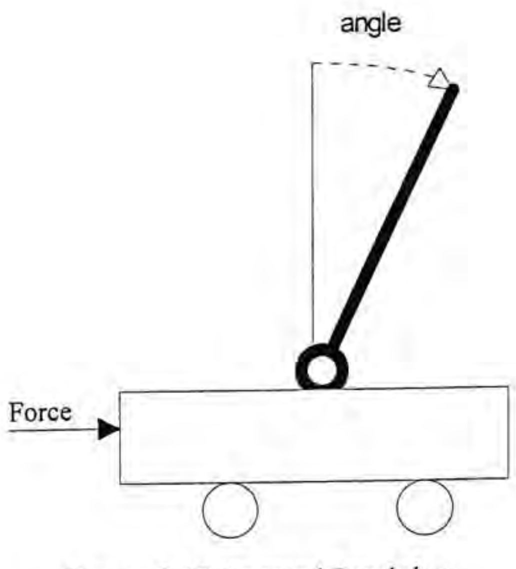


Figure 5.10 Inverted Pendulum

The dynamics of the inverted pendulum system are characterized by two state variables: θ (angle of the pole with respect to the vertical axis), $\dot{\theta}$(angular velocity of the cart). The behavior of these two state variables is governed by the following differential equations[14]:

$$\ddot{\theta} = \frac{g * \sin\theta + \cos\theta * \left(\frac{-F - m * l * \dot{\theta}^2 * \sin\theta}{m_c + m} \right)}{l * \left(\frac{4}{3} - \frac{m * \cos^2\theta}{m_c + m} \right)} \tag{5.7}$$

where g is acceleration due to gravity, m_c is the mass of the cart, m is the mass of the pole , l is the half-length of the pole and F is the applied force in newtons. Our control goal here is to balance the pole without regard to the cart's position and velocity.

5.4.1 Stage Adaptive Network

Figure 5.11 shows a block diagram of a feedback control system consisting of a fuzzy controller and a plant. We assume the delay through the controller is small and the state variables are accessible with accuracy.

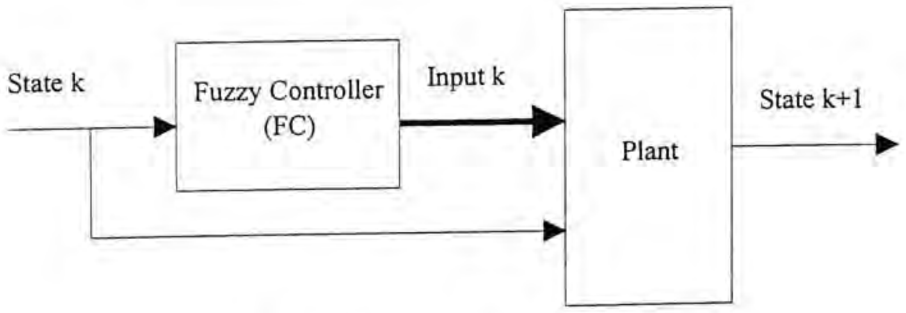


Figure 5.11 : feedback control system

An obvious candidate for implementing the FC block in the figure is the ANFIS architecture, since it has exactly the same function as a fuzzy controller. If we have p inputs to the plant, then the FC block can be implemented by our modified ANFIS with p outputs (section 3.2.6). For the implementation of the plant block, ANFIS is also used, because it has a model-insensitive attribute which make it able to represent the input-output behavior of the plant. Consequently, the block diagram of figure 5.11 can also be viewed as an adaptive network containing 2 subnetworks, the FC block and the plant block. Subsequently, the adaptive network of figure 5.11 is referred to as SAN_k which represents the stage adaptive network[14] at time stage k.

5.4.2 Trajectory Adaptive Network

Given the state of the plant at time $t=k+h$, the FC will generate an input to the plant and the plant will evolve to the next state at time $(k+1)\times h$. By repeating this process starting from $t=0$, we obtain a plant state trajectory determined by the initial state and the parameters of the FC. The state transition from $t=0$ to $m\times h$ is show conceptually in figure 5.12 in which the adaptive network consisting of m SAN_k*'s*, $k=0$ to $m-1$ is called Trajectory Adaptive Network (TAN)[14].

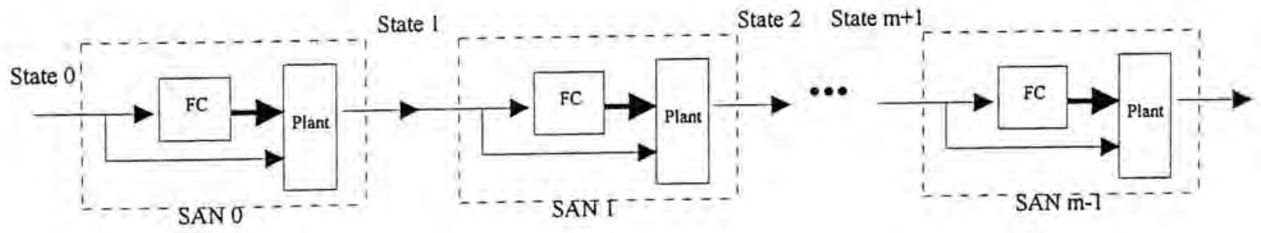


Figure 5.12 : Trajectory Adaptive Network

Accordingly, we can still apply the back-propagation gradient descent[1] to minimize the differences between adaptive network outputs and desired outputs. In order to make the inputs and outputs more explicit, we redraw figure 5.12 to get the trajectory adaptive network shown in figure 5.13, where the inputs to the network

[1] LSE is not applicable in temporal back propagation.

are the initial state of the plant at time = 0, the outputs of the network are the state trajectory from $t=h$ to $m\times h$, and the adjustable parameters are all pertaining to the FC block implemented as an ANFIS. Hence each entry of the training data is of the form : *(Initial State; Desired Trajectory)*, and the corresponding error measure to be minimized is

$$E=\sum_{k=1}^{m}\|\overset{\rho}{x}(h\times k)-\overset{\rho}{x}_d(h\times k)\|^2+\lambda\sum_{k=0}^{m-1}\|\overset{\rho}{ih}(h*k)\|^2, \tag{5.8}$$

where $\overset{\rho}{ih}(h\times k)$ is the controller's output at time $h\times k$[14]. By a proper selection of λ, a compromise between trajectory error and control effort can be obtained. Though there are m FC blocks, all of them refer to the same fuzzy controller at different time stages. In other words, there is only one parameter set for all m FC blocks at different time stages.

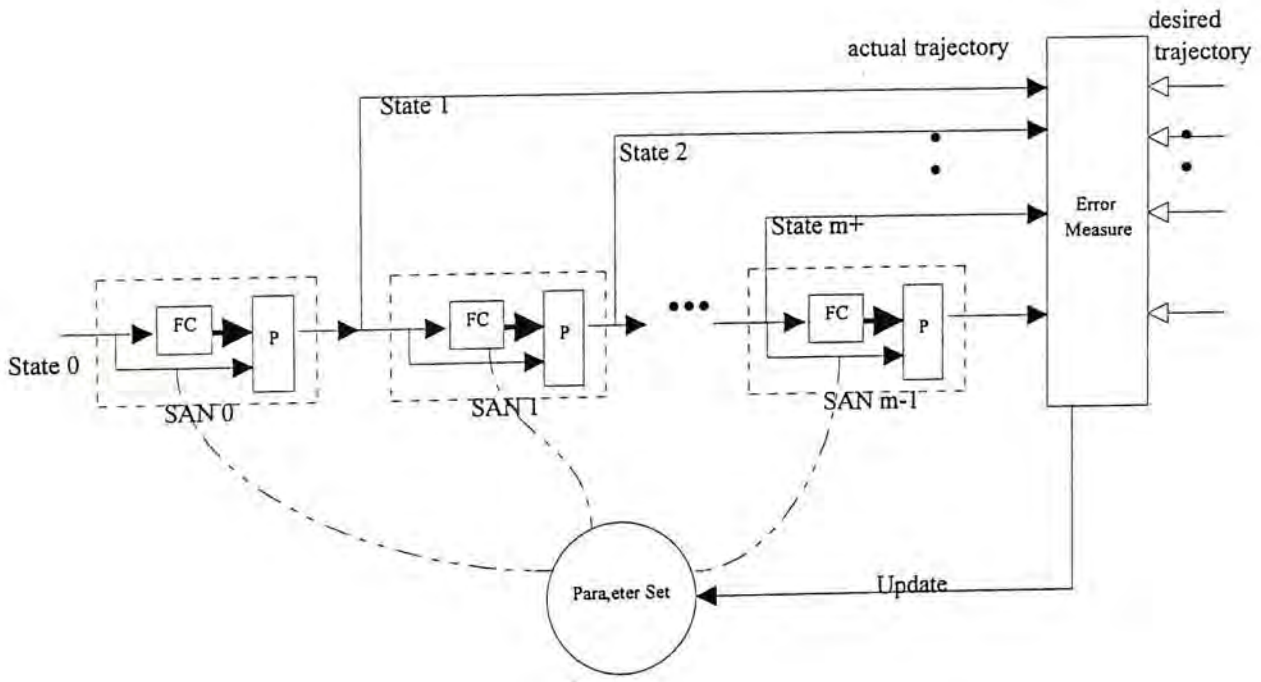


Figure 5.13 modified trajectory adaptive network

5.4.3 Architecture and Training data

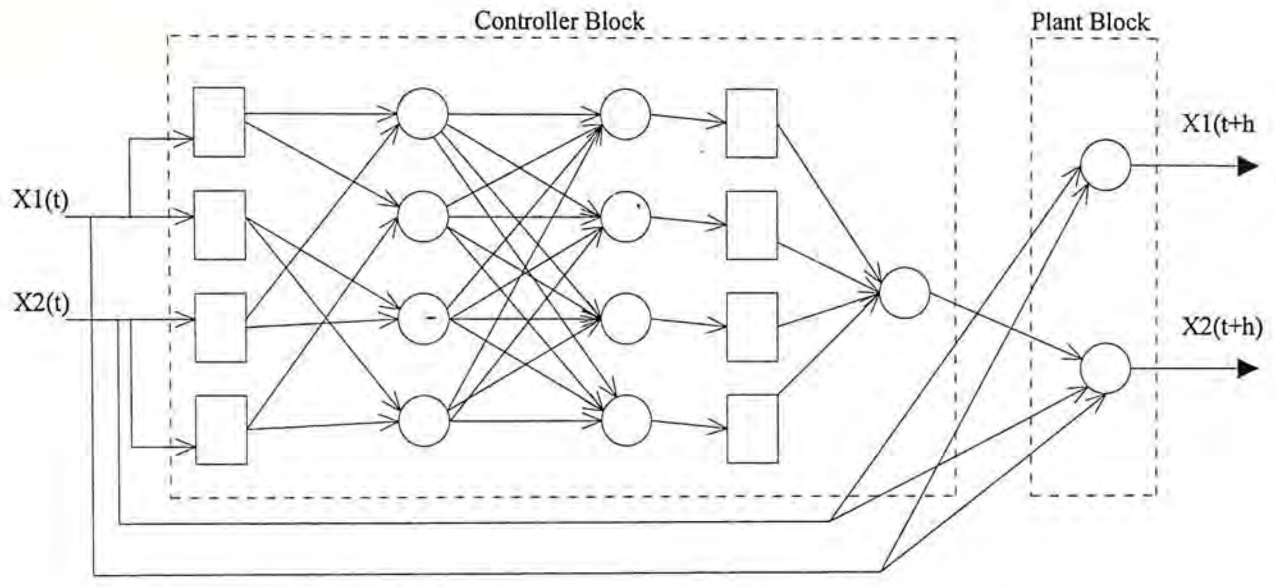


Figure 5.14 Stage adaptive network used in the simulation

Figure 5.14 shows the stage adaptive network used in our simulation. The plant is a deterministic nonlinear dynamic system with precisely defined differential equations, so we can just use two nodes to calculate the state variables at the next time step by linear approximation[14]:

$$\begin{cases} x_1(t+h) = h\dot{x}_1(t) + x_1(t) \\ x_2(t+h) = h\dot{x}_2(t) + x_2(t) \end{cases} \tag{5.9}$$

where $x_1(\cdot) = \theta(\cdot),\ and\ x_2(\cdot) = \dot{\theta}(\cdot)$. These two equations are the node functions of the plant block in figure 5.11.

The controller is implemented as an ANFIS with two inputs, each of which is assigned with two membership functions, so it is a fuzzy controller with only four fuzzy rules.

In the training we employ 100 stage adaptive networks to construct the trajectory adaptive network, and each stage adaptive network corresponds to the time transition of 10ms. That is, the time step (*h*) used is 10ms, and the trajectory adaptive network corresponds to a time interval from *t=0* to *t=1*s. If *h* is too small, a large network has to be built to cover the same time span, which increases the signal propagation time and thus delays the whole learning process. On the other hand, if *h* is too big, then the linear approximation of the plant behavior may not be precise enough and a higher order approximation has to be used instead. For the training data pair, the initial states is a two-element vector which specifies the initial condition of the pole while the desired trajectory is a 100-element vector which contains the desired pole angle at each time step. In our simulation, only two entries of training data are used : the initial conditions are (10,0) and (-10,0), respectively, and the desired trajectory is always a zero vector. That is, we expect that the trajectory adaptive network not only can learn to balance the pole from an initial pole angle of +10° or -10°, but also can achieve the control goal in an near-optimal manner which minimizes the error measure.

5.4.4 Results and Discussions

The result of this application is very attractive. In the first place, ANFIS is trained with 10 epochs, then the system bases on the parameters from the training set to evaluate other un-trained checking data. It is amazing to observe that the FC is able to balance the pole in a very short time and also without any serious oscillation around the vertical as shown in Figure 5.15.

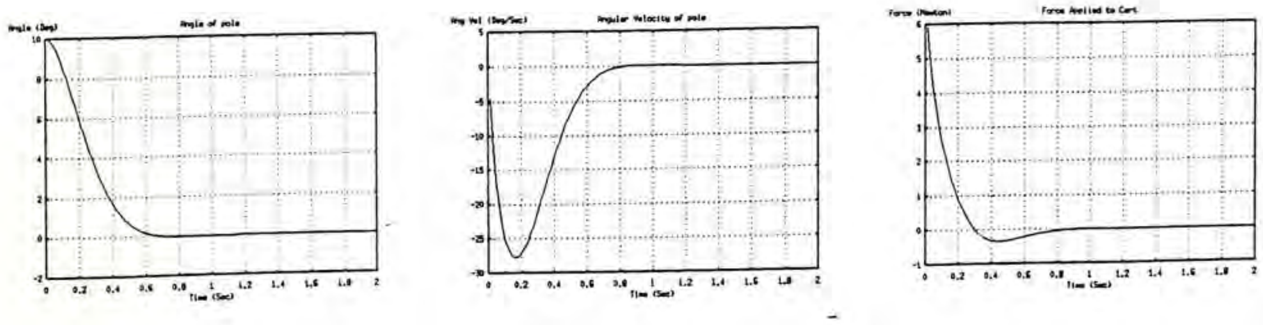


Figure 5.15 Result of simulation (a) angle (b) angular velocity (c) force applied

In the learning phase, we supply only two training data, corresponding to initial conditions (10, 0) and (-10, 0) of the pole. It would be interesting to know how the FC deals with other initial conditions, or with a longer or shorter pole. Simulations have been performed around these test cases, and the result is also very good. Again, the same fuzzy controller can perform the control task starting from the various initial conditions. Figure 5.16 and figure 5.17 shows the robustness and fault tolerance of the fuzzy controller obtained from the TBP ANFIS.

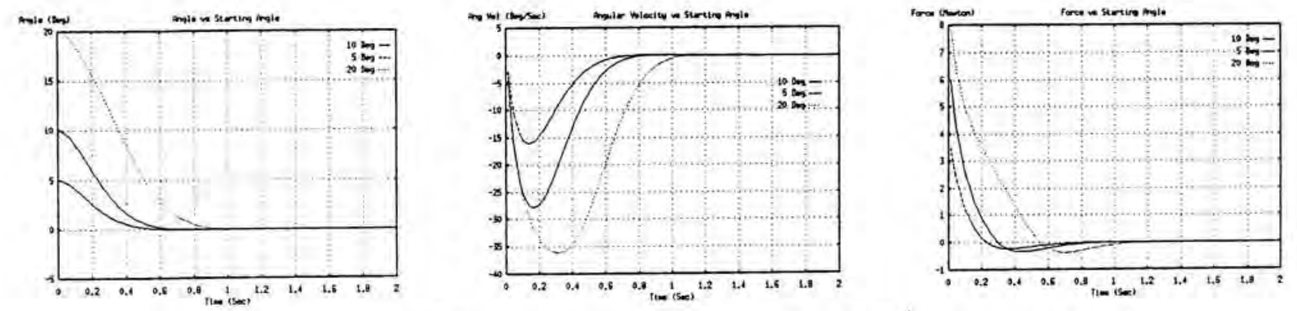


Figure 5.16 : Result of controller for simulation with different starting angle.

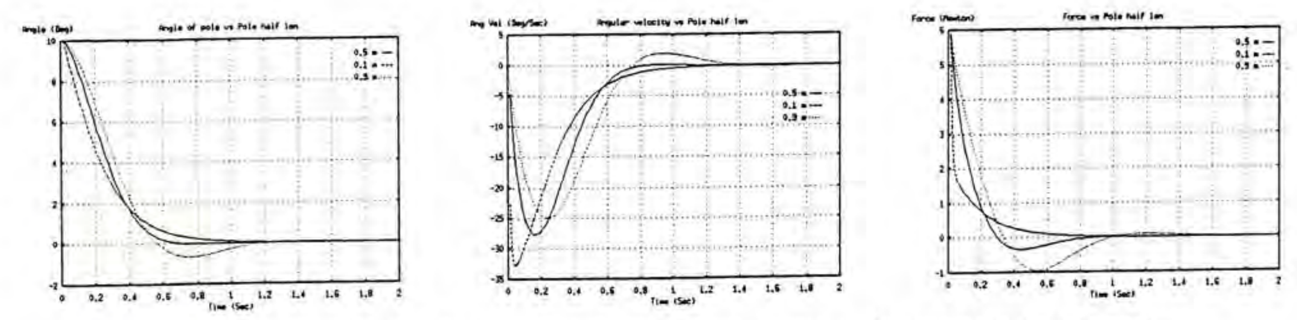


Figure 5.17 : Result of controller for simulation with different pole half length.

One thing should be noted is that the proposed control strategy is quite general and can be used to control plants with diverse characteristics. Moreover, the *a priori* knowledge that we have about the plant can be applied in an auxiliary manner to speed up the learning process, this is one of the strengths of ANFIS over the traditional neural networks.

5.5 Chinese Characters Recognition

The objective of this application is to recognize 300 printed 24×24 Chinese characters bitmap under noise based on the ANFIS. Unlike the previous applications which accept the input values directly, we have to transform the input before the input data are used by ANFIS. Transformation of the input data means to create an input vector from the input data, such input vector will be processed by ANFIS.

For instances, a Chinese character is represented by a 24×24 bitmap, we cannot take the whole bitmap as training vector for ANFIS because its dimension is too large[2]. Instead of using the bitmap as the input vector, we have to transform (preprocess) the bitmap into another form which has smaller dimension.

5.5.1 Feature Extraction

In fact, the transformation of the bitmap is the process of feature extraction. Feature extraction is a process to extract the information of the image (e.g. no. of pixels). This information is very important to the image. Based on this information, the original character can be identified, or at least, reduce the search space, so that the search of the character can be done easily and quickly.

[2] *If the whole bitmap is used, the input vector will have dimension of 576 which is very high dimension.*

In this application, signal transformation techniques are proposed as the method of feature extraction. Transformation is to transfer the variable (information) from one domain to another domain. After the transformation, small region of the frequency domain carries most information of the original signal. In other words, small region of the frequency domain information can be used to restore most part of the original signal. Actually, such signal transformation techniques are frequently used in signal analysis.

Beside one dimension signal, this signal transformation technique can be applied to two dimension signal, such as image. Instead of using 2D-Fourier Transform, we use another transformation technique which is called Discrete Cosine Transformation (DCT). The reason is that the result of fourier transform has two components (real and imaginary), for each frequency, two components have to be considered. This will double the dimension of input feature vectors.

Discrete Cosine Transformation (DCT) is similar to 2D Fourier Transformation. It can represent most of the signal (image) information in the low frequency components while there is only one component (real) in frequency domain.

Since most of the information of the original image is represented by the low frequency components in the frequency domain. Even we ignore the high frequency components, by using the Inverse DCT, we can still construct the original image with some distortions. Therefore, DCT can also be a method of feature extraction. Moreover, the size of the dimension of feature vector will not be doubled because the frequency domain of DCT has only one real component.

We can implement the DCT directly based on the following equation[31] :

$$C(u,v) = \alpha(u)\alpha(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\cos\left[\frac{(2x+1)u\pi}{2N}\right]\cos\left[\frac{(2y+1)v\pi}{2N}\right]. \qquad (5.10)$$

The inverse DCT can be done by using the following equation[31] :

$$f(x,y) = \sum_{u=0}^{N-1}\sum_{v=0}^{N-1} \alpha(u)\alpha(v)C(u,v)\cos\left[\frac{(2x+1)u\pi}{2N}\right]\cos\left[\frac{(2y+1)v\pi}{2N}\right]. \qquad (5.11)$$

where $\alpha(u)\begin{cases}\sqrt{1/N} & \text{for u} = 0 \\ \sqrt{2/N} & \text{for u} = 1,2,\ldots,\text{N - 1}\end{cases}.$

Although DCT only produce real number in frequency domain. there is still a problem for the direct implementation of DCT. For a 24×24 Chinese character, each pixel is either on or off. There is no gray level for each pixel. This characteristic will lead to DCT produce high amplitude in high frequency components.

Since we want to minimize the number of dimensions of feature vectors, we shall select the low frequency components for constructing the feature vectors. Hence, it will be better for the feature vectors representation if most of the information shift from high frequency components to low frequency components.

In order to reduce the amplitude of high frequency components, we try to form a grid of 2×2 pixel and provide a gray scale to each grid. For each grid, it can have 5 gray level (0-4). The gray level is assigned according to the number of "on" pixels in a grid. This method is called "Gray Level Preprocessing (GLP)". After GLP, the original word pattern is still conserved. Figure 5.18 shows the Chinese character before and after GLP.

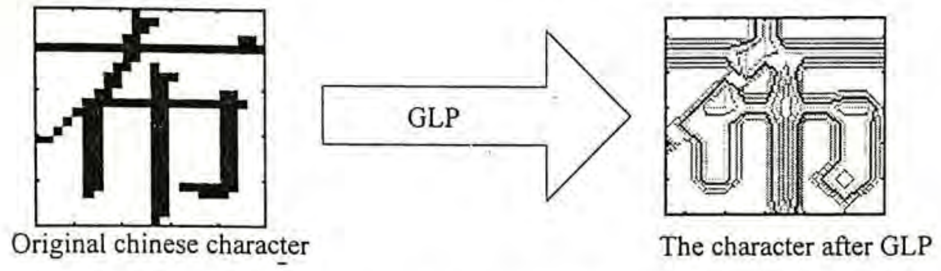


Figure 5.,18 : A chinese character before and after GLP, notice that after GLP the size of the character is 12x12 with each pixel ranges from 0 to 4. The character after GLP is shown as a contour picture.

An experiment has been conducted on comparing the direct implementation of DCT and DCT with GLP. The experiment selected 100 Chinese characters randomly, and evaluated the distribution amplitude in frequency domain for each of the above approaches. The result shows that the DCT with GLP can greatly reduce the amplitude in high frequency components.

In order to decide which frequency components should be used in the feature vectors, another experiment is constructed to evaluate the distribution of amplitude in each frequency component. Figure 5.19 shows the result of this experiment.

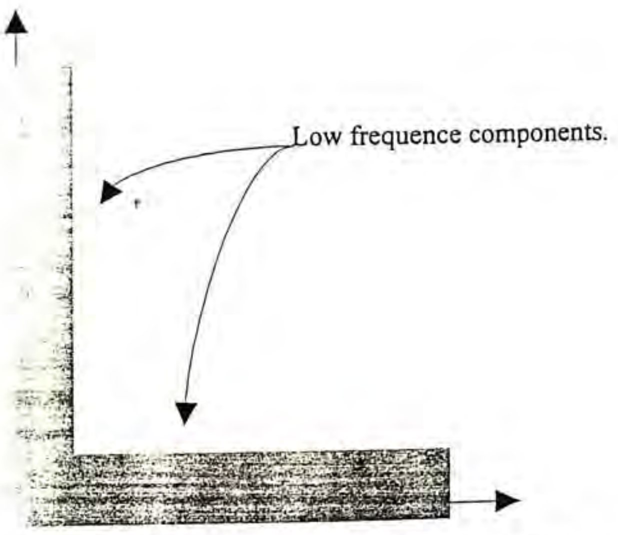


Figure 5.19 : The low frequency components contain the most DCT energy

Based on the result of the experiment, the frequency components f(0,0) to f(0,11) and f(1,0) to f(11,0) are used to constructed the feature vector. Figure 5.20 shows the DCT of a Chinese character.

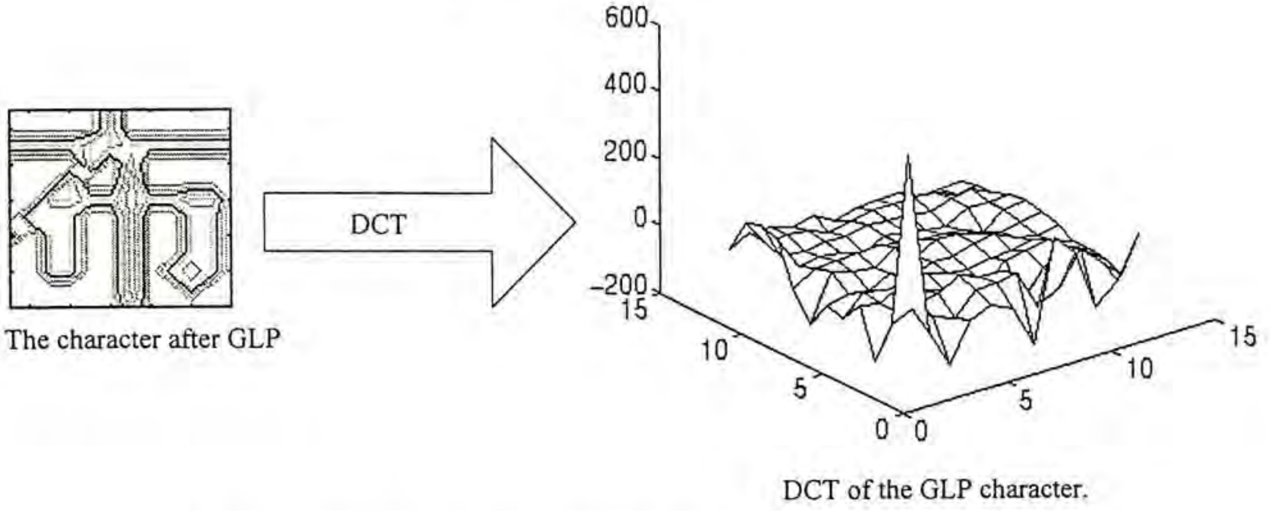


Figure 5.20 : The figure shows the DCT of a GLP character.

Although the DCT is already have the noise[3] filtering capability, a noise filter is still embedded in Feature Extraction process in order to improve the performance of the system.

In order to reduce the noise, a noise filter is added before the DCT is applied on the characters. After the gray level is assigned to the grids, for those grid with gray level 1. its gray level will be changed to 0. The result shows that the DCT with GLP gives more than 10% improvement over the system without GLP under the noise level[4] 0 to 10. Figure 5.21 shows a noise filter removing the noise from a GLP character.

---

[3] *Noise refers to the while noise which change the states of pixels randomly. Under the noise environment, some "on" pixels may change to "off" and vice verse.*

[4] *Noise Level refers to the percentage of the bitmap pixels of the words that have changed states.*

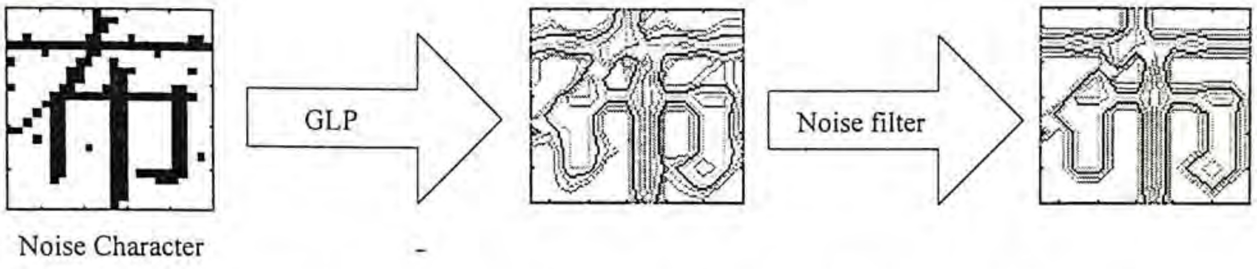


Figure 5.21 : The figure shows how the noise filter removing noise.

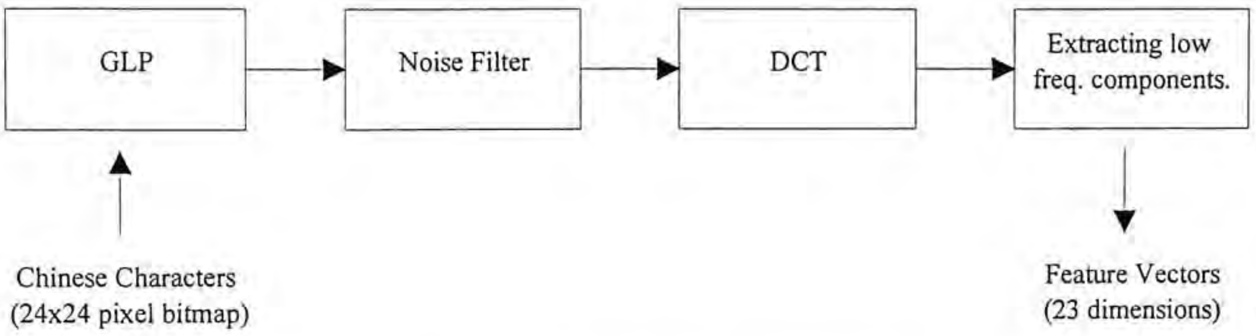

GLP
Noise Filter
DCT
Extracting low freq. components.
Chinese Characters (24x24 pixel bitmap)
Feature Vectors (23 dimensions)

Figure 5.22 : The procedure of feature extraction.

Figure 5.22 shows the procedure of feature extraction. Figure 5.23 shows the feature vectors of a Chinese character with and without noise. The deviation between two vectors is 1.8447°. The deviation was calculated by the cross angle between two vectors. We found that the deviation between two vectors is very small and can conclude that DCT has noise toleration capability.

Normal Feature Vector	Noise Feature Vector
0.8023	0.8015
-0.0263	-0.0248
-0.2066	-0.2034
-0.0230	-0.0194
0.0104	-0.0000
-0.0291	-0.0338
-0.2261	-0.2225
0.2301	0.2198
0.1387	0.1425
-0.1336	-0.1237
-0.0446	-0.0439
0.1353	0.1415
0.0056	0.0149
-0.2017	-0.1955
0.0170	0.0122
-0.0784	-0.0874
-0.0564	-0.0673
-0.1834	-0.1889
-0.0595	-0.0601
0.1297	0.1336
0.1638	0.1722
0.1352	0.1446
-0.0228	-0.0152

Figure 5.23 : Difference between feature vectors with and without noise

Figure 5.24 shows the reconstruction of the original character by Inverse DCT, the reconstruction is based on the feature vector only (low frequency region) and the rest coefficients (high frequency region) are set to zero. The figure shows that the reconstructed word can still be recognized by human being.

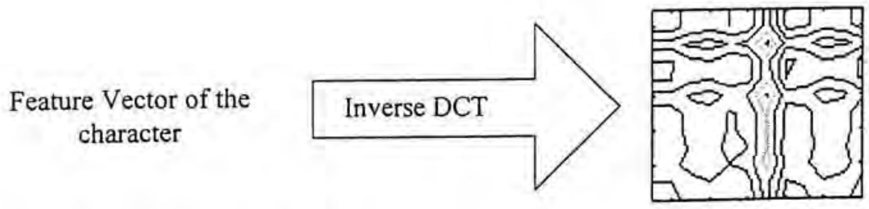


Figure 5.24 : The figure shows the reconstruction of the character by the feature vector though the Inverse DCT.

Since Chinese characters are distinct, after inverse DCT, the original character can still be recognize visually, therefore we can conclude that our method of feature extraction can generate a distinct set of feature vectors, which at least can be classified manually.

After the feature vectors are created, they can be used as input to ANFIS for training and recognition.

5.5.2 Architecture

Since the feature vectors contain 23 dimensions, the ANFIS has 23 input variables. For each input, we intended to assign 2 membership functions to it, however, we found that the number of fuzzy rules increase rapidly as the number of membership functions increases. If each input variable has 2 membership functions, then there will be 2^{23} fuzzy rules which are difficult to be handled.

Instead of assigning 2 membership functions to each variable. 6 input variables are assigned with 2 membership functions, the rest are assigned with 1 membership function. With this architecture, there are only $2^6 = 64$ rules and it is less of structured knowledge representation and more of a black-box model (like neural networks).

For the outputs of ANFIS, we used the multiple outputs ANFIS model discussed earlier (section 3.2.6). The ANFIS contains 10 output nodes, each of them produce either 0 or 1. In other words, the Chinese characters are represented by 10-bit binary codes.

5.5.3 Training Data

In this application, we use 300 Chinese characters for training and recognition. First of all, the input feature vector of each character is generated, then the corresponding character code is generated. The training data set contains 300 noise data with noise at 2% (i.e. there is 600 training data, 300 noise + 300 no noise). The reason for including the noise data in the training data set is to improve the noise tolerate capability of ANFIS.

5.5.4 Results and Discussions

The training error of ANFIS is acceptable, after 65 training epochs, the error is 0.001. However, for each training epoch, it takes more than 2 hours. For 65 training epochs, it is about 130 computation hours.

The first reason for such long training time is the use of LSE. When the training set is large and the dimension of each training data is high. LSE will take a long time to be finished. This is the defect of using LSE as well as hybrid learning.

The second reason is the number of outputs. For the previous application, the number of output of ANFIS is 1, while in this application, the number of output is 10. It is not difficult to see that the computational time increase in a greater extent as long as the number of outputs increased.

Although the training time of ANFIS is quite long, it gives a remarkable result on this application. In order to evaluate the performance of ANFIS, we use the original 300 training data to test the accuracy of ANFIS at different noise levels and the result is shown in table 5.8. Figure 5.25 shows a Chinese character at different noise levels. The high accuracy in noise level 2 is due to the fact that we have added to the training data set a set of noise level 2 data.

Noise Level	*Accuracy*
0	100%
1	98%
2	100%
3	85%
4	72%
5	63%
7	40%
9	40%

Table 5.8 : Accuracy of ANFIS at different noise levels.

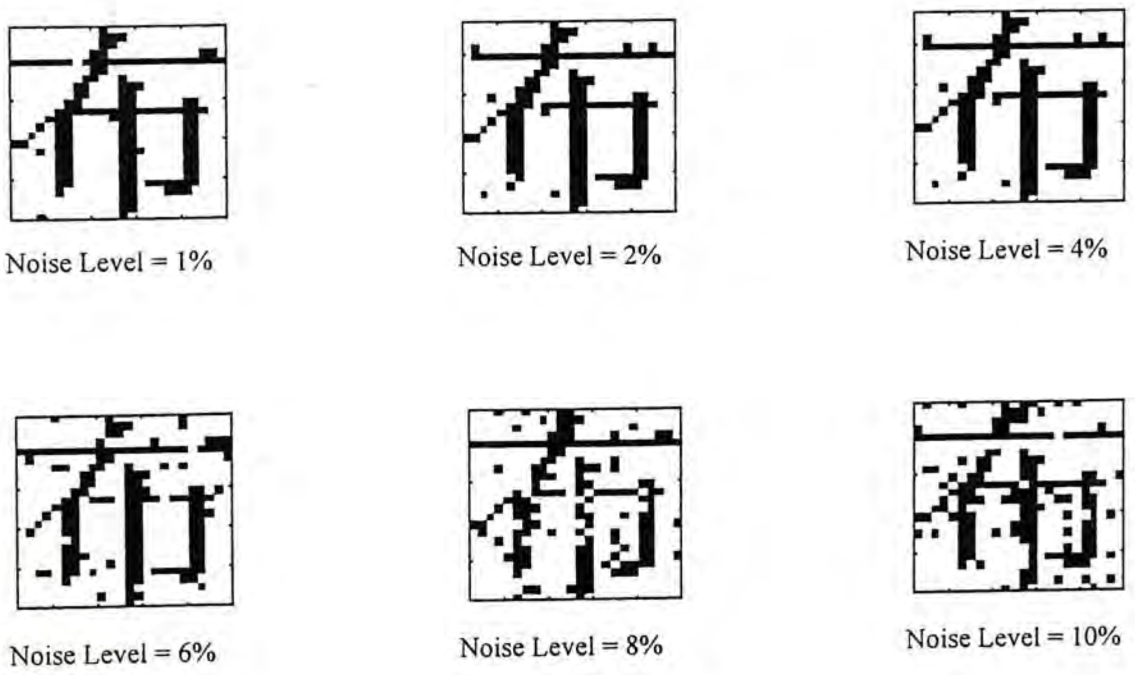


Figure 5.25 : The figure shows a chinese character at different noise levels.

The above result shows that ANFIS can have a very good performance on problem of classification which is a classical problem of neural networks. Moreover, ANFIS shows it noise tolerate capability for classifying the noise patterns correctly.

5.6 Image Coding

Since there is a limitation on the bandwidth of communication channels, when image files are transferred through the communication channels, source coding of images is needed to reduced the amount of data to be transferred, hence, save the bandwidth.

The source coding of images can be divided into three steps which are shown in figure 5.26.

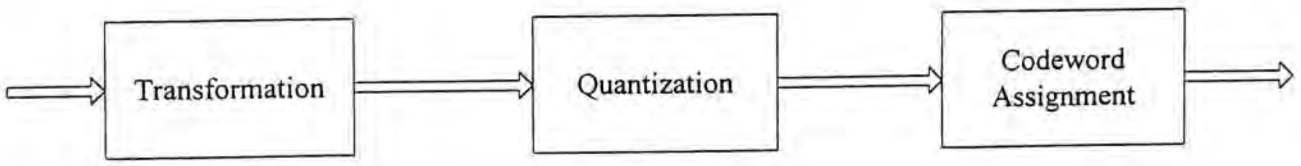


Figure 5.26 : The three steps for the source encoding of images.

Transformation aims to convert an image to the most suitable domain for quantization and codeword assignment. Quantization aims to represent a continuous value by discrete value. Codeword assignment aims to use as less bit as possible to represent the quantization levels.

This application applies ANFIS in one of the quantization methods - DPCM. ANFIS will be used as a prediction engine of DPCM. Our implementation is based on the idea proposed by Woods and O'Neil. First of all, subband coding will be applied on the image. Then, each band will be coded by DPCM with ANFIS.

5.6.1 Subband Coding

In this application, the image will be divided into 16 bands by using subband coding. The advantage of using subband coding of digital waveforms is that it splits the full frequency band into several subbands, each of which can be encoded more accurately. Coding error is confined to individual frequency subbands. Further, by varying the bit assignment among the subbands, the noise spectrum can be shaped to suit the human perception. In the subband coding of speech, the noise spectrum can be shaped according to the subjective noise perception of the human ear.

For the 2-Dimensional processing of images, the image is divided in a 2-dimensional way. For an ideal filter, the 4-band partitioning of the image is shown in figure 5.27. The splitting of the image into subband can be done as shown in the system diagram in figure 5.28.

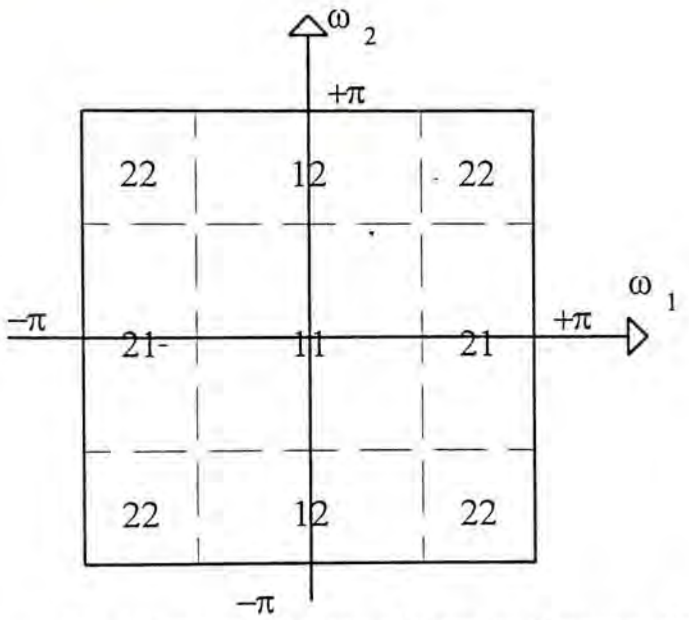


Figure 5.27 : 4-band partitioning of image in frequency domain.

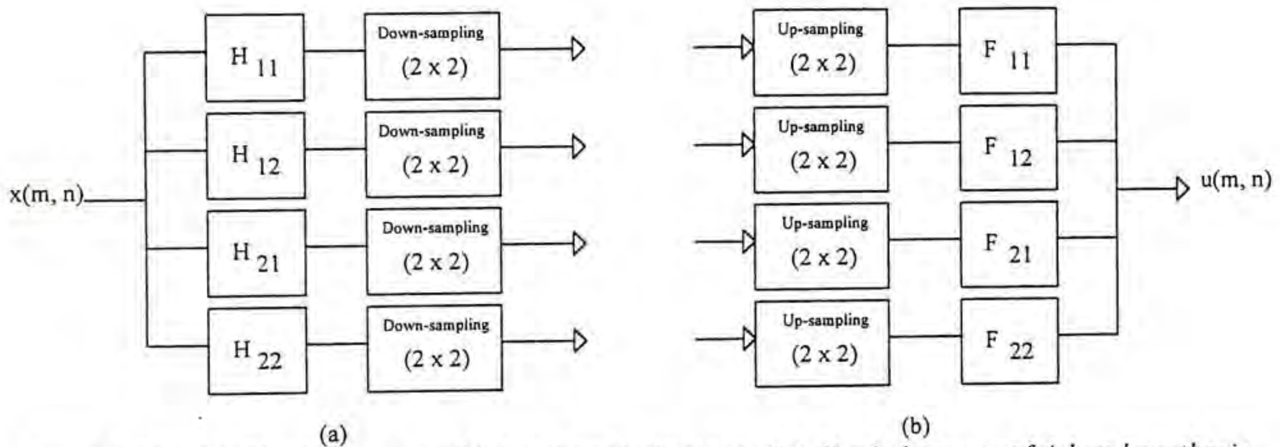


Figure 5.28 : (a) Block diagram of 4-band analysis stage. (b) Block diagram of 4-band synthesis stage.

If an ideal filter is available, the image can be reconstructed perfectly by using the analysis and synthesis stages as shown in figure 5.28. Nevertheless, an ideal filter is not available in practice. Instead, a FIR filter is used to approximate the ideal IIR filter. In filter design, the main concern is the perfect reconstruction of the image.

In the following, H_{ij}, $i \in \{1, 2\}, j \in \{1, 2\}$, denote the filters for the analysis stage, and F_{ij}, $i \in \{1, 2\}, j \in \{1, 2\}$, denote the filters for the synthesis stage, where 1 corresponds to the low frequency bank and 2 corresponds to the high frequency bank. Referring to Figure 4(a) and the initial four-band splitting, we require that the four subband filters H_{11} through H_{22} have mirror-image conjugate symmetry about their mutual boundaries, which for real h_{ij} is equivalent to

$$\begin{aligned} H_{12}(\omega_1,\omega_2) &= H_{11}(\omega_1,\omega_2+\pi) \\ H_{21}(\omega_1,\omega_2) &= H_{11}(\omega_1+\pi,\omega_2) \\ H_{22}(\omega_1,\omega_2) &= H_{11}(\omega_1+\pi,\omega_2+\pi) \end{aligned} \tag{5.12}$$

Denoting the outputs of the filters in figure 5.28(a) as x_{11} through x_{22}, we have the outputs Y_{ij} after (2, 2) downsampling,

$$Y_{ij}(\omega_1,\omega_2) = \frac{1}{4}\sum_{k=0}^{1}\sum_{l=0}^{1} H_{ij}(\tfrac{\omega_1+k\pi}{2},\tfrac{\omega_2+l\pi}{2})\cdot X(\tfrac{\omega_1+k\pi}{2},\tfrac{\omega_2+l\pi}{2}). \tag{5.13}$$

After synthesis, the outputs U_{ij} of the interpolation filters are

$$U_{ij}(\omega_1,\omega_2) = Y_{ij}(2\omega_1,2\omega_2)F_{ij}(\omega_1,\omega_2), \tag{5.14}$$

and the final output U is

$$U(\omega_1,\omega_2) = \frac{1}{4}\sum_{k=0}^{1}\sum_{l=0}^{1} X(\omega_1+k\pi,\omega_2+l\pi)[\sum_{i=0}^{1}\sum_{j=0}^{1} H_{ij}(\omega_1+k\pi,\omega_2+l\pi)F_{ij}(\omega_1,\omega_2)] \tag{5.15}$$

From (5.15), we can see that there is an aliasing component U_a generated[19], where

$$U_a(\omega_1,\omega_2) = \frac{1}{4}\sum_{(k,l)\neq(0,0)} X(\omega_1+k\pi,\omega_2+l\pi)[\sum_{i=0}^{1}\sum_{j=0}^{1} H_{ij}(\omega_1+k\pi,\omega_2+l\pi)F_{ij}(\omega_1,\omega_2)] \tag{5.16}$$

which will vanish if and only if

$$\sum_{i,j} H_{ij}(\omega_1+k\pi,\omega_2+l\pi)F_{ij}(\omega_1,\omega_2) = 0 \qquad \text{for} \quad (k,l)\neq(0,0). \tag{5.17}$$

By choosing the same reconstruction filters[19] as

$$\begin{aligned} F_{11}(\omega_1,\omega_2) &= 4H_{11}(\omega_1,\omega_2) \\ F_{12}(\omega_1,\omega_2) &= -4H_{12}(\omega_1,\omega_2) \\ F_{21}(\omega_1,\omega_2) &= -4H_{21}(\omega_1,\omega_2) \\ F_{22}(\omega_1,\omega_2) &= 4H_{22}(\omega_1,\omega_2) \end{aligned} \tag{5.18}$$

and found that the aliased terms automatically vanish for $(k, l) = (0, 1)$ or $(1, 0)$. However, in the case $(k, l) = (1, 1)$ the aliased term will vanish iff

$$H_{11}(\omega_1,\omega_2)H_{11}(\omega_1+\pi,\omega_2+\pi) = H_{11}(\omega_1,\omega_2+\pi)H_{11}(\omega_1+\pi,\omega_2). \tag{5.19}$$

We can show that any separable filter H_{11} would clearly satisfy (5.19). If we define the 2-D baseband filter h_{11} as the separable product

$$h_{11}(m,n) \overset{\Delta}{=} h_1(m)h_1(n). \tag{5.20}$$

By taking the Fourier transform on (5.20) and substitute it into (5.19), we can find that the 2-D QMF filters can be taken as a separable product of identical 1-D QMF filters[19] and the 2-D analysis filter banks can be obtained by simply multiplying two 1-D filters, i.e.

$$H_{ij}(\omega_1,\omega_2) = H_i(\omega_1)H_j(\omega_2), \qquad 1 \le i,j \le 2. \tag{5.21}$$

Consider a 1-D Subband filter pair h_1 and h_2 which satisfy

$$\begin{aligned} &h_1(n) = h_1(L-1-n) \qquad 0 \le n \le L/2-1 \\ &h_2(n) = (-1)^n h_1(n) \\ &\left|H_1^2(\omega)\right| + \left|H_2^2(\omega)\right| = 1 \end{aligned} \tag{5.22}$$

Base on (5.18) and (5.21), all filters (H_{ij} and F_{ij}) in the system can be found. By employing a linear phase symmetric $L \times L$ FIR filter for h_{11} with L even, we have

$$h_{11}(m,n) = h_{11}(L-1-m, L-1-n), \qquad 0 \le m,n \le \tfrac{1}{2}L-1. \tag{5.23}$$

and we can show that the phase of the output signal U would have a linear phase[19].

By selecting the low-pass filter h_l , we can obtained the 4-subband filter as shown in figure 5.29.

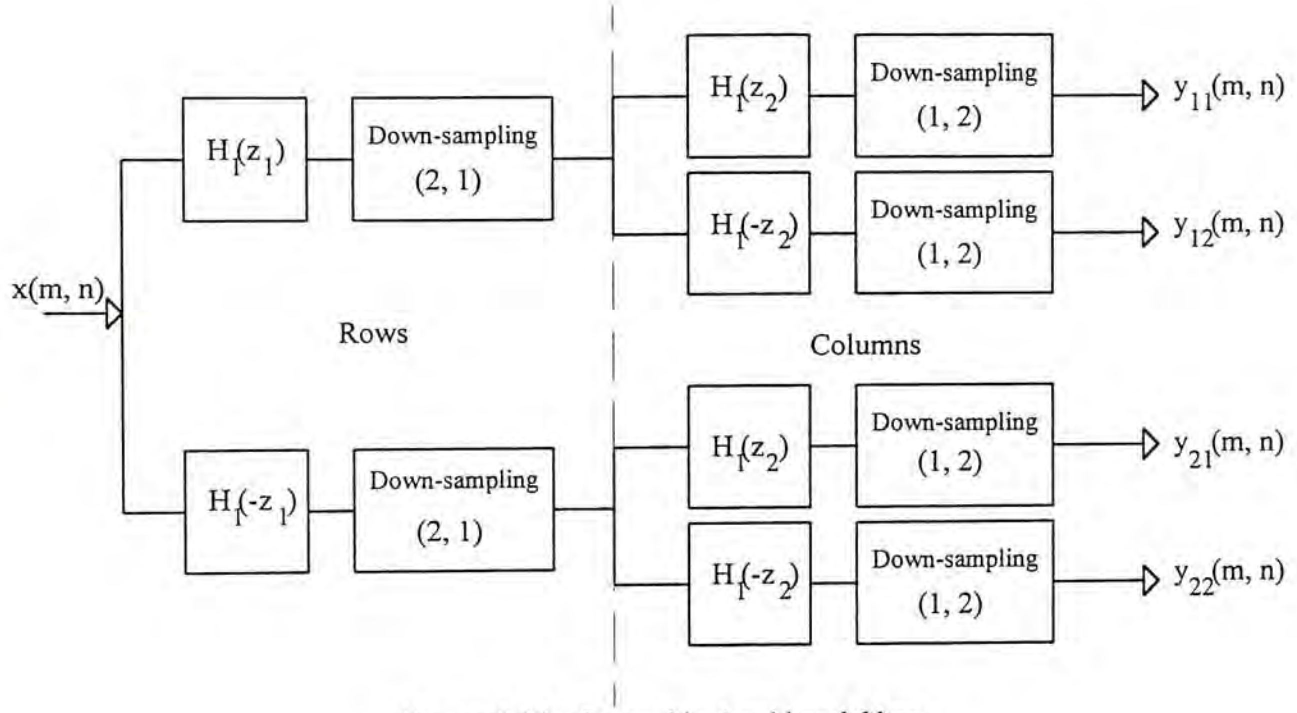


Figure 5.29 : Separable 4-subband filter.

We can build the 16-subband filter by cascade a 4-subband filter with another four 4-subband filters as shown in figure 5.30.

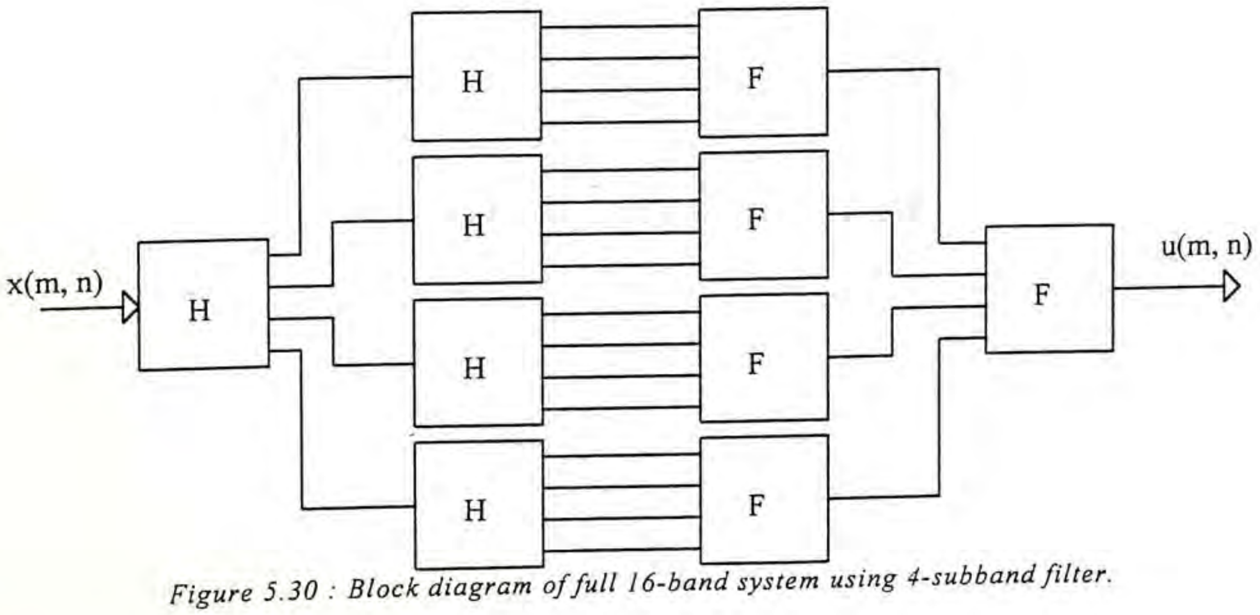


Figure 5.30 : Block diagram of full 16-band system using 4-subband filter.

5.6.2 DPCM with ANFIS

After the image is splitted into 16 different subbands, an ANFIS will be applied on different subbands for DPCM, the number of training data for each ANFIS is less than that of directly code the image with ANFIS. This is one of the reasons of using subband coding before DPCM with ANFIS, it aims to reduce to training data set of each ANFIS model. Hence, the accuracy and training time of ANFIS can be improved greatly.

Instead of using the conventional N-tap linear predictor, we use ANFIS as the predictor engine. If ANFIS gives a more accuracy prediction than N-tap linear predictor, the performance of DPCM can be improved. In other words, less bits can be used to code the data. Figure 5.31 shows the conventional DPCM system (a) and DPCM with ANFIS (b).

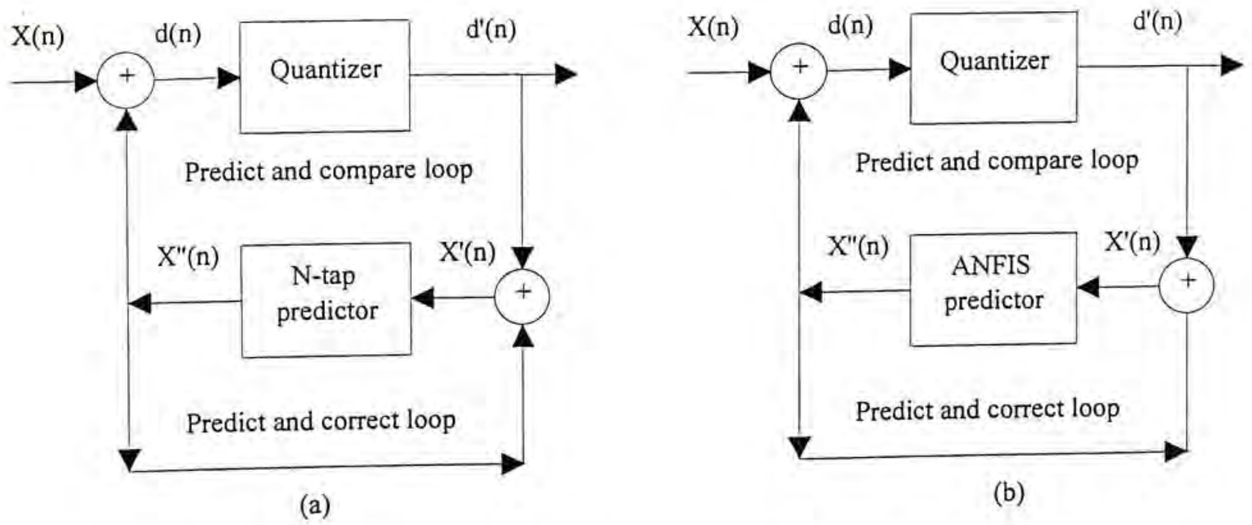


Figure 5.31 : Two approaches to DPCM, the equation of predict and correct loop is $X'(n)=X''(n)+d'(n)$. (a) One-tap predictor DPCM. (b) ANFIS predictor DPCM.

Nonuniform quantization is used to code the prediction error. The compression function is based on the μ-law compander which is the North American Standard. It is in the form of

$$y = y_{max} \frac{ln[1 + \mu(|x| / x_{max})]}{ln(1 + \mu)} sgn(x). \qquad (5.24)$$

where $sgn(x)$ is the sign of x and $y_{max} = 1$.

The parameter μ in the μ-law compander had originally been set to 255 for 8-bit converter[2].

The bit assignment of the nonuniform quantizing for each subband is according to the following equation,

$$B_k = B + \frac{1}{2} log_2 \left[\frac{\sigma^2_{p,k}}{\sigma^2_{gm}} \right], \quad 1 \le k \le M, \qquad (5.24)$$

where $\sigma^2_{p,k}$ is the prediction error variance on band k, and $\sigma^2_{gm} = (\prod_{k=1}^{M} \sigma^2_{p,k})^{1/M}$ is the geometric mean of the $\sigma^2_{p,k}$ [18][19].

5.6.3 Architecture and Training data

In this application, each ANFIS model has 3 input variables and 1 output variable. For each input variable, 3 membership functions are associated with it.

In order to generate the training set, first of all, subband coding will be applied on the image to divide the image into 16 subbands. A set of training data will be generated from each subband. Each training data set will be used to training an ANFIS model, hence, there are totally 16 ANFIS models.

Training data are generated by considering a 2×2 grid with 4 pixels. The upper left, upper right and lower left pixel are taken to be the input data for ANFIS. The lower right pixel is the predicted pixel (output) of ANFIS. Figure 5.32 shows how the training data is generated.

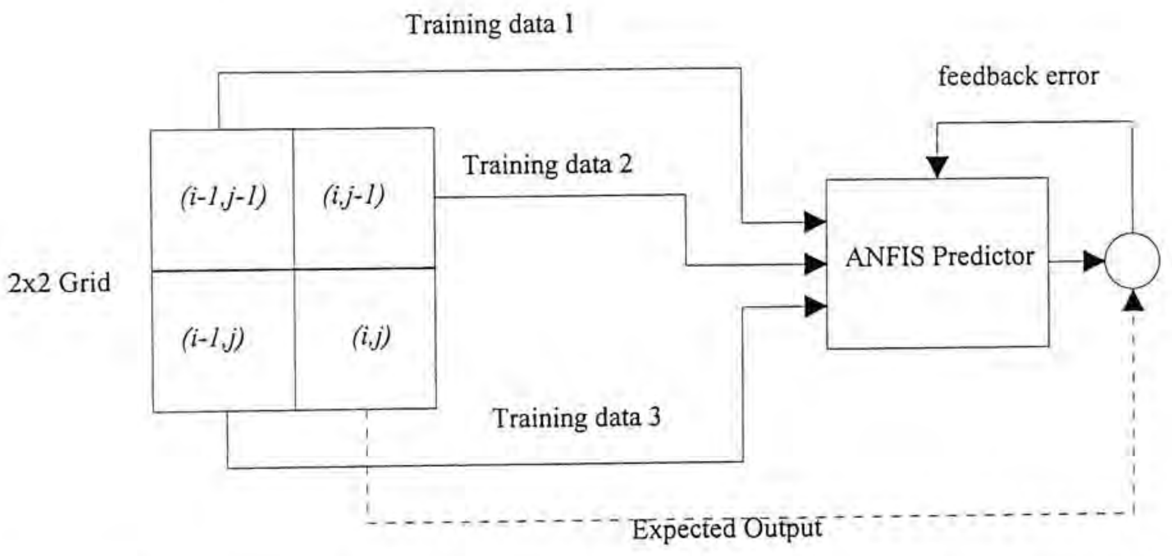


Figure 5.32 : The ANFIS training model and generation of training data from a 2×2 grid.

For a $n \times n$ subband, the total number of training data is $(n-1)\times(n-1)$. For example, a subband of 64×64 pixels, it has 3969 training data.

5.6.4 Results and Discussions

The proposed Subband coding and DPCM with ANFIS coding scheme is used to code the "Lena" image which is shown in figure 5.33.

Figure 5.33 : Original "Lena" image.

After the image is divided into 16 subbands by SBC, each subband is trained with an ANFIS model and the information of the prediction error is recorded in table 5.9.

Band[5]	Prediction error mean	Prediction error variance	Minimum prediction error	Maximum prediction error
11-11	0.0940	352.77	-120.83	230.94
11-12	-0.0022	59.12	-66.51	96.97
11-21	-0.0043	5.96	-14.58	18.2
11-22	-0.0152	19.13	-31.62	28.56
12-11	0.0412	19.70	-36.47	40.09
12-12	-0.023	16.80	-38.53	26.53
12-21	-0.0022	3.64	-12.94	16.72
12-22	-0.0170	9.47	-33.19	28.68
21-11	0.0330	2.02	-16.24	20.34
21-12	-0.0052	1.66	-8.55	12.46
21-21	0.0000	1.31	-8.68	11.97
21-22	0.0004	1.58	-9.19	9.14
22-11	-0.0170	4.61	-29.09	19.9
22-12	-0.0030	6.72	-19.51	19.8
22-21	0.0120	3.82	-27.52	65.32
22-22	0.0030	4.95	-17.61	19.20

Table 5.9 : Information of prediction error.

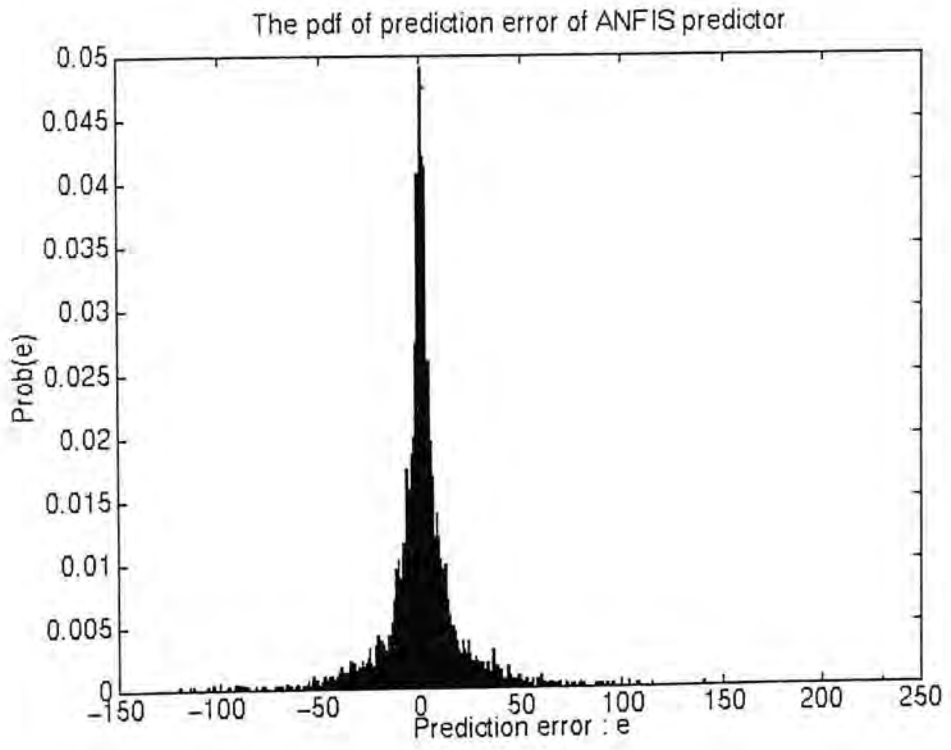
[5] *ij-kl refers to the subband resulting from filtering the image by H_{ij} and then by H_{kl}.*

A comparison has been done on the DPCM with 3-tap predictor and ANFIS predictor on the band 11-11. Table 5.10 shows the results of the comparison. The coefficients of the 3-tap predictor are from the "IEEE video course : Digital and Video Compression coding" which was presented by John W. Woods and James Modestino[18]. The reason of using the 3-tap predictor is to compare with the ANFIS with 3 input variables.

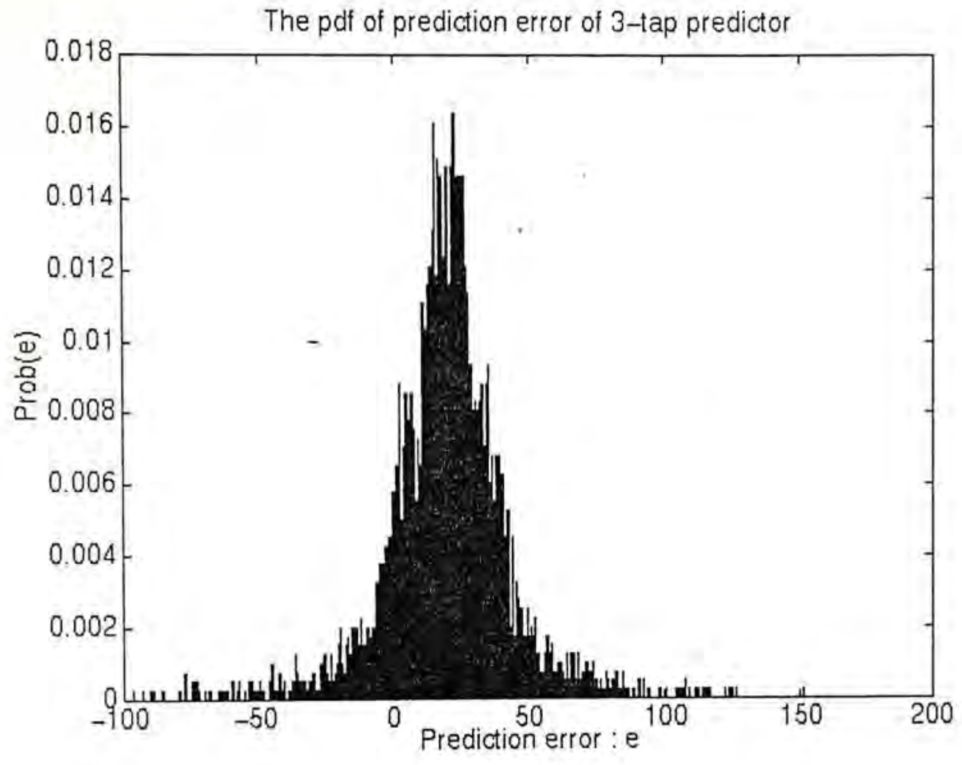
DPCM with	Prediction error mean	Prediction error variance	Minimum prediction error	Maximum prediction error
3-tap predictor	19.4088	478.91	-96.80	151.69
ANFIS predictor	0.0940	352.77	-120.83	230.94

Table 5.10 : Comparison of the predictor error on band 11-11.

Figure 5.33 shows the pdf of the prediction error of 3-tap predictor and ANFIS predictor. Based on table 5.10 and figure 5.34, we can conclude that the prediction capability of ANFIS is more accurate than that of 3-tap predictor.



(a)



(b)

Figure 5.34 : The pdf of prediction error of (a) ANFIS predictor, (b) 3-tap predictor.

In order to evaluate the quality of the reconstructed image, The Peak Signal to Noise Ratio (PSNR) is used.

$$PSNR(dB) = 10 \log \left[\frac{255^2}{\frac{1}{N} \sum_{j=1}^{N} (x_j - \bar{x}_j)^2} \right] \tag{5.25}$$

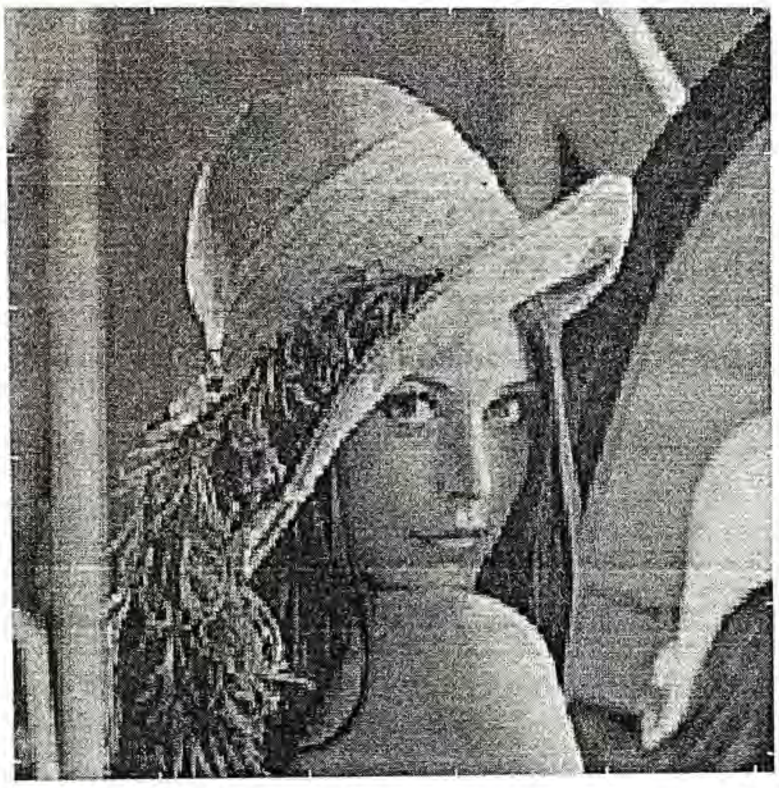
Table 5.11 shows the comparison of PSNR[6] of "Lena" image from 3-tap predictor and ANFIS predictor at different bit per pixel (bpp). It shows that ANFIS can give a better performance at each level.

[6] *The PSNR is calculated before the entropy coding.*

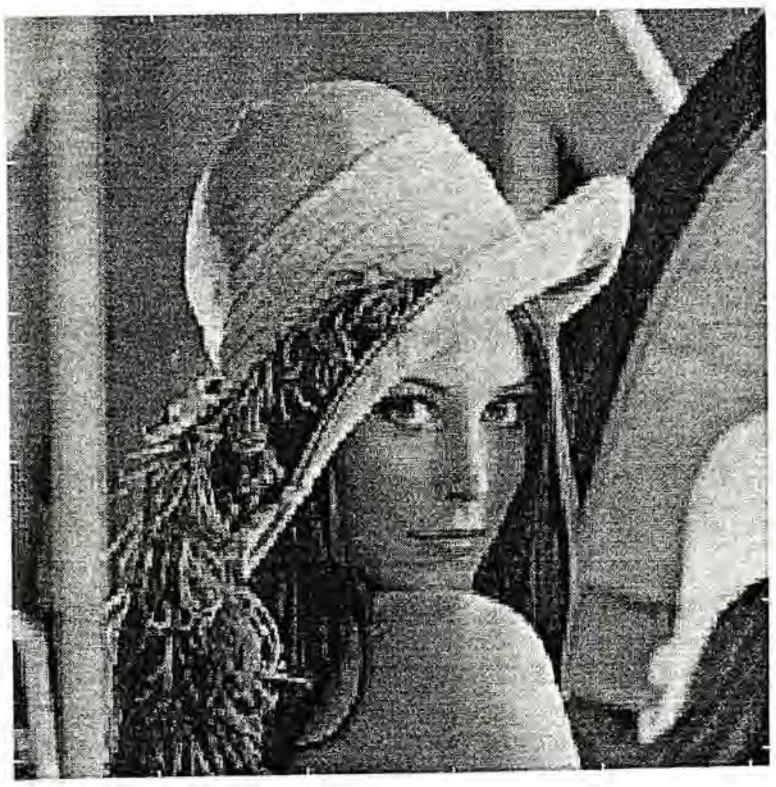
Bit per pixel	PSNR (ANFIS)	PSNR (3-tap)
1 bpp	29.2951	28.8172
2 bpp	31.9848	31.0918
3 bpp	33.4526	33.2240

Table 5.11 : PSNR of reconstructed image of ANFIS and 3-tap predictor at different bpp level.

(a)



(b)



(c)

Figure 5.35 : Reconstructed image "Lena" at (a) 1bpp, (b) 2bpp, (c) 3bpp.

Figure 5.35 shows the reconstructed image of "Lena" at different bpp. From this application, the prediction capability of ANFIS has been used in image coding.

6 Concluding Remarks

The report has discussed the idea of neural network based fuzzy inference system (Neural Fuzzy System). The main idea is to embed the fuzzy inference system into an adaptive network or feed forward neural networks. Base on the learning algorithm of neural networks, the parameters of the fuzzy system can be fine tuned and the semantic of the knowledge can be extracted from the input-output patterns.

Among a number of neural fuzzy systems, ANFIS (Adaptive-Networks-Based Fuzzy Inference System) from Jang was adopted for implementation. The architecture and learning algorithms have been discussed. The reason of adopting ANFIS in the project was because of the clear specifications of the ANFIS by Jang's paper and a number of remarkable results were reported for ANFIS.

The implementation was based on the development of a C++ library for ANFIS. The library was used to build a number of applications for the illustration of ANFIS capabilities. The applications are ranged simple logical operators to complex classification or control problems. The implementation shows that the library can be used as a software prototype for Neural-Fuzzy applications, so that engineers can simulate their Neural-Fuzzy applications before put them into hardware (e.g. VLSI) implementation.

Although all applications yielded the remarkable results from ANFIS, ANFIS is not a problem free model. Because of the lack of expert knowledge, in most cases, all possible rules are embedded in ANFIS. In the report, we have shown that the training time grows exponentially with respect to the number of fuzzy rules. Instead, ANFIS should only embed those rules with higher firing strengths and this is the problem of systems identification. For this problem, generic heuristic search (such as tabu search) from the field of Artificial Intelligence can be considered as a solution. This can be a further enhancement of the project.

Besides the explosion of training time, the proposed hybrid learning algorithm do has its limitation when the number of training data is large. The LSE of hybrid learning will take a long time to approximate the solution of the matrix equation (3.32). Moreover, dealing with the real time applications, such as control problems, hybrid learning is not applicable and gradient descent learning has to be used instead.

In the applications of ANFIS, ANFIS shows it capability in functional mapping[1]. The results are outstanding and better than those of neural networks. The explicit structural knowledge representation can account for the outstanding performance of ANFIS.

In this project, ANFIS has been proven to have good performance. This implies the adaptive networks with explicit structural knowledge representation have a better performance than traditional neural networks with implicit knowledge representation. I do think the future research direction should approach to the explicit structural knowledge representation.

[1] *Actually, all implemented applications belong to the catalogue of functional mapping and interpolation.*

7 References

[1] Bart Kosko, "Neural Networks and Fuzzy Systems", PHI, 1992.

[2] Bernard Sklar, "Digital Communications : Fundamentals and Applications", PHI, 1988.

[3] Blum Adam, "Neural Networks in C++ : an object-oriented framework for building connectionist system", Wiley, 1992.

[4] Charles Elkan, "The Paradoxical Success of Fuzzy Logic", Department of Computer Science and Engineering, University of California, San Diego, Nov 1993.

[5] Cox Earl, "Solving Problems with Fuzzy Logic", AI Expert, Mar 1992.

[6] Don Tveter, "Getting a fast break with Backprop", AI Expert, July 1991.

[7] Earl Cox, "Integrating Fuzzy Logic into Neural Nets", AI Expert, June 1992.

[8] Gary William Flake, "Nonmonotonic Activation Functions in Multilayer Perceptrons", PhD disssertation, pp. 6-18, Institute for Advance Computer Studies, Department of Computer Science, University of Maryland, 1993.

[9] Hamid R. Berenji, "Fuzzy Logic Controllers", An Introduction to Fuzzy Logic Applications in Intelligent Systems, Kluwer Academic Publishers.

[10] Herbert Schildt, "C++: the Complete Reference", McGraw-Hill, 1991.

[11] Hopefield, "Neural Networks and Physical Systems with Emergent Collective Properties", Proceedings of the National Academic Science, Vol. 1 and 2, MIT Press, 1992.

[12] J-S Roger and C. T. Sun, "Functional Equivalence between Radial Basis Function Networks and Fuzzy Inference Systems", 1992.

[13] J-S Roger Jang, "ANFIS: Adaptive-Network-Based Fuzzy Inference System", IEEE Transactions on Systems, Man, and Cybernetics, 1992.

[14] J-S Roger Jang, "Self-Learning Fuzzy Controller Based on Temporal Back Propagation", IEEE Transactions on Neural Networks, Vol. 3, No. 5, Sept 1992.

[15] James A. Freeman, David M. Skapura, "Neural Networks : Algorithms, Applications and Programming Techniques", Addison-Wesley Publishing Company, 1992.

[16] James M. Keller, Ronald R. Yager, Hossein Tahani, "Neural network implementation of fuzzy logic", Fuzzy Sets and Systems (1992) 1-12, North-Holland.

[17] John Hertz, Anders Krogh and Richard G. Palmer, "Introduction to the Theory of Neural Computation", Addision-Wesley Publishing Company, 1991.

[18] John Woods and James Modestino, "Digital Image and Video Compression Coding" (video tapes), IEEE Educational Activities Board, Sept 1992.

[19] John Woods, "Subband Coding of Images", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. ASSP-34, No. 5, Oct 1986.

[20] Kaufmann, "Introduction to the Theory of Fuzzy Subsets", Academic Press, 1975.

[21] Kohonen T., "Self-Organization and Associative Memory", Springer-Verlag, 1988.

[22] Larry O'Brien, "Developing a Neural Network with Turbo C++", AI Expert, Oct 1990.

[23] Lin Chin-Teng and Lee C. S. George, "Neural-Network-Based Fuzzy Logic Control and Decision System", IEEE Transactions on Computers, Vol. 40, No. 12, Dec 1991.

[24] Lotfi A. Zadeh, "Fuzzy Logic, Neural Networks and Soft Computing", Communications of the ACM, Vol. 37, No. 3, Mar 1994.

[25] Lotfi A. Zadeh, "Knowledge Representation in Fuzzy Logic", An Introduction to Fuzzy Logic Applications in Intelligent Systems, Kluwer Academic Publishers.

[26] Lotfi A. Zadeh, "The Calculus of Fuzzy If/Then Rules", AI Expert, Mar 1992.

[27] Maureen Caudill, "Neural Networks Primer", Part I - VIII, AI Expert, Dec 1987, Feb 1988, Jun 1988, Aug 1988, Nov 1988, Feb 1989, May 1989, Aug 1989.

[28] Minsky M. and Papert S., "Perceptrons : an Introduction to Computational Geometry, MIT Press, 1969.

[29] Paul M. Embree, Bruce Kimble, "C Language Algorithms for Digital Signal Processing", PHI, 1991.

[30] Philip D. Wasserman, "Advanced Methods in Neural Computing", Van Nostrand Reinhold, 1993.

[31] Rafael C. Gonzalez, Richard E. Woods, "Digital Image Processing", Addision-Wesley Publishing Company, 1992.

[32] Rao K. R. and Yip P., "Discrete Cosine Transform : algorithms, advantages, applications", Academic Press, 1990.

[33] Richard M. Karp, "Combinatorics, Complexity, and Randomness", Turning Award Lecture, 1985.

[34] Russell C. Eberhart, Roy W. Dobbins, "Neural Network PC Tools", Academic Press, 1990.

[35] Schwartz Daniel G., "Fuzzy Logic flowers in Japan", IEEE Spectrum, July 1992.

[36] Schwartz Tom J., "Fuzzy Tools for Expert Systems", AI Expert, Feb 1991.

[37] Shin-ichi Horikawa, Takeshi Furuhashi and Yoshiki Uchikawa, "On Fuzzy Modeling Using Fuzzy Neural Networks with the Back-Propagation Algorithm", IEEE Transactions on Neural Networks, Vol. 3, No. 5, Sept 1992.

[38] Takeshi Yamakawa, "A Fuzzy Inference Engine in Nonlinear Analog Mode and Its Application to a Fuzzy Logic Control", IEEE Transactions on Neural Networks, Vol. 4, No. 3, May 1993.

[39] TH Goth, PZ Wang, HC Lui, "Learning Algorithm for the Enhanced Fuzzy Perceptron", Institute of Systems Science, National University of Singapore, 1991, unpublished.`

[40] Toshiro Terano, Kiyoji Asai, Michio Sugeno, "Fuzzy Systems Theory and its Applications", Academic Press, 1992.

[41] Williams Tom, "Fuzzy Logic is anything but Fuzzy", Computer Design, Apr 1992.

[42] Xiao-Hu Yu, "Can Backpropagation Error Surface Not Have Local Minima", IEEE Transactions on Neural Networks, Vol. 3, No. 6, Nov 1992.