UNIVERSITÉ DE
SHERBROOKE

Faculté de Génie

Département de génie électrique et de génie informatique

HIGH DIMENSIONAL NEURAL FUZZY CONTROLLER FOR NONLINEAR SYSTEMS

Mémoire de maîtrise ès sciences appliquées

Spécialité: génie électrique

Xiaodong Tan

Sherbrooke Quebec Canada Juin 2007

Your file Votre référence
ISBN: 978-0-494-49586-5
Our file Notre référence
ISBN: 978-0-494-49586-5

RÉSUMÉ

De nos jours, la théorie de contrôle joue un rôle significatif dans presque tous les domaine de la science et de l'ingénierie. Les contrôleurs linéaires PID sont les applications principales de la théorie de contrôle, et ils se basent sur les systèmes de contrôle simples. Mais beaucoup de vrais systèmes possèdent des caractéristiques non-linéaires. Dans la pratique, il est nécessaire de faire beaucoup de linéarisations. Quand nous employons le contrôleur classique dans un système non-linéaire fortement complexe, les difficultés augmentent exponentiellement. Pour éviter les imperfections, on peut employer des contrôleurs flous. Le contrôleurs flous se basent sur le système de connaisance. Ce sont des outils importants dans le domaine de l'automatique. Ils possèdent beaucoup plus d'avantages que les contrôleurs classiques "PID", mais ils ont besoin d'experts pour concevoir les règles de base. La limite principale des contrôleurs flous est la difficulté d'établir les règles de base.

Maintenant, beaucoup de recherches sont consacrées à la fusion des réseaux de neurones et de systèmes flous dans une nouvelle structure (les réseaux de neuro-floue). Cette approche combine les avantages de deux paradigmes puissants dans une capsule simple, et fournit un cadre puissant pour extraire des règles floues des données numériques.

Cependant, cette technologie n'est pas parfaite. Il reste quelques difficultés: beaucoup de règles floues sont nécessaires, les algorithmes sont complexes et la fiabilité est basse (Par exemple, pour un même modèle ou fonction, les résultats dépendent des ensembles d'apprentissage). Pour éviter les difficultés, ce mémoire présente une nouvelle méthode, appelée "inférence neuro-floue de haute-dimension".

L'idée fondamentale de cette méthode proposé est de considérer chaque donné dans ce système comme point avec la haute dimension. Chaque dimension d'entrée sera traitée en même temps dans les mêmes sous-ensembles de haute dimension.

L'algorithme proposé a été examiné sur différentes applications, et les résultats ont été comparés aux données éditées sur trois problèmes de repère.

Cet algorithme est simple à employer, et les résultats expérimentaux prouvent que le nombre de faisceaux exigés est inférieur à ceux rapportés dans la littérature. L'exactitude de rendement est bonne dans beaucoup d'applications.

REMERCIEMENTS

Au terme de ce travail, je tiens tout d'abord à exprimer ma profonde gratitude à l'endroit de mon directeur de recherche, le professeur et doyen au Faculté de génie du Université de Sherbrooke Dr. Gérard Lachiver. Je le remercie aussi pour ses conseils et son support financier durée de ce projet.

Enfin, je remercie tout spécialement ma femme wei He, pour leur soutien moral tout au long de ce travail, ainsi que mes parents et beaux-parents, wanhua tan, jiaping Lu, keyu He, et yufeng Xia, pour leur soutien et leur intérêt continus.

Encore une fois, Merci beaucoup, il n'y a aucun de mot qui peut exprimer ma gratitude.

TABLE OF CONTENTS

1. INTRODUCTION 3

2. ARTIFICIAL NEURAL NETWORKS 6

2.1 Artificial Neural Networks 6

2.1.1 Supervised learning 9

2.1.2 Unsupervised learning 13

2.1.3 Clustering 17

3 FUZZY LOGIC 18

3.1 Fuzzy sets and membership function 18

3.2 Linguistic variables and linguistic rules 20

3.3 Fuzzy system 21

3.4 Fuzzy control system 36

3.4.1 SISO AND MISO 37

3.4.2 MIMO 38

4 DESIGN OF A HIGH-DIMENSIONS NEURAL FUZZY CONTROLLERS (HDFIN) FOR NONLINEAR SYSTEMS 40

4.1 High-dimensions fuzzy inference 41

4.2 Input space clustering 46

4.3 Membership function forming 47

4.4 Identification parameters matrix derivation for T-S fuzzy rules 48

4.5 Conclusion 49

5. RESULTS AND COMPARATIVE ANALYIS 50

5.1 Nonlinear function approximation 50

5.2 Mackey-Glass time series prediction 53

5.3 Nonlinear system control: the ball and beam example 57

6. CONCLUSION 66

7. APPENDIX A..... 68

8. APPENDIX B..... 71

9. BIBLIOGRAPHY..... 78

1. INTRODUCTION

Today, the control theory plays a significant role in almost every field of science and engineering. The classical theory of automatic control systems has been widely used in modern society and it ranges from simple application, such as in washing machine control systems, to highly sophisticated systems such as space shuttles, satellites and intelligent robots. Throughout the history of control, the goal has been to mimic the human worker interacting with machines and to process events without human interaction. In recent years, researchers work to design controllers that have the ability to "learn" and "think" like human expert [1] [2].

Fuzzy theory is a powerful problem-solving method with wide applications in industrial control and information processing [3] [4] [5]. It provides a simple way to draw definite conclusions from vague, ambiguous and imprecise information. Unlike classic approach which needs a deep understanding of system dynamics and the knowledge of exact equations and precise numerical values, fuzzy logic incorporates simple rule-based "IF X AND Y THEN Z" approach to solve control problem rather than trying to model it mathematically. Fuzzy modeling based on numerical data, which was first explored systematically by Takagi and Sugeno [6], has found many successful applications to complex system modeling. Fuzzy controllers are the most important application of the fuzzy theory. They work rather differently than conventional controllers by using expert knowledge (rules) instead of differential equations to describe a system. The knowledge could be expressed in a natural way with "linguistic variables", which are described by "fuzzy sets". Fuzzy logic has many advantages but also has some limitations. For example, fuzzy systems needs expert for rule discovery and they cannot learn the rules themselves.

Artificial Neural Network (ANN) is an important concept in artificial intelligence. These networks are based on the parallel architecture of human brain. The true power and advantage of neural networks lies in their ability to represent both linear and non-linear relationships and in their capacity to learn these relationships directly from training examples. The training examples must be selected carefully otherwise useful time is wasted or even worse the network might be functioning wrongly. ANN is made of many highly interconnected processing elements (neurons) working in unison to solve specific problems. It can create its own organisation or representation of the information it receives during learning time but the knowledge represented by the network is difficult to understand. Also, the mathematical theories used to guarantee the performance of an applied neural network are still under development.

Many researches are devoted to fusion of neural networks and fuzzy system into new structures called "Neuro Fuzzy Networks". These approaches combine the benefits of two powerful paradigms into a single capsule and provide a powerful framework to extract fuzzy rules from numerical data [7], [8], [9], [10], [11], [12]. Neuro fuzzy networks are widely used in many fields. In fact, a fuzzy system can be seen as a special neural network and represented as a three-layer feedforward network [13] [14]. However this technology is not perfect. Neuro fuzzy networks behave like "black boxes" whose internal rules of operation are unknown. Nodes and links in a neuro fuzzy network correspond to a specific component of a fuzzy system, some represent linguistic terms, some input or output variables and some are used for representing fuzzy rules. All nodes are not fully connected to the nodes in the neighboring layers. This arises several questions such as: how to optimize a neuro fuzzy network, how to set a large number of parameters, how to reduce convergence time and how to realize effective training and adaptation?

The aim of this work is to answer these questions by proposing a new clustering neuro fuzzy method. The objective is to improve the accuracy in modeling applications, and to reduce the number of fuzzy rules and to realize reliable adaptation in fuzzy control applications.

The method uses clustering techniques and Takagi-Sugeno modeling. The learning phase starts with a clustering algorithm which divides the input space into a series of clusters according to the input data distribution. These clusters are then used to construct a series of membership functions for the input space mapping. As a result, a collection of fuzzy sets represents all inputs. Finally an identification parameters matrix **P** is drawn by Takagi-Sugeno modeling and matrix operations. This matrix holds all identification parameters for the consequent part of the T-S fuzzy rules. In a certain extent, this matrix is equivalent to a rule base. After training, the matrix will be use to compute the system output. The main advantage of the proposed method is its simplicity, a fast convergence time and great accuracy.

This work is divided as follows: chapter 2 and chapter 3 introduce fuzzy logic and neural networks. They describe essential knowledge and information needed to design neuro fuzzy networks. In chapter 4, the proposed method is developed in details. Performances and results are evaluated and discussed for three different types of application in chapter 5: prediction, approximation, and control. Finally, conclusions are given in chapter 6.

2. ARTIFICIAL NEURAL NETWORKS

Neuro fuzzy systems have the structure of Artificial Neural Networks and the abilities of fuzzy inference. Therefore, a good understanding of ANN is important to design neuro fuzzy systems.

2.1 Artificial Neural Networks

ANN is an information processing paradigm inspired by our knowledge on biological nervous system. This idea was established before the arrival of computers. In 1943, a neurophysiologist Warren McCulloch and a young mathematician Walter Pitts wrote a paper on how neurons might work [15]. To do so, they modeled a simple neural network with electrical circuits. With computers, it is now possible to simulate more complex neural networks and to mimic human learning skill.

Neural networks try to reproduce the structure of neurons in the human brain. They are a powerful data-modeling tool. Knowledge in neural networks is stored within the synaptic weights like inter-neuron connection strengths. Neural networks have typically a multi-layer architecture where layers of nodes are connected [16].

In a neural network, the basic element is the artificial neuron which is a device with several inputs and one output. Figure 2.1 shows the model of a single artificial neuron.

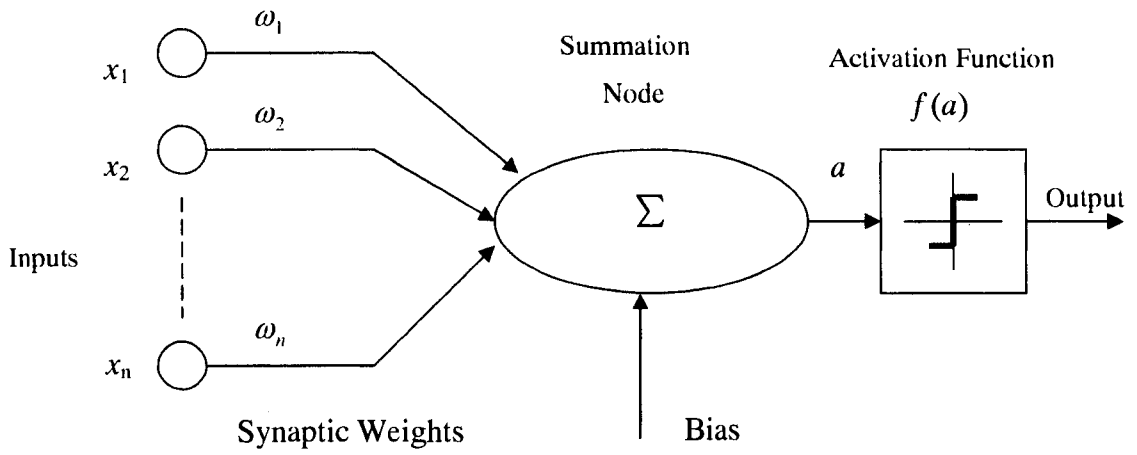


Fig 2.1 Artificial Neuron Model

A single neuron, called a perceptron, is characterized by the parameters:

$$\theta = (\omega_1, \omega_2, \ldots, \omega_n, b, f). \tag{2-1}$$

Each connection link has a synaptic weight that multiplies the signal transmitted. Each neuron has an activation function being applied to the weighted sum of the inputs signal and the bias to produce an output signal y :

$$y = f\left(\sum_{i=1}^{n} \omega_i x_i - b\right) \tag{2-2}$$

The activation function can be a threshold function, a piecewise-linear function or a sigmoid function.

A multi-layer neural network is a neural network with more than one layer of neurons. In this case, the activation functions of the different neurons can be different. There are no connections between neurons of the same layer, only weighted connections with neurons in the next layer [16], [17]. Figure 2.2 shows a typical multi-layer neural network architecture.

By using proper input and output training samples, an ANN can learn to approximate complex relationships between inputs and outputs. Through the learning process, an ANN can be configured for a specific application, such as approximation, pattern recognition or data classification. However, the accuracy of the output is limited because the variables are effectively treated as analog variables, and the recursive least squares algorithm used during the learning process does not necessarily lead to a null error. Also, the needed time for proper training of a neural network can be long.

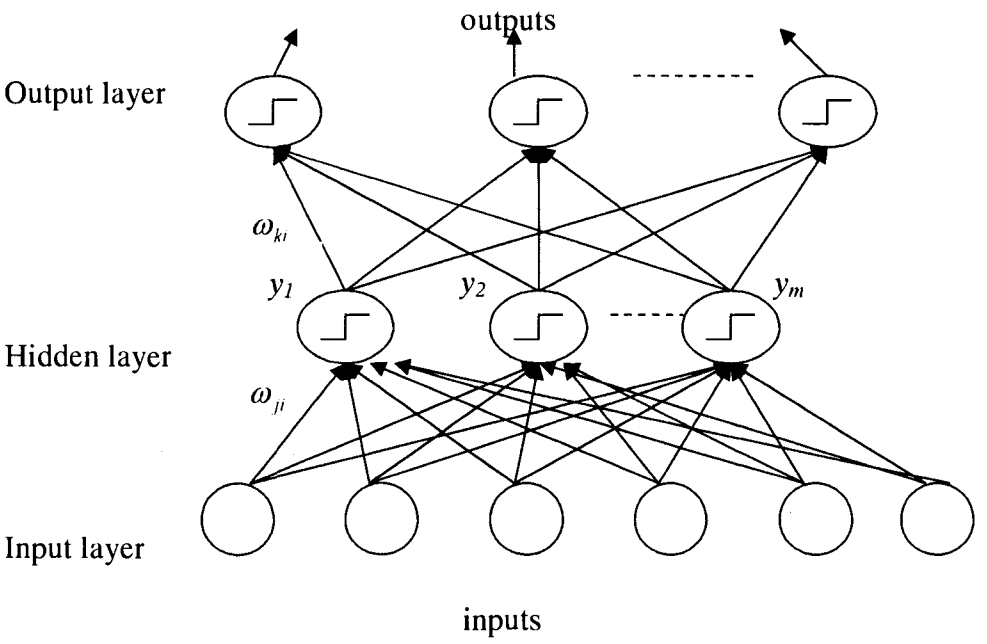


Fig 2.2 Typical Neural Network Architecture

Neural networks need to be trained before they can be used. They improve their performances during the learning process. Learning occurs through an iterative process of adjustment applied to all weights and bias of the network. Learning methods used for neural networks can be classified into two main categories: supervised learning and unsupervised learning [18].

2.1.1 Supervised learning

An essential factor of supervised or active learning is the availability of an external teacher, as showed in figure 2.3. The term "supervised" originates from the fact the desired response is provided by an external "teacher". Network parameters are adjusted under a combined influence of the training vector and the error signal; the error signal is defined as the difference between the actual response of the network and the desired response. Finally, through step by step adjustments, the error will approach 0. The backpropagation algorithm is widely used to implement this technique.

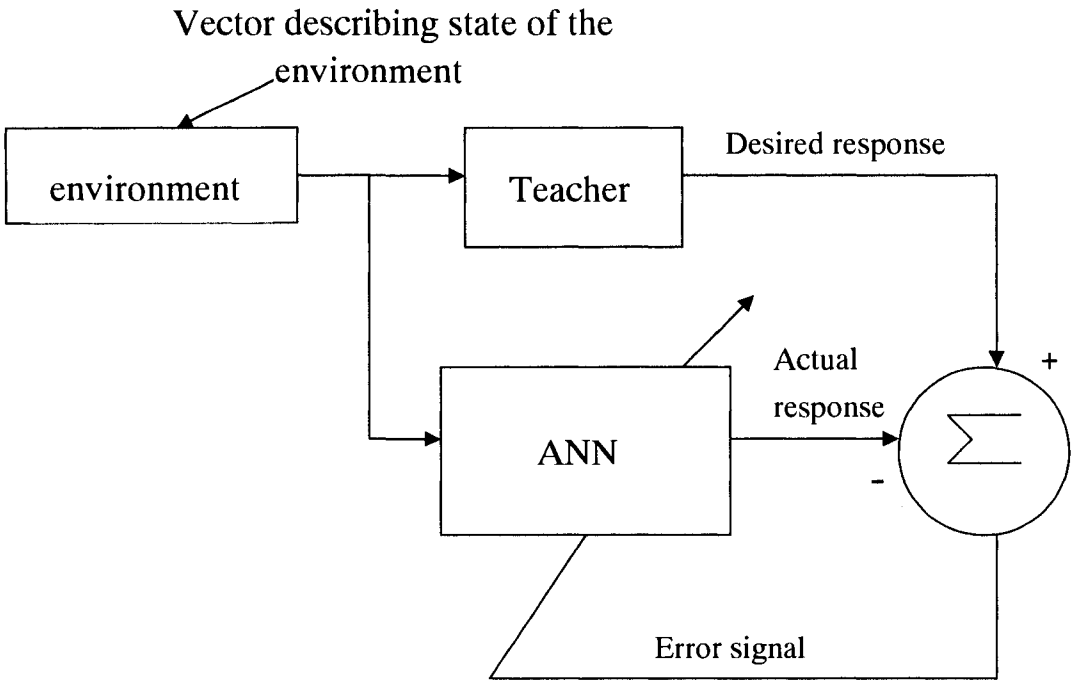


Fig 2.3 Supervised Learning

Backpropagation is a systematic method for training multiple-layer (three or more)

artificial neural networks. Standard backpropagation is a gradient descent algorithm, in which the network weights are moved along the negative of the gradient of a performance function. Properly trained backpropagation networks tend to give reasonable answers when presented with inputs that they have never seen before. In backpropagation, the output error on the training examples is used to adjust the network weights. Figure 2.4 shows a typical three layers network.

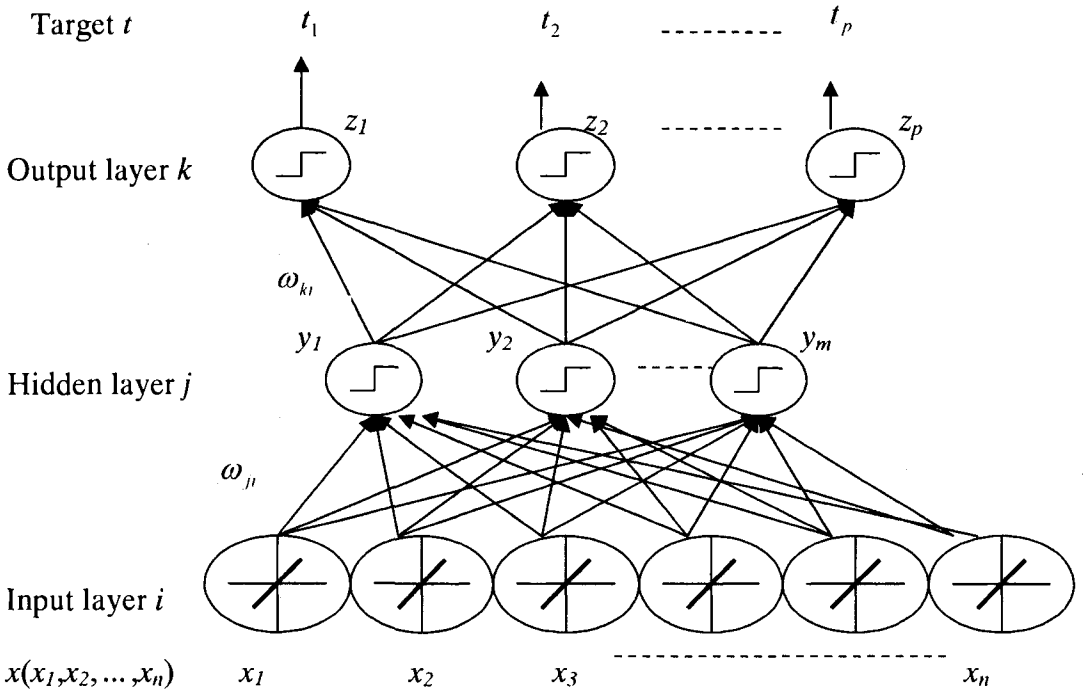


Fig 2.4 Typical BackPropagation Network

Let t_k be the k-th target (or desired) output, z_k be the k-th computed output with

$k = 1,\ldots, p$ and ω represents all the network weights and x the inputs.

The training error is given by:

$$\varepsilon(\omega) = \frac{1}{2}\sum_{k=1}^{p}(t_k - z_k)^2 \qquad (2\text{-}3)$$

The backpropagation learning rule is based on gradient descent. The change in each weight vector component is proportional to the negative of its gradient:

$$\Delta\omega = -\eta \frac{\partial \varepsilon}{\partial \omega} \tag{2-4}$$

Where η is a constant called the learning rate chosen between 0 and 1.

$$\omega(m+1) = \omega(m) + \Delta\omega(m) \tag{2-5}$$

where m is the m-th pattern presented.

Backpropagation involves two passes. In the forward pass, the input signals propagate through the network to the output. In the reverse pass, the calculated error signals propagate backwards through the network where they are used to adjust all the weights. Calculating the output is carried out, layer by layer, in the forward direction. The output of one layer is the input to the next layer. In the reverse pass, using the delta rule, the error value of each output neuron guides the adjustment of the associated weights. Since the middle-layer neurons have no target values, the error must be propagated back through the network layer by layer [19].

Suppose net_j is the inner product of the input layer signals with input weights for hidden unit j:

$$net_j = \sum_{i=1}^{n} x_i \omega_{ji} \tag{2-6}$$

The hidden unit output is then computed: $y_j = f(net_j)$, where $f(.)$ is a nonlinear activation function. The error on the hidden–to-output weights is given by:

$$\frac{\partial \varepsilon}{\partial \omega_{kj}} = \frac{\partial \varepsilon}{\partial net_k} \cdot \frac{\partial net_k}{\partial \omega_{kj}} = -\delta_k \frac{\partial net_k}{\partial \omega_{kj}} \tag{2-7}$$

where:

$$\delta_k = -\frac{\partial \varepsilon}{\partial net_k} = \frac{\partial \varepsilon}{\partial z_k} \cdot \frac{\partial z_k}{\partial net_k} = (t_k - z_k) f'(net_k) \tag{2-8}$$

Suppose net_k is the inner product of the input layer signals with input weights at the output unit k:

$$net_k = \sum_{j=1}^{m} y_j \omega_{kj} \tag{2-9}$$

On the output layer, the change in each weight vector component is:

$$\Delta \omega_{kj} = \eta \delta_k y_j = \eta (t_k - z_k) f'(net_k) y_j \tag{2-10}$$

Equations for the hidden layer are the same as for the output layer except the error term must be generated without a target vector.

Error on the input-to-hidden units is given by:

$$\frac{\partial \varepsilon}{\partial \omega_{ji}} = \frac{\partial \varepsilon}{\partial y_j} \cdot \frac{\partial y_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial \omega_{ji}} \tag{2-11}$$

where:

$$\begin{aligned} \frac{\partial \varepsilon}{\partial y_j} &= \frac{\partial \left[\frac{1}{2} \sum_{k=1}^{p} (t_k - z_k)^2 \right]}{\partial y_j} = -\sum_{k=1}^{p} (t_k - z_k) \frac{\partial z_k}{\partial y_j} = -\sum_{k=1}^{p} (t_k - z_k) \frac{\partial z_k}{\partial net_k} \cdot \frac{\partial net_k}{\partial y_j} \\ &= -\sum_{k=1}^{p} (t_k - z_k) \mathrm{f}'(net_k) \omega_{kj} \end{aligned} \tag{2-12}$$

According to (2-8) one can define:

$$\delta_j = f'(net_j) \sum_{k=1}^{p} w_{kj} \delta_k \tag{2-13}$$

The learning rule for the input-to-hidden weights is:

$$\Delta\omega_{ji} = \eta x_i \delta_j = \eta\left(\sum w_{kj}\delta_k\right) f'\left(net_j\right) x_i \qquad (2\text{-}14)$$

Backpropagation technique summary [20]:

1. Randomize the weights to small random values (both positive and negative) to ensure the network is not saturated by large values of weights.
2. Select a training pair from the training set.
3. Apply the input vector to network input.
4. Calculate the network output.
5. Calculate the error, the difference between the network output and the desired output.
6. Adjust the weights of the network in a way that minimizes this error.
7. Repeat steps 2-8 for each pair of input-output vectors in the training set until the error for the entire system is acceptably low.

The training algorithm updates the parameters to match one input-output pair at a time. Others algorithms, such as recursive least squares, minimize the summation of the matching error for all the input-output pairs.

2.1.2 Unsupervised learning

In unsupervised or self-organized learning there is no external teacher or critic to oversee the learning process, as showed in figure 2.5. The word "unsupervised" means that no target values are needed. In fact, for most varieties of unsupervised learning, the targets are the same as the inputs. Unsupervised learning usually performs tasks like: classify data or compressing information from inputs. A typical unsupervised learning technique is competitive learning.

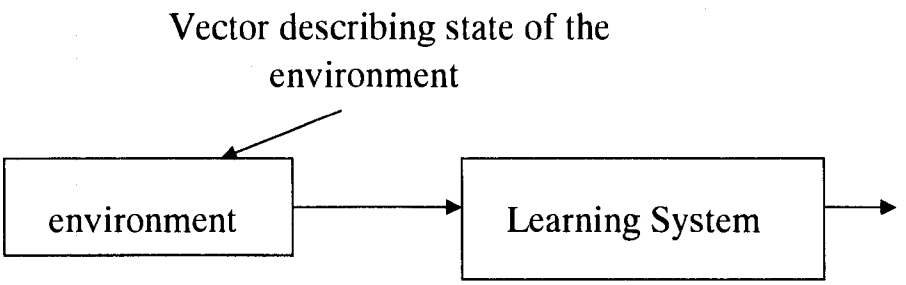


Fig 2.5: Unsupervised Learning

A basic competitive learning network has one layer of input neurons and one layer of output neurons (fig 2.6). An input pattern **X** is a sample point in the *n*-dimensional real vector space (R^n). Competitive learning is an unsupervised learning paradigm, so it just extracts information from the input patterns alone without the need for a desired response. In a competitive-learning network, a neuron's synaptic weight vector typically represents a set of related data points, and is able to find the largest or smallest output. It associates those groups with one another and with a specific proper response. Normally, when competition for learning is in effect, only the weights belonging to the winning processing element will be updated. This element is called "winner" and is determined as the element with the largest weighted-sum Net_i^k, where:

$$Net_i^k = w_i^T x^k \tag{2-15}$$

Where x^k is the current input. Thus the i^{th} element is the winning unit if:

$$w_i^T x^k \geq w_j^T x^k \quad \text{for all j} \neq \text{i} \tag{2-16}$$

which may be written as:

$$\|w_i - x^k\| \leq \|w_j - x^k\| \quad \text{for all j} \neq \text{i} \tag{2-17}$$

The concept of choosing a winner is a fundamental issue in competitive learning. In general, the winner is the best output (depending on the criteria) and this system is called "winner-take–all" network. The simplest architecture is a single layer of units, each receiving the same input $X \in R^n$ and producing an output Y. It is also assumed that only one unit is active at a given time see Figure 2.6.

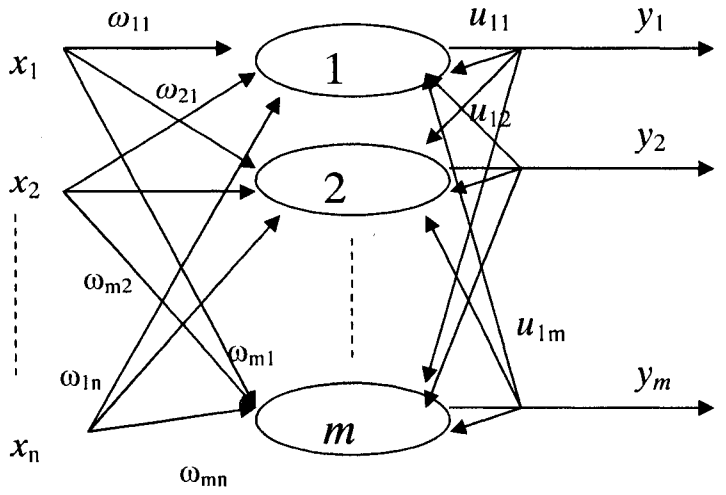


Fig 2.6: Single Layer Competitive Network

For a given input x_k drawn from a random distribution $\mathbf{p}(x)$, the weights of the winning unit are updated (the weights of all other units are left unchanged) according to:

$$\omega_i(t+1) = \omega_i(t) + \alpha(t)\cdot(x(t) - \omega_i(t)) \text{ if } i \text{ is winner index} \qquad (2\text{-}18)$$

$$\omega_i(t+1) = \omega_i(t) \quad \text{if } i \text{ is not winner index} \qquad (2\text{-}19)$$

Where α is a scalar called *adaptation gain* or *step rate*, $0 < \alpha(t) < 1$. The variable α controls how large the update's rate is at each step. If the step rate is close to 1, the network will converge quickly, but any new input can upset the cluster. On the other hand, if α is too small, the convergence will be too slow. A compromise is to use a variable step rate $\alpha(t)$ during learning. In general, there are three types of step

rate.

a. Constant learning rate

Suppose the learning rate is constant, so:

$$\alpha(t) = \alpha_0 \tag{2-20}$$

b. *K*-means learning rate:

$$\alpha(t) = \frac{1}{m} \tag{2-21}$$

Where the time parameter m stands for the number of input signals for which this particular unit has been winner so far.

c. Exponentially decaying learning rate

The exponential decay learning rate is given by:

$$\alpha(t) = \alpha_i \left(\frac{\alpha_f}{\alpha_i}\right)^{t/t_{max}} \tag{2-22}$$

Where α_i and α_f are the initial and final value of the learning rate, $t_{\max}$ is the total number of adaptation steps which is taken.

Every update strategy has its advantages and disadvantages: *K*-means is simple, but the harmonic series $\lim_{n\to\infty} \sum_{t=1}^{n} \frac{1}{t} = \infty$ diverges, so even after a large number of input signals and similarly low values of the learning rate arbitrarily large modifications of each input vector may occur. Constant learning rate has no convergence, but its practical effect is better than *K*-means in many cases. Exponentially decaying learning rate has good effect than the two other strategies, but in many cases $t_{\max}$ is unknown.

2.1.3 Clustering

To fuzzify input signals in a fuzzy system, it is important to have a series of proper membership functions. Clustering technology is a good way to compose these membership functions. The key idea is to use clustering technology to class the input space into a series of subspaces, and then, to use cluster to construct membership function for each subspace.

Clustering is one of the most important unsupervised learning technologies. A loose definition of clustering is "*the process of organizing objects into groups whose members are similar in some way*". The goal of clustering is to reduce the amount of data by categorizing or grouping similar data items together. Such grouping is pervasive in the way humans process information, and one of the motivations for using clustering algorithms is to provide automated tools to help in forming categories or taxonomies [21] [22].

A cluster is a collection of objects which have similitudes between them and are "dissimilar" to the objects belonging to other clusters. Competitive rule allows a single-layer linear network to group and represent data samples that lie in a neighborhood of input space. Each neighborhood is represented by a single output neuron. From the view of the input space, clustering divides the space into local regions, each of which is associated with an output neuron. If the vectors are in the same cluster then they are similar. It usually means that they are "close" to one another in the input space. Clustering is a mechanism that changes a continuous space to a series of discrete vectors. If the number of clusters is enough, the error between a data sample in the neighborhood and its center is small, so the fidelity is high. However at present time there is no algorithm to find the optimal clusters for an input space.

3 FUZZY LOGIC

Lotfi Zadeh, professor at the University of California at Berkeley, first introduced fuzzy logic in the mid-1960's. Fuzzy logic is a powerful problem-solving theory with many applications in control engineering and information processing. Fuzzy logic provides a remarkably simple way to draw definite conclusions from vague, ambiguous or imprecise information. In a sense, fuzzy logic resembles human decision making with its ability to work from approximate data and find precise solutions. Fuzzy logic incorporates an alternative way of thinking which allows modeling complex systems using a higher level of abstraction originating from our knowledge and experience. Fuzzy Logic allows to expresse this knowledge with subjective concepts such as very well or bright red to be maped into exact numerical ranges [23] [24].

3.1 Fuzzy sets and membership function

Fuzzy sets are extension sets of the classic sets. Fuzzy sets, unlike classic sets, have no crisp boundary, and they provide a gradual transition between "belonging to" and "not belonging to" a set. A fuzzy set in a universe of discourse **X** (which contains all the possible elements of concern in each particular context or application) is characterized by a membership function $\mu_A(x)$ that takes values in the interval [0, 1]. In other words, a fuzzy set is a generalization of a classical set by allowing the membership function to take any values in the interval [0, 1]. The fuzzy set theory makes it possible for an object or a case to belong to a set to a certain degree [25]. For instance, we might think that age 20 is "young" with membership value of 1. Age 30 has a membership value of 0.95. Age 60 has a membership value of 0.1, and so forth. That is to say, every people is "young" to a certain degree. Figure 3.1 describes a possible relationship between fuzzy sets "young" and "old".

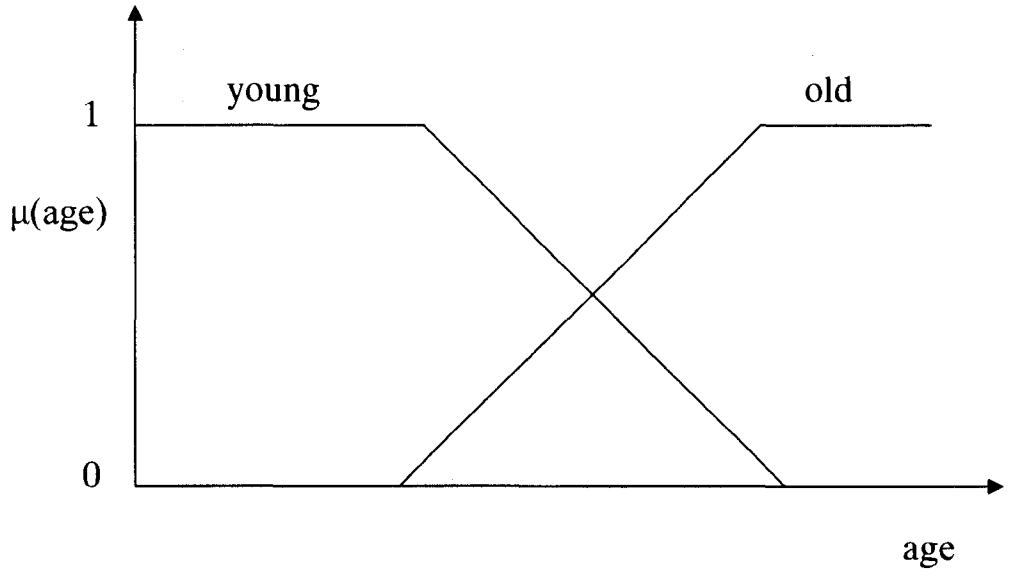


Fig 3.1 Fuzzy sets young and old

Curves in figure 3.1 are the membership function of the fuzzy set "young" and the fuzzy set "old". The membership value of each element is between 0 and 1. The values 0 and 1 describe "not belonging to" and "belonging to" a classic set respectively. Elements of a fuzzy set are taken from a *universe of discourse*, also called its universe. The universe contains all elements that can come into consideration. In general, fuzzy sets can either be discrete or continuous. For example, a fuzzy set A in universe $\mathbf{X}$ may be represented as a set of ordered pairs of a generic element x and its membership value, that is,

$$A = \{(x, \mu_A(x)) | x \in \mathbf{X}\} \tag{3-1}$$

When $\mathbf{X}$ is a discrete set:

$$\mathbf{X} = \{ x_1, x_2, x_3, \cdots, x_n \} \tag{3-2}$$

$$A = \sum_{x} \frac{\mu_A(x)}{x} \tag{3-3}$$

Where the summation sign does not represent arithmetic addition; it denotes the collection of all points $x \in \mathbf{X}$ with the associated membership function $\mu_A(x)$.

When **X** is a continuous and infinite set:

$$A = \int \frac{\mu_i}{x_i} \tag{3-4}$$

3.2 Linguistic variables and linguistic rules

Just like an algebraic variable takes numbers as values, linguistic variables take words or sentences as values. A linguistic variable is characterized by a quintuple $(\mathbf{X};\ \mathbf{T(X)};\ \mathbf{U};\ \mathbf{G};\ \mathbf{M})$ in which **X** is the name of the variable, **T(X)** is the *term set*, **U** is a *universe of discourse*, **G** is a *syntactic rule* to create the elements of **T(X)** and M is a *semantic rule* for associating meaning with the linguistic values of **X** [6]. In figure 3.1, the "age" is a linguistic variable.

Fuzzy ***if–then*** rules (linguistic rules) or fuzzy conditional statements are expressions of the form ***if*** *A* ***then*** *B*, where *A* and *B* are labels of fuzzy sets characterized by appropriate membership functions. Inputs of a fuzzy system are associated with the premise, and outputs are associated with the consequence. Assume that a fuzzy system has $m+k$ linguistic variables: m inputs x_1, x_2.....,x_m and k outputs y_1, y_2....., y_k and that n linguistic rules defined by:

R_1: *if* x_1 *is* $\mathbf{A}_{11}$ *and* x_2 *is* $\mathbf{A}_{12}$ *and*,.....,*and* x_m *is* $\mathbf{A}_{1m}$
Then y_1 *is* $\mathbf{C}_{11}$ *and* y_2 *is* $\mathbf{C}_{12}$ *and*,.....,*and* y_k *is* $\mathbf{C}_{1k}$

.....

R_n: *if* x_1 *is* $\mathbf{A}_{n1}$ *and* x_2 *is* $\mathbf{A}_{n2}$ *and*,.....,*and* x_m *is* $\mathbf{A}_{nm}$

Then y_1 *is* $\mathbf{C}_{n1}$ *and* y_2 *is* $\mathbf{C}_{n2}$ *and,.....,and* y_k *is* $\mathbf{C}_{nk}$

3.3 Fuzzy system

Fuzzy systems have been applied to a wide variety of fields ranging from control, signal processing and communication. However, the most significant applications are in control problems. In general, a fuzzy system consists of four components: fuzzification, fuzzy inference, defuzzification and fuzzy rule base (Figure3.2). The function of fuzzification is to map a crisp input to a linguistic value. After fuzzification, the inference engine uses a fuzzy ***if-then*** rules base and linguistic values to form linguistic output. Once linguistic output is available, the defuzzification step produces a final crisp value from linguistic values.

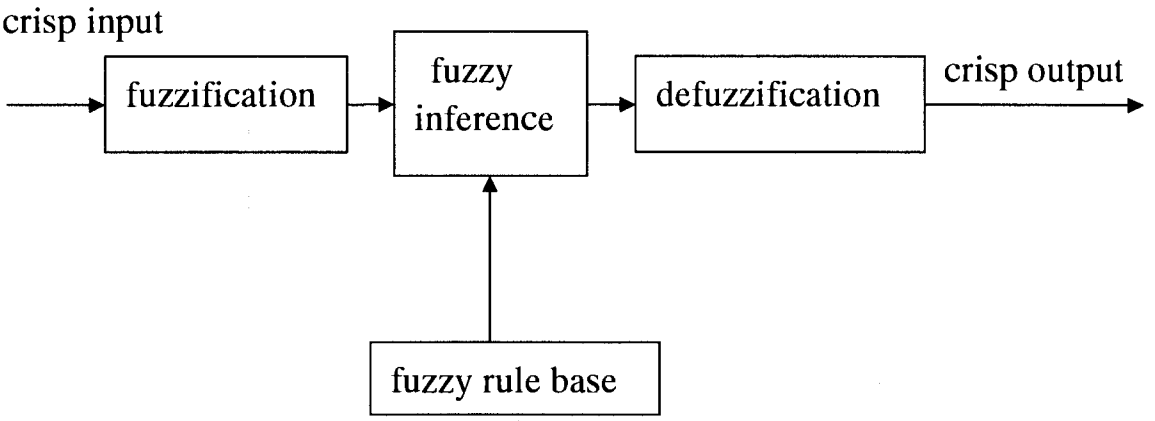


Fig 3.2 Fuzzy System

3.3.1 Fuzzification

Fuzzification transforms crisp values x_i into grades of membership for linguistic

terms of fuzzy sets A_i. The simplest form of fuzzification can be done using a fuzzifier function f that fixes the degree of fuzziness $\mu(A_i)$ in a set.

For example, suppose $\mathbf{A}_i$ is the input space, x_i is an input vector and $x_i \in \mathbf{A}_i$. The fuzzification performs to transform x_i to every fuzzy set A_i defined by function f :

$$\mu(A_i) = f(x_i) \tag{3-3}$$

In general, Gaussian or triangular membership functions are used as fuzzifier.

A Gaussian function is defined by:

$$\mu(x) = \exp(-\frac{(x-\alpha)^2}{2b^2}) \tag{3-4}$$

Where α is the peak value and b is the width of the membership function (fig 3.3).

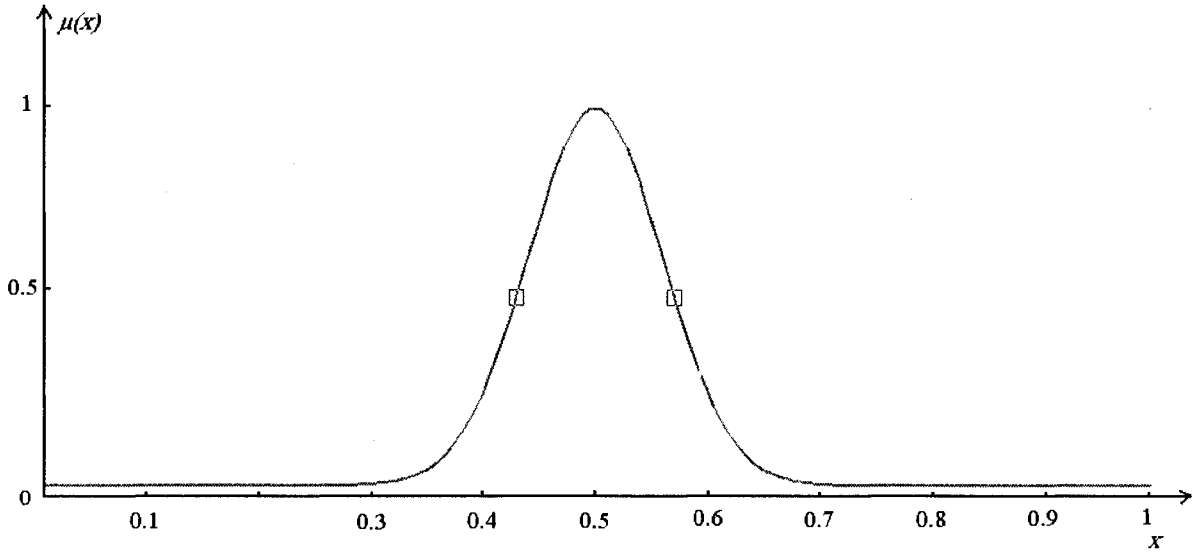


Fig 3.3 Gaussian Function

If membership functions are defined as symmetric or triangular function (fig 3.4), then their

general form is (3-5):

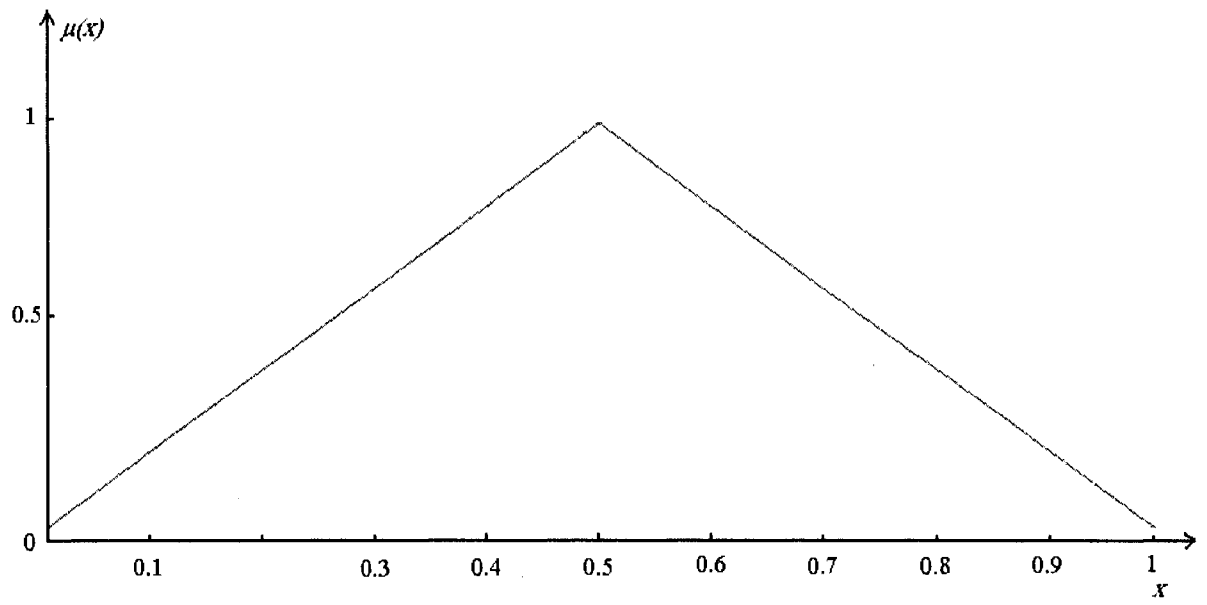


Fig 3.4 Triangular Function

$$\begin{cases} \mu(x) = 1 - (2 \cdot |x - a|) / b & \text{for } \dfrac{a-b}{2} < x < \dfrac{a+b}{2} \\ = 0 & \text{for otherwise} \end{cases} \qquad (3\text{-}5)$$

In triangular or Gaussian functions, there are two unknown parameters, center and width. In most case, the center points can be found by clustering technique, and the width factors can be determined by two ways.

a. Constant width

If the width is constant for all the membership functions, then the width can be defined by:

$$\delta = \frac{d(\max)}{\sqrt{2M}} \tag{3-6}$$

Where M is number of subsets, and $d(\max)$ is the maximum distance between those centers of subsets. This form would be close to the optimal solution if the data were uniformly distributed in the input space leading to a uniform distribution of the centers. Unfortunately, most real-life problems show non-uniform data distributions. The method is thus inadequate in practice and an identical width for all Gaussian kernels should be avoided since their widths should depend on the clusters position, which in turn depends on the data distribution in the input space.

b. Independent width

If the width of each Gaussian function is independent, the following form can be used to define the width of each function:

$$\delta_L = \frac{\min\{\|V_i - V_L\| \quad i, L = 1, 2, \cdots, k \quad i \neq L\}}{2} \tag{3-7}$$

Where k is the number of subsets, V_i and V_L are the center of subset, and i is not equal L. In practice, another form, based on the r-nearest neighbor of the center is often used:

$$\delta_L = \frac{(\sum_{i=1}^{r} \|V_L - V_i\|^2)^{\frac{1}{2}}}{r} \tag{3-8}$$

Where the V_i are the r-nearest neighbors of center V_L. A suggested value for r is $2k$. The form (3-7) and (3-8) offer the advantage of taking the distribution variations of the data into account. In practice, they are able to perform much better as they offer a greater adaptability to the data than a fixed-width procedure.

3.3.2 Defuzzification

The outputs of a fuzzy system are fuzzy. They often need to be converted into a scalar. This process is called defuzzification. Defuzzification is especially necessary for hardware application because conventional systems work with crisp data exchange. However, selecting a single scalar value from a series of fuzzy outputs is a challenging task. There are many defuzzification methods: Centroid Average (CA), Maximum Center Average (MCA), Mean of Maximum (MOM), Smallest of Maximum (SOM) and Largest of Maximum (LOM) Centroid Average and Maximum Center Average methods belong to continuous methods and are frequently used in control engineering and process modeling. The rest represents discontinuous methods, which are mainly used in decision-making and pattern recognition applications. For convenience, we could divide all the defuzzification methods into two basic mechanisms: centroid and maxima. The centroid methods are based on finding a point within the total geometric figure such as the weight of each fuzzy set, the area of the largest fuzzy set, or the area of the highest intersection.

a. The Center of Area Defuzzification:

The Center of Area (COA) method is often referred to as the Center-of-Gravity because it calculates the centroid of the composite area representing the output fuzzy term (3-9). It specifies the crisp value y* as the center of the area covered by the membership function of fuzzy set A.

$$y^* = \frac{\sum_{i=1}^{N} y_i \mu_A(y_i)}{\sum_{i=1}^{N} \mu_A(y_i)} \tag{3-9}$$

The summation is carried overvalues of the universe of discourse y_i sampled at N points. See figure 3.5.

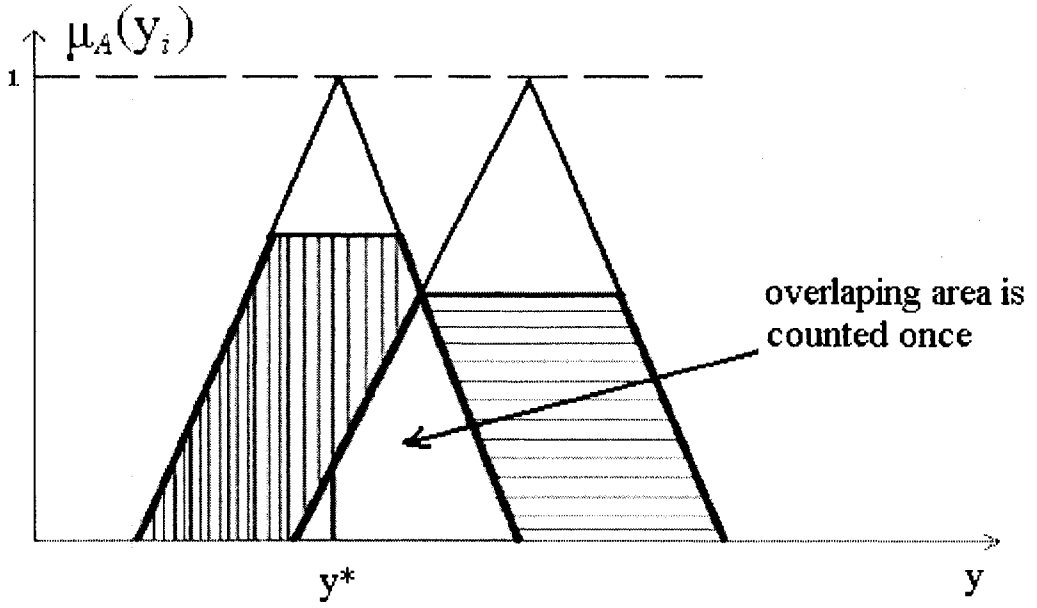


Fig 3.5: COA Defuzzification

If we regard $\mu_A(y_i)$ as the probability density function of a random variable, then the center of area defuzzification gives the mean value of the random value. The advantage of COA lies in its intuitive likelihood. The disadvantage is the difficulty to calculate the summation in most real case since. COA favors "central" values in universe of discourse and that, because of its complexity, it may lead to rather slow inference cycles If the areas of two or more contributing rules overlap, the overlapping area is counted only once. This defuzzification method is often used [26] [27].

b. The center of sums defuzzification

The Center of Sums (COS) method builds the resulting membership function by taking the sum of output from each contributing rule. So, this sum is not just the union. The overlapping areas are counted more than once. This method can be carried out easily and leads to rather fast inference cycles and it is the most commonly used defuzzification method.

$$y^* = \frac{\sum_{i=1}^{N} y_i \sum_{k=1}^{n} \mu_A(y_i)}{\sum_{i=1}^{N} \sum_{k=1}^{n} \mu_A(y_i)} \qquad (3\text{-}10)$$

Where $\mu_A(y_i)$ is the membership function which is at point y_i of the universe of discourse resulting from the firing of the k^{th} rules (figure 3.6).

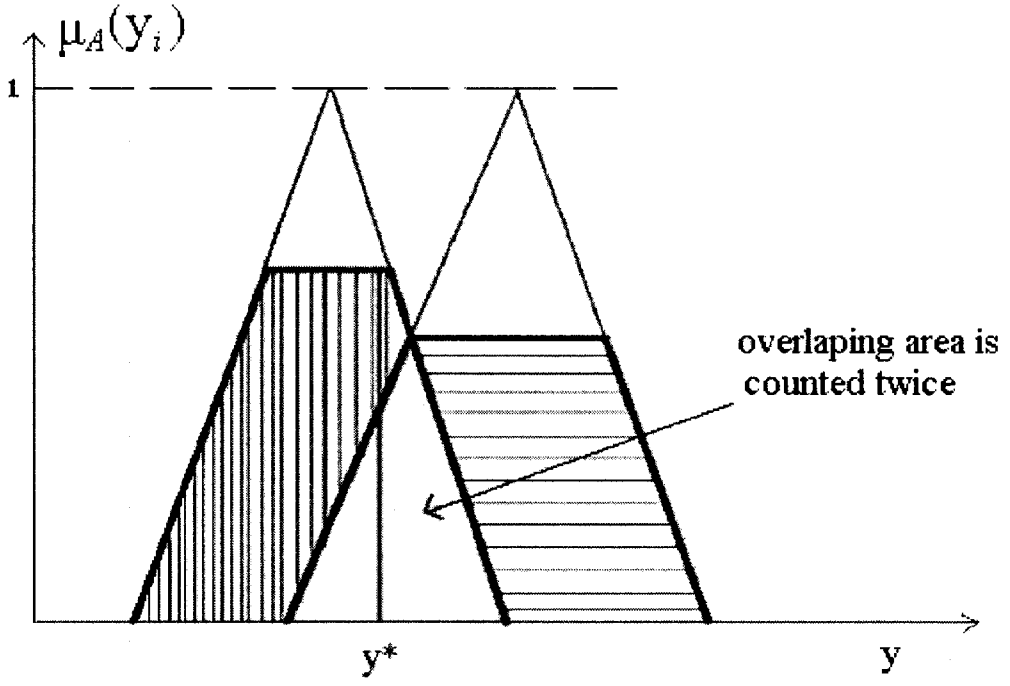


Fig 3.6 COS Defuzzification

c. The Mean of Maxima method:

The Mean of Maxima method (MOM) a simple method for doing the defuzzification. The output takes the crisp value of the highest degree of membership in $\mu_A(y_i)$. If a system has more than one element in the universe of discourse having the same maximum value, a random selection can be used or a mean value is performed. Suppose, there are M such maxima in a discrete universe of discourse. The crisp output can be obtained by:

$$y^* = \sum_{m=1}^{M} \frac{y_m}{M} \tag{3-11}$$

Where y_m is the m^{th} element in the universe of discourse where the membership function of $\mu_A(y_i)$ is at the maximum value, and M is the total number of such elements. MOM defuzzification is faster than COA. Furthermore, it allows the controller to reach values near the edges of the universe of discourse. This method does not consider the overall shape of the fuzzy output $\mu_A(y_i)$ (fig 3.7).

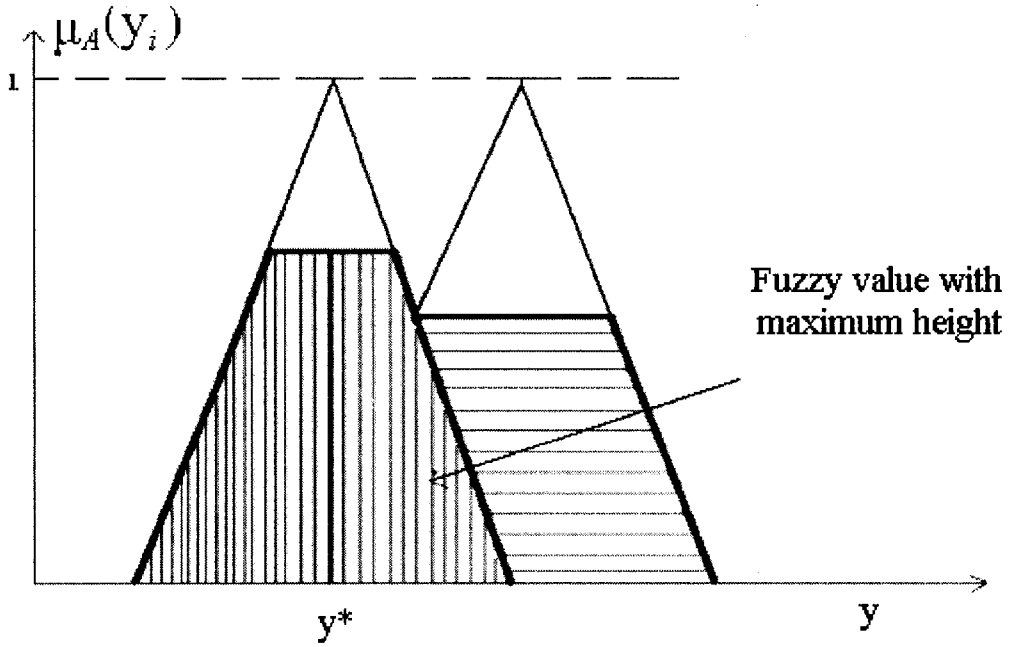


Fig 3.7 MOM Defuzzification

3.3.3 Fuzzy Inference

To draw conclusions from a rule base, a mechanism that can produce an output from a collection of ***If-then*** rules is necessary. This mechanism is called fuzzy inference. There are two major inference methods: T-S fuzzy inference and Mamdani fuzzy

inference. The Mamdani fuzzy inference method is the most common method.

a. Mamdani fuzzy inference:

The Mamdani method was proposed in 1975 [28], to control a steam engine and boiler combination. In this application, the set of fuzzy rules was supplied by experienced human operators.

The following example is a typical Mamdani fuzzy rule.

R_i: *if* x_1 *is* A_{i1} *and* x_2 *is* A_{i2} *and*.....*and* x_r *is* A_{ir} then y is C_i (3-12)

Where R_i is the i^{th} fuzzy rule; x is an input and A is a fuzzy set.
The following two examples explain the Mamdani fuzzy inference.

Single fuzzy rule

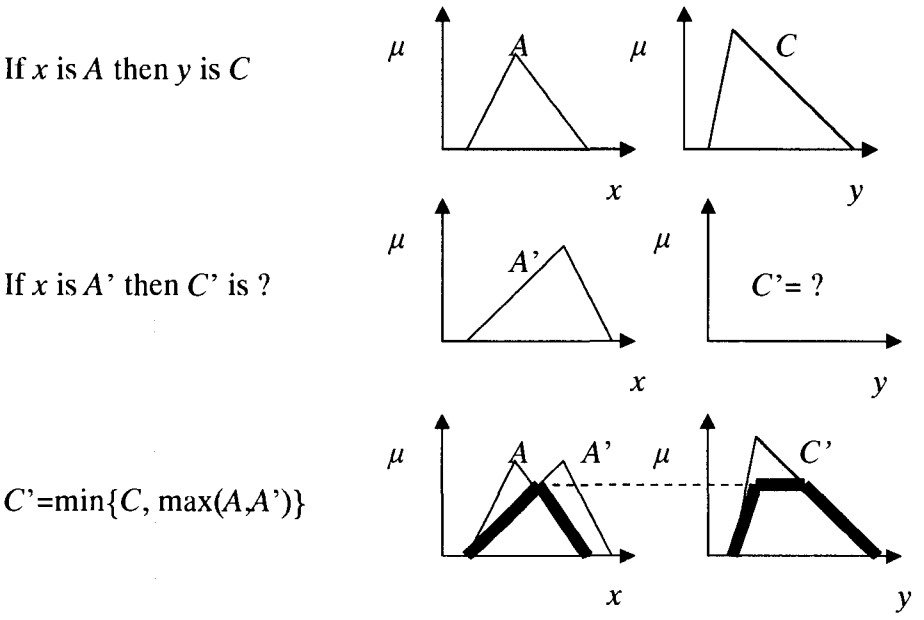


Fig 3.8 Simple fuzzy rule Mamdani method

Multiple fuzzy rules

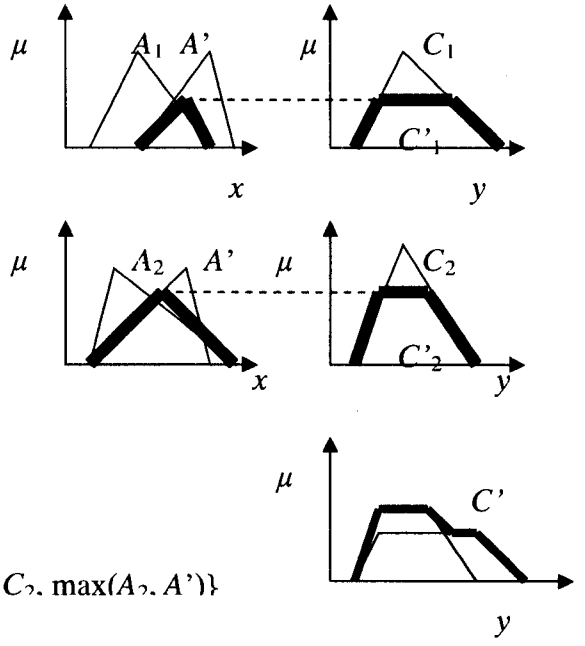


Fig 3.9 Multiple Fuzzy Rules of Mamdani Method

b. Takagi-Sugeno fuzzy inference

The Takagi-Sugeno fuzzy inference method was introduced in 1984 and it is similar to the Mamdani method. The first two parts of the fuzzy inference process, inputs fuzzification and fuzzy operators, are the same. The main difference between Mamdani and Sugeno is Takagi-Surgeon approximates a nonlinear system by a combination of several linear systems (see figure 3.10). It breaks down the input space into several subspaces and represents the input/output relation in each subspace by a linear equation. A typical model of Takagi-Sugeno fuzzy inference is:

If input $x = A_i$ and input $y = B_j$, then output is $z = ax + by + c$. (3-13)

Where a, b, c are numerical constants.

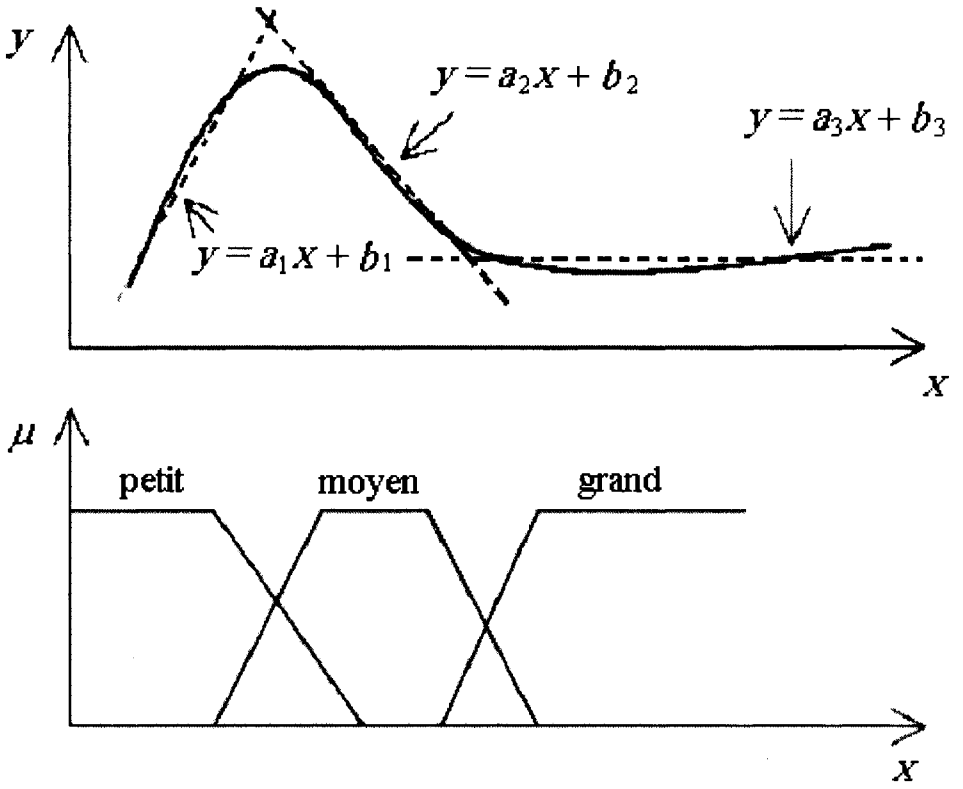


Fig 3.10 Linear approximation by 3 fuzzy rules

In general, Takagi-Sugeno rules have the following form:

If x_1 is A_{i1}, and ... and x_r is A_{ir} then $y^i = f_i(x_1, x_2, \cdots, x_r) = b^i + a_1^i x_1 + \cdots + \alpha_m^i x_m$ (3-14)

Where f_i is the linear function, a and b are parameters of this linear function, y_i is the inference consequence of this rule.

T-S method can be easily represented and evaluated by a three-layers' neural network. Let us suppose a multiinput system with n inputs $\mathbf{D}(x_1, x_2, x_3, x_4, ..., x_n)$ and an output $\mathbf{Y}(y_1, y_2, y_3, y_4, ..., y_k)$ (figure3.11).

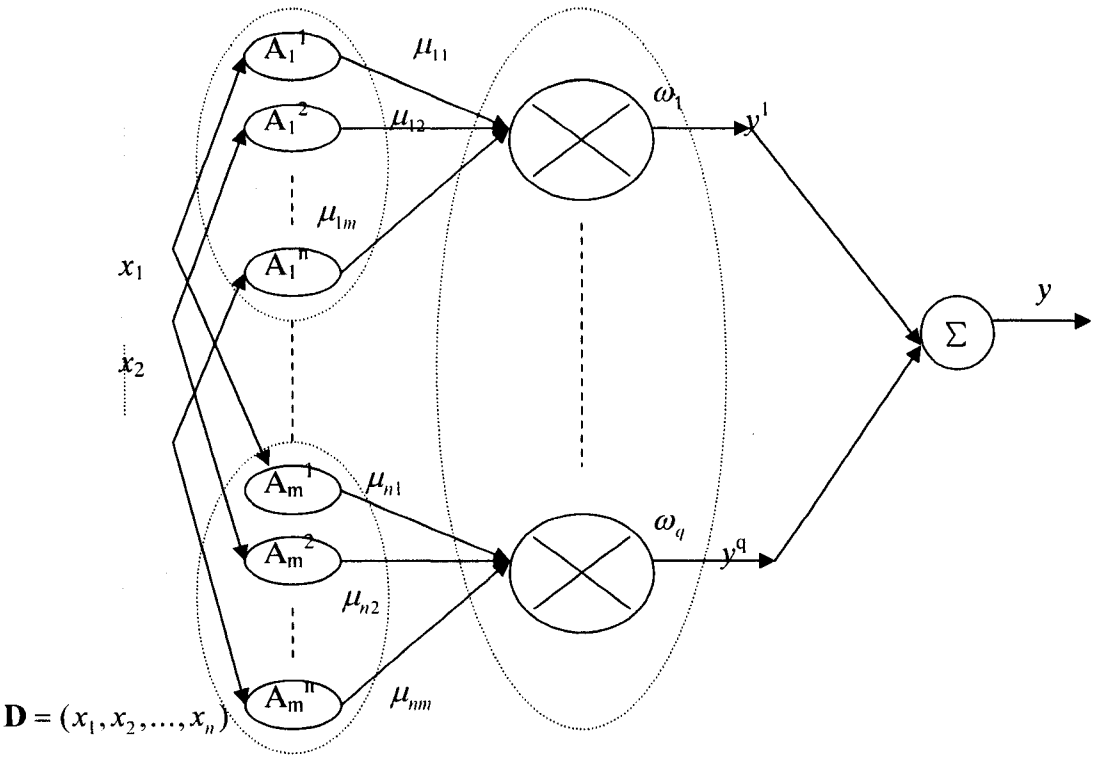


Fig 3.11 Takagi-Sugeno Fuzzy System

Layer 1:

The nodes of first layer receive input vector $\mathbf{D}(x_1, x_2, x_3, x_4, \ldots, x_n)$ and act as fuzzy sets representing the terms of the matching input variable. Nodes in this layer are arranged into *n* groups; each group representing the ***if***-part of fuzzy rule. Each node acts as a fuzzifier. Hence, the node outputs are in the range [0, 1] and are calculated by the following function:

$$\mu_{ij}(x_i) = \exp\frac{-(x_i - a_{ij})^2}{2b_{ij}^2} \qquad (3\text{-}15)$$

Layer 2:

The number of nodes in this layer is equal to the number of fuzzy rules. A node in this layer represents a fuzzy rule. Each node in this layer is a product operator. The "min" operator is not used, because the function "min" is not differentiable in the set of real numbers $\Re$. Outputs of this node are:

$$\omega_i = \mu_{1i}(x_1)\,\mu_{2i}(x_2)\cdots\mu_{mi}(x_m)\,(1 \le i \le q) \tag{3-16}$$

$$= \prod_{j=1}^{n} \mu_{ij}(x_i) \tag{3-17}$$

ω is called the firing strength.

Layer 3:

Nodes in this layer represent the output variables of the system. Each node acts as a defuzzifier. Outputs of this node are:

$$y = \frac{\omega_1}{\sum_{i=1}^{q}\omega_i}.y^1 + \frac{\omega_2}{\sum_{i=1}^{q}\omega_i}.y^2 + \cdots + \frac{\omega_q}{\sum_{i=1}^{q}\omega_i}.y^q \tag{3-18}$$

$$= \frac{\sum_{i=1}^{q}\omega_i.y^i}{\sum_{i=1}^{q}\omega_i} \tag{3-19}$$

$$= \sum_{i=1}^{q}\overline{\omega}_i y^i \tag{3-20}$$

The firing strengths ω_i are normalised and outputs y are the crisp output sets weighted by the firing strengths.

A fuzzy rule (3-14) has four basic elements: input $(x_1, x_2, x_3, x_4 ..., x_n)$, input space $(A_1^l, A_2^l, \cdots, A_m^l)$, output y and parameters of consequence $(b^l, a_1^l, \cdots, a_m^l)$. In certain degree, the object of training is to draw the parameters of consequence, so

the following contents will introduce how to draw the parameters of consequence matrix **P.**

Suppose that we have n rules R^i $(i=1,2,\cdots,n)$ of the form:

R^i: IF x_1 is $A_1{}^i$ and x_2 is $A_2{}^i$ and ... and x_m is $A_m{}^i$, then $y^i=b^i+a_1{}^i x_1+a_2{}^i x_2+\ldots+a_m{}^i x_m$

If we apply the j^{th} sample to the i^{th} rule R^i, we obtain:

R_j^i: IF x_{1j} is $A_{1j}{}^i$ and x_{2j} is $A_{2j}{}^i$ and ... and x_{mj} is $A_{mj}{}^i$, Then $y_j^i=b^i+a_1{}^i x_{1j}+a_2{}^i x_{2j}+\ldots+a_m{}^i x_{mj}$

Where $i=1,2,\ldots,n$ and $j=1,2,\ldots,r$.

So the weights of the rules are:

$$w_j^i = x_{1j} \text{ is } A_{1j}{}^i \text{ and } x_{2j} \text{ is } A_{2j}{}^i \text{ and} \ldots \text{and } x_{mj} \text{ is } A_{mj}{}^i$$

$$= \prod_{j=1}^{n} \mu_{A_{kj}}(x_{kj}) \qquad k=1,2,\ldots,m \tag{3-21}$$

According to (3-19), the output y_j is given by:

$$y_i = \frac{\sum_{i=1}^{n} \omega_j^i y_j^i}{\sum_{i=1}^{n} \omega_j^i}$$

$$= \sum_{i=1}^{n} \beta_{ij} y_j^i \tag{3-22}$$

Where :

$$\beta_{ij} = \frac{\omega_j^i}{\sum_{i=1}^{n} \omega_j^i} \tag{3-23}$$

The output y_j can be expressed by:

$$y_j = \begin{bmatrix} \beta_{1j} & \beta_{2j} & \cdots & \beta_{nj} \end{bmatrix} \begin{bmatrix} y_j^1 \\ y_j^2 \\ \vdots \\ y_j^n \end{bmatrix}$$

$$= \begin{bmatrix} \beta_{1j} & \beta_{2j} & \cdots & \beta_{nj} \end{bmatrix} \begin{bmatrix} b^1 & a_1^1 & \cdots & a_m^1 \\ b^2 & a_1^2 & \cdots & a_m^2 \\ \vdots & \vdots & \vdots & \vdots \\ b^n & a_1^n & \cdots & a_m^n \end{bmatrix} \begin{bmatrix} 1 \\ x_{1j} \\ \vdots \\ x_{mj} \end{bmatrix} \qquad (3\text{-}24)$$

So, the vector of output **Y** can be expressed by:

$$\mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_r \end{bmatrix} = \begin{bmatrix} \beta_{11} & \beta_{21} & \cdots & \beta_{n1} \\ \beta_{12} & \beta_{22} & \cdots & \beta_{n2} \\ \vdots & \vdots & \vdots & \vdots \\ \beta_{1r} & \beta_{2r} & \cdots & \beta_{nr} \end{bmatrix} \begin{bmatrix} b^1 & a_1^1 & \cdots & a_m^1 \\ b^2 & a_1^2 & \cdots & a_m^2 \\ \vdots & \vdots & \vdots & \vdots \\ b^n & a_1^n & \cdots & a_m^n \end{bmatrix} \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_{11} & x_{12} & \cdots & x_{1r} \\ \vdots & \vdots & \vdots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mr} \end{bmatrix} \qquad (3\text{-}25)$$

We can rewrite in the form:

$$\mathbf{Y} = \mathbf{X}\,\mathbf{P} \qquad (3\text{-}26)$$

Where:

$$\mathbf{Y} = \begin{bmatrix} y_1 \\ \vdots \\ y_r \end{bmatrix} \qquad (3\text{-}27)$$

$$\mathbf{X} = \begin{bmatrix} \beta_{11} & \cdots & \beta_{n1}, & x_{11}\beta_{11} & \cdots & x_{11}\beta_{n1} & \cdots & x_{m1}\beta_{11} & \cdots & x_{m1}\beta_{n1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \beta_{1r} & \cdots & \beta_{nr}, & x_{1r}\beta_{1r} & \cdots & x_{1r}\beta_{nr} & \cdots & x_{mr}\beta_{1r} & \cdots & x_{mr}\beta_{nr} \end{bmatrix} \qquad (3\text{-}28)$$

$$\mathbf{P}=\begin{bmatrix} b^1 \\ \vdots \\ b^n \\ \vdots \\ a_m^1 \\ \vdots \\ a_m^n \end{bmatrix} \tag{3-29}$$

X is a vector matrix with $r \times n(m+1)$ dimensions,

Y is a vector with *r* dimensions,

P is a vector with $n \times (m+1)$ dimensions.

Finally, vector **P** is expressed by:

$$\mathbf{P} = [\mathbf{X}^T\mathbf{X}]^{-1}\mathbf{X}^T\mathbf{Y} \tag{3-30}$$

3.4 Fuzzy control system

Control systems are classified in two categories: open loop control systems, in which the control action is independent of physical system output, and closed loop system, also known as feedback control systems. The physical system under control is called a plant. In a closed loop system, a sensor measures the output signal and returns it to the input signal. To guarantee satisfactory responses and performance, it is necessary to use an additional system, known as a controller or a compensator into the loop. Currently, most controllers are based on proportional integral derivative (PID) technology. PID controllers work well in many control applications. But in practice, the system to control always contains uncertainty, is complex, poorly understood, or nonlinear. So it is difficult to build a precise mathematic model and is difficult to linearize. A fuzzy controller unlike a PID controller which needs a deep understanding of the system, exact equations and precise numerical values, uses a simple, rule-based *if x* AND *y then z* approach rather than trying to model a system mathematically. Fuzzy logic provides an alternative solution for nonlinear

applications. Fuzzy inference processes results in improved performance, simpler implementation, and reduced design costs. In many applications, fuzzy logic can lead to better control performances than linear, piecewise linear, or lookup table techniques. Most control applications have multiple inputs and require modeling and tuning of many parameters which makes implementation tedious and time-consuming. Fuzzy rules can help to simplify implementation by combining multiple inputs into single *if-then* statements while still handling nonlinearity.

In general, a fuzzy control system includes three major classes: single input single output (SISO), multi-input single output (MISO), and multi-input multi-output (MIMO).

3.4.1 SISO AND MISO

The "SISO" fuzzy controller is widely applied in practice. This class of controllers is simple (Fig 3.12).

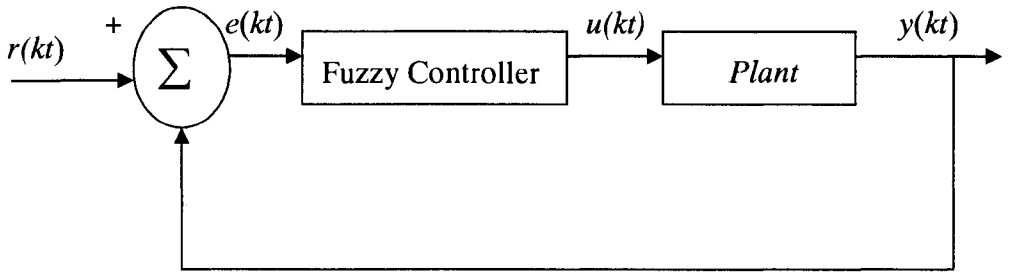


Fig 3.12 Structure of a SISO

However, for dynamic systems, it is better to track also the trend of the error's change as shown in (Fig 3.13).

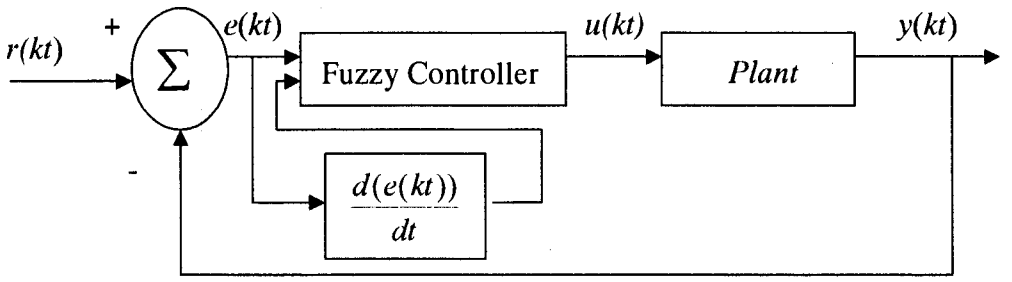


Fig 3.13 Structure of a MISO with differential input

3.4.2 MIMO

The design of controllers for MIMO systems has always been a hard problem even for the linear ones. Currently, there is no complete theory to design MIMO systems. Most of existed techniques are quite complicated. The main difficulties is the derivation of rules is not easy and the number of rules is too high, however it is possible to describe a "MIMO" system (figure 3.14) with several "MISO" systems (figure 3.15).

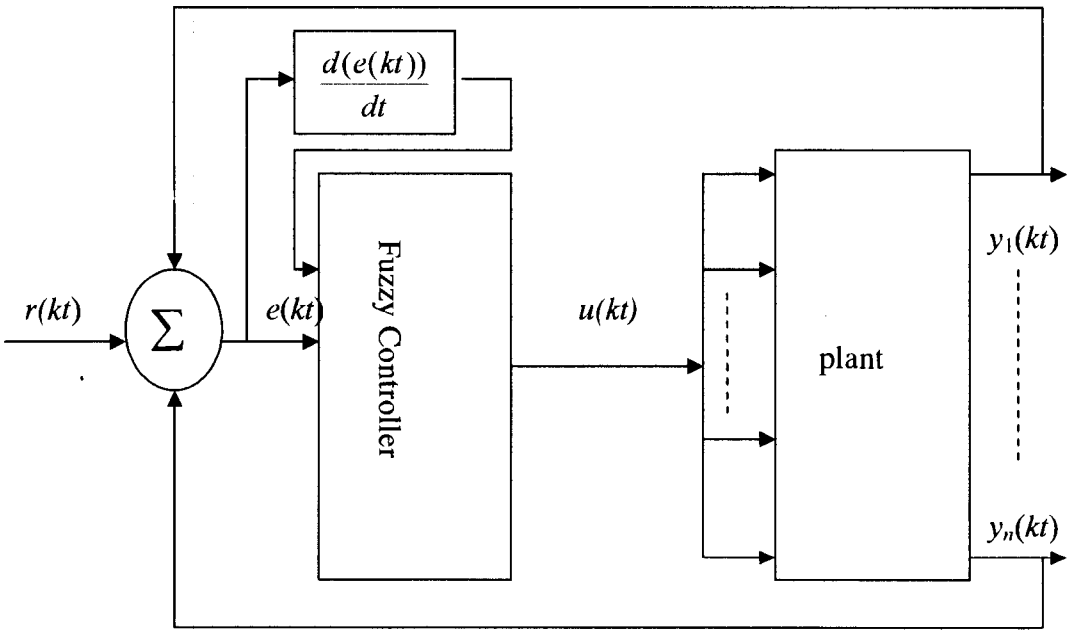


Figure 3.14 Structure of a simple MIMO

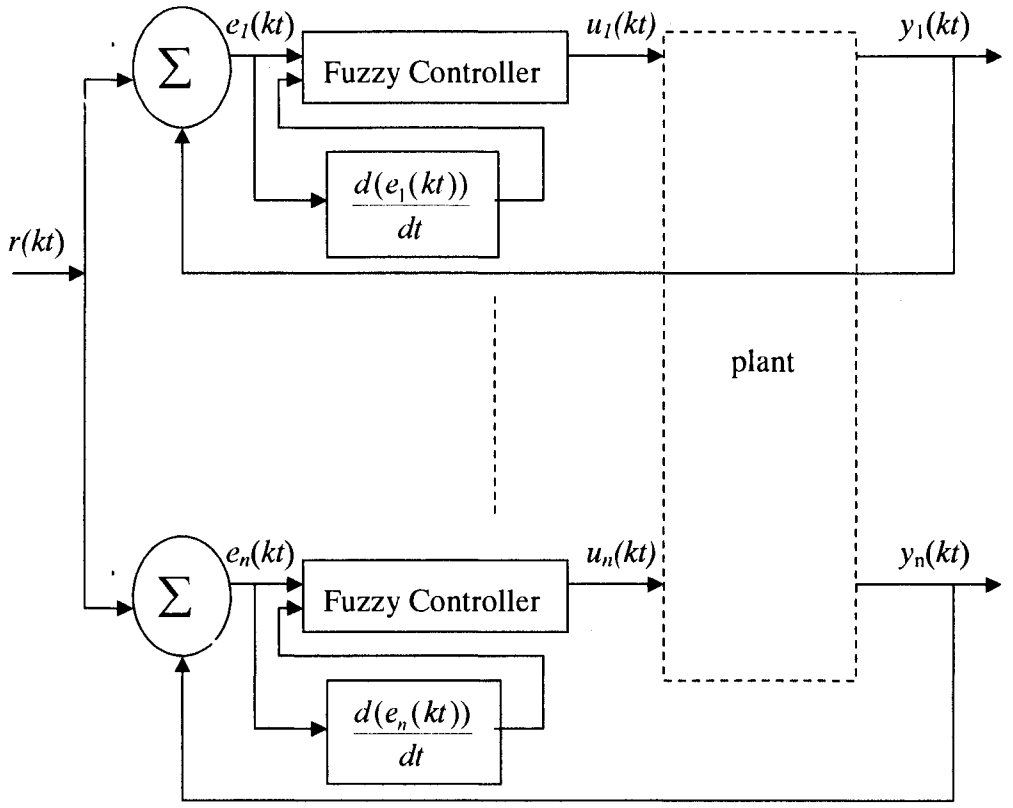


Figure 3.15 Implementation of a MIMO system with several MISO systems

4 DESIGN OF A HIGH-DIMENSIONS NEURAL FUZZY CONTROLLERS FOR NONLINEAR SYSTEMS

Fuzzy controller are widely applied and fuzzy systems have been proved to be able to approximate nonlinear functions with arbitrary accuracy [29] [30]. Generally speaking, fuzzy systems and neural networks have much equivalence [31] [32]. In fact, fuzzy control systems can be regard as special neural networks control systems and many neural networks learning algorithms can be used in fuzzy control systems. Neural fuzzy systems have good capacity for approximations; however, there are still some difficulties.

- Many fuzzy rules are needed, which leads to excessive training time, particularly with multidimensional input.
- Algorithms are complex and problem adapted.
- The reliability is low. For example, for a same model or function, the learning results are dependent on the training sets.

To solve these problems, Jian-qin Chen and Yu-Geng Xi [33] developed a learning method. The basic idea is to use competitive learning for partition the input space to fix the decision boundaries for local regions in an input space, and then to learn the consequent part of the fuzzy rules by RLS (Recursive Least Square). In this method, the boundaries decision is an important step. The choice of decision boundaries is useful for dealing with the conflict between over fitting and generalization as well as reducing the number of local input regions in certain degree. The following section describes the decision boundaries algorithm:

Step 1. Freeze the cluster centers v_i (i=1,2,... ,*cluster_number*).

Step 2. Let r_i=0 (i=1,2,... , *cluster_number*) at first

Step 3. For any input $x(t)$ find out v_c by

$$\|x(t)-v_c\| = \min\|x(t)-v_j\|. \qquad (4\text{-}1)$$

Step 4. Update r_c

$$\text{if} \quad \|x(t) - v_c\| > r_c, \quad \text{then} \quad r_c(k+1) = \|x(t) - v_c\| \tag{4-2}$$

$$\text{if} \quad \|x(t) - v_c\| \le r_c, \quad \text{then} \quad r_c(k+1) = r_c(k) \tag{4-3}$$

Where v_i is the i^{th} cluster and r_i is the radius of the local regions which center is v_i.

The algorithm for determination of decision boundaries is efficient and simple. However in practice, this algorithm has some drawbacks. For example; two very closed points A and B, with similar characteristics, can belong to different clusters ("A" belongs to cluster N and M. "B" belongs to cluster M and P). This may lead to output errors (called *boundary error*). To decrease this type of errors and to improve the precision, it is necessary to increase the number of clusters and to lessen theirs radiuses. This method needs many clusters to guarantee a good accuracy.

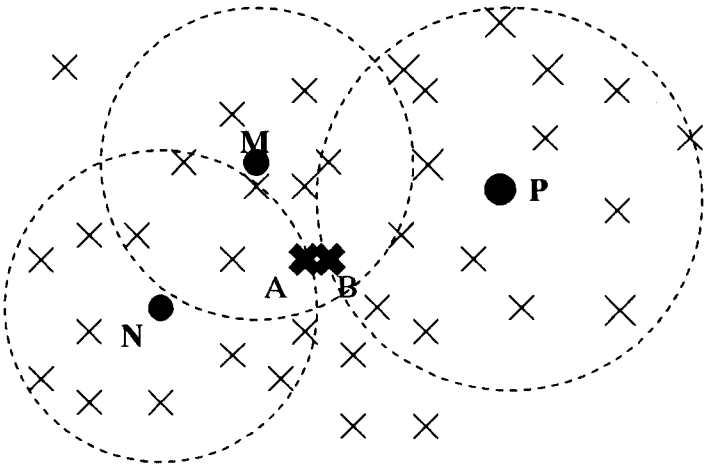


Fig 4.1 Boundary error during a clustering process

4.1 High-dimensions neural fuzzy inference (HDNFI)

To solve the problem mentioned above, in inference phase, a high-dimension neural fuzzy inference (HDNFI) which based on T-S model is proposed in this work (see figure 4.2)

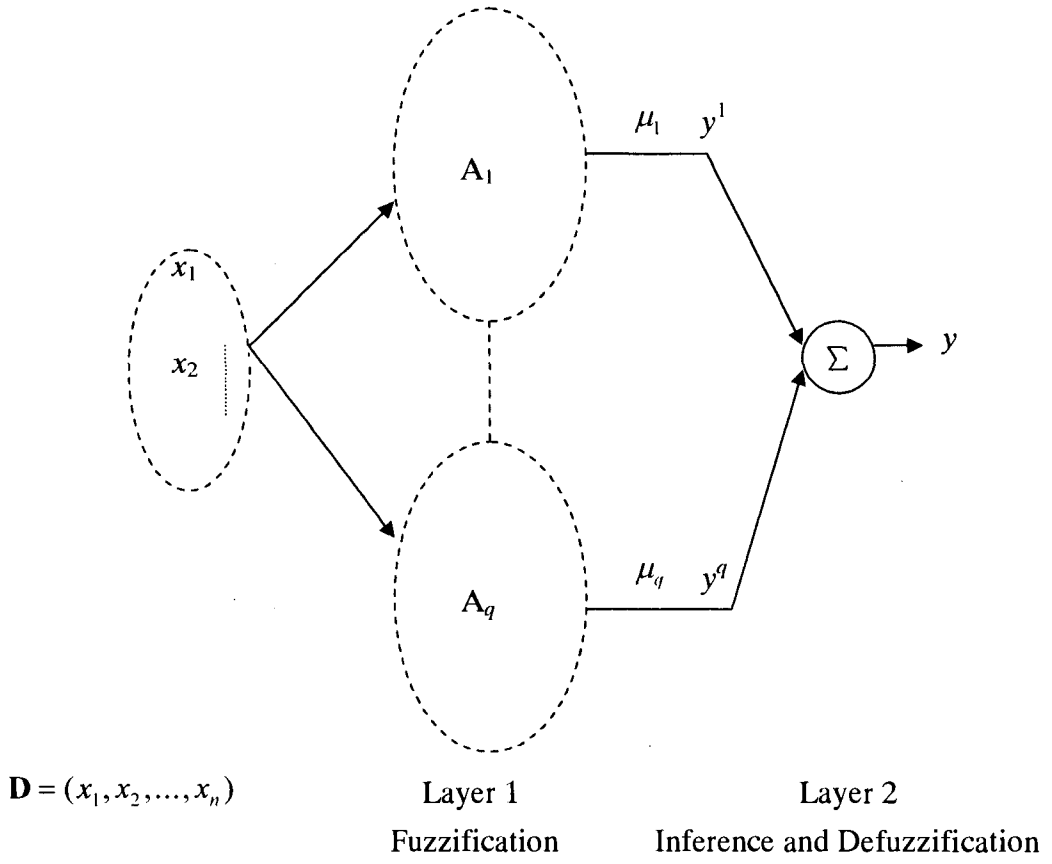


Fig 4.2 High-dimension fuzzy inference

The basic idea of this method is to consider each data in this system as a point with high-dimension. Each dimension of input **D** will be processed at same time in same high-dimension subsets.

Layer 1:

The nodes of first layer receive input vector $\mathbf{D}(x_1, x_2, x_3, x_4, \ldots, x_n)$ and each node acts as a fuzzifier. Therefore, the node outputs are in the range [0, 1] and are calculated by the following function:

$$\mu_i(\mathbf{D}) = \exp\frac{-(\mathbf{D}-\mathbf{c}_i)^2}{2b_i^2} \tag{4-4}$$

Where $\mathbf{c}_i$ is the center and b_i is the width of membership function (4-4), $\mathbf{c}_i$ and $\mathbf{D}$ has same dimensions.

Layer 2:

This layer includes fuzzy inference and defuzzification. In this layer, the variable μ_i is the degree of fuzziness of input $\mathbf{D}$ in subset $\mathbf{A}_i$ and suppose the firing strength ω_i equal to μ_i. The ouput is given by:

$$y = \frac{\omega_1}{\sum_{i=1}^{q}\omega_i}.y^1 + \frac{\omega_2}{\sum_{i=1}^{q}\omega_i}.y^2 + \cdots + \frac{\omega_q}{\sum_{i=1}^{q}\omega_i}.y^q \tag{4-5}$$

$$= \frac{\sum_{i=1}^{q}\omega_i.y^i}{\sum_{i=1}^{q}\omega_i} \tag{4-6}$$

$$= \sum_{i=1}^{q}\bar{\omega}_i y^i \tag{4-7}$$

Firing strengths ω_i are normalised and outputs y are the crisp output sets weighted by the firing strengths.

Suppose that we have q rules R^i $(i = 1,2,\cdots,q)$ of the form:

R^i: If input $\mathbf{D}$ **is** in $\mathbf{A}_i$ then output is $z^i = \mathbf{a}^i \times \mathbf{D} + b^i$

If we apply the j^{th} sample to the i^{th} rule R^i, we get:

R_j^i: If input $\mathbf{D}_j$ is in $\mathbf{A}_i$ then output is $z^i = \mathbf{a}_j^{\ i} \times \mathbf{D}_j + b^i$

Where $i=1,2,\ldots,q$ and $j=1,2,\ldots,r$.

According to (4-6), the output y_j is given by:

$$y_j = \frac{\sum_{i=1}^{q} \omega_j^i y_j^i}{\sum_{i=1}^{q} \omega_j^i}$$

$$= \sum_{i=1}^{q} \beta_{ij} y_j^i \tag{4-8}$$

Where :

$$\beta_{ij} = \frac{\omega_j^i}{\sum_{i=1}^{q} \omega_j^i} \tag{4-9}$$

The output y_j can be expressed by:

$$y_j = \begin{bmatrix} \beta_{1j} & \beta_{2j} & \cdots & \beta_{qj} \end{bmatrix} \begin{bmatrix} y_j^1 \\ y_j^2 \\ \vdots \\ y_j^r \end{bmatrix}$$

$$= \begin{bmatrix} \beta_{1j} & \beta_{2j} & \cdots & \beta_{qj} \end{bmatrix} \begin{bmatrix} b^1 & a_1^1 & \cdots & a_n^1 \\ b^2 & a_1^2 & \cdots & a_n^2 \\ \vdots & \vdots & \vdots & \vdots \\ b^q & a_1^q & \cdots & a_n^q \end{bmatrix} \begin{bmatrix} 1 \\ x_{1j} \\ \vdots \\ x_{nj} \end{bmatrix} \tag{4-10}$$

So, the vector of output **Y** can be expressed by:

$$\mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_r \end{bmatrix} = \begin{bmatrix} \beta_{11} & \beta_{21} & \cdots & \beta_{q1} \\ \beta_{12} & \beta_{22} & \cdots & \beta_{q2} \\ \vdots & \vdots & \vdots & \vdots \\ \beta_{1r} & \beta_{2r} & \cdots & \beta_{qr} \end{bmatrix} \begin{bmatrix} b^1 & a_1^1 & \cdots & a_n^1 \\ b^2 & a_1^2 & \cdots & a_n^2 \\ \vdots & \vdots & \vdots & \vdots \\ b^q & a_1^q & \cdots & a_n^q \end{bmatrix} \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_{11} & x_{12} & \cdots & x_{1r} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nr} \end{bmatrix} \tag{4-11}$$

We can rewrite in the form :

$$\mathbf{Y} = \mathbf{X}\,\mathbf{P} \tag{4-12}$$

Where:

$$\mathbf{Y} = \begin{bmatrix} y_1 \\ \vdots \\ y_r \end{bmatrix} \tag{4-13}$$

$$\mathbf{X} = \begin{bmatrix} \beta_{11} & \cdots & \beta_{q1}, & x_{11}\beta_{11} & \cdots & x_{11}\beta_{q1} & \cdots & x_{n1}\beta_{11} & \cdots & x_{n1}\beta_{q1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \beta_{1r} & \cdots & \beta_{qr}, & x_{1r}\beta_{1r} & \cdots & x_{1r}\beta_{qr} & \cdots & x_{nr}\beta_{1r} & \cdots & x_{nr}\beta_{qr} \end{bmatrix} \tag{4-14}$$

$$\mathbf{P} = \begin{bmatrix} b^1 \\ \vdots \\ b^q \\ \cdots \\ a_1^1 \\ \vdots \\ a_n^1 \\ \vdots \\ a_1^q \\ \vdots \\ a_n^q \end{bmatrix} \tag{4-15}$$

X is a matrix with $r \times q(n+1)$ dimensions,

Y is a matrix with r dimensions,

P is a matrix with $q(n+1)$ dimensions.

Finally, matrix **P** is expressed by:

$$\mathbf{P} = [\mathbf{X}^T\mathbf{X}]^{-1}\mathbf{X}^T\mathbf{Y} \tag{4-16}$$

Obviously, in this method, the number of fuzzy rules is equal to the number of subsets (number of membership functions) and input **D** is processed by every membership function during the inference phase, that is to say input **D** is the member of all subsets, there is no step for choosing subsets. So, there is no boundary error. The points A and B in figure 4.1 will give similar output

if using the proposed method and this result is more reasonable than [33]. In this method, the characteristics of input data will be represented more accurately by those membership functions. The output accuracy will be improved. In other word, for a certain requirement of accuracy, this method can decrease the number of membership functions.

The following section presents the design process. Appendices A and B provide the detail flow chart and pseudo-code.

4.2 Input space clustering

A typical input space may contain a large number of inputs. But, it is not necessary to deal with each input. They can be grouped according to their common characteristics and then be represented by a point. This process is called clustering and this chosen point is called a cluster. Clusters can be multi-dimensional points, and competitive learning is often used to perform the clustering. The following algorithm is used for clustering.

Step 1. Fix the cluster number "*cluster_number*", initialize these clusters *cluster*(*i*) ($i = 1, 2, \cdots, cluster\_number$).

Step 2. By using a traditional learning algorithm to decide a winner, dead units (clusters which are never updated) may appear. To solve this problem, the following algorithm is used [33].

$$p_{winner} \| x(j) - cluster(winner\_ID) \| = \min_i p_i (time\_win(i) \times \| x(j) - cluste(i) \|) \quad (4\text{-}17)$$

$$\text{where } p_i = \frac{n_i}{\sum_{k=1}^{m} n_k} \quad (4\text{-}18)$$

Where the *cluster* (*winner_ID*) (*winner_ID*=1,2,..., *cluster_number*) is the nearest cluster of $x(j)$ and $x(j)$ is the j^{th} input of training sample, p_i is roughly

equal to the probability of input data near the cluster (i), n_i is the number of cluster(i) being chosen as a winner and m is the number of cluster [33]. The variable probability p is used to avoid dead units. So every cluster has a chance to competition.

Step 3. Update the *cluster*(*win_ID*) according to $x(t)$ by:

$$cluster(win\_ID) = cluster(win\_ID) + step\_rate \times (x(j) - cluster(win\_ID)) \quad (4\text{-}19)$$

Where $x(j)$ is the input vector ($j = 1, 2, \cdots, number\_input$). The variable *step_rate* is used to avoid oscillation during the training time.

Step 4. Update *step_rate* (2-20)

$$\alpha(t) = \alpha_i \left(\frac{\alpha_f}{\alpha_i}\right)^{t/t_{max}} \quad (4\text{-}20)$$

In general, it is difficult to know the exact value of each $t_{\max}$, but it can be roughly set as the maximum number of iterations.

4.3 Membership function forming

At the end of the preceding section, the input space clustering is done. That is to say all the centers of the membership functions are fixed. However, just having centers is not enough to compose membership functions. It is necessary to find another parameter such as the width to describe each membership function. If the input space is uniformly distributed, then width can be inferred by equation (3-6). This method is simple and rapid. However if the input space is not uniformly distributed, then we need to use the independent width method (3-7) or (3-8) as following.

Step 1. Find the maximal distance between arbitrary two centers by

$$distance = \min(\| cluster(i) - cluster(b) \|) \tag{4-21}$$

Step 2. Calculate out widths by:

$$width = \frac{distance}{2} \tag{4-22}$$

4.4 Identification parameters matrix derivation for T-S fuzzy rules

In this proposed neural fuzzy system, the purpose of training process is to get an identification matrix **P.** This matrix contains all the identification parameters of consequent part of T-S fuzzy rules. It can be gained by the matrix operation (4-12) (4-13) (4-14) (4-15) (4-16). Matrix **P** is the core of the proposed controller. To get a desired output, a simple multiplication between fuzzy input and matrix **P** is performed.

Step 1. Fuzzification by Gaussian function or the triangular function (3-4) or (3-5)

$$dis = x(i) - cluster(j) \tag{4-23}$$

$$u(j,i) = e^{\frac{-dis}{2 \times width^2}} \tag{4-24}$$

Step 2. Calculate β with equation (4-9)

Step 3. Use β to construct input matrix **X** (4-14)

Step 4. Calculate matrix **P** with equation (4-16)

Step 5. Calculate the final output with equation (4-12). If the final error is more than the expected error, then increase the number of clusters and retrains the controller.

dis is the distance between given input and a cluster. $x(i)$ is the input vector ($i = 1, 2, \cdots, m$). "m" is the total number of inputs.

4.5 Conclusion

This method has three main parts. The first part is a competitive learning phase used to cluster the input space. Exponentially decaying learning rate is used in this phase, which can avoid oscillations during the training time. The second part forms membership functions. Independent widths are used to make membership functions more suitable for the input space. The third part is based on fuzzy inference and matrix operations to draw the identification parameters matrix **P.** This method has been validated on three different benchmarking problems and results are presented in the next chapter.

5. RESULTS AND COMPARATIVE ANALYIS

The validity of the proposed procedure has been validated by means of extensive computer simulations on three benchmark problems commonly used in the literature:

- Nonlinear function approximation.
- Mackey-Glass time series prediction.
- Nonlinear system control.

5.1 Nonlinear function approximation

Let us suppose a nonlinear system with a single output variable [34] [35] whose model is given by:

$$y(t+1) = \frac{\{y(t)y(t-1)y(t-2)u(t-1)[y(t-2)-1]+u(t)\}}{[1+y^2(t-1)+y^2(t-2)]} \qquad (5\text{-}1)$$

In this nonlinear function, the output value depends on its previous values y(t), y(t-1), y(t-2) and input values $u(t)$, $u(t-1)$. So, to approximate this function, a 5 dimension input vector **x** is defined where: $\mathbf{x}(t) = [u(t), u(t-1), y(t), y(t-1), y(t-2)]$ and an output **y** as a single dimension vector $[y(t+1)]$. This classic nonlinear function is often used to test the performances of fuzzy systems to approximate nonlinear functions.

Suppose $u(t) = 0$ and $y(t) = 0$ for $t \le 0$. The learning procedure randomly generates input signal $u(t)$ uniformly distributed in the range [-1, 1]. Using equation (5-1), the corresponding output $y(t+1)$ is computed, thus composing the training samples $\{x(t), y(t+1)\}$.

Then the training samples train the proposed neural fuzzy system using the following parameters:

- Initial cluster number = 1;
- Initial update rate = 0.1;

- Final update rate = 0.001. An exponential decay update strategy is used as given by equation (2-20);
- Independent membership function widths (equation 3-7) are used;
- If the final output error is more than 0.0001, increase cluster number by 1 and retrain the system.

Once the training is completed, the fuzzy inference system is tested with a new input given by:

$$u(t)=\sin\frac{2\pi t}{250} \text{ for } t\le 500 \text{ and } u(t)=0.8\sin\frac{2\pi t}{250}+0.2\sin\frac{2\pi t}{25} \text{ for } t>500. \quad (5\text{-}2)$$

The following figures show the resulting performances. Real output and expected output curves are shown in figure 5.1. As observed, these two curves almost completely overlap. The computed difference between the two outputs, called approximation error is illustrated in figure 5.2.

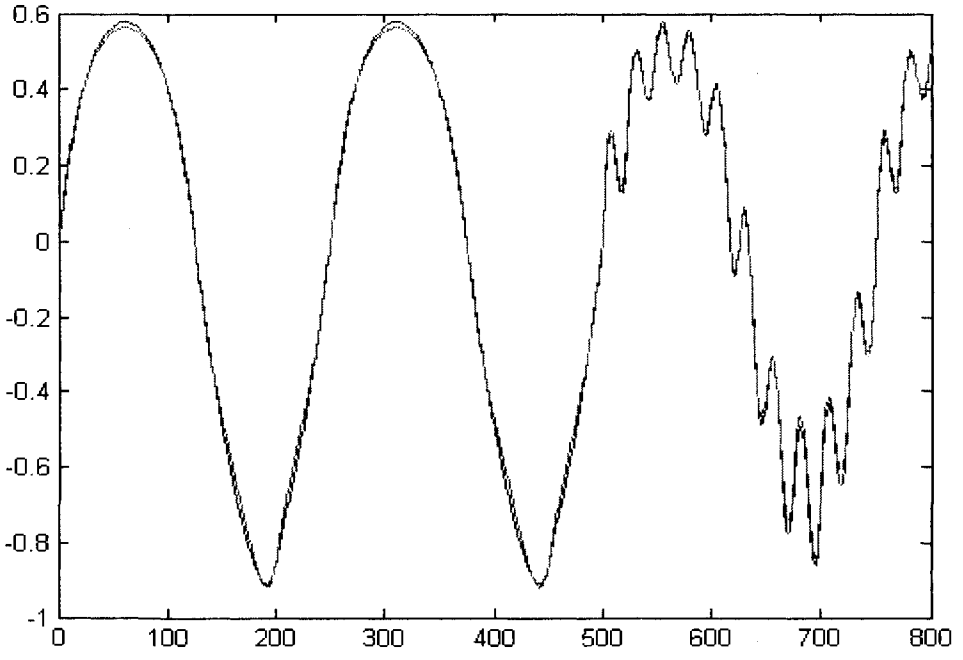


Fig 5.1 Real output and expected output curves

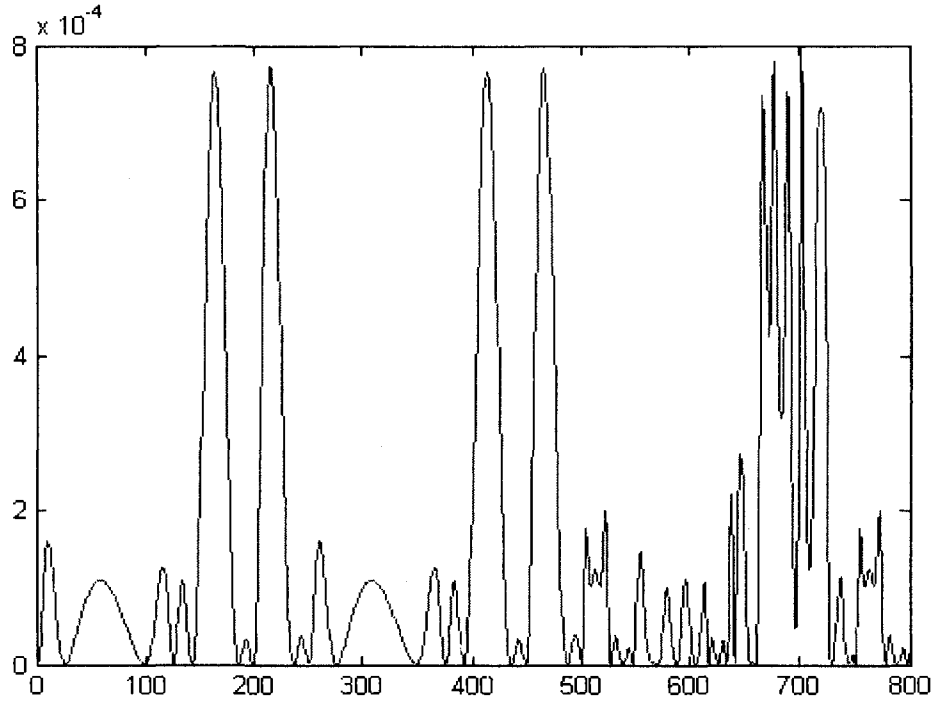


Figure5.2: Approximation error.

Several authors have reported on the same experiment [33] [36]. To compare results with them, the same performance index as in [58] is used. Let's defined:

$$PI = \frac{1}{N}\sum_{t=1}^{N}\left[y(t) - y^*(t)\right]^2 \tag{5-3}$$

In [33], the author used 50 clusters. The best case reported gives PI = 0.0012, the worst case PI = 0.0052 and the average case PI = 0.0029. In [36], the performance index is only 0.0052 when the author tests the generalization and he reported a PI = 0.00028 only when the training sets are chosen to be the same as the testing sets in which the generalization is ignored.

With the method described in this work, with 50 clusters used, the best performance index obtained is PI = 0.00008 and the worst case is PI = 0.0015. The average case is PI = 0.000308. This experiment shows that the proposed method is more accurate than those found in the literature and has good generalization properties.

5.2 Mackey-Glass time series prediction

The prediction of time-series is an important practical problem with applications found in economics, business planning, inventory and production control, weather forecasting, signal processing, and control. In this experiment, the proposed method is used to predict a Mackey-Glass chaotic time series [37]. This time series is created with the use of the Mackey-Glass time-delay differential equation defined here:

$$\frac{dx(t)}{dt} = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1x(t) \tag{5-4}$$

This time series, originally developed to model white blood cell production, is commonly used to test the performance of neural networks. This extremely chaotic series makes it an ideal representation of nonlinear oscillations found in many physiological processes. It has no obvious time period, is neither convergent nor divergent and its value at any time may depend on its entire previous history.

When $x(0)=1.2$, $\tau=17$, this series is a non-periodic, non-convergent time series and is very sensitive to initial condition ($x(t)=0$ for $t<0$). So in this experiment, $x(0)=1.2$ and $\tau=17$ has been used as initial conditions. The word *prediction* means that at time t the function value is known and the object is to predict the value at future time $t+p$. The standard method for this type of prediction is to create a mapping from D sample data points, sampled every Δ units in time, $(x(t-(D-1)\Delta,\ldots,x(t-\Delta),x(t))$, to a predicted future value $x(t+p)$. To compare with other articles [33] [38] [39] [40] [41], the conventional settings $D=4$ and $\Delta=p=6$ are used in this experiment.

A four dimensional input vector is defined in the following form:

$$\mathbf{W}(t) = \left[x(t-18),\ x(t-12),\ x(t-6),\ x(t)\right] \tag{5-5}$$

A one dimensional prediction vector is defined in the following form:

$$output(t) = x(t+6) \tag{5-6}$$

A MATLAB demo called "Time-Series prediction" illustrates a Mackey-Glass series. This demo was used to generate training samples and test samples for the experiment. The samples was saved in a file mgdata.dat with ($x(0) = 1.2$, $\tau = 17$). The following MATLAB program is used:

```
load mgdata.dat
t = mgdata(:,1);
x = mgdata(:,2);
plot(t,x);
for t=118:1117,
   Data(t-117,:) = [x(t-18) x(t-12) x(t-6) x(t) x(t+6)];
end
trnData = Data(1:500,:); % first 500 data pairs used to construct training samples.
chkData = Data(501:end,:);  % last 500 data pairs used to construct test samples.
```

In this program, the time range t is 118 to 1117 to be able to compare with results published by other authors. The *trnData* and *chkData* are 5-dimension arrays. The matrix *trnData* holds 500 training pairs, and matrix *chkData* holds 500 test pairs. The first four-dimension of *trnData* and *chkData* are used to construct training and test input vectors and the last dimensions of *trnData* and *chkData* are used to construct the training and test output vector.

The proposed neural fuzzy system was trained with the following parameters:

- Initial cluster number = 1;
- Initial update rate = 0.1;
- Final update rate = 0.001. An exponential decay update strategy is used as given by equation (2-20);
- Independent membership function width (equation 3-7);
- If the final output error is greater than 0.0001, increase cluster number by 1 and retrain the system.

Figure 5.3 gives the prediction error while figure 5.4 shows the computed output and the expected output in the same coordinate system. The difference is so small that it is almost impossible to distinguish the two curves.

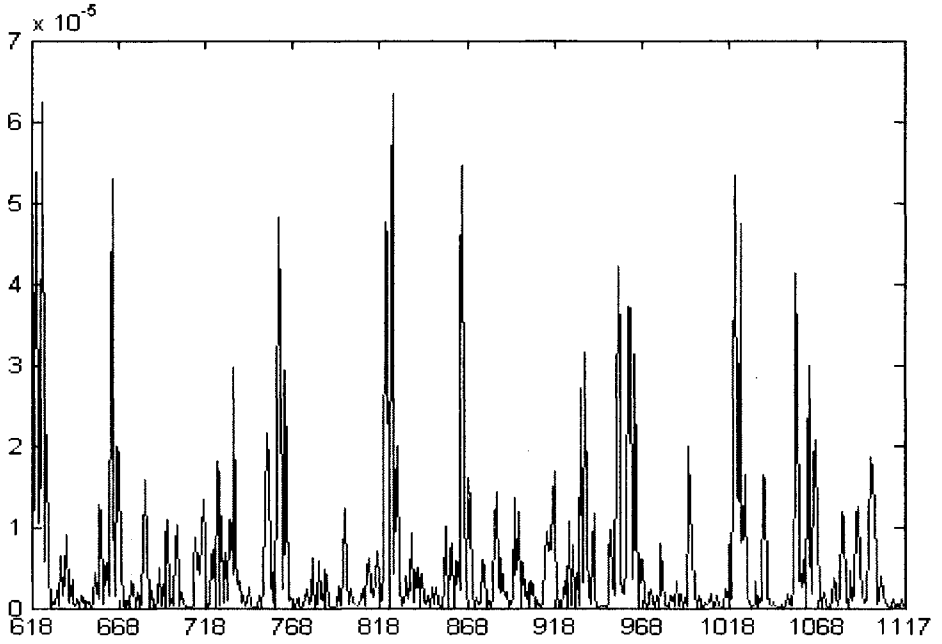


Fig 5.3 Prediction error.

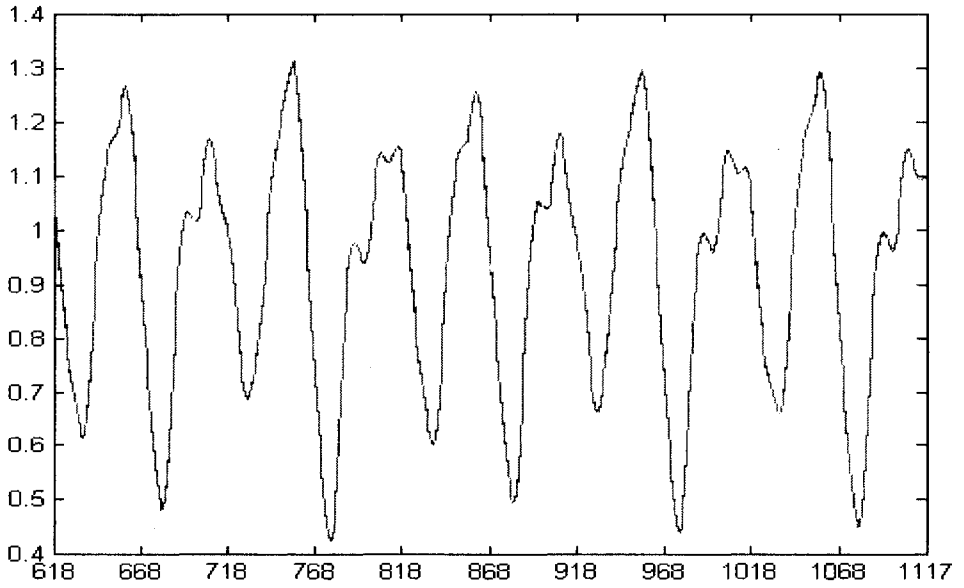


Fig 5.4 Mackey–Glass prediction:

The same experiment was reported in [33] [38] [39] [40] [41]. To compare with experiment *D* of [33], *PI* is used, which is defined as (5-3). Table I lists the simulating results represented by *PI*.

TABLE I

Test Results between [33] and HDNFI

MODEL	CLUSTER NUMBER	TRAINING DATA	WORST CASE (*PI*)	AVERAGE CASE (*PI*)	BEST CASE (*PI*)
[33]	40	2500	0.00246	0.00172	0.00142
HDNFI	**15**	**500**	**0.0019**	**0.000056**	**0.00000875**

In average case $PI_{(HDNFI)}$=0.000056 and in best case $PI_{(HDNFI)}$=0.00000875, which is a perfect result. And this result show us the performance of HDNFI is much better than the [33]'s. To show more performances of HDNFI, a comparison was also conducted with [33] [38] [39] [40] [41]. For these cases a non-dimensional error index (NDEI) is defined as the root mean square error (RMSE) divided by the standard deviation of the target series. Table II lists the simulating results represented by NDEI (main data are from [41]).

TABLE II

Test Results on Mackey–Glass Data

MODEL	TEST NDEI
WKNN	0.06
CC-NN model	0.06
6^{th}-order Polynomial	0.04
MLP(BP)	0.02
HYFIS [38]	0.01
ANFIS [39]	0.007
DENFIS [40]	0.006
NFI [41]	0.004
HDNFI	**0.0033**

The table II shows that the method NFI which proposed by [41] possesses better performance than other method except HDNFI. Because all the fuzzy membership functions in [41] are one-dimension, but in HDNFI, all the fuzzy membership functions are high-dimensions, which can effectively decrease the output error. And in [41], the cluster radiuses are used as the widths of fuzzy membership functions, but in HDNFI, the form (3-7) and (3-8) for calculating the widths are proposed, which is more suitable of many kinds of input spaces.

Obviously, above reasons make the proposed algorithm HDNFI perform well than the other models. With results clearly superior to those previously found in the literature.

5.3 Nonlinear system control: the ball and beam example

The ball and beam system is a frequently given example of nonlinear dynamic system. It is a popular and important laboratory model to teach control systems engineering because it is open loop unstable.

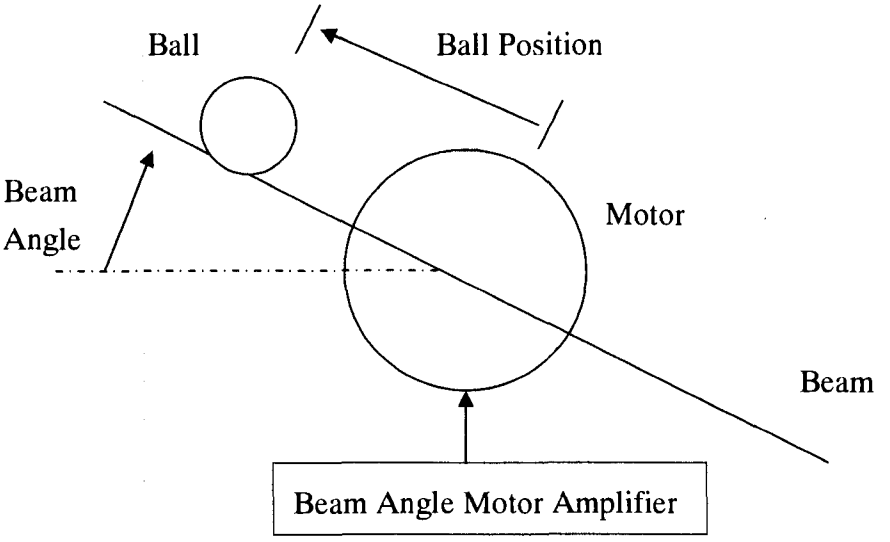


Fig 5.5 Ball and beam system

The system shown in Figure 5.5 is very simple. A beam is mounted on the output shaft of a motor. The beam can be tilted around its centre axis by applying an electrical control signal to a motor amplifier. The position of the ball on the beam is measured by a sensor. The ball moves with an acceleration that is proportional to the tilt of the beam. That is to say the ball position increases without limit for a fixed beam angle. A control system must be used to keep the ball in a desired position on the beam. This system has been used to test the capacity of the proposed controller to control the ball and beam system.

A MATLAB demo, called "Ball and beam" is readily available with a fuzzy logic controller (figure 5.6). The controller is a MISO controller. It uses four state variables as input vector $[A,B,C.D]$: position, angle and their respective derivatives. The output is a one-dimension vector $[E]$: motor voltage.

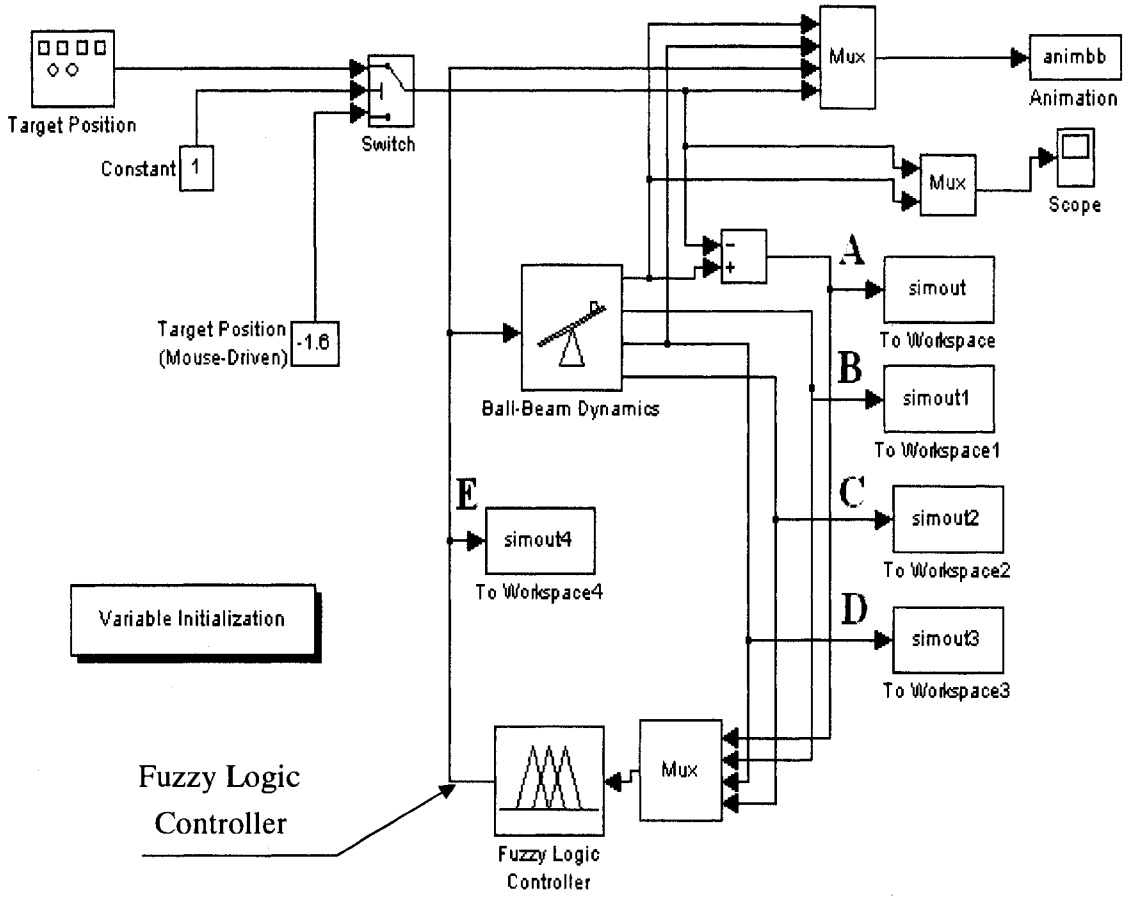


Figure 5.6 Original MATLAB Ball and beam controller

This experiment was modified for the following purposes:

- To acquire the proper "control knowledge" by learning from the original fuzzy logic controller;
- To replace the original controller and stabilize the ball.

Five steps are needed to conduct this experiment from the MATLAB demo :

Step 1 Add 5 *simouts* in the original SIMULINK program at points labeled A, B, C, D, E in figure 5.6.

Simout is a workspace block of SIMULINK. It is used to return output trajectories to the MATLAB workspace. The block writes its output to an

array or structure that has the name specified by the block's variable name parameter. These 5 *simouts* are used to obtain training data. Data from *simout*, *simout1*, *simout2* and *simout3* are input data, so the input of this system is a 4-dimensions vector [*simout*, *simout1*, *simout2*, *simout3*]. And [*simout4*] is a one-dimension output vector.

Step 2 Run the original SIMULINK program for more than 8 seconds then stop the program. Check workspaces. Every *simout* block kept more than six hundred data. The first 500 data of each *simout* is used to construct the training sample in this work.

Step 3 Use the 500 input-output pairs $[simout, simout1, simout2, simout3]$, $[simout4]$ to train the new controller more than ten times.

Step 4 Replace the original fuzzy controller by the new controller.

Step 5 Run the simulation.

The animation shows us that the ball is under control while moving on the beam. Figure 5.8 allows the comparison between the target position and the real position when the proposed controller is used. Figure 5.9 shows the target position and real position when the MATLAB controller is used.

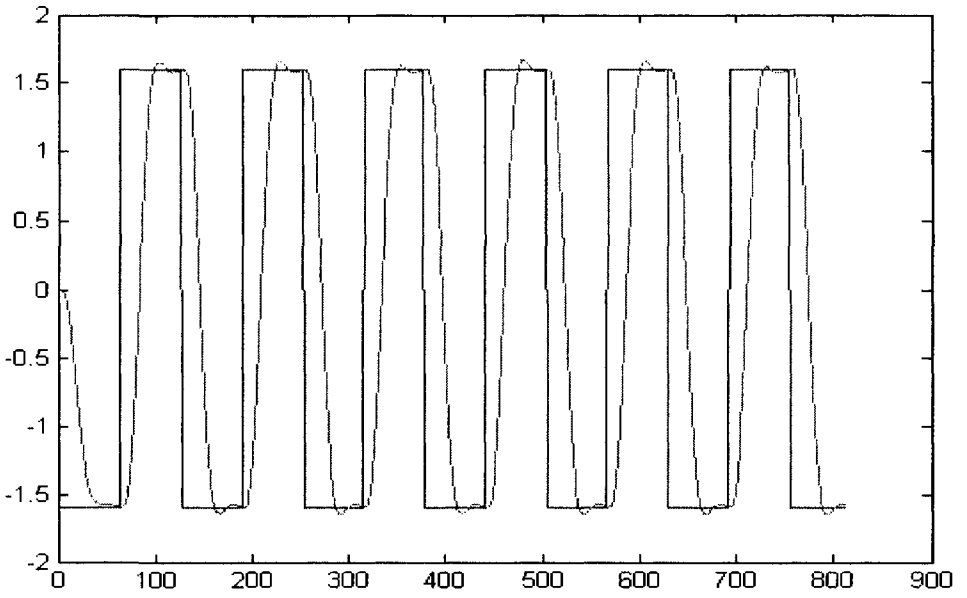


Fig 5.8 Position and tracking with the proposed controller

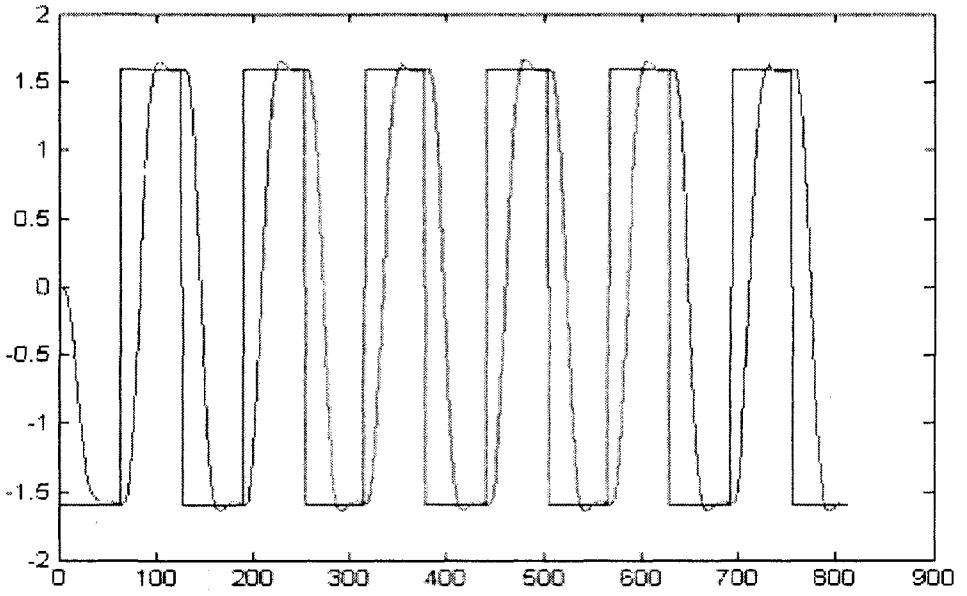


Fig 5.9 Position and tracking with the original controller.

Results were compared with those published in papers [42] and [43] with different initial conditions [position, angle position's derivative, angle's derivative]:

$x(0)$ = [2.4, -0.1, 0.6, 0.1], [1.6, 0.05, -0.6, -0.05], [-2.4, 0.1, -0.6, -0.1], [-1.6, -0.05, 0.6, 0.05], [1, 0, 0, 0], [2, 0, 0, 0] and [3, 0, 0, 0].

Figure 5.10 shows the ball position of the closed-loop ball and beam system using the proposed controller for the following initial conditions ***A*** { $x(0)$ = [2.4, -0.1, 0.6, 0.1], [1.6, 0.05, -0.6, -0.05], [-2.4, 0.1, -0.6, -0.1] and [-1.6, -0.05, 0.6, 0.05] }.

Figure 5.11 shows the ball position of the closed-loop ball and beam system using the SVM-based fuzzy basis function inference system as published in [43] for the following initial conditions ***A*** { $x(0)$ = [2.4, -0.1, 0.6, 0.1], [1.6, 0.05, -0.6, -0.05], [-2.4, 0.1, -0.6, -0.1] and [-1.6, -0.05, 0.6, 0.05] }. This inference system is a hybrid of fuzzy basis function inference system [36] and the support vector machine. It chooses the fuzzy basis function as the kernel function of the support vector machine to fuse those two mechanisms into a new fuzzy inference system. This inference system possesses satisfactory generalization ability and over-fitting prevention capability. This overall fuzzy inference system can be represented as series expansion of fuzzy basis functions, and this also makes the inference system itself to be interpretable.

Figure 5.12 shows the ball position of the closed-loop ball and beam system using the proposed controller for the following initial conditions ***B*** { $x(0)$ = [1, 0, 0, 0], [2, 0, 0, 0] and [3, 0, 0, 0] }.

Figure 5.13 shows the results found in [36] for initial conditions ***B*** {$x(0)$ = [1, 0, 0, 0], [2, 0, 0, 0] and [3, 0, 0, 0]}.

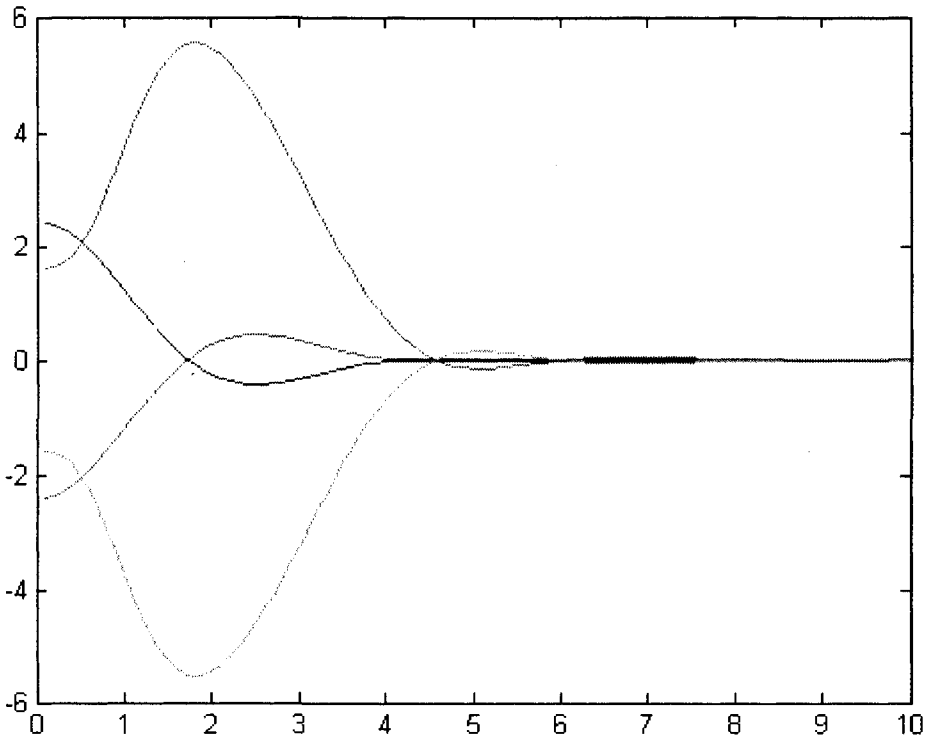


Fig 5.10 Ball position with the proposed controller with initial conditions A

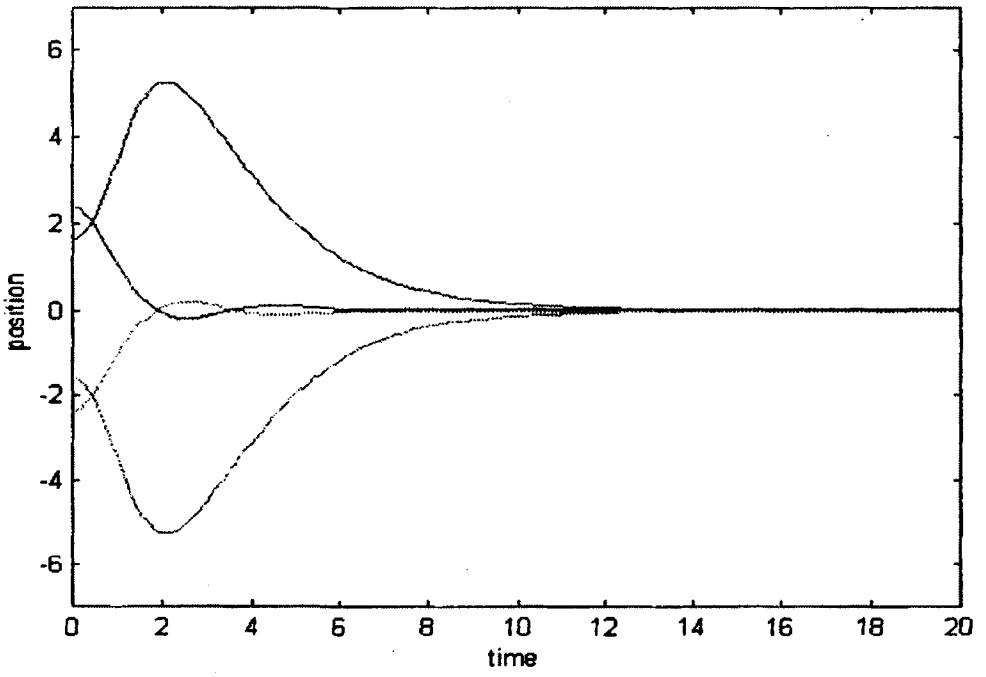


Fig 5.11: Ball position found in [43] with initial conditions A

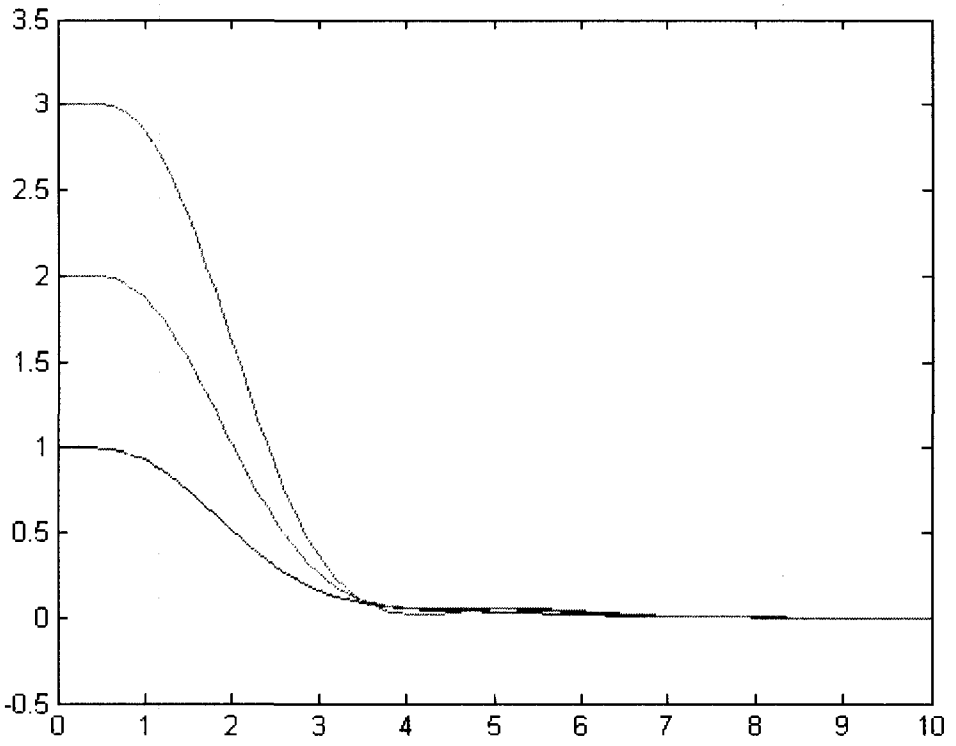


Fig 5.12: Ball position with the proposed controller with initial conditions ***B***

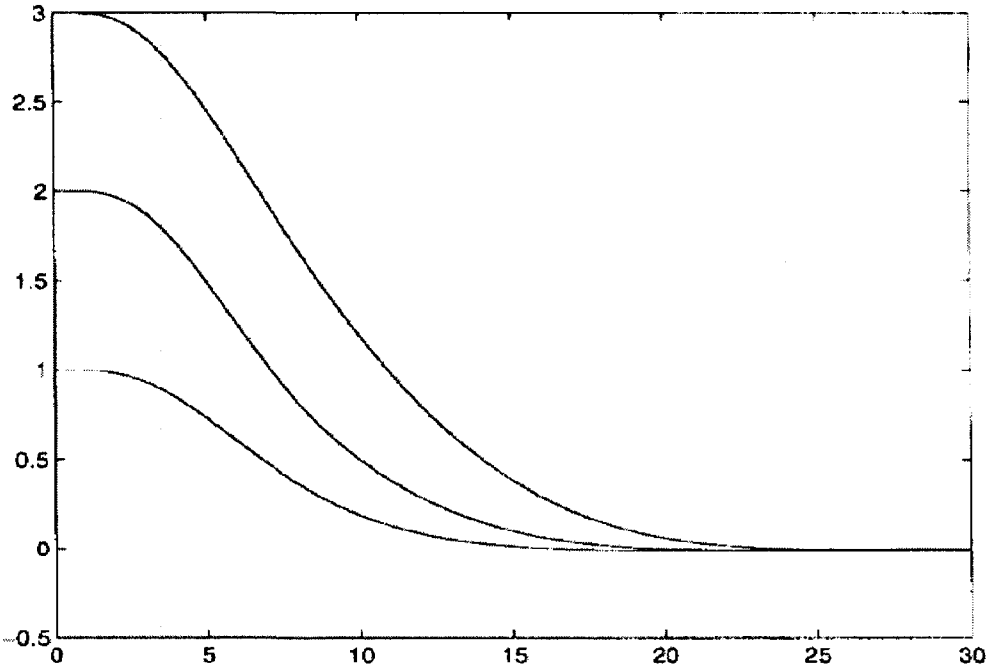


Fig 5.13: Outputs of the closed-loop ball and beam system in [42] with initial conditions ***B***

The figure 5.10 and figure 5.11 shows that HDNIF controller and the controller which proposed by [43] have similar rise time and peak overshoot at initial conditions ***A***. But their settling time are difference, $t_s(\text{HDNFI}) = 7.5$ and $t_s([43]) = 12.5$. The figure 5.12 and figure 5.13 shows that at initial conditions ***B***, the rise time $t_r(\text{HDNFI}) = 1.5$, $t_r([42]) = 8$ and the settling time $t_s(\text{HDNFI}) = 8$ and $t_s([42]) = 24$.

The above experiment results show us that the performance of proposed controller HDFIN is satisfactory and it may provide better generalization capability than other controller in article [42] and [43].

6. CONCLUSION

Fuzzy logic is a powerful tool for control engineering. However serious drawbacks affect the design of fuzzy controllers. One of them is the rule base construction. To overcome these problems, authors propose solutions to construct the rule base automatically by combining artificial neural networks techniques and fuzzy logic in neuro fuzzy architectures.

Several researches have been conducted on the subject and algorithms are available, most of them give good results, but they generally have some limitations. In many cases these algorithms result in a large number of fuzzy rules, proper learning results dependent on the quality of training sets and accuracy is not always acceptable.

To address these limitations, a new clustering neuro fuzzy procedure was developed in this work. The main concept is to use clusters to construct membership function in the input data space. Then based on Tagaki-Sugeno fuzzy inference engine and matrix operations, a matrix **P** is constructed. This matrix includes all fuzzy rules parameters and is called rule base matrix or identification parameters matrix.

During the training phase, the clustering technique and an exponential decay update rate are applied to find the proper clusters for the input data space. These clusters and independent widths used to construct membership functions are well adapted to different types of data space. For the fuzzy inference phase, to avoid boundary errors, a high-dimension T-S fuzzy inference is used. Each input data is processed by every membership function to construct *if-part* of fuzzy rules base. This strategy improves the output accuracy and decreases the number of clusters.

The proposed algorithm has been tested on different applications and results compared with published data on three benchmark problems.

The proposed algorithm is simple to use and experimental results show that the number of clusters required is less than those reported in the literature. Output accuracy is good in many applications.

The proposed algorithm has many advantages but also has some difficulties. Since its structure is based on neural networks and the mathematical theories used to guarantee the performance of an applied neural network are still under development. So we cannot guarantee getting same training result every time even using same training samples. Sometimes, for drawing a perfect result, we need to do more times training. So the further direction for research is to improve the stability of neural networks and optimize the membership functions in high-dimensions space.

7. APPENDIX A

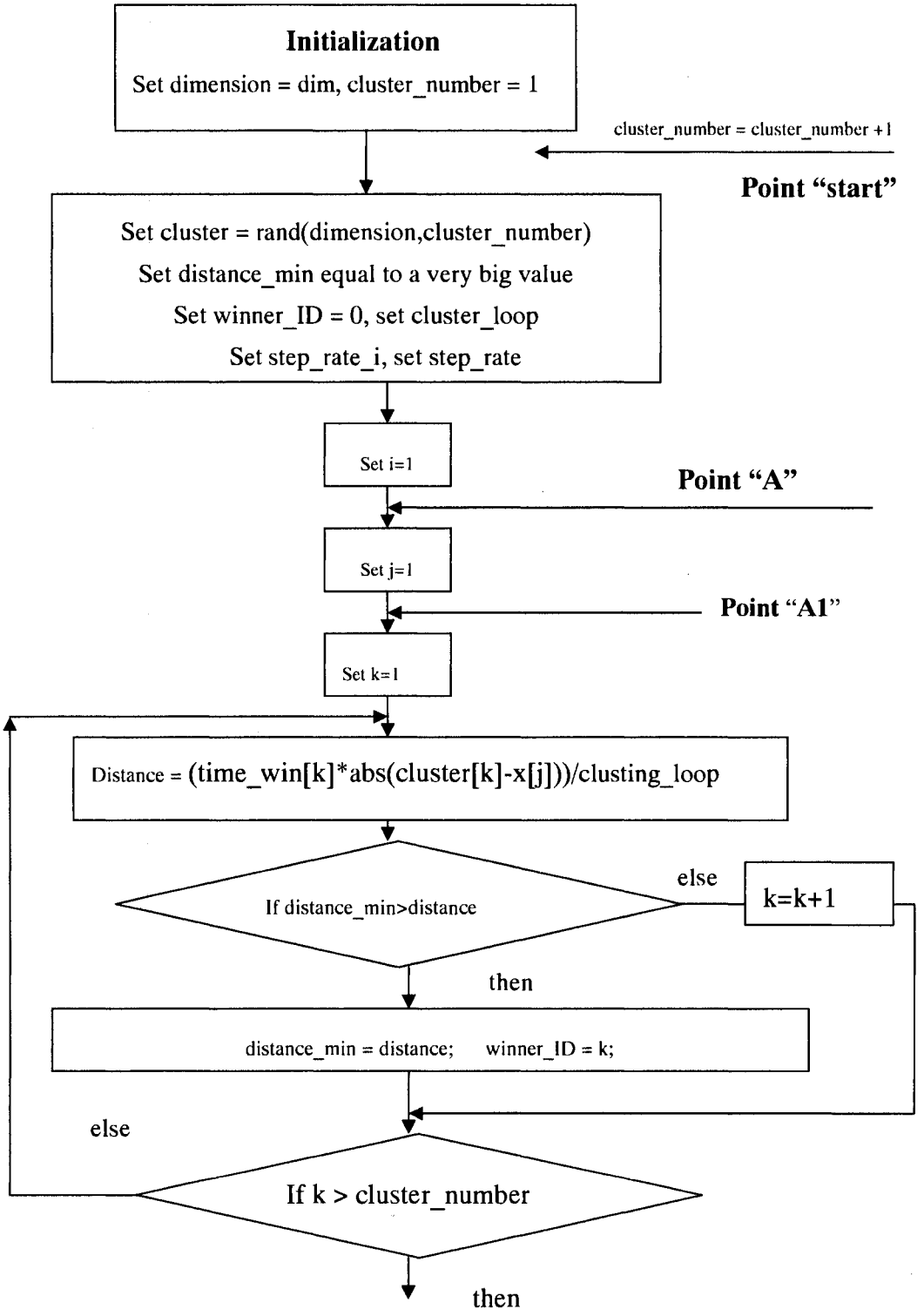
PSEUDOCODE DESCRIPTION

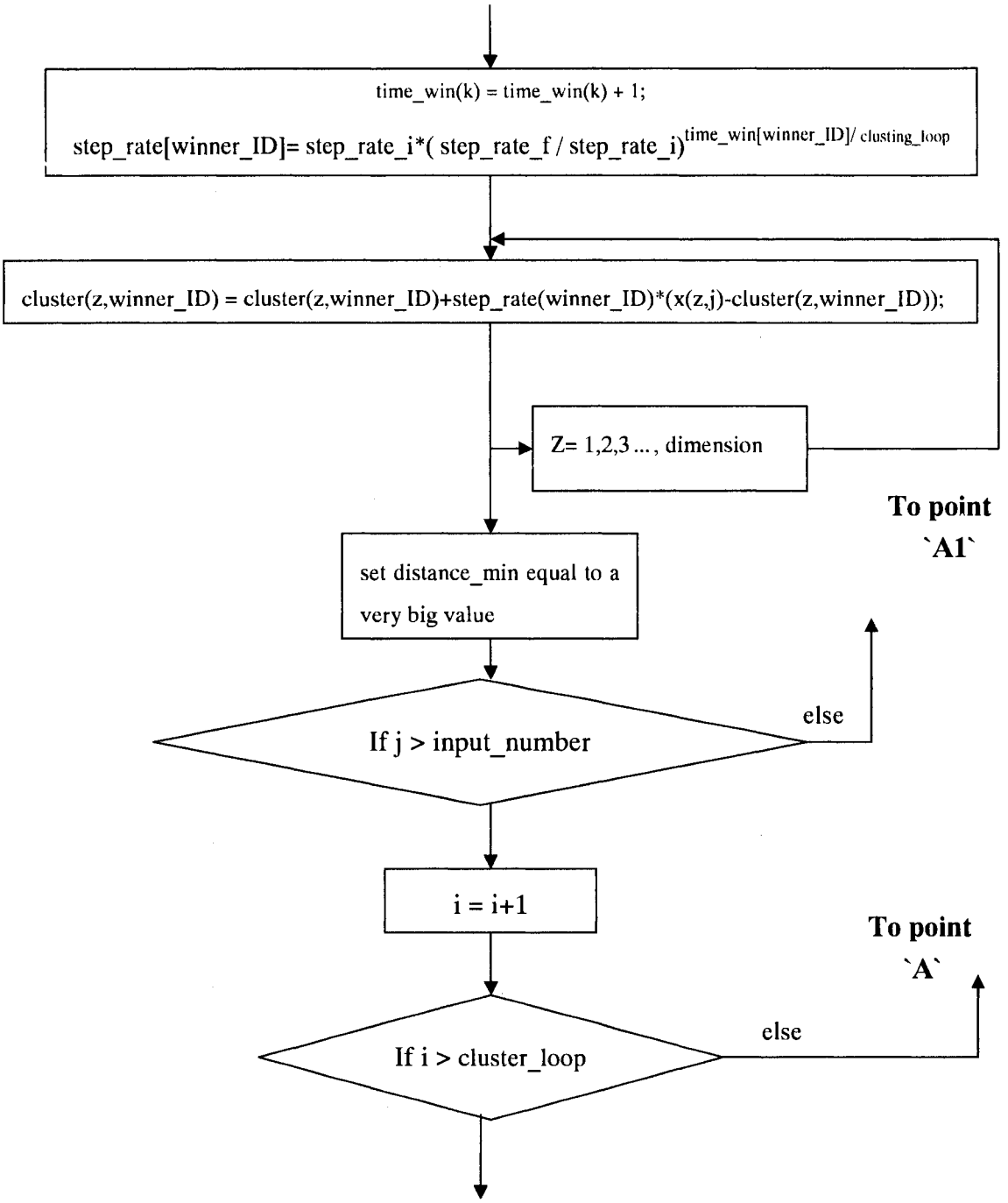
```
set dim   // to indicate the dimension of input
set cluster_number=1 // At the beginning of training phase, the number of clusters is set 1
error=1  // to start the following "while" loop
set input_number // Number of input data in training sample
for i=1 to cluster_number
    time_win[i]=1   // to indicate times the i^th cluster wins
endfor
set step_rate_i // set initial update rate
set step_rate_f // set final update rate
while (error>0.0001)          //     if the final error exceeds this value, the cluster number is
                                increased and system is  trained again.
        cluster_number=cluster_number+1    // at the beginning we assume there is one cluster.
                                              Every time the final error is more than 0.0001,
                                              the cluster_number is increased by 1 and the
                                              system is  trained again
        set cluster[dim][cluster_number]   //   random set the initial value of each cluster
        set distance_min // set this variable equals to a large value
        winner_ID = 0   // to indicate which cluster wins in the current loop

        set clusting_loop    // iteration number for clustering
        for i=1 to cluster_number
        step_rate[i]= step_rate_i   // initialize update rate
        endfor

        //**********To determine the winner and to update the winner**********
        for i= 1 to clusting_loop
          for j= 1to input_number
            for k=1 to cluster_number
                distance = (time_win[k]*abs(cluster[k]-x[j]))/clusting_loop
                if distance_min >distance
                    distance _min = distance
                    winner_ID=k

                else
                endif
            endfor  // end of variable "k"
            time_win(k)= time_win(k)+1
            step_rate[winner_ID]= step_rate_i*( step_rate_f / step_rate_i)^(time_win[winner_ID]/ clusting_loop)
            for h=1 to dim   // update winner
```

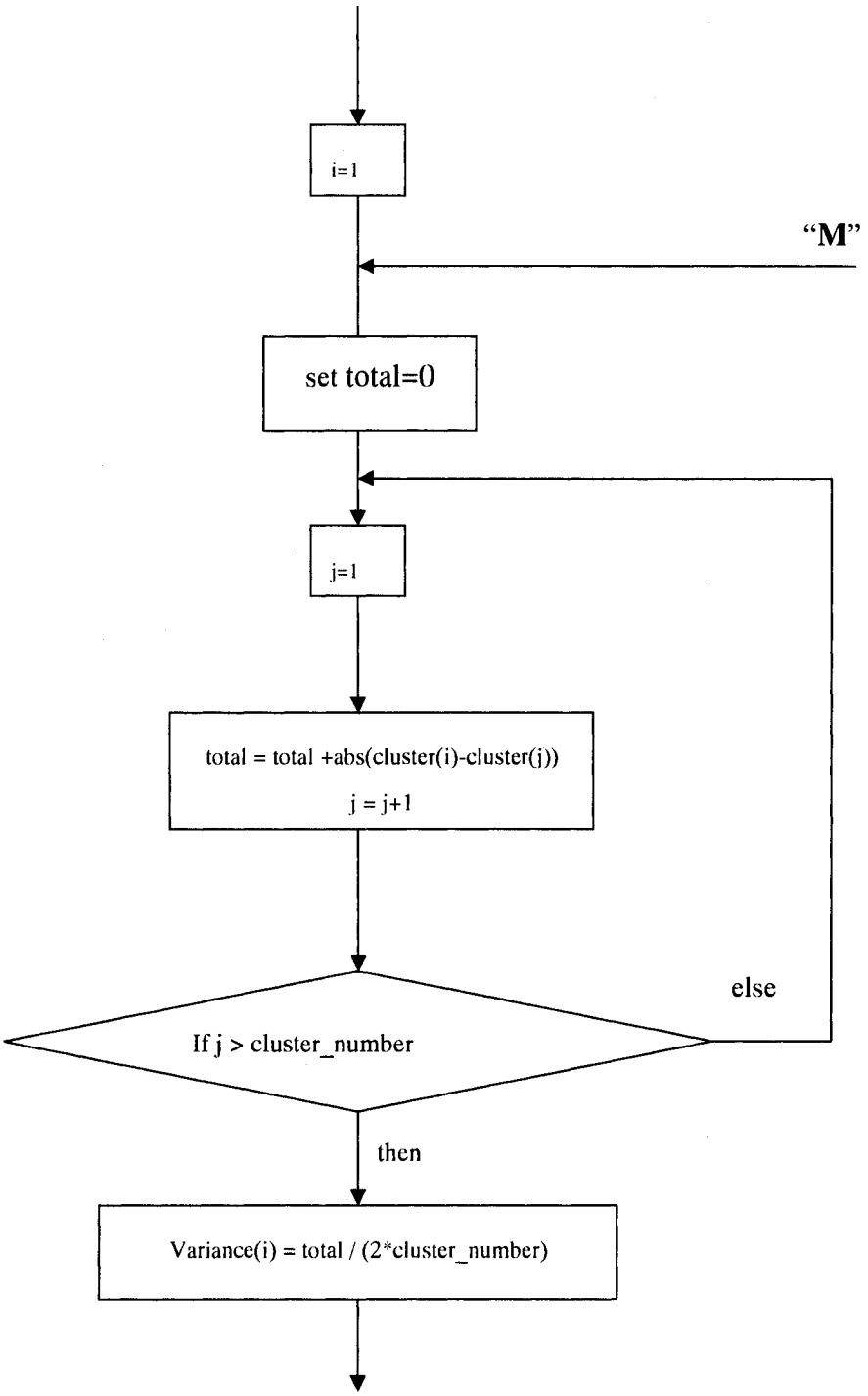
```
                cluster[h][winner_ID]=
                cluster[h][winner_ID]+step_rate[winner_ID]*( cluster[h][ winner_ID]-x[h][j])
            endfor // end of variable "h" for update clusters
            set distance_min equal to a very big value
            winner_ID = 0
          endfor // end of variable "j"
        endfor // end of variable "i"
//*************************************************
// **********calculate out the width of each membership function******

for i= 1 to cluster_number
    total=0
    for j=(i+1) to cluster_number
        total =total + abs(cluster[i]- cluster[j])
    endfor // end of variable "j"
    variance[i] = total / (2* cluster_number) // width of membership function
endfor // end of variable "i"

//******************fuzzy inference ((3-27),(3-28),...,(3-34))***********
for i= 1 to input_number
    for j= 1 to cluster_number
          dis=abs(cluster[j]-x[i])
          u[j][i]=exp(-dis/(2*variance* variance)) // fuzzify input (calculate out the degree of
                                                    fuzziness)
    endfor //      end of variable "j"
endfor //     end of variable "i"

set w[j][i]=u[j][i] // w is the firing strength
total_w=0
for i=1 to input_number
    for j= 1 to cluster_number
        total_w= total_w+w[j][i]
    endfor   //        end of variable "j"

    for k=1 to cluster_number
        beta[k][i] = w[k][i]/total_w
    endfor //        end of variable "k"
    total_w=0
endfor   //        end of variable "i"

for i=1 to input_number
    for j=1 to cluster_number
        x_beta(j,i) = beta(j,i)
    endfor   //        end of variable "j"

    for k= cluster_number+1 to cluster_number*2
```

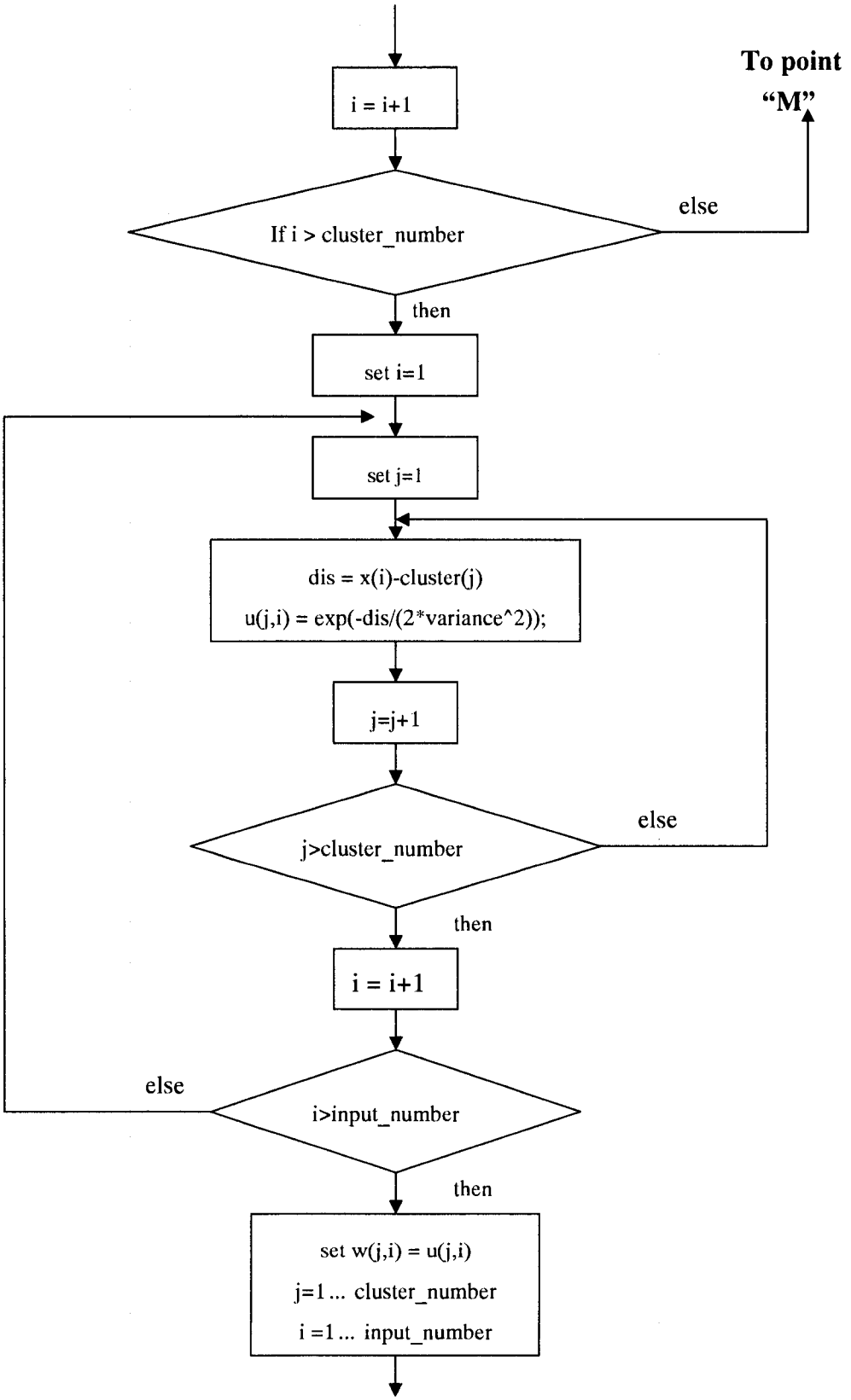
```
        x_beta(k,i) = x(i,1) * beta(k-cluster_number,i)
    endfor   //        end of variable "k"

    for m=cluster_number*2+1 to cluster_number*3
        x_beta(m,i) = x(i,2) * beta(m-cluster_number*2,i);
    endfor  //        end of variable "m"

    for n=cluster_number*3+1 to cluster_number*4
        x_beta(n,i) = x(i,3) * beta(n-cluster_number*3,i);
    endfor  //        end of variable "n"

    for o=cluster_number*4+1 to cluster_number*5
        x_beta(o,i) = x(i,4) * beta(o-cluster_number*4,i);
    end   //        end of variable "o"
endfor  //        end of variable "i"

parameter =inv(x_beta*(x_beta'))*(x_beta)*(y_expect)
y_real=x_beta'*parameter

PI=0  // performance index
for i= 1 to input_number
        PI = PI +( y_real[i] - y_expect[i])* (y_real[i] - y_expect[i])
endfor   //        end of variable "i"
PI = PI / input_number
endwhile
```

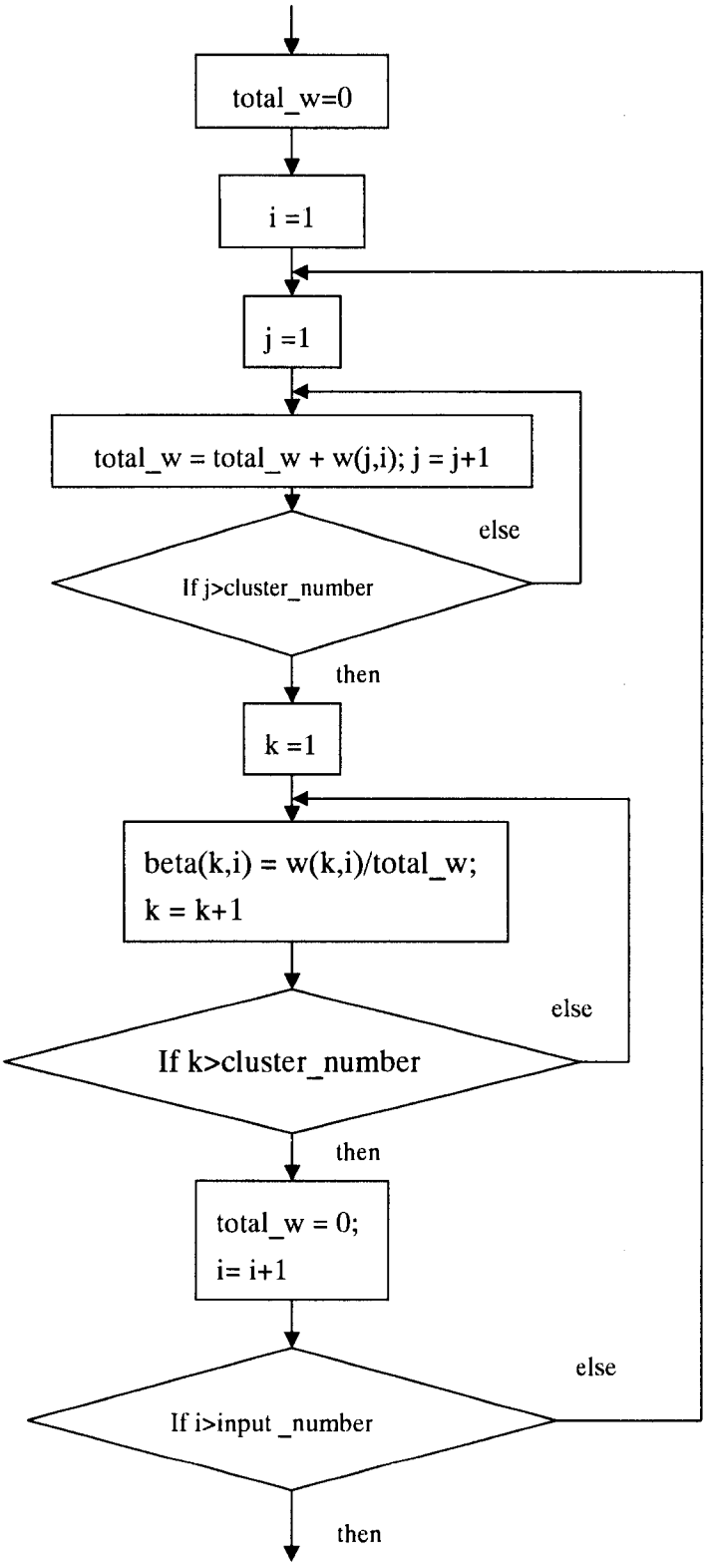
8. Appendix B

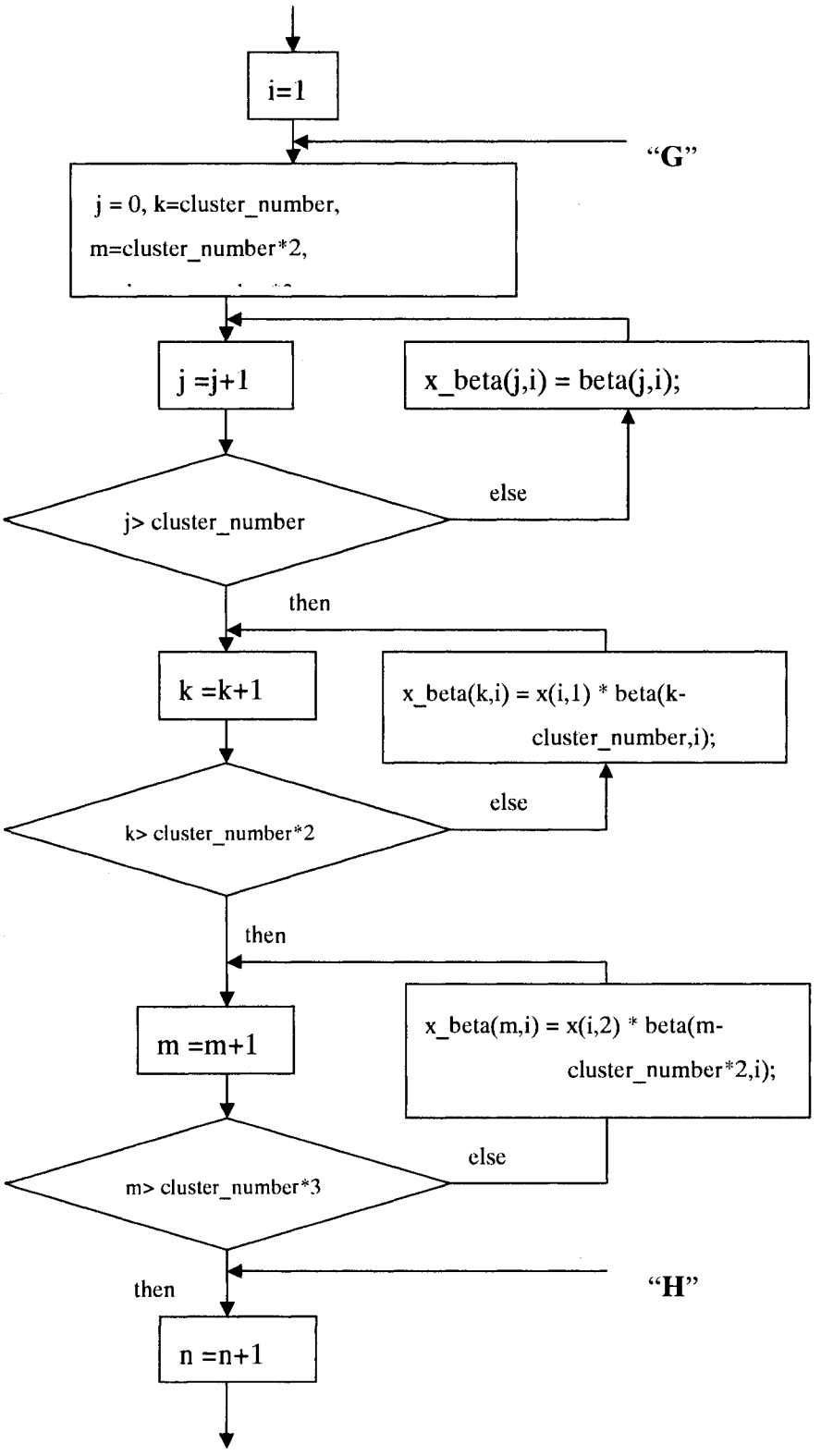

Initialization
Set dimension = dim, cluster_number = 1
cluster_number = cluster_number +1
Point "start"
Set cluster = rand(dimension,cluster_number)
Set distance_min equal to a very big value
Set winner_ID = 0, set cluster_loop
Set step_rate_i, set step_rate
Set i=1
Point "A"
Set j=1
Point "A1"
Set k=1
Distance = (time_win[k]*abs(cluster[k]-x[j]))/clusting_loop
If distance_min>distance
else
k=k+1
then
distance_min = distance; winner_ID = k;
else
If k > cluster_number
then

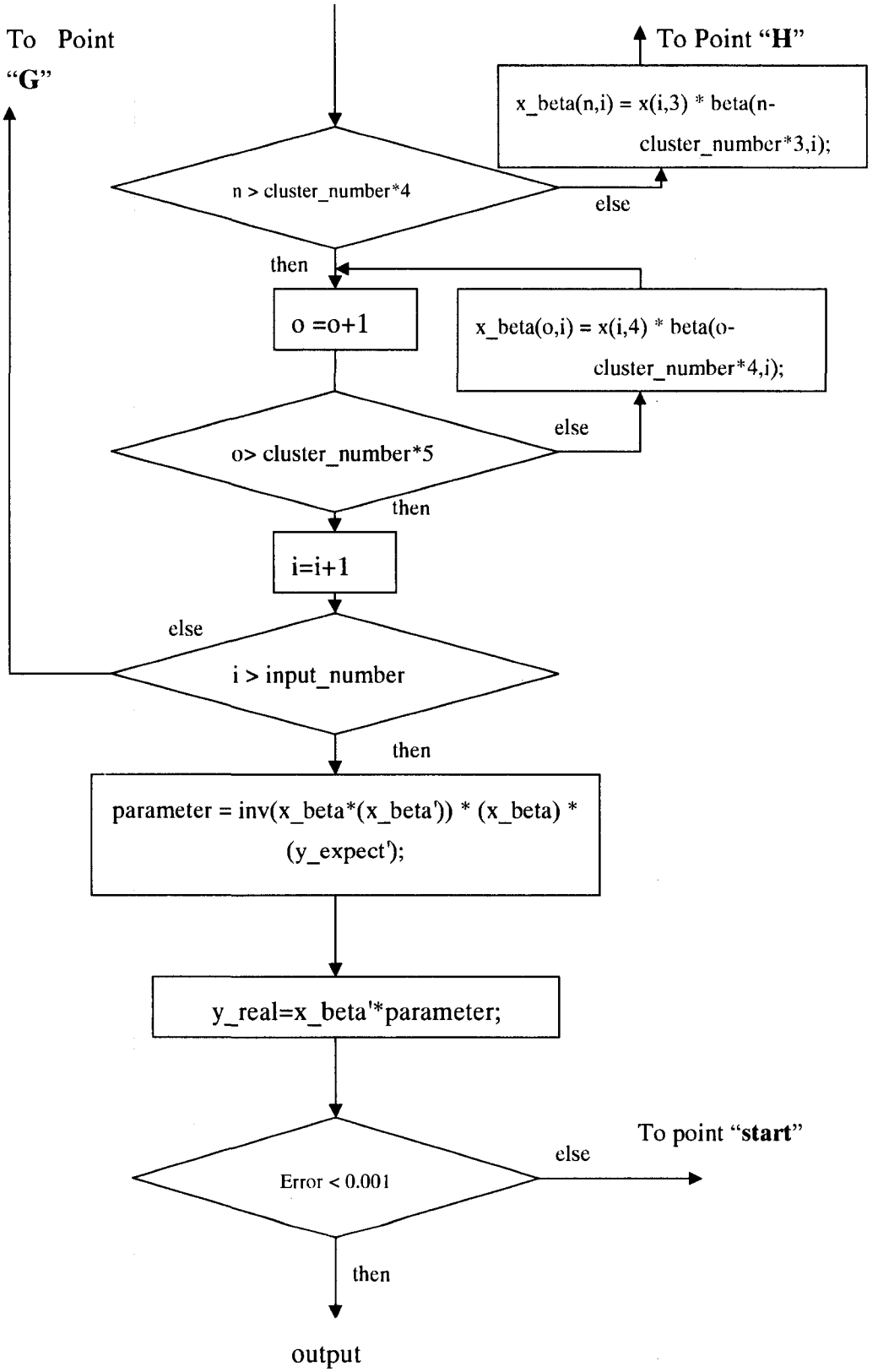

time_win(k) = time_win(k) + 1;
step_rate[winner_ID]= step_rate_i*(step_rate_f / step_rate_i)^time_win[winner_ID]/ clusting_loop
cluster(z,winner_ID) = cluster(z,winner_ID)+step_rate(winner_ID)*(x(z,j)-cluster(z,winner_ID));
Z= 1,2,3 ... , dimension
To point `A1`
set distance_min equal to a very big value
If j > input_number
else
i = i+1
To point `A`
If i > cluster_loop
else


i=1
"M"
set total=0
j=1
total = total +abs(cluster(i)-cluster(j))
j = j+1
else
If j > cluster_number
then
Variance(i) = total / (2*cluster_number)


To point
"M"
i = i+1
If i > cluster_number
else
then
set i=1
set j=1
dis = x(i)-cluster(j)
u(j,i) = exp(-dis/(2*variance^2));
j=j+1
j>cluster_number
else
then
i = i+1
i>input_number
else
then
set w(j,i) = u(j,i)
j=1 ... cluster_number
i =1 ... input_number


total_w=0
i =1
j =1
total_w = total_w + w(j,i); j = j+1
else
If j>cluster_number
then
k =1
beta(k,i) = w(k,i)/total_w;
k = k+1
else
If k>cluster_number
then
total_w = 0;
i= i+1
else
If i>input _number
then


i=1
"G"
j = 0, k=cluster_number,
m=cluster_number*2,
j =j+1
x_beta(j,i) = beta(j,i);
else
j> cluster_number
then
k =k+1
x_beta(k,i) = x(i,1) * beta(k-
cluster_number,i);
else
k> cluster_number*2
then
m =m+1
x_beta(m,i) = x(i,2) * beta(m-
cluster_number*2,i);
else
m> cluster_number*3
then
"H"
n =n+1


To Point "G"
To Point "H"
x_beta(n,i) = x(i,3) * beta(n-cluster_number*3,i);
n > cluster_number*4
else
then
o =o+1
x_beta(o,i) = x(i,4) * beta(o-cluster_number*4,i);
else
o> cluster_number*5
then
i=i+1
else
i > input_number
then
parameter = inv(x_beta*(x_beta')) * (x_beta) * (y_expect');
y_real=x_beta'*parameter;
To point "start"
else
Error < 0.001
then
output

9. BIBLIOGRAPHY

1. M. Funabashi, A. Maeda, Y. Morooka, and K. Mori, "Fuzzy and neural hybrid expert systems: Synergetic AI," *IEEE Expert*, vol. 10, pp. 32–40, Aug. 1995.
2. Kim C. Ng, *Member, IEEE*, and Mohan M. Trivedi, *Senior Member, IEEE*, "A Neuro-Fuzzy Controller for Mobile Robot Navigation and Multirobot Convoying", *IEEE T Transactions on Systems, Man and Cybernetics* part B:, 1998 vol. 28, no 6.
3. Eichfeld, H., Künemund, T. & Menke ,M. (1996). "A 12b General-Purpose Fuzzy Logic Controller Chip", *IEEE Transactions on Fuzzy Systems*. 4 (4); 460-475.
4. Guilherme N. DeSouza and Avinash C. Kak, "Vision for Mobile Robot Navigation: A Survey", *IEEE Transactions on Pattern analysis and machine intelligence*, vol. 24, no. 2, Feb. 2002
5. Umut Uludag and Anil K. Jain, "Fuzzy Fingerprint Vault", Springer, 2003, Computer Science and Engineering, Michigan State University, East Lansing, MI, 48824, USA
6. T. Takagi and M. Sugeno, "Fuzzy Identification Of Systems and Its Applications to Modeling and Control," *IEEE Trans. Syst., Man, Cybern.*, vol.15, pp. 116–132, 1985.
7. J.S.R. Jang, C.T. Sun, "Neurofuzzy modeling and control", *IEEE Trans. Fuzzy Sys*, vol. 3, no. 3, March 1995, pp. 378-406.
8. L.X. Wang and J. Mendel, "Generating fuzzy rules by learning from Examples", *IEEE Trans. Syst., Man, and Cyb.*, vol. 22, 1992, pp. 1414- 1427.
9. D.A. Linkens, M-Y. Chen, "Input selection and partition validation for fuzzy modeling using neural network", *Fuzzy Sets and Systems*, vol. 107, 1999, pp. 299-308.
10. M. Figueiredo, F. Gomide, "Design of fuzzy systems using neurofuzzy networks," *IEEE Trans. on NeuralNetworks*, vol. 10, no. 4, July 1999, pp.815-827.
11. C. Lin, Y. Lu, "A neural fuzzy system with supervised learning", *IEEE Trans.*

Syst., Man, and Cyb. - Part B, vol. 26, 1996, pp. 744-763.

12. J.S.R. Jang, C.T. Sun, "Neurofuzzy modeling and control," *IEEE Trans. Fuzzy Sys.*, vol. 3, no. 3, March 1995, pp. 378-406.
13. L.X. wang and J.M. Mendel, "Analysis and design of fuzzy logic controller," *USC SIPI Rep.*184, 1991
14. L.X. wang and J.M. Mendel, "Back-propagation fuzzy systems as nonlinear dynamic system identifiers," in *Proc. IEEE 1992 Int. Conf. Fuzzy systems* (San Diego, CA) Mar.1992, pp. 1409-1418
15. Warren McCulloch, Walter Pitts, "A Logical Calculus of Ideas Immanent in Nervous Activity", Bulletin of Mathematical Biophysics 5:115-133, 1943.
16. Mohamad H. Hassoun, "Fundamentals of Artificial Neural Networks", The MIT Press, Cambridge, Massachusetts, USA, 1995.
17. L.P.J Veelenturf , "Analysis and Applications of Artificial Neural Networks", New York: Prentice Hall, 1995.
18. Simon.Haykin, "Neural Networks", MacMaster university, ISBN 0-13-226556-7, Neural networks (Computer science), QA76.87.H39, 1994.
19. Lefteri H. Tsoukalas and Robert E. Uhrig, "Fuzzy and Neural Approaches in engineering," ISBN 0-471—16003-2, pp. 240.
20. Lefteri H. Tsoukalas and Robert E. Uhrig, "Fuzzy and Neural Approaches in engineering," ISBN 0-471—16003-2, pp. 229-288.
21. Jardine. N. and Sibson. R., "Mathematical Taxonomy," Wiley, London, 1971
22. Sneath, P.H.A. and Sokal, R.R, "Numerical Taxonomy," Freeman, San Francisco, CA, 1973.
23. Kevin M.Passino , Steohen Yurkovich, "Fuzzy Control", Menlo Park, Calif. : Addison-Wesley: World Scientific, ISBN 0-201-18074, 1998.
24. Henk B.Verbruggen, Robert Babuska "Fuzzy Logic Control Advances in Applications", World scientific series in robotics and intelligent systems, vol 23 , Singapore; River Edge, NJ, ISBN 981-02-3825-8, 1999.
25. L. A. Zadeh, "Fuzzy Sets", Information and Control, 8, pp. 338-353, 1965.
26. Lee, C. C., "Fuzzy logic in control systems: fuzzy logic controller B Part I", *IEEE Transactions on Systems, Man and Cybernetics,* 1990, vol. 20, no:2,

p.404-418.

27. Lee, C. C., "Fuzzy logic in control systems: fuzzy logic controller B Part II", *IEEE Transactions on Systems, Man and Cybernetics,* 1990, vol. 20, no:2, p.419-435.
28. Mamdani, E.H. and S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller," *International Journal of Man-Machine Studies*, Vol. 7, No. 1, pp. 1-13, 1975.
29. L.X. Wang and J.M. Mendel, "Generating fuzzy rules by learning from examples," *IEEE Trans. Syst., Man, Cybern.*, vol. 22, pp. 1414-1427, June 1992
30. L.X. Wang and J.M. Mendel, "Fuzzy basis function, universal approximation, and orthogonal least-squares learning," *IEEE Trans. Neural Networks*, vol. 3, PP.807-814, sept.1992.
31. R.J.S. Jang and C. T. Sun, "Functional equivalence between radial basis function networks and fuzzy inference systems," *IEEE Trans. Neural Networks*, vol. 4, pp. 156-159, Jan. 1993.
32. J.J. Bucley, Y.Hayashi, and E. Czogala, "On the equivalence of neural networks and fuzzy expert systems", in *Int. Joint Conf. Neural network.* Baltimore, MD, pp.691-695, June 7-11, 1992.
33. Jian-Qin chen and Yu-Geng Xi "Nonlinear System Modeling by Competitive Learning and Adaptive Fuzzy Inference System", *IEEE Man and Cybernetics*, Part C: *Application and Reviews*, vol. 28, pp. 231-238, No. 2, may 1998.
34. J.Nie, "constructing fuzzy model by self-organizing counterpropagation network," *IEEE Trans. Syst., Man,Cybern.*, vol. 25, pp. 963-970, June 1995.
35. K.Narendra and K.Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Networks*, vol. 1, pp. 4-27, Jan. 1990.
36. J.Nie, "Constructing fuzzy model by self-organizing counterpropagation network," *IEEE Trans., Man, Cybern.,* vol. 25, pp.963-970, Ju
37. Mackey, M. C. and L. Glass. 1977. "Oscillations and chaos in physiological control systems," *Science* 197: 287-289.
38. J.S. Kim and N.Kasabov, "HyFIS: Hybrid connectionist fuzzy inference for

adaptive dynamic systems," *Neural Networks*, Vol.12, NO. 9, pp. 1301-1321, 2001.

39. R.Jang, "ANFIS: Adaptive network-based fuzzy inference system," *IEEE Trans.Syst., Man, Cybern.*, vol. 23, no.3, pp. 665-685, Jun,1993.
40. N.Kasabov and Q. Song, "DENFIS: Dynamic, evolving neural-fuzzy inference systems and its application for time-series prediction," *IEEE Trans. Fuzzy Syst.*, vol 10, no. 2, pp. 144-154, Apr.2002.
41. Qun Song and Nikola K.Kasabov, "NFI: A Neural-Fuzzy Inference Method for Transductive Reasoning", *IEEE Trans., Fuzzy Syst.*, vol. 13, No. 6, pp. 799-808, December 2005.
42. Li-xin Wang, "Stable and Optimal Fuzzy control of linear systems", *IEEE Trans., Fuzzy Syst.*, vol 6, No 1, pp 137-143, Feb. 1998.
43. Jung-Hsien Chang, Pei-yi Hao, "Support Vector Learning Mechanism for Fuzzy Rule-Based Modeling: A New Approach", *IEEE Trans., Fuzzy Syst.*, vol 12, No.1, pp 1-12, Feb 2004.