



The Institution of Engineering and Technology

Younger Member Section

IET YMSC

ONE DAY WORKSHOP

ON

SOFT COMPUTING TECHNIQUES

(15th September 2007)

COURSE MATERIAL



St. Xavier's Catholic College of Engineering

Chunkankadai .629 807

Kanyakumari Dist., Tamilnadu

India.

CHAPTER-1

Fuzzy Logic Controller

Fuzzy Logic control (FLC) has proven effective for complex, non-linear and imprecisely defined processes for which standard model based control techniques are impractical or impossible. Fuzzy Logic, unlike Boolean or crisp logic, deals with problems that have vagueness, uncertainty and use membership functions with values varying between 0 and 1. Fuzzy Logic tends to mimic human thinking that is often fuzzy in nature.

In fuzzy logic a particular object has a degree of membership in a given set, which is in the range of 0 to 1. The essence of fuzzy control algorithms is a conditional statement between a fuzzy input variable A and a fuzzy output variable B. This is expressed by a linguistic implication statement such as

IF A THEN B.

In general a fuzzy variable is expressed through a fuzzy set, which in turn is defined by a membership function μ .

1.1 Configuration of FLC

The basic configuration of an FLC is given in Fig. 1.1. It comprises of four principal components :

1. A fuzzification interface
2. A knowledge base
3. A decision-making logic and
4. A defuzzification interface.

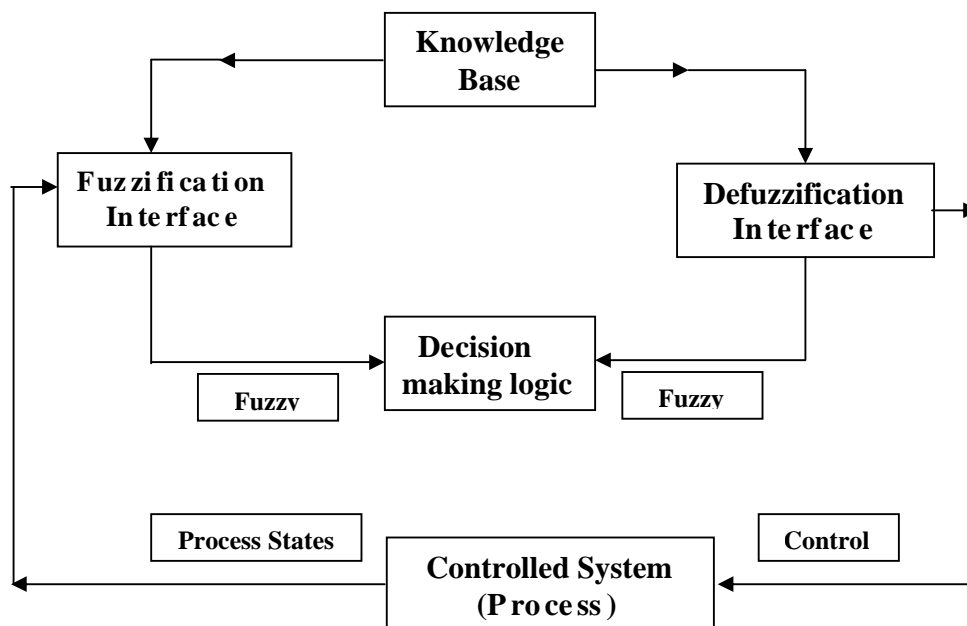


Fig.1.1. Basic Configuration of Fuzzy Logic Controller

1. The fuzzification interface involves the following functions.

- (a) measures the values of input variable.
- (b) performs a scale mapping that transfers the range of values of input variable into corresponding universe of discourse.
- (c) performs the function of fuzzification that converts input data into suitable linguistic values.

2. The knowledge base consists of database and a linguistic control rule base.

- (a) The database provides necessary definitions, which are used to define linguistic control rules.
- (b) The rule base characterized the control goals and control policy of the domain experts by means of a set of linguistic control rules.

3. The decision-making logic is the kernel of an FLC. It has the capability of simulating human decision-making based on fuzzy concepts and of inferring fuzzy control actions employing fuzzy implication and the rules of inference in fuzzy logic.

4. The defuzzification interface performs the following functions.

(a) A scale mapping, which converts the range of values of output variables into corresponding universe of discourse.

(b) Defuzzification, which yields a non-fuzzy control action from an inferred fuzzy control action.

Thus the idea behind the FLC is to fuzzify the controller inputs, and then infer the proper fuzzy control decision based on defined rules. The output is then produced by defuzzifying this inferred control decision.

1.1.1. Fuzzification and membership functions

Fuzzification is a process of transferring the crisp control variables to corresponding fuzzy variables. Selection of the control variables relies on the nature of the system and its desired output. The FLC input and output signals are interpreted into a number of linguistic variables. The number of linguistic variables varies according to the application. Increasing the number of linguistic variables results in a corresponding increase in the number of rules. Each linguistic variable has its fuzzy membership function. The membership function maps the crisp values into fuzzy variables.

A fuzzy set A in X is defined as

$$A = \{(x, \mu_A(x)) / x \in X\}$$

where $\mu_A(x)$ is called membership function (MF). The membership grade of each element of X is in the range 0-1.

Some of the one-dimensional membership functions commonly used are:

1. Triangular membership functions.
2. Gaussian membership functions.
3. Trapezoidal membership functions.

The general description of fuzzy classes of Triangular MF's is shown in Fig.1.2(a). It is specified by three parameter $\{a, b, c\}$ with $(a < b < c)$ are the x-coordinates of the three corners of the MF.

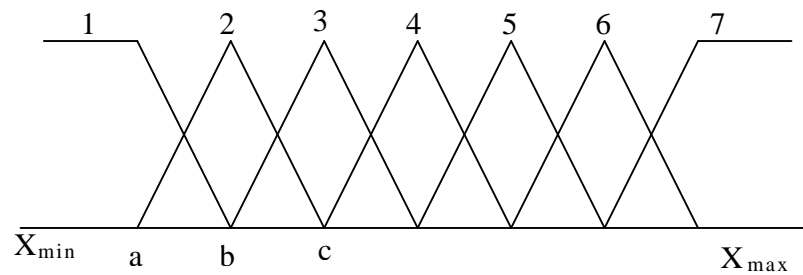


Fig.1.2(a). Triangular Membership functions

The class of Gaussian membership functions is shown in Fig.1.2(b). It is given by

$$\mu_{A_i}(x_i) = \frac{1}{\sigma_i \sqrt{2\pi}} e^{-\frac{(c_i - x_i)^2}{2\sigma_i^2}}$$

where c_i and σ_i are the center and width of the 'i'th fuzzy set A_i respectively.

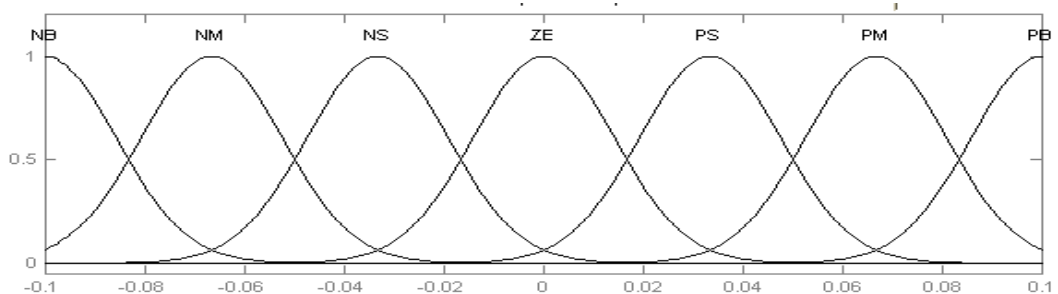


Fig.1.2(b). Gaussian membership functions

The Trapezoidal membership function is specified by four parameters $\{a, b, c, d\}$. The parameters $\{a, b, c, d\}$ (with $a < b \leq c < d$) determine the x-coordinates of the four corners of the underlying trapezoidal MF. The membership function description of trapezoidal family is described in Fig. 1.2 (c).

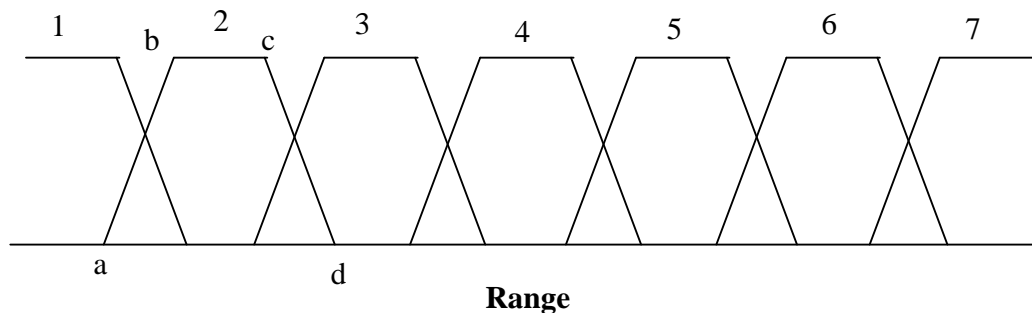


Fig.1.2(c). Trapezoidal membership functions

1.1.2. Rules Creation and Inference

In general, fuzzy systems map input fuzzy sets to output sets. Fuzzy rules are relations between input/output fuzzy sets. The modes of deriving fuzzy rules are based either of the following.

- Expert experience and control engineering knowledge.
- Operator's control actions.
- Learning from the training examples.

In this thesis the fuzzy rules are derived by learning from the training examples. The general form of the fuzzy control rules in this case is

$$\text{IF } x \text{ is } A_i \text{ AND } y \text{ is } B_i \text{ THEN } z = f_i(x, y)$$

where x , y and z are linguistic variables representing the process state variables and the control variable respectively. A_i , B_i are the linguistic values of the linguistic variables, $f_i(x, y)$ is a function of the process state

variables x , y and the resulting fuzzy inference system (FIS) is called a first order sugeno fuzzy model.

The function of the inference engine is to calculate the overall value of the control output variable based on the individual contributions of each rule in the rule base. (i.e.) the defuzzification process. There is no systematic procedure for choosing defuzzification. In first-order sugeno fuzzy model each rule has a crisp output and overall output is obtained as weighted average thus avoiding the time consuming process of defuzzification required in a conventional FLC.

CHAPTER-2

Artificial Neural Networks

Artificial Neural Networks (ANNS) are systems that are deliberately constructed to make use of some organizational principles resembling those of human brain. The key factors that distinguish Artificial Neural Networks from other computational techniques are

- ANNS are nonlinear; Able to classify patterns and capture complex interactions among the input variables in the system.
- ANNS are adaptive: they can take data and learn from it. (i.e.) online training.
- ANNS can generalize: they can correctly process data that broadly resembles the data they were trained originally.
- ANN is a parallel-distributed information processing structure.

2.1 Artificial Neuron model:

The model of an artificial neuron is shown in Fig 2.1. In this model the processing elements (neuron) computes the weighted sum of its inputs and outputs according to whether this weighted input sum is above or below a certain threshold θ_k . The externally applied bias has the effect of lowering the net input of the activation function

$$y_k = f(u_k - \theta_k) \quad 2.1$$

$$\text{where } u_k = \sum w_{kj} x_j \quad 2.2$$

Here x_1, x_2, \dots, x_p are input signals and $w_{k1}, w_{k2}, \dots, w_{kp}$ are interconnection weights of the neuron k . u_k is the linearly combined output.

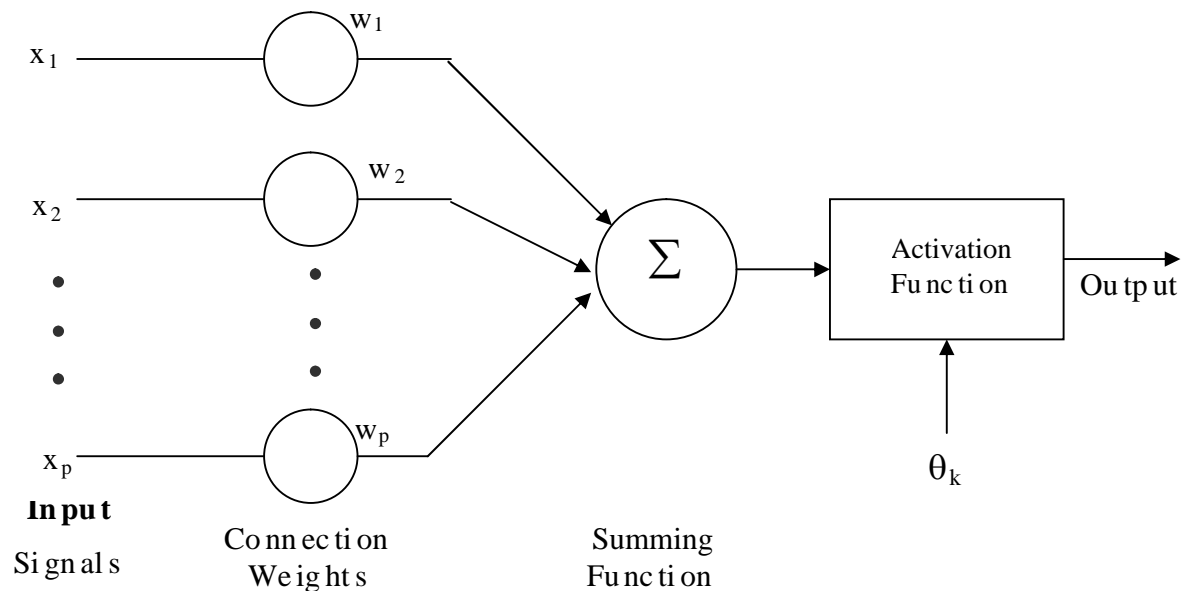


Fig.2.1. Artificial Neuron Model

Some commonly used activation functions are

- Step function $f(x) = \begin{cases} 1 & \text{for } x \geq 0, \\ 0 & \text{otherwise.} \end{cases}$ 2.3.
- Ramp function $f(x) = \begin{cases} 1 & \text{for } x > 1, \\ x & \text{for } 0 \leq x \leq 1, \\ 0 & \text{for } x < 0. \end{cases}$ 2.4
- Sigmoid function $f(x) = 1 / (1 + e^{-ax})$. 2.5.

where 'a' is the slope parameter of the function.

2.2 Neural Network Connections

ANNS are weighted directed graphs in which neurons are nodes and directed edges (with weights) are connected between neuron outputs and neuron inputs.

Based on connection patterns ANNS are grouped as

- Feed-forward networks in which graphs have no loops.
- Recurrent or feedback networks in which loops occurs.

Feed-forward networks are static as they produce only one set of output values for a given input. These networks are memory-less in the sense that their response to an input is independent of the previous network state. In the simplest form of single layer feed-forward networks these is an input layer of source codes that project into an output layer of neurons. The simple layer feed forward networks are shown in Fig. 2.2 (a). The interconnection of several layers form a multilayer feed-forward networks as shown in Fig .2.2(b). The layer between the input and output layer is called a hidden layer and its function is to intervene between the external input and output. The hidden layer had no direct contact with the external environment. The radial basis function network is a special class of multilayer feed-forward networks. The hidden layer in this employs a radial basis function, such as a Gaussian kernel as the activation function.

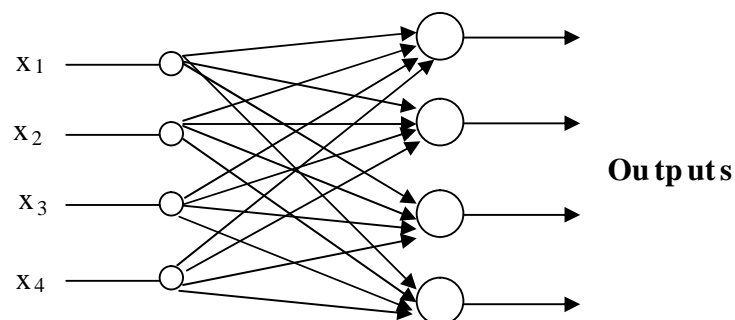


Fig.2.2. (a)Single layer feed forward network

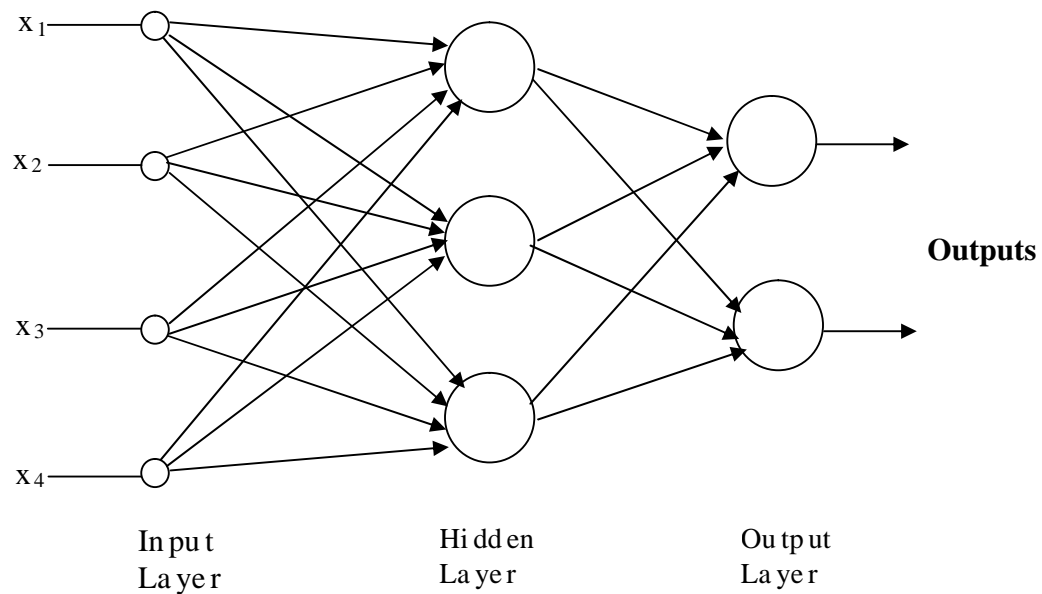


Fig. 2.2(b) Multilayer feed forward network

Feedback networks that have closed loops are called recurrent networks. In single layer recurrent network the processing element output is feedback to itself or to other processing element or to both. In the multilayer recurrent network the neuron output can be directed back to the nodes in the preceding layers.

2.3 Learning in ANN

Learning is not a unique process; there are different learning processes, each suitable to different species. In ANN the concepts of learning processes have been borrowed from the behaviorist's lab and implemented in electronic circuitry.

Learning is a process by which neural network adapts itself to stimulus and eventually (after making the proper parameter adjustments to itself) it produces a desired response. Learning is also a continuous classification process of input stimuli. During the process of learning the network adjusts its parameters, the synaptic weights in response to and

input stimulus so that its actual output response converges to the derived output response.

Back propagation Learning Algorithm

Based on this algorithm, the network learns a distributed associative map between the input and output layers. What makes this algorithm different than the others is the process by which the weights are calculated during the learning phase of the network. In general, difficulty with multilayer perceptrons is calculating the weights of the hidden layers in an efficient way that results in the least (or zero) output error; the more hidden layers there are; the more difficult it becomes. To update the weights, one must calculate an error. At the output layer this error is easily measured; this is the difference between the actual and desired (target) outputs. At the hidden layers however, there is no direct observation of the error, hence some other technique must be used to calculate error, as this is the ultimate goal.

Learning with Back propagation Algorithm

During the training session of the network, a pair of patterns is presented (x_k, d_k) , where x_k is the input pattern and d_k is the target or desired pattern. The x_k pattern causes output responses at each neuron in each layer and, hence actual output O_k at the output layer. At the output layer, the difference between the actual and target outputs yields an error signal. This error signal depends on the values of the weights of the neurons in each layer. This error is minimized, and during this process new values for the weights are obtained. The speed and accuracy of the learning process (i.e.) the process of updating the weights also depends on a factor known as the learning rate.

The basis for this weight update algorithm is simply the gradient – descent method as used for simple perceptrons with differentiable units. For

a given input – output pair (x_k, d_k) the back – propagation algorithm performs two phase of data flow. First, the input pattern a_x is propagated from the input layer to the output layer and, as a result of this forward flow of data, it produces an actual output y_k . Then the error signals resulting from the difference between d_k and y_k are back - propagated from the output layer to the previous layers for them to update their weights.

Let us consider 'm' PEs (processing elements) in the input layer, 'l' PEs in the hidden layer and 'n' PEs in the output layer as shown in Fig (2.3).

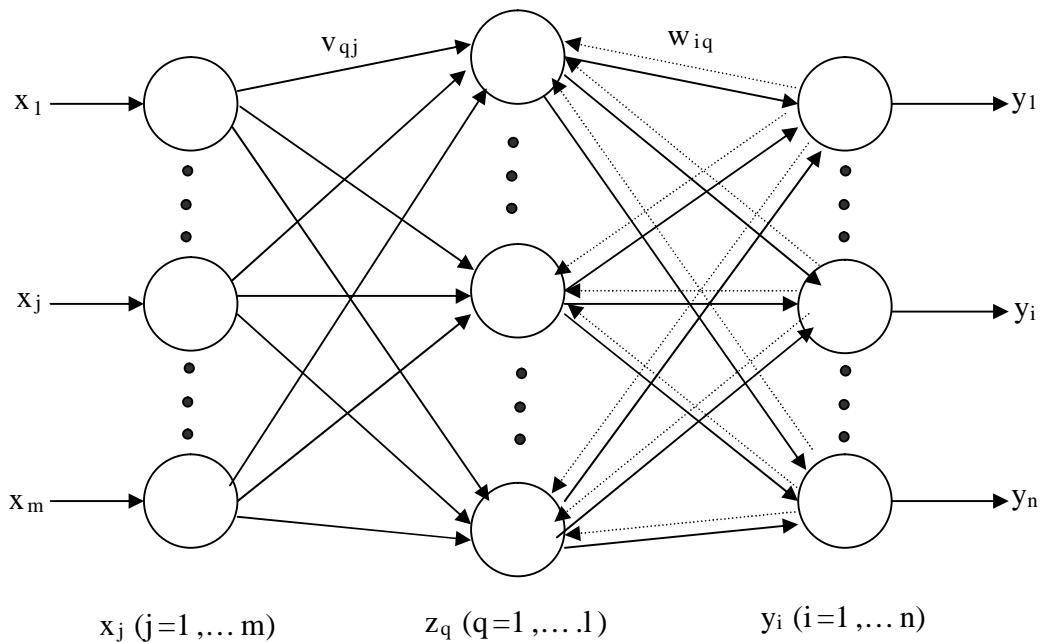


Fig.2.3. Back propagation Network

Consider an input – output training pair (x, d) a PE q in the hidden layer receives a net input of

$$\text{net } q = \sum v_{qi} x_j \quad j=1, \dots, m. \quad 2.6.$$

and produces an output of

$$z_q = a(\text{net } q) = a\left(\sum v_{qi} x_j\right) \quad 2.7.$$

The net input for a PE 'i' in the output layer is then

$$\text{net } i = \sum w_{iq} z_q = \sum w_{iq} a\left(\sum v_{qi} x_j\right) \quad q=1, \dots, l. \quad 2.8.$$

and it produces an output of

$$\begin{aligned} y_i &= a(\text{net } i) = a\left(\sum w_{iq} z_q\right) \\ &= a\left(\sum w_{iq} a\left(\sum v_{qi} x_j\right)\right) \end{aligned} \quad 2.9.$$

The above equations indicate the forward propagation of input signals through the layers of neurons. Next, we shall consider the error signals and their back propagation.

$$E(w) = 1/2 * \sum (d_i - y_i)^2 \quad i=1, \dots, n.$$

Then according to the gradient – decent method the weights in the hidden to output connections are updated by

$$\begin{aligned} \Delta w_{iq} &= -\eta \partial E / \partial w_{iq} \\ &= -\eta \left[\partial E / \partial y_i \right] \left[\partial y_i / \partial \text{net}_i \right] \left[\partial \text{net}_i / \partial w_{iq} \right] \\ &= \eta \left[d_i - y_i \right] \left[a'(\text{net}_i) \right] \left[z_q \right] \\ &= \eta \delta_{oi} z_q. \end{aligned}$$

The process of computing the gradient and adjusting the weights is repeated until a minimum error is found. In practice, one develops on algorithm termination criterion so that the algorithm does not continue this iterative process forever.

In summary the error back – propagation algorithm can be outlined as

Step 1: Initialize all weights to small random values.

Step 2: Choose an input-output training pair.

Step 3: Calculate the actual output from each neuron in a layer by propagating the signal forward through the network layer by layer (forward propagation).

Step 4: Compute the error value and error signals for output layer.

Step 5: Propagate the errors back ward to update the weights and compute the error signals for the preceding layers.

Step 6: Check whether the whole set of training data has been cycled once, yes – go to step 7; otherwise go to step 2.

Step 7: Check whether the current total error is acceptable; yes- terminate the training process and output the field weights, otherwise initiate a new training epoch by going to step 2.

CHAPTER-3

ADAPTIVE NEURO-FUZZY CONTROLLERS

The fusion of ideas from fuzzy control and neural networks had acknowledged a significant role in improving controller performances. Fuzzy logic has proven effective for complex, non-linear and imprecisely defined systems. The common bottleneck in fuzzy logic is the derivation of fuzzy rules and the parameter tuning for the controller. The neural networks have powerful learning abilities, optimization abilities and adaptation. The fuzzy logic and neural networks can be integrated to form a connectionist Adaptive network based Fuzzy logic controller. This integrated adaptive system modifies the characteristics of the rules, topology of fuzzy sets and/or the structure of control system.

3.1. Fuzzy Adaptive Learning Control Network

The basic concept of Neuro-Fuzzy control models [29,30,31] is first to use structure-learning algorithms to find appropriate fuzzy logic rules and then use parameter-learning algorithms to fine-tune the membership functions and other parameters. The fuzzy adaptive learning control Network (FALCON) is a feed forward multilayer network, which integrates the basic elements, and functions of a traditional fuzzy logic controller into a connectionist structure thus had distributed learning abilities. In this connectionist structure, the input and output nodes represent the input states and output control or decision signals respectively, and in hidden layers there are nodes functioning as membership functions and fuzzy logic rules.

The FALCON can be constructed from training examples by neural learning techniques and the connectionist structure can be trained to develop fuzzy logic rules and determine proper input-output membership function. Expert knowledge can also be incorporated into the FALCON. The rule base of connectionist structure contains fuzzy IF-THEN rules of

sugeno's first order type in which the output of each rule is a linear combination of input variables plus a constant term. For a first-order sugeno fuzzy model the common rule set with two fuzzy IF-THEN rules is 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 final output is the weighted average of each rule's output.

3.2 Adaptive Neuro-Fuzzy Inference System Architecture

The architecture of the Adaptive Neuro-fuzzy Inference System (ANFIS) is shown in Fig.3.1. Consider a first-order sugeno fuzzy model with two input x and y and one output with above-mentioned fuzzy IF-THEN rules.

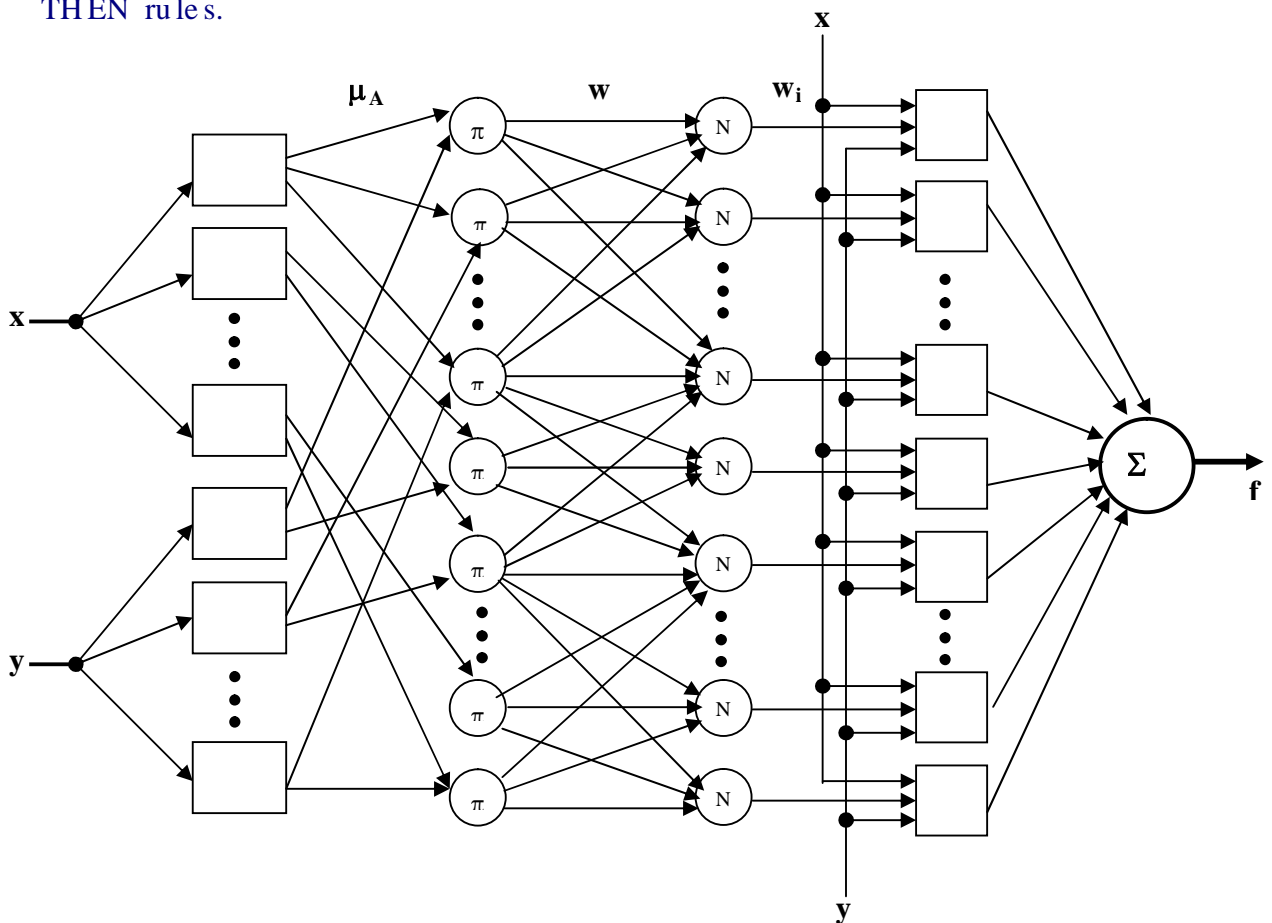


Fig.3.1. Architecture of Adaptive Neuro-Fuzzy Inference System

The system has a total of five layers.

Layer 1:

Every node i in this layer is an adaptive node performing membership function.

$$O_{1,i} = \mu_{A_i}(x_i) \quad i=1, \dots, n. \quad 3.1$$

where x_i is the input to node i .

The membership function can be any appropriate parameterized membership function. Parameters in this layer are referred to as premise parameters.

Layer 2:

Every node in this layer is a fixed node whose output is the product of all the incoming signals. (i.e.) these nodes perform the fuzzy AND operation.

$$O_{2,i} = w_i = \mu_{A_i}(x_i) \cdot \mu_{B_i}(y) \quad i=1, \dots, n. \quad 3.2$$

Each node output represents the firing strength of a rule.

Layers 3:

The nodes of this layer calculate the normalized firing strength of each rule.

$$O_{3,i} = \bar{w}_i = w_i / \sum_i w_i \quad i=1, \dots, n. \quad 3.3$$

w_i –firing strength of a rule.

Layer 4:

Every node i in this layer is an adaptive node with a node function.

$$O_{4,i} = \bar{w}_i f_i = \bar{w}_i (p_i x + q_i y + r_i) \quad i=1, \dots, n. \quad 3.4$$

where \bar{w}_i a normalized firing strength from layer 3 and $\{p_i, q_i, r_i\}$ the parameter set. Parameters in this layer are referred to as consequent parameters.

Layer 5:

The single node in this layer computes the overall output as the summation of all incoming signals.

$$O_{5,1} = \frac{\sum w_i f_i}{\sum w_i} \quad i=1, \dots, n. \quad 3.5$$

where $O_{1,i}$ denote the output of the 'i'th node in layer 1. This structure can update membership functions and rule base parameters accordingly to the gradient descent update procedure.

CHAPTER-4

GENETIC ALGORITHMS

4.1 Introduction

Evolutionary algorithms include a wide variety of population based algorithms which can be applied to different kinds of problems. The main idea there is to form a group of alternative solutions for the problem, that is, a population, and then to breed this initial population according to evolutive laws. A fitness function is used to measure the quality of each individual and to select the best individuals to produce offspring.

The basic and most common form of evolutionary algorithms are genetic algorithms, where a solution for a typically numerical optimization problem is searched for. Individuals are simply (binary) coded strings describing possible combinations of the parameter values, and the fitness function is the function to be optimized.

A more complex form of evolutionary algorithms is used in genetic programming. There each individual is a computer program which includes some calculations and performs some task. Fitness function describes how well the programs perform the given task, thus favouring better solutions.

4.2 Basics of Genetic Algorithms

A typical genetic algorithm consists of the following steps:

1. Create the initial population.
2. Evaluate the fitness of each individual.
3. Select the best individuals and perform recombination.
4. Mutate the new generation.
5. If termination condition is not reached, go back to step 2.

The calculation can be terminated for example when a certain fitness level is reached or after a certain number of iterations is performed. Also, if it seems that the solutions will not get any better for a long time, it can be deduced that it is best to stop the calculation. Next, the operations given above are explained in more detail.

4.2.1 Recombination

Recombination is the operation where two individuals in the old generation are used to produce two offspring for the new generation. Selection of the parent individuals is done based on their fitness values, and the typical methods are either roulette wheel selection or tournament selection. In roulette wheel selection, an individual is selected with probability which is proportional to its relative fitness value. So the better individuals will be selected more often, but also the other solutions can be selected. In tournament selection, a fixed number of solutions (e.g. two) is randomly chosen from the generation. Then the one of them with the best fitness value is selected to be the parent. This way, also the not so good solutions have a chance to be chosen. Assuming that the two parents are now selected, the recombination is simply performed by picking a random index of the bit strings and swapping the ends of the strings as presented in Figure 4.1.

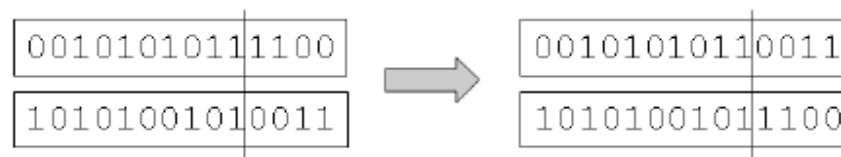


Figure 4.1: Recombination of two parent individuals (left) results to two offspring (right).

4.2.2 Elitism

To make sure that the best solution in the old generation is preserved, elitism can be used. It means that a fixed number of best solutions in the old generation is directly copied to the new generation. This way the algorithm may converge faster to the desired optimum.

2.3 Mutation

An important part of a genetic algorithm is mutation, where each bit of the individuals in the new generation is reversed with a small probability. This mutation probability is typically chosen to be around 0.01.

The idea of mutation is to increase the randomness of the algorithm, which again helps to prevent the convergence to a local optimum.

4.2.4 Advantages

There is couple of advantages in genetic algorithms when compared with more traditional optimization methods. First of all, the main advantage is that genetic algorithms can handle the local optima quite well unlike the simple gradient based methods. They can also be applied to a wide variety of problems, including high dimensional ones, as far as the solutions can be effectively coded to binary strings.

The main drawback of genetic algorithms is the large number of free parameters and problem dependent choices for example in the algorithm structure. This is because the method is purely heuristic and is not based on any global theory. The computational cost can also be quite high, especially if the calculation of the fitness function is complicated. Additionally, in the basic form only discrete solutions are allowed since the coding is done using binary strings.