NEAR EAST UNIVERSITY

GRADUATE SCHOOL OF APPLIED
AND SOCIAL SCIENCES

NEURO-FUZZY SYSTEMS FOR MODELING NONLINEAR PROCESSES

Marwan Alakhras

Master Thesis

Department Of Computer Engineering

Nicosia 2003

Marwan Alakhras : Neuro-fuzzy system for modeling nonlinear processes

Approval of The Graduate School of
Applied and Social Sciences

Prof. Dr. Fakhraddin Mamedov
Director

We certify that this thesis is satisfactory for the award of the degree of Master of Science in Computer Engineering

Examining Committee in charge:

Prof. Dr. Fakhraddin Mamedov, Committee Chairman, Dean of Engineering Faculty, NEU

Assoc. Prof. Dr. Adnan Khashman, Committee Member, Chairman of Electrical and Electronic Engineering Department, NEU

Assoc. Prof. Dr. Raşad Aliyev, Committee Member, Department of Applied Mathematic And Computer Sciences, EMU

Assoc. Prof. Dr. Rahib Abiyev, Supervisor, Computer Engineering Department, NEU

AKNOWLEDGMENET

When I started my thesis few months a go I was just wondering if I can success in this mission which assigned to me. But because of the belief of the human of intelligence, strangeness, willpower, and the ability of creativity I resolve to complete this mission.

First of all I am indebted to a number of individuals, who contributed directly or indirectly to the thesis text. In particular I would like to express my thanks to my advisor Assoc. Prof. Dr. Rahib Abiyev for his help, valuable guideness, and patience. I would like to thank Assoc. Prof. Dr. Adnan Khashman for introducing me to the fields of artificial intelligence.

I would like to thank my family, especially my father, mother, bothers and sisters for their material and moral support, encouragement and patience, I would like to thank my friends here in North Cyprus, home mete Emad, Alla, and my younger brother Omran, and many others.

I would like to express my thanks to the jury members as well for their contribution, I would like to thank every one who should be thanked, every one I met on this long journey.

Last but not least, I am very grateful to Near East University, its staff, students, and every one I met their.

ABSTRACT

Taking into account all possible factors and create a model of given object on the base of traditional methods is very difficult and impossible for its practical use. Such as in the deterministic models the consideration of all factors is impossible, the use of these models leads to ineffective determination of control parameters. In such case for operative control of the processes, man-operator makes decision using his long-term experiences. Taking into consideration all these it is important to develop an intelligent system on the base of knowledge of the experienced specialists. For such cases one of effective means for information processing is the use of fuzzy logic [1,2].

In the thesis the implementation of a fuzzy system for modeling nonlinear systems and predicting exchange rate using parallel architecture and learning capabilities of adaptive neural network based fuzzy system in MATLAB programming package is considered.

To solve these problems the state of application problem neuro-fuzzy system for solving different problems is given. The fuzzy rule base systems have been introduced. Fuzzy variables, fuzzy membership functions, structure of fuzzy inference system are given, the functions of main blocks of fuzzy inference system and its operational principles are described. Also structures and learning algorithms of neural network are given. mathematical model of neuron, neural network structures such as feedforword, feedback and different neural network's learning algorithms are described. The fuzzy inference system structure and neural network's learning associated with matlab tool box are used for modelling nonlinear systems and predicting exchange rate. The simulation for modelling of nonlinear objects and predicting exchange rate problems is carried out on the base of neuro-fuzzy system. Simulation is realized in MATLAB package, the results of simulation are analyzed. In conclusion the obtained important results and contribution from the thesis are presented.

CONTENTS

INTRODUCTION

The traditional artificial intelligence systems mainly based on the symbolic information processing. Despite successes of traditional artificial intelligence systems in developing of different systems for solving problems, automatically proving theorems, recognizing patterns as well as in constructing game systems, expert systems, natural language understanding systems, the expectations have not been approved. All traditional artificial intelligent systems widely used in various areas of human activity have been realized on the base of hard computing, often using computers. In the last years the significant improvement can be noticed in number of applied artificial intelligence systems based on neural networks, fuzzy logic, evolutionary programming. There are number of research works, publications, which are devoted to fuzzy logic, genetic algorithms, neural computing etc. This allows the scientists to focus their investigations on artificial intelligence systems that make a shift nearer to Soft Computing [2].

Soft Computing (SC) methodology includes new computational approaches as fuzzy logic (NN), neural networks (NN), evolutionary computation, probabilistic reasoning (PR) and so on. These approaches allow solving many important real-world problems, where their solutions were impossible using traditional artificial intelligence methods.

In soft computing fuzzy logic is concerned in the main with imprecision and approximate reasoning, neural network with learning, probabilistic reasoning with uncertainty, and genetic algorithm with global optimization and searching and chaos theory with nonlinear dynamics. SC methodology considers the development of the system which combines of the above mentioned components. In this thesis we will consider two of them fuzzy logic and neural networks and their combination.

Fuzzy set theory was found to be a very effective mathematical tool for dealing with the modeling and control aspects of complex industrial and not industrial processes as an alternative to other, much more sophisticated mathematical models. Further, the latter circumstance led to the appearance at the beginning of the 1970's of fuzzy logic computer controllers which became a powerfully tool for coping with the complexity and uncertainty with which we are faced in many real world problems of industrial process control. The first investigations in this field had to answer the question: Is it

possible to realize a process controller which deals like a man with the involved linguistic information? The result of these inquires led to the design of the first fuzzy control systems which implemented in hardware and software a linguistic control algorithm. A control engineer can formulate such a control algorithm on the base of the interviews and with human experts who currently work as process operators. But fuzzy system has such disadvantage as knowledge acquisition, learning properties. To avoid from this problem the combination of fuzzy logic and neural network is considered.

Neural networks called connectionist systems are designed to model certain aspects of the human brain. They consist of simple processing elements (neurons) that exchange signals along weighted connections. By modeling the operation principle of neural structures one can get adequate mathematical models. Neural networks have such characteristics as: vitality, parallelism of computations, learning and generalization abilities, analytic description of linear and non-linear problems etc. Due to these characteristics neural network becomes of great importance for application in such areas such as artificial behavior, artificial intelligence, theory of control and decision making, identification, optimal control, robotics etc. [22].

Neuro-fuzzy combinations are considered for several years already. However, the term *neuro-fuzzy* still lacks of proper definition. *Neuro-fuzzy* means the employment of heuristic learning strategies derived from the domain of neural network theory to support the development of a fuzzy system. Neuro-fuzzy approaches provide simple and efficient learning algorithms to derive fuzzy systems from data. Where neuro-fuzzy methods are useful for small problem sizes, graphical models can be used to model less simple interdependencies of variables for larger problems. Like Bayesian networks in probabilistic modeling, probabilistic networks can be learned from data to discover and to describe complex relationships [3-7].

In chapter one the state of application problem neuro-fuzzy system for solving different problems is given. The importance and advantages of used methodology have been clarified.

In chapter two fuzzy rule base has been introduced. Fuzzy variables, fuzzy membership functions, structure of fuzzy inference system are given, the functions of main blocks of fuzzy inference system and its operational principles are described.

In chapter three structures and learning algorithms of neural network are given. Mathematical model of neuron, neural network structures such as feedforword, feedback and different neural network's learning algorithms are described.

In chapter four development of neural network based fuzzy inference system (NFIS) is given. Structure, operation principles and algorithms of NFIS are presented.

In chapter five development and simulation of neuro fuzzy system for modeling of nonlinear objects and predicting exchange rate problems is carried out. Simulation is realized by using MATLAB package, the results of simulation are analyzed. In conclusion the obtained important results and contribution from the thesis are presented.

Due to the advantages of neuro-fuzzy systems they began widely used in different areas. The aims of the work presented with this thesis are:

1. To obtain neural network learning of fuzzy inference systems (neuro-fuzzy system) to solve two different dynamic and statistic problems.
2. Modeling of nonlinear object.
3. Predicting exchange rate of Turkish Lira versus US Dollar.
4. To prove the efficiency of the solved different problems.

CHAPTER 1: NEURAL LEARINING OF FUZZY SYSTEM AND ITS APPLICATION FOR SOLVING INDUSTRIAL PROBLEMS

1.1. Overview

In this chapter state of application problems of neuro-fuzzy systems for solving different industrial problems are considered. *The main branches of the development of* neural network and fuzzy systems in addition to their advantages are described.

1.2. State of application problems of neuro-fuzzy systems

Fuzzy logic and Neural networks have been successfully applied to many of industrial spheres, in robotics, in complex decision-making and diagnostic system, for data compression, in TV and others. Fuzzy sets can be used as a universal approximator, which is very important for modeling unknown objects. Fuzzy technology has such characteristics as interpretability, transparency, plausibility, graduality, modeling, reasoning, imprecision tolerance. Neural networks have such capabilities as parallel processing, learning characteristics, and description of nonlinear function. In spite of these characteristics they have some disadvantages. Weakness of fuzzy logic is knowledge acquisition, learning, neural networks- black box, interpretability

To increase the quality of the control system the development of them on the base of combination of soft computing components are preferable. These combinations allow creating hybrid systems for helping to overcome these disadvantages.

Hybrid systems are defined in many different ways. In a simple way, hybrid systems are those composed by more than one intelligent system. Hybrid systems are expected to be more powerful due to the combining advantages of different intelligent techniques. Among the most popular hybrid models are the Neuro-Fuzzy systems, Neuro-Genetic systems, Neural-Statistic systems and Fuzzy-Genetic systems.

The complexity and dynamics of real-world problems, especially in engineering and manufacturing, require sophisticated methods and tools for building on-line, adaptive intelligent systems. Such systems should be able to grow as they operate, to

update their knowledge and refine the model through interaction with the environment, that is the systems should have an ability of learning [26].

Among the different approaches to intelligent computation, fuzzy logic provides a strong framework for achieving robust and yet simple solutions. Fuzzy logic can be further strengthened by the introduction of learning capabilities, such as those of artificial neural networks. A large number of adaptive fuzzy or neuro-fuzzy models have been reported in the literature, which aim at amalgamating the benefits of both computational approaches, namely the learning capabilities of neural networks and the representation power and transparency of fuzzy logic systems [12].

A Neuro-Fuzzy system combine the learning capabilities of neural networks with the linguistic rule interpretation of Fuzzy inference systems. The basic idea of a neuro-Fuzzy system is the implementation of a Fuzzy Inference System under the distributed parallel architecture of a neural net, thus taking advantage of the learning capabilities of the neural networks. There are many research works in the world about application of neuro-fuzzy system for solving different problems.

Classical control theory usually requires a mathematical model for designing the controller [27]. The inaccuracy of mathematical modelling of the plants usually degrades the performance of the controller, especially for nonlinear and complex, control problems. Recently, the advent of the fuzzy logic controllers (FLC's) and the neural controllers based on multilayered back-propagation neural networks (BPNN's) has inspired new resources for the possible realization of better and more efficient control [31]. They offer a key advantage over traditional adaptive control systems. That is, they do not require mathematical models of the plants. The concept of fuzzy logic has been applied successfully to the control of industrial /processes. Conventionally, the selection of fuzzy if-then rules often relies on a substantial amount of heuristic observation to express proper strategy knowledge. Obviously, it is difficult for human experts to examine all of the input-output data from a complex system to find a number of proper rules for the FLC. For a BPNN, its nonlinear mapping and self-learning abilities have been the motivating factors for its use in developing intelligent control systems. Although (BPNN's) here demonstrated high potential in the non-conventional branch of adaptive control, their long training time usually discourages their applications in industry. Moreover, when they are trained on-line to adapt to plant

variations, the over-tuned phenomenon usually occurs. To overcome the weakness of the BPNN, in this a neural fuzzy inference network (NFIN) is proposed to be suitable for adaptive control of practical plant systems in general. The NFIN is inherently a modified Takagi-Sugeno-Kang (TSK) type fuzzy rule-based model possessing a neural network's learning ability. In contrast to the general adaptive neural fuzzy networks, where the rules should be decided in advance before parameter learning is performed, there are no rules initially in the NFIN- The rules in the NFIN are created and adapted as on-line learning proceeds via simultaneous structure and parameter identification. The NFIN has been applied to a practical control system. As compared to the BPNN under the same training procedure, the simulated results show that not only can the NFIN greatly reduce the training time and avoid the over-tuned phenomenon, but the NFIN also has perfect regulation ability. The performance of the NFIN is also compared to that of the traditional, controller and fuzzy logic controller (FLC) on the control system [13]. The three control schemes are compared through experimental studies with respect to set-points regulation, ramp-points tracking, and the influence of unknown impulse noise and large parameter variation in the control system. It is found that the NFIN control scheme has the best control performance of the other control schemes.

The architecture and learning procedure underlying ANFIS (adaptive network fuzzy system), which is a fuzzy inference system implemented in the framework of adaptive networks by using a hybrid learning procedure.

When the design of fuzzy controller is considered, the main inconvenient of Fuzzy Controller Systems is to determine the rule base, due to the complexity of the plant. Sometimes it is also very difficult to create an appropriate rule base when using the aid of an expert-man. The Neuro-Fuzzy Controller system offers the possibility to create this rule base automatically through a learning phase, evaluating the error response of the system.

A neural fuzzy inference network is proposed to overcome the disadvantages of the BPNN and FLC. The NFIN is a fuzzy rule-based network possessing a neural network's learning ability. Compared to other existing neural fuzzy networks [12], a major characteristic of the network is that no pre-assignment and design of the rules is required. The rules are constructed automatically during the on-line operation. The structure-learning phase and the parameter-learning phase, are adopted on-line for the construction task.

One important task in the structure identification of the NFIN is the partition of the input space, which influences the number of fuzzy rules generated. On-line input space-partition methods reduce not only the number of rules generated but also the number of fuzzy sets in each dimension. Another feature of the NFIN is that it can optimally determine the consequent part of fuzzy if-then rules during the structure-learning phase.

There are different ways of creating a neuro-fuzzy system. One in the ANN-fuzzy controller structure proposed in[Lin 94]. This structure has five layers. Layer1 is a transparent layer; it just transmits input values directly to the next level. Layer2 calculates the membership degrees. Layer3 find the matching degrees of any rule. Layer4 integrates the rules strengths and Layer5 calculates the output.

The idea of using neural networks to design membership functions was proposed by Takagi and Hayashi, and it involved the design of multidimensional membership functions. Many models with this approach have been used. Lin and Lee proposed a neural-network-based model for fuzzy logic control/decision systems. The model represents a feed-forward neural network. Input nodes represent input signals and output nodes represent output decisions or signals. Nodes in the hidden layers implement membership functions and fuzzy logic rules. The system is a fuzzy inference system; however, it uses distributed representation and learning algorithms of a neural network. Parameters representing membership functions are determined using a back-propagation learning algorithm or a gradient-descent technique.

Pal and Mitra [16] proposed a similar model. In their model, inputs are fed to a preprocessor block, which performs the same functions as that of the fuzzifier block in a fuzzy inference system. The output of the preprocessor represents fuzzy membership values. For each input variable term, variables such as *low*, *medium*, and *high* are used. If input consists of n variables, then the preprocessor block yield m x n outputs, where m represents the number of term values used in the model. The output of the preprocessor block is then fed to a multilayer perceptron model. The perceptron model implements the inference engine. Pal and Mitra have used the model for classifying vowels. Kulkarni has developed a similar model and has used it for multispectral image analysis [16].

Kosko suggested models for fuzzy associative memory. In unsupervised classification, the application of fuzzy logic to classical clustering algorithms has resulted in a number of models [4].

An algorithm called the adaptive fuzzy leader clustering has been suggested [18]. The algorithm is similar to the fuzzy adaptive resonance theory. Also, models for fuzzy competitive learning have been developed. In classical clustering methods, a sample is assigned to one class only. However, with fuzzy clustering, for any given input sample, membership values for all output classes are evaluated. The cluster centers or weights representing the centers are updated using these membership values.

In [15] the constructing of neuro-fuzzy system for controlling water distribution network. This system is characterized by two types of faults or uncertainties in such a system: measurement errors caused by equipments and topological errors caused by faults due to the leakage and wrong valve status. Measurement errors are not correlated and thus the measurements with error can be discarded. But the detection and identification of topological errors are still not studied comprehensively. The Generalized Fuzzy Min-Max Neural Networks (GFMM NN) approach for clustering and classification is applied for this purpose. This is a fully integrated hybrid structure.

The neuro-fuzzy recognition system is used to identify and detect a leakage in the system. The training data set is generated by the system state estimation procedure combined with the Confidence Limit Algorithm (CLA) for the quantification of inaccuracies of system state estimation due to uncertainties in input data. The state estimation procedure is based on the mass balances in each node and the specific measurements taken on the node. The neuro-fuzzy recognition considered here is based on the hyperbox fuzzy sets. The hyperbox defines a region of n-dimensional pattern space and is defined by its min-max points. The hyperbox created during the training can represent a distinctive state of the system such as the normal operating state, a leakage between two nodes etc.

A neural network that implements the GFMM clustering/classification algorithm is a three- layered feedforward network. The input layer has $2*n$ number of nodes, two for each of the n-dimensions of the input pattern. Each node in the second layer represents a hyperbox fuzzy set. The connections in the 1^{st} and 2^{nd} layer are min-max points and the transfer function is the hyperbox membership function. Each node in the third layer represents a class. The connections between 2^{nd} and 3^{rd} layer are binary

values: 1 if the second layer hyperbox fuzzy set is a part of the class represented by the output layer node and 0 otherwise. They are stored in a matrix form. The output can be either fuzzy or crisp.

Generation of the training data set of the networks is done in 3 stages: simulation of the state, estimation of accurate measurements and the CLA. Leakage of the system is simulated as a demand between two nodes and not as a pressure difference. Reservoir inflows and the other network consumption are adjusted to compensate the additional demand. The wrong operation of a valve is simulated in a way that the valves those are usually open, remain as closed.

The recognition system developed is a two-level system where the first level is to distinguish the typical behavior of the system (such as night load, peak load etc.) and the second level is to detect the anomalies. The result of this approach shows that the neuro-fuzzy system can be trained successfully for the estimated system-state as well as the residuals with their confidence limit. Both have advantages and disadvantages, however, the simulation of a system based on the estimated system state gives better results in terms of accuracy.

The stand alone structure of the hybrid system is applied for controlling the tank level in solvent dewaxing (oil refinery) plant. The controlling purpose is to keep the tank level stable and to change the outflow rate from the tank as smoothly as possible in order to keep the whole process normal and continuous. The whole dewaxing process itself is difficult to be controlled because of uncertainties and complexities, therefore, usually is controlled by the experts. The difficulty occurs due to the following reasons: the inflow rate to the tank varies with oil filter plugging, feed oil is switched on frequently, the heater has a limit in the change of the flow rate, two different states to control (steady and transient) etc. The steady state is when the tank level goes down and up periodically by stopping and washing one of the oil filters. A transient state occurs when in addition to the above, the feed oil is changed completely, which results the tank level to drop down rapidly.

In order to deal with these states, the neuro-fuzzy controller is built with the following three components:

1. a statistical component to calculate long time tendencies of the flow rate from the historical operational data.

2. a correction component (fuzzy logic) for compensating the flow rate from statistical component to stabilize the tank level. The rule base is built on the basis of the experts' knowledge.
3. a prediction component (neural networks) to predict the inflow rate when the oil is being changed. That is the target of the fuzzy logic controller.

Application of the neuro-fuzzy controller smoothes the tank levels not only in a steady state, but also in transient state when the feed oil is changing. For example, applying the neuro-fuzzy system the tank level ranges between 35%-75%, while on the basis of experts' knowledge it ranges between 30%-80% [15].

Detection of faults in anaerobic process using the fuzzy-neural network has been considered in [13]. The process is very complex as well as unstable and depends on the incoming flow rate, influent organic load etc. There are problems that are less predictable such as pipe clogging which causes an increase in valve opening, foam forming which changes gas flow rate etc. Besides, there are local controllers used to control the individual processes. But there is no technique using on-line measurements and handling an ill-defined process as a whole.

The structure of the neuro-fuzzy system is as follows: The measured signals are transferred into fuzzy variables depending on whether the variable is deviated from the mean value or not. By using additional fuzzy rules, the occurrence of a faulty situation in the system is determined.

Another loosely coupled hybrid model of an ANN and fuzzy logic has been applied for diagnosis of anaerobic treatment plant [13]. The raw data has been processed by fuzzy logic to build a pattern vector (training data), where training data set is classified into pre-specified categories indicating the state of the system. An ANN is then used to classify the process states and to identify the faulty and dangerous states. The hybrid model recognizes the situations caused by pipe clogging, foam forming and bad temperature regulation. The approach can be seen as a tool able to handle with large number of problems in a simple frame.

The neuro-fuzzy system has been found to be a suitable approach for a multi-purpose reservoir system operation. The study concentrates on the application of a fuzzy neural network (stand-alone hybrid structure) and a fuzzy system for reservoir operation and presents a comparison of results obtained. The mathematical expression of dam operations [17] is difficult and somewhat vague because of the presence of many

different constraints which need to be considered. On the other hand the inflow can be predicted on the basis of abundantly available hydrological information within the catchments. The composed system is applied for determining the operation of a reservoir for irrigation and flood control purposes.

The operation line is determined on the basis of the water level in the reservoir, changing inflow, inflow and precipitation coupled with actual historical operation. The neural network used 7 input variables (rainfall, river discharge, predicted flow, changing inflow, water level and release discharge), one hidden layer with three nodes (response to dam basin, discharge and the state of the reservoir) and an output layer with one neuron (describing release of discharge, storage volume or conservation of water level in the reservoir).

For irrigation purposes, the fuzzy control and neuro-fuzzy control give smoother release of discharge. The fuzzy control gives better result in terms of storage volume. For flood control purpose during the typhoon both the controllers give the maximum release, however, the fuzzy neural networks give higher peak value than the fuzzy controller.

The above principle was investigated with a six-layered neural network, in which each layer performs specific actions to represent the fuzzy inference mechanism. The six layers are the input layer, a fuzzification layer, two layers for fuzzy inferences, a defuzzification layer and an output layer. The designed controller is applied for experimental fluid beam balancing system, which balances an unstable beam contained in two tanks, one at each end pumping back or forward from the tanks. The problem was formulated as Multiple Input and Single Output (MISO) problem and the real-time control was evaluated against a PID controller. After a short simulation, the algorithm gave reasonable results compared to the PID controller and further investigation of RTC was suggested.

1.3. Implementation of Neuro-Fuzzy Systems Through Interval Mathematics

Neural network performance is dependent on the quality and quantity of training samples presented to the network. In cases where training data is sparse or not fully representative of the range of values possible, incorporation of fuzzy techniques optimizes performance. That is, while neural networks are excellent classifiers,

introducing fuzzy techniques allow the classification of imprecise data. The neuro-fuzzy system presented here is a neural network that processes fuzzy numbers. It uses interval mathematics in its implementation [20].

The neuro-fuzzy system uses a standard feed-forward network as its basis. The novelty lies in the fact that it processes fuzzy numbers. Specifically, α-cuts of the fuzzy numbers are represented by interval vectors. The back-propagation with momentum learning rule is derived for interval variables. The resulting equations are then employed for training of the system. Thus, the input and output vectors are interval vectors, and the neuronal operations are modified to deal with the interval numbers. Summation of the resultant α-cuts (interval numbers) provide the final fuzzy valued output.

Results of the antecedent mentioned Experiments [12, 13, 15, 16, 17] show that the neuro-fuzzy system's performance is vastly improved over a standard neural network and other existing methods for speaker-independent speech recognition, an extremely difficult classification problem.

Processing fuzzy numbers can be accomplished in a variety of ways. One of the most elegant, because of its simplicity, is by using interval methods.

1.4. Summary

The online self construction and organization property of neuro-fuzzy systems reduces the design efforts and error as compared to other existing intelligent systems such as Neural Network and Fuzzy System, also this property makes it able to deal with problems of a changing environment or plant, which can not be handled perfectly by conventional systems or controllers. The advantages of neuro-fuzzy systems have been verified.

In this chapter the applications of combination of neural networks and fuzzy system for solving different problems are considered. The types of neuro-fuzzy system are described. The obtained results from solving identification, control, classification, optimization, forecasting, etc, problems satisfy the efficiency of application of neuro-fuzzy system to different areas [13, 15, 17].

CHAPTER 2: FUZZY RULE BASED SYSTEMS

2.1. Overview

The fuzzy method differs significantly from conventional ones. Usually a control strategy and a controller itself is synthesized on the base of mathematical models of the object under control involve quantitative, numeric calculations and commonly are constructed in advance, before realization.

There is a different method where, instructions are comprehend and strategies are generated based on a priori verbal communication. Most engineers would accept intuitively that mathematical modelling, which they perform in translating their concept of a control strategy into an automatic controller, is completely different from their own approach to a manual performance of the same task. On the other hand, linguistic description of control seems to be similar to its manual implementation.

Fuzzy systems mainly based on knowledge of experts, this knowledge is often formulated through fuzzy rule-based format. In the fuzzy rule-based input and output variables are often characterized by linguistic values. These linguistic values are described by membership functions.

In this chapter the main elements of fuzzy rule-based system, its structure, main blocks, and their operation principles are presented.

2.2. Fuzzy IF-THEN Rules

Fuzzy if-then rules or fuzzy conditional statements are expressions for the form

IF A THEN B,

where A and B are labels of fuzzy sets characterized by appropriate membership functions. A is premise, B is consequent parts of fuzzy rule. Due to their concise form, fuzzy if-then rules are often employed to capture the imprecise modes of reasoning that play an essential role in the human ability to make decisions in an environment of uncertainty and imprecision.

Usually, the inputs of the fuzzy systems are associated with the premise, and the outputs are associated with the consequence. These If-Then rules can be represented in many forms. Its simple form is Single Input Single Output (SISO). This form has the format

$$\text{If } u \text{ is } A \text{ Then } y \text{ is } B \tag{2.1}$$

Other standard forms, Multi-Input Multi-Output (MIMO) and Multi-Input Single-Output (MISO), are considered here. The MISO form of a linguistic rule is

$$\text{If } u_1 \text{ is } A_1^j \text{ and } u_2 \text{ is } A_2^k \text{ and ,....., and } u_n \text{ is } A_n^l \text{ Then } y_q \text{ is } B_q^p \tag{2.2}$$

It is an entire set of linguistic rules of this form that the expert specifies on how to control the system. Note that if u_1="velocity error" and A_1^j= "positive large", then "u_1 is A_1^j", a single term in the premise of the rule, means "velocity error is positive large". It can be easily shown that the MIMO form for a rule (i.e. one with consequents that have terms MISO rules using simple rules from logic. For instance, the MIMO rule with n inputs and m =2 outputs

$$\text{If } u_1 \text{ is } A_1^j \text{ and } u_2 \text{ is } A_2^k \text{ and ,....., and } u_n \text{ is } A_n^l \text{ Then } y_1 \text{ is } B_1^r \text{ and } y_2 \text{ is } B_2^s \tag{2.3}$$

Is linguistically (logically) equivalent to the two rules

$$\text{If } u_1 \text{ is } A_1^j \text{ and } u_2 \text{ is } A_2^k \text{ and ,....., and } u_n \text{is } A_n^l \text{ Then } y_1 \text{ is } B_1^r$$

$$\text{If } u_1 \text{ is } A_1^j \text{ and } u_2 \text{ is } A_2^k \text{ and ,....., and } u_n \text{is } A_n^l \text{ Then } y_2 \text{ is } B_2^s$$

This is the case since the logical "and" in the consequent of the MIMO rule is still represented in the two MISO rules since it still assert that the both the first "and" second rule are valid. For implementation, then two fuzzy systems should be specified, one with output y_1 and the other with the output y_2. The logical "and" in the consequent of the MIMO rule is still represented in the MISO case since by implementation two fuzzy systems asserting that the ones set of rules is true "and" another is true. An example that describes a simple fuzzy rule is

If pressure is high, then volume is small

Where *pressure* and *volume* are linguistic variables, *high* and *small* are linguistic values or labels that are characterized by membership functions.

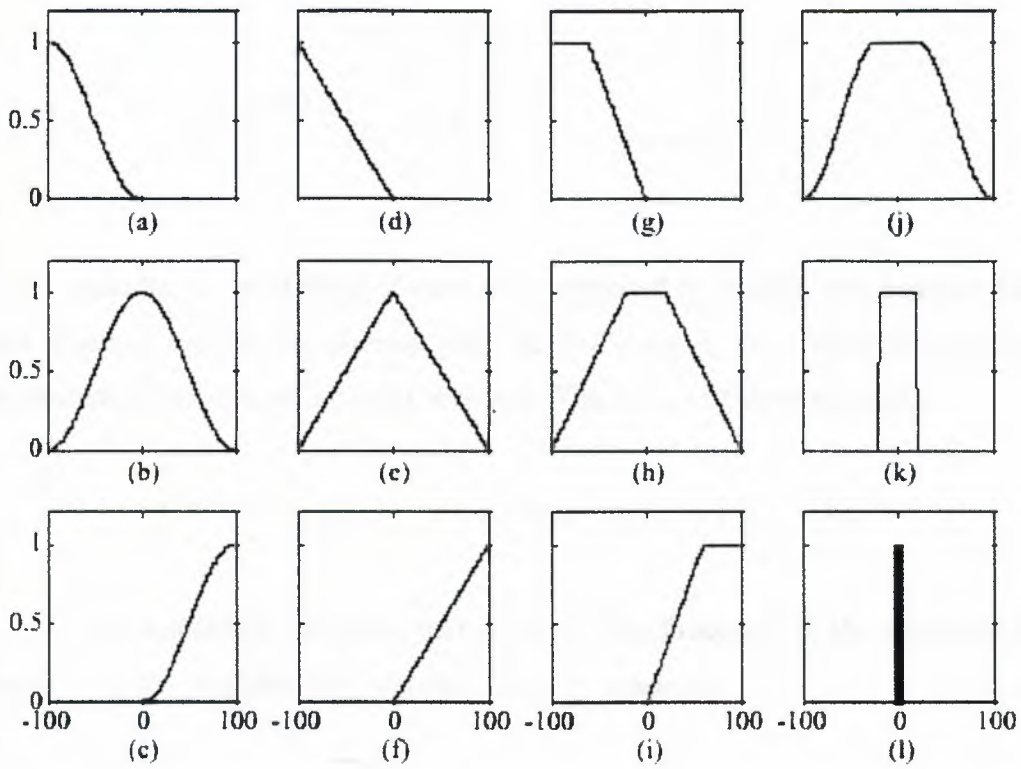


Figure 2.1 Examples of membership functions. Read from top to bottom, left to right: (a) s—function, (b) π— function, (c) z—function, (d-f) triangular versions, (g-i) trapezoidal versions, (j) flat π— function. (k) rectangle. (I) singleton.

Membership functions can be flat on the top, piece-wise linear and triangle shaped, rectangular, or ramps with horizontal shoulders. Figure.2.1 shows some typical shapes of membership functions.

A common example of a function that produces a bell curve is based on the exponential function,

$$\mu\ (x)\ =\ \exp\ \left[\frac{-\ (x\ -\ x_0\ )^2}{2\sigma^{\ 2}}\right] \tag{2.4}$$

This is a standard Gaussian curve with a maximum value of 1 ,x is the independent variable on the universe, x to is the position of the peak relative to the universe, and σ is the standard deviation. Another definition which does not use the exponential is

$$\mu\,(x)=\left[1+\left(\frac{x-x_0}{\sigma}\right)^2\right]^{-1} \tag{2.5}$$

Triangle membership function is described by

$$\mu(x) = \begin{cases} 1 - \dfrac{\bar{x} - x}{\bar{x} - x_l} & , \quad x_l < x < x \\ 1 - \dfrac{x - \bar{x}}{x_r - \bar{x}} & , \quad \bar{x} < x < x_r \end{cases} \tag{2.6}$$

Another form of fuzzy if-then rule, proposed by Takagi and Sugeno, has fuzzy sets involved only in the premise part. In this method, the consequent part is just a mathematical function of the input variables. The format of the method is:

$$\text{if } A_1(x_1),\ A_2(x_2),\ \ldots,\ A_n(x_n) \text{ then } Y = f(x_1, x_2, \ldots, x_n). \tag{2.7}$$

The antecedent (premise) part is fuzzy. The function f in the consequent part is usually a simple mathematical function, linear or quadratic:

$$f = a_0 + a_1 * x_1 + a_2 * x_2 + \ldots + a_n * x_n \tag{2.8}$$

By using Takagi and Sugeno's fuzzy if-then rule, the resistant force on a moving object can be described as follows:

$$\textit{If velocity is high, then force} = k * (\textit{velocity})^2$$

Here, again, ***high*** in the premise part is linguistic label characterized by an appropriate membership function. However, the consequent part is described by a nonfuzzy equation of the input variable, velocity.

Both types of fuzzy if-then rules have been used extensively in both modelling and control. Through the use of linguistic labels and membership functions, a fuzzy if-then rule can easily capture the spirit of a 'rule of thumb' used by humans. From another point of view, due to the qualifiers on the premise parts, each fuzzy if-then rule can be viewed as a local description of the system under consideration. Fuzzy if-then rules form a core part of the fuzzy inference system to be introduced below.

2.3. Fuzzy Inference Systems

Fuzzy inference systems are also known as fuzzy-rule-based systems, fuzzy associative memories (FAM) [14], or fuzzy controllers when used as controllers. Basically a fuzzy inference system is composed of five functional blocks (see figure.2.2);

- a rule base containing a number of fuzzy if-then rules;
- a database which defines the membership functions of the fuzzy sets used in the fuzzy rules;
- a decision-making unit which performs the inference operations on the rules;
- a fuzzification inference which transform the crisp inputs into degrees of match with linguistic values;
- a defuzzification inference which transform the fuzzy results of the inference into crisp output.

Usually, the rule base and the database are jointly referred to as knowledge base.

The steps of fuzzy reasoning (inference operations upon fuzzy if-then rules) performed by fuzzy inference systems are:

1. Compare the input variable with the membership functions on the premise part to obtain the membership values (or compatibility measures) of each linguistic label. (This step is often called fuzzification).
2. Combine (through a specific *T-norm* operator [14], usually multiplication or min.) the membership values on the premise part to get firing strength (weight) of each rule.
3. Generate the qualified consequent (either fuzzy or crisp) of each rule depending on the firing strength.
4. Aggregate the qualified consequent to produce a crisp output. (This step is called defuzzification.)

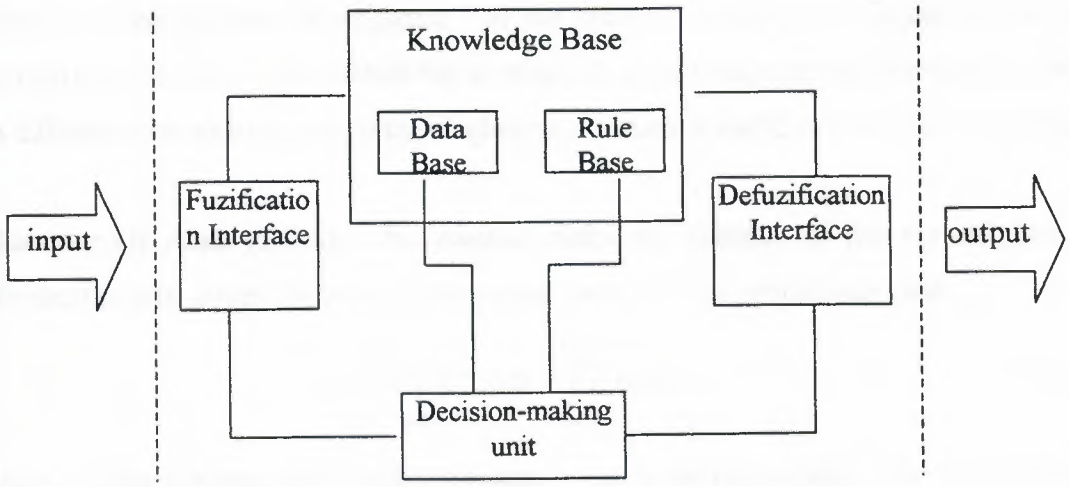


Figure.2.2 Structure of fuzzy inference system

2.4. Defuzzfication Methods

There are many defuzzification methods that can be used in fuzzy inference system and the following are the most known ones.

Centre Of Gravity (COG): The crisp output value u is the abscissa under the centre of gravity of the fuzzy set,

$$u = \frac{\sum_i \mu(x_i) x_i}{\sum_i \mu(x_i)} \tag{2.9}$$

Here X_i is a running point in a discrete universe, and $\mu(x_i)$ is its membership value in the membership function. The expression can be interpreted as the weighted average of the elements in the support set. For the continuous case, replace the summations by integrals. It is a much used method although its computational complexity is relatively high. This method is also called *centroid of area* [27].

Centre of gravity method for singletons (COGS): If the membership functions of the conclusions are singletons, the output value is

$$u = \frac{\sum_i \mu(s_i) s_i}{\sum_i \mu(s_i)} \tag{2.10}$$

Here s_i is the position of singleton i in the universe. and $\mu(s_i)$ is equal to the firing strength α_i of rule i. This method has a relatively good computational complexity and u is differentiable with respect to the singletons s_i, which is useful in neuro-fuzzy systems.

Bisector Of Area (BOA): This method picks the abscissa of the vertical line that divides the area under the curve in two equal halves. In the continuous case,

$$u = \left\{ x \left| \int_{Min}^{x} \mu(x)dx = \int_{x}^{Max} \mu(x)dx \right. \right\} \tag{2.11}$$

Here x is the running point in the universe, $\mu(x)$ is its membership. *Min* is the leftmost value of the universe, and *Max* is the rightmost value. Its computational complexity is relatively high, and it can be ambiguous. For example, if the fuzzy set consists of two singletons any point between the two would divide the area in two halves; consequently it is safer to say that in the discrete case. BOA is not defined.

Center Of Average (COA): A crisp output y_q^{Crisp} is chosen using the centers of each of the output membership functions and the maximum certainty of each of the conclusions represented with the implied fuzzy sets, and is given by

$$y_q^{Crisp} = \frac{\sum_{i=1}^{R} b_i^q \sup_{yq}\{\mu B_q^i(y_q)\}}{\sum_{i=1}^{R} \sup_{yq}\{\mu B_q^i(y_q)\}} \tag{2.12}$$

where "*sup*" denotes the "supermum [27]" (i.e., the least upper bound which can often be thought of as maximum value). Hence, $\sup_x\{\mu(x)\}$ can be simply thought as the highest value of $\mu(x)$.

Max Criterion: A crisp output y_q^{Crisp} is chosen as the point on the output universe of discourse y_q for which the overall implied fuzzy set B_q achieves a maximum-that is,

$$y_q^{Crisp} \in \left\{ \arg\sup_{y_q} \{\mu B_q(y_q)\} \right\} \tag{2.13}$$

Here, "$\arg\sup_x\{\mu(x)\}$" returns the value of x that results in the supermum of the function $\mu(x)$ being achieved. For example, suppose that $\mu_{Overall}(u)$ denotes the

membership function for the overall implied fuzzy set that is obtained by taking the maximum of the certainty values of $\mu(1)$ and $\mu(2)$ over all u.

Mean Of Maxima (MOM): An intuitive approach is to choose the point with the strongest possibility i.e. maximal membership. It may happen. Though, that several such points exist, and a common practice is to take the *mean of maxima* (MOM). This method disregards the shape of the fuzzy set, but the computational complexity is relatively good.

Leftmost Maximum (LM), and Rightmost Maximum (RM): Another possibility is to choose the leftmost maximum (LM), or the rightmost maximum (RM). In the case of a robot, for instance, it must choose between left and right to avoid an obstacle in front of it. The defuzzifier must then choose one or the other, not something in between. These methods are indifferent to the shape of the fuzzy set, but the computational complexity is relatively small.

2.5. Types of Fuzzy Systems

Several types of fuzzy reasoning have been proposed in the literature [11]. Depending on the types of fuzzy reasoning and fuzzy if-then rules employed, most fuzzy inference systems can be classified into three types (see figure.2.3)

Type 1: the overall output is the weighted average of each rule's crisp output introduced by rule's firing strength (the product or minimum of the degrees of match with the premise part) and the output membership functions. The output membership functions used in this scheme must be monotonic functions.

Type 2: the overall fuzzy output is derived by applying 'max' operation to the qualified fuzzy outputs (each of which is equal to the minimum of firing strength and the output membership function of each rule). The above mentioned schemes are used to choose the final crisp output based on the overall fuzzy output; for example, centroid of area, bisector of area, mean of maxima, maximum criterion, etc.

Type 3: Takagi and Sugeno's fuzzy if-then rules are used. The output of each rule is linear combination of input variables plus a constant term, and the output is the weighted average of each rule's output. Figure 2.3 utilizes a two-rule two-input fuzzy inference system to show different types of fuzzy rules and fuzzy reasoning mentioned above. Be aware that most of the differences come from the specification of the consequent part (monotonically non-decreasing or bell-shaped membership functions, or crisp function) and thus the defuzzification schemes (weighted average, centroid of area, etc) are also different.

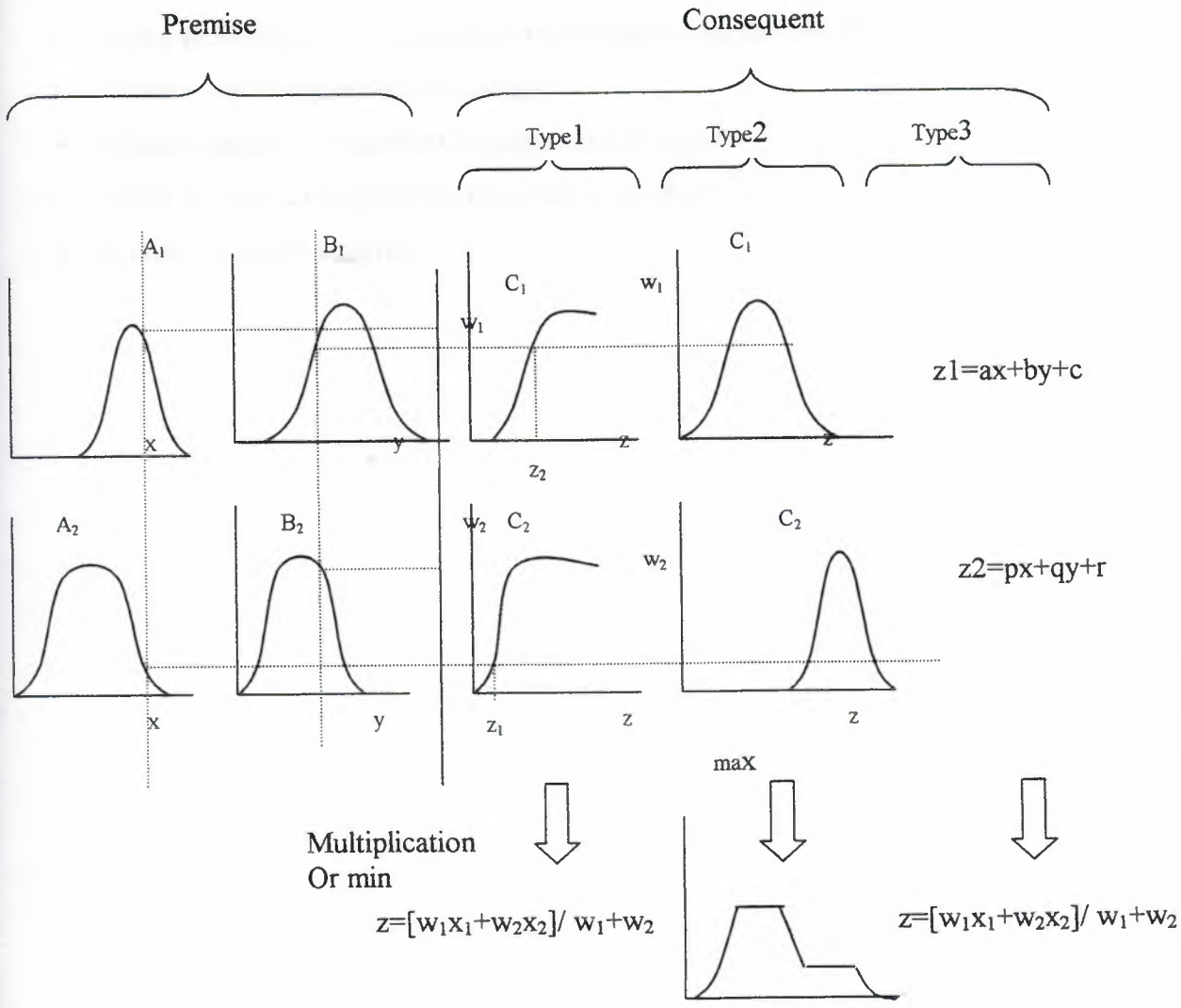


Figure.2.3. Types of fuzzy reasoning mechanisms

2.6. Summary

Fuzzy systems mainly based on knowledge of experts, or generated form sample data points, this knowledge is often formulated through fuzzy rule-based to fuzzy sets. Fuzzy sets allow partial memberships. Commonly used membership functions are triangular, trapezoidal, bell shaped, and Gaussian curves In this chapter the different forms of fuzzy rules, structure of fuzzy inference system, its main blocks and operation principle are discussed. The main types of fuzzy rule base and its reasoning mechanism have been described.

Here we can summarize, why fuzzy system?

- Ability to translate imprecise/vague knowledge of human experts.
- Simple, easy to implement technology.
- software design and hardware implementation support.
- results are easy to transfer from product to product.
- smooth controller behavior.

CHAPTER 3: NEURAL NETWORKS AND LEARNING ALGORITHMS

3.1. Overview

A neural network is a massively parallel-distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:

1. Knowledge is acquired by the network through a learning process.
2. Neuron inter neuron connection strengths known as synaptic weights are used to store the knowledge.

The procedure used to perform the learning process is called learning algorithm, the function of which is to modify the synaptic weights of the network in an orderly fashion so as to attain a desired design objective. The modification of synaptic weights provides the traditional method for the design of neural networks.

In this chapter the different mathematical models of neurons, neural networks structure are considered. The different learning algorithms of neural networks are presented.

3.2. Models of a neuron

A neuron is an information-processing unit that is fundamental to the operation of a neural network.

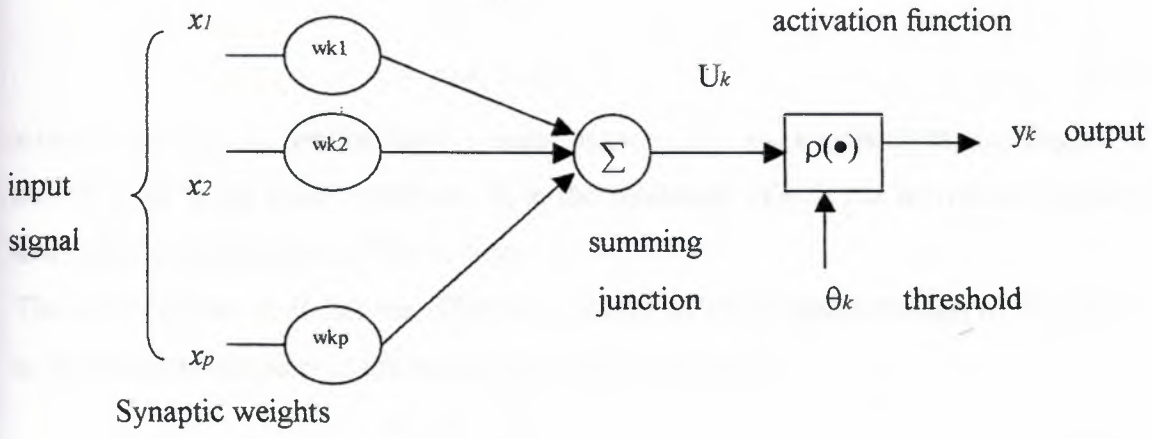


Figure 3.1 Non-linear model of neuron.

Three basic elements of neuron can be identified:

1. A set of synapses or connecting links, each of which is characterized by a weight or strength of its own. Specifically, a signal x_j at the input of synapse j connected to neuron k is multiplied by the synaptic weight w_{kj}. It is important to make a note of the manner in which of which of the subscript of the synaptic weight w_{kj} is written. The first subscript refers to the neuron in question and the second subscript refers to the input end of the synapse to which the weight refers; the reverse of this notation is also used in the literature. The weight w_{kj} is positive if the associated synapse is excitatory and negative if the synapse is inhibitory.
2. An adder for summing the input signals, weighted by the respective synapses of the neuron; the operation described here constitutes a linear combiner.
3. An activation function for limiting the amplitude of the output of a neuron. The activation function is also referred in the literature as a "squashing function" in that it squashes (limits) the permissible amplitude range of the output signal to some finite value. Typically the normalized amplitude range of the output of a neuron is written as the closed unit interval [0,1] or alternatively [-1,1].

The model of a neuron shown in the figure above includes an externally applied threshold θ_k, that has the effect of lowering the net input of the activation function. On the other hand, employing a bias term rather than a threshold may increase the net input of the activation function; the bias is negative of the threshold.

In mathematical terms a neuron k may be described by writing the following equations

$$u_k = \sum_{j=1}^{p} w_{kj} x_j \tag{3.1}$$

$$y_k = \rho(u_k - \theta_k) \tag{3.2}$$

where $x_1, x_2, \ldots\ldots, x_p$ are the input signals, $w_{k1}, w_{k2}, \ldots\ldots, w_{kp}$ are the synaptic weights of neuron k, u_k is the linear combiner, θ_k is the threshold, $\rho(\bullet)$ is the activation function, and y_k is the output signal of the neuron.

The use of threshold θ_k has the effect of applying an affine transformation to the output u_k of the linear combiner in the model shown above as follow:

$$v_k = u_k - \theta_k \tag{3.3}$$

In particular, depending on whether the threshold θ_k is positive or negative, the relationship between the effective internal activity level or activation potential V_K of neuron k and the linear combiner output u_k is modified in manner illustrated in the figure below, note that as a result of this affine transformation, the graph of v_k versus u_k no longer passes through the origin.

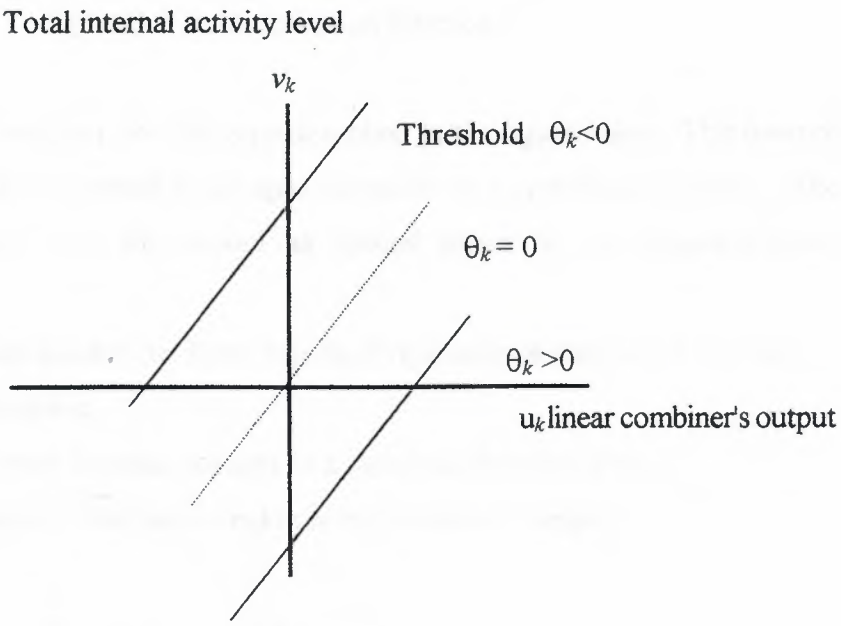


Figure 3.2 Affine transformation produced by the presence of a threshold.

3.2.1. Types Of Activation Function

The activation function denoted by $\rho(\bullet)$, defines the output of a neuron in terms of the activity level at its input. There are three basic types of activation functions:

a) Hard Activation Function, for this type of activation functions described in the figure below, the output of neuron k employing such a threshold function is expressed as:

$$y_k = \begin{cases} 1, & if\ v_k \geq 0 \\ 0, & if\ v_k < 0 \end{cases}$$

Where v_k is the internal activity level of the neuron; that is

$$v_k = \sum_{j=1}^{p} w_{kj} x_j - \theta_k \qquad (3.4)$$

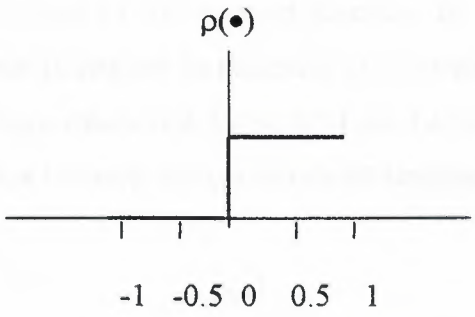


Figure 3.3 Hard activation function.

b) Piecewise-Linear Function, for this type described in the figure below. This form of activation function may be viewed as an approximation to a non-linear amplifier. The following two situations may be viewed as special forms of the piecewise-linear function:

1- A linear combiner arises if the linear region of operation is maintained without running into saturation.

2- The piecewise-linear function reduces to a threshold function if the amplification factor of the linear region is made infinitely larger.

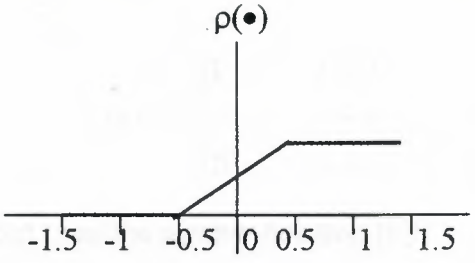


Figure 3.4 Piecewise linear function.

$$p(v) = \begin{cases} 1, & v \geq 1/2 \\ v, & +1/2 > v > -1/2 \\ 0, & v \leq -1/2 \end{cases}$$

c) Sigmoid Function, is the most common form of activation functions used in the construction of artificial neural network. It is defined as strictly increasing function that exhibits smoothness and asymptotic properties, an example of sigmoid function is the logistic function, defined by:

$$p(v) = \frac{1}{1 + \exp(-av)}$$

where a is the slope parameter of the sigmoid function. By varying the parameter a sigmoid function of different slopes can be obtained, as illustrated in the figure below. In fact, the slope at the origin equals $a/4$. In the limit, as the slope parameter approaches infinity, the sigmoid function becomes simply threshold function.

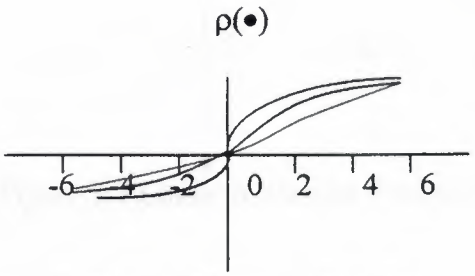


Figure 3.5 Sigmoid functions.

Whereas a threshold function assumes the value of 0 or 1, a sigmoid function assumes a continuous range of values from 0 to 1.

It is noted that the sigmoid function is differentiable whereas the threshold function is not. It is some times desirable to have activation function of range from -1 to +1, in which case the activation function assumes an antisymmetric form with respect to the origin.

$$p(v) = \begin{cases} 1, & v > 0 \\ v, & v = 0 \\ 0, & v < 0 \end{cases}$$

Which is commonly referred to as the *signum function* [9].

d) Gaussian Activation Function. The Gaussian activation function is a radial function (symmetric about the origin) that requires a variance value, v>0, to shape the Gaussian function. In some networks is used
in conjunction a dual set of connections.

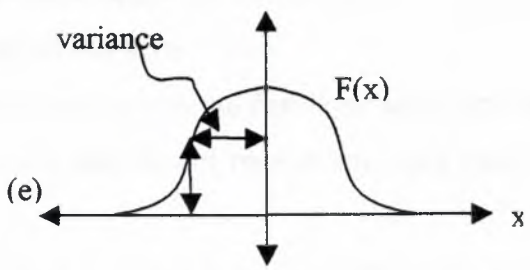


Figure 3.6 Gaussian Activation Function

e) Linear Activation Function. A linear function produces a linearly modulated output from the input.

$$p(v) = av + b$$

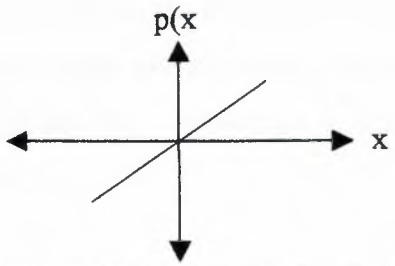


Figure 3.7 Linear Activation Function

3.2.2. Communication and Types Of Connections

Neurons are connected via a network of paths carrying the output of one neuron as input to another neuron. These paths is normally unidirectional, there might however be a two-way connection between two neurons, because there may be another path in reverse direction. A neuron receives input from many neurons, but produces a single output, which is communicated to other neurons.

The neuron in a layer may communicate with each other, or they may not have any connections. The neurons of one layer are always connected to the neurons of at least another layer.

3.2.2.1. Inter-Layer Connections

There are different types of connections used between layers; these connections between layers are called inter-layer connections.

- **Fully connected**; each neuron on the first layer is connected to every neuron on the second layer.
- **Partially connected**; a neuron of the first layer does not have to be connected to all neurons on the second layer.
- **Feed forward**; the neurons on the first layer send their output to the neurons on the second layer, but they do not receive any input back form the neurons on the second layer.
- **Bi-directional**; there is another set of connections carrying the output of the neurons of the second layer into the neurons of the first layer.

Feed forward and bi-directional connections could be fully or partially connected.

- **Hierarchical**; if a neural network has a hierarchical structure, the neurons of a lower layer may only communicate with neurons on the next level of layer.
- **Resonance**; the layers have bi-directional connections, and they can continue sending messages across the connections a number of times until a certain condition is achieved.

3.2.2.2. Intra-Layer Connections

In more complex structures the neurons communicate among themselves within a layer, this is known as intra-layer connections. There are two types of intra-layer connections.

- **Recurrent;** The neurons within a layer are fully- or partially connected to one another. After these neurons receive input form another layer, they communicate their outputs with one another a number of times before they are allowed to send their outputs to another layer. Generally some conditions among the neurons of the layer should be achieved before they communicate their outputs to another layer.
- **On-center/off surround;** A neuron within a layer has excitatory connections to itself and its immediate neighbours, and has inhibitory connections to other neurons. One can imagine this type of connection as a competitive gang of neurons. Each gang excites itself and its gang members and inhibits all members of other gangs. After a few rounds of signal interchange, the neurons with an active output value will win, and is allowed to update its and its gang member's weights. (There are two types of connections between two neurons, excitatory or inhibitory. In the excitatory connection, the output of one neuron increases the action potential of the neuron to which it is connected. When the connection type between two neurons is inhibitory, then the output of the neuron sending a message would reduce the activity or action potential of the receiving neuron. One causes the summing mechanism of the next neuron to add while the other causes it to subtract. One excites while the other inhibits.)

3.3. Neural Network's Architectures

Often, it is convenient to visualize neurons as arranged in layers. Typically, neurons in the same layer behave in the same manner. Key factors in determining the behavior of a neuron are its activation function and the pattern of weighted connections over which it sends and receives signals. Within each layer, neurons usually have the same activation function and the same pattern of connections to other neurons. To be more specific, in many neural networks, the neurons within a layer are either fully interconnected or not interconnected at all. If any neuron in a layer (for instance, the layer of hidden units) is connected to a neuron in another layer (say, the output layer), then each hidden unit is connected to every output neuron.

The arrangement of neurons into layers and the connection patterns within and between layers is called the *net architecture.* Many neural nets have an input layer in which the activation of each unit is equal to an external input signal. The net illustrated in figure 3.8 consists of input units, output units, and two hidden units (the units that are neither an input unit nor an output unit).

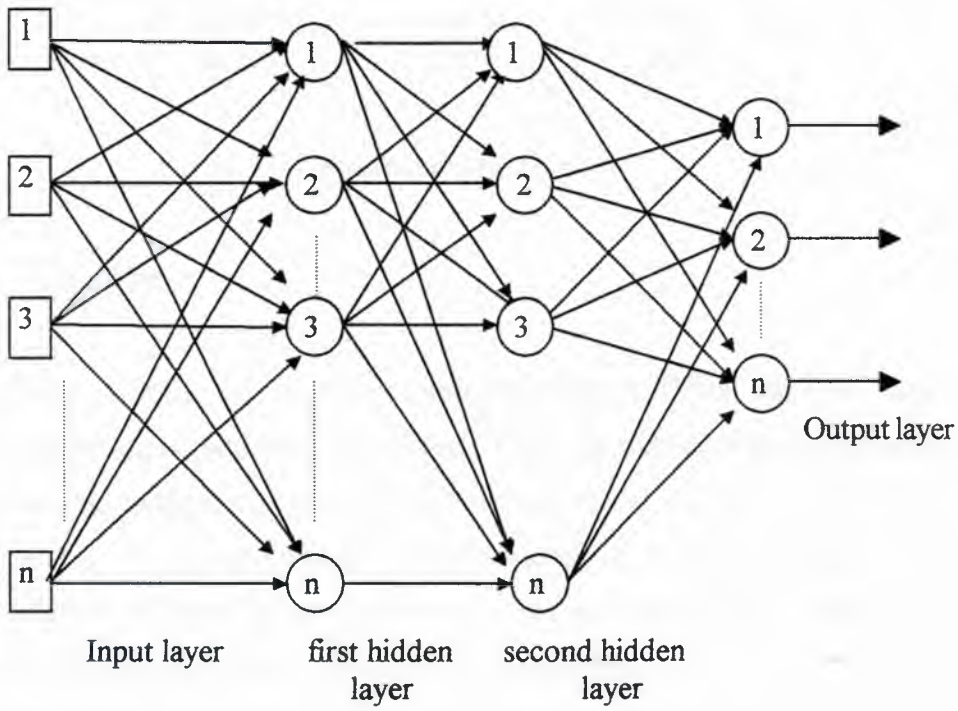


Figure 3.8 Architectural graph of a feed-forward neural network with two hidden layers.

Neural nets are often classified as single layer or multilayer. In determining the number of layers, the input units are not counted as a layer, because they perform no computation. Equivalently, the number of layers in the net can be defined to be the number of layers of weighted interconnect links between the slabs of neurons. This view is motivated by the fact that the weights in a net contain extremely important information. The net shown in Figure 2.8 has three layers of weights. The multilayer net illustrated in Figures 2.8 is example of *feedforward* net-net in which the signals flow from the input units to the output units, in a forward direction.

The fully interconnected competitive net in figure 3.9 is an example *of* a *recurrent* net, in which there are closed-loop signal paths from a unit back to itself.

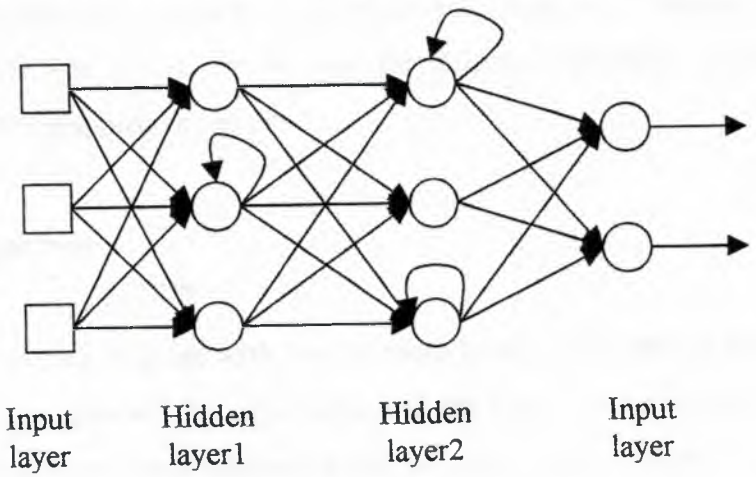


Figure 3.9 Recurrent neural network

3.3.1. Single-Layer Net

A single-layer net has one layer of connection weights. Often, the units can be distinguished as input units, which receive signals from the outside world, and output units, from which the response of the net can be read. In the typical single layer net shown in Figure 3.10, the input units are fully connected to output units but are not connected to other input units, and the output units are not connected to other output units. By contrast, the Hopfield net architecture, shown in Figure 3.10, is an example of a single-layer net in which all units function as both input and output units. For pattern classification, each output unit corresponds to a particular category to which an input vector may or may not belong.

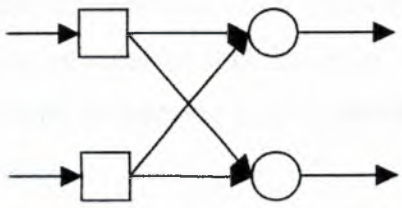


Figure 3.10 a single layered

For a single layer net, the weights for one output unit do not influence the weights for other output units. For pattern association, the same architecture can be used, but now the overall pattern of output signals gives the response pattern associated with the input signal, that caused it to be produced. These two examples illustrate the fact that the same type of net can be used for different problems, depending on the interpretation of the response of the net.

3.3.2. Multilayer Net

A multilayer net is a net with one or more layers (or levels) of nodes (the so - called hidden units) between the input units and the output units. Typically, there is a layer of weights between two adjacent levels of units, (input, hidden, or output). As shown in figure 3.8. Multilayer nets can solve more complicated problems than, can single-layer nets, but training may be more difficult. However, in some cases, training may be more successful, because it is possible to solve a problem that a single-layer net cannot, be trained to perform correctly at all.

3.4. Learning Process

Among the many interesting properties of neural network, the property that is of a primary significance is the ability of the network to learn from its environment, and to improve its performance through learning; the improvement of performance takes place over time in accordance with some prescribed measure.

The neural network learns through an iterative process of adjustments applied to its synaptic weights and thresholds. Ideally, after each iteration of the learning process the network becomes more knowledgeable about the environment.

Learning is a process by which the free parameters of a neural network are updated through a continuing process of stimulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameter changes take place.

This definition of the learning process implies the following sequence of events:

1- The neural network is stimulated by an environment.

2- The neural network undergoes changes as a result of this stimulation.

3- The neural network responds in a new way to the environment, because of the changes that have carried in its internal structure.

To be specific, a pair of node signals x_j and v_k connected by a synaptic weight w_{kj}, are considered as shown in the figure below. Signal x_j represents the output of neuron j, and signal v_k represents the internal activity of neuron k. In the context of synaptic weight w_{kj}, the signals x_j and v_k are commonly referred to as pre-synaptic and postsynaptic activities, respectively. Let $w_{kj}(n)$ denotes the value of the synaptic weight w_{kj} at a time n. at time n an adjustment $\Delta w_{kj}(n)$ is applied to the synaptic weight $w_{kj}(n)$, yielding the updated value $w_{kj}(n+1)$, where

$$w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n) \tag{3.5}$$

Where $w_{kj}(n)$ and $w_{kj}(n+1)$ may be viewed as old and new values of the synaptic weight w_{kj}, respectively.

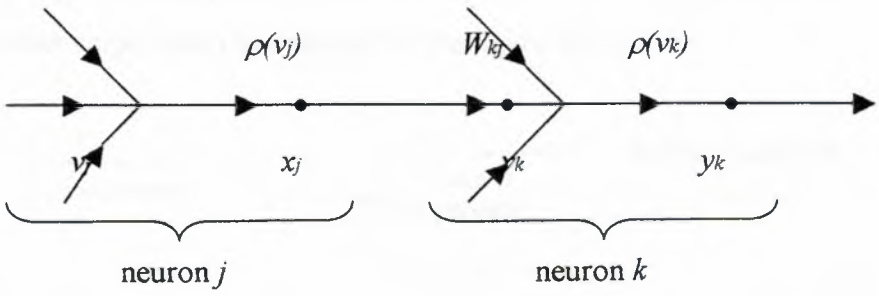


Figure 3.11 Signal-flow graph depicting a pair of neurons j and k embedded in a neural network; both neurons are assumed to have the same activity function $\rho(\bullet)$.

The above mentioned equation sums up the overall effect of events 1 and 2 implicit in the definition of the learning process presented above. In particular, the adjustment $\Delta w_{kj}(n)$ is computed as a result of stimulation by the environment (event 1), and the updated value $w_{kj}(n+1)$ defines the changes made in the network as a result of this stimulation (event 2). Event 3 takes place when the response of the new network, operating with updated set of parameters $\{w_{kj}(n+1)\}$, is revaluated.

A prescribed set of well-defined rules for the solution of a learning problem is called a *learning algorithm* [30].

3.5. Learning Algorithms

There is no unique learning algorithm for the design of the neural networks. Rather, a "kit of tools" are represented by a diverse variety of learning algorithms, each of which efforts advantages of its own. Basically, learning algorithms differ from each other in the way in which the adjustment Δw_{kj} to the synaptic weight w_{kj} is formulated. Another factor to be considered is the manner in which the neural network relates to its environment.

3.5.1. Supervised Learning

An essential ingredient of supervised or active learning is the availability of an external teacher (supervisor) as indicated in the figure below

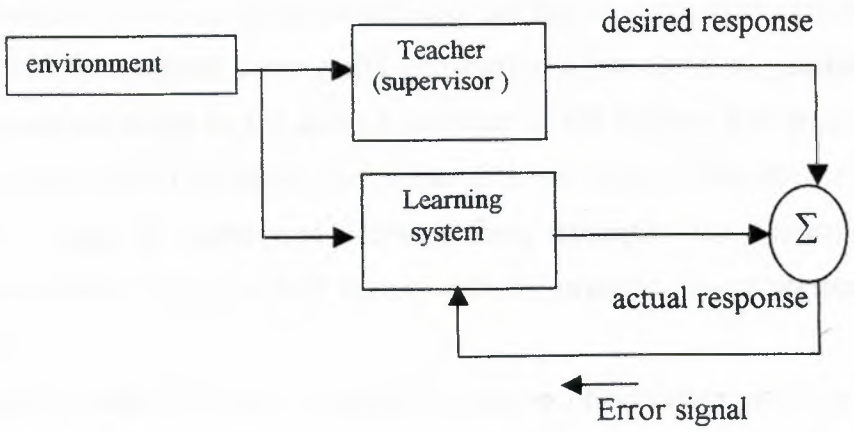


Figure 3.12 Block diagram of supervised learning system

Conceptually, it is thought that the supervisor or the teacher as having knowledge of the environment that is represented by a set of input-output examples. The environment is, however, unknown to the neural network. Suppose that the teacher and the N.N are both exposed to a training vector drown from the environment.

By virtue of built in knowledge, the teacher is able to provide the neural network with a desired or target response for that training vector. Indeed the desired response represents the optimum action to be performed by the N.N. The network parameters are adjusted under the 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. This adjustment is carried out iteratively in a step-by-step fashion with the aim of eventually making the neural network emulate the teacher, the emulation is presumed to be optimal in some statistical sense. In other words, knowledge of the environment available to the teacher is transferred to the neural network as fully as possible. When this condition is reached then the teacher may be dispensed and the neural network deals with the environment completely by it self (i.e. in an unsupervised fashion).

The supervised learning is a closed-loop feedback system. The mean-squared error defined as a function of the free parameters may be used as a performance measure of the system. This function may be visualized as a multidimensional error-performance surface. The true error surface is averaged over all possible input-output examples. Any given operation of the system under the teacher's supervision is represented as a point on the error surface. For the system to improve performance over time and therefore learn from the teacher, the operating point has to move down successively toward a minimum point of the error surface. A supervised learning system is able to do this by virtue of some useful information it has about the gradient of the error surface corresponding to the current behavior of the system. The gradient of an error surface at any point is a vector that points in the direction of the steepest descent. In fact in the case of supervised learning from example; the system uses an instantaneous estimate of the gradient vector, with the example indices presumed to be those of time.

The use of such an estimate results in a motion of operating point on the error surface that is typically in the form of a "random walk." Nevertheless, given an algorithm designed to minimize the cost function of interest, and given an adequate set

of input-output examples and enough time permitted to do the training, a supervised learning system is usually able to perform such tasks as pattern classification and function approximation satisfactorily.

An example of supervised learning algorithms is the back-propagation learning algorithm. Supervised learning algorithm can be performed in an off-line or on-line manner. In the off-line case, a separate computational facility is used to design the supervised learning system. Once the desired performance is accomplished, the design is "frozen", which means that the N.N operates in a static manner.

On the other hand in on-line learning, the learning procedure is implemented solely within the system it self, not requiring a separate computational facility. In other words, learning is accomplished in real time, with the result that the neural network is dynamic.

An advantage of supervised learning, regardless of whether it is performed off-line or on-line, is the fact that without the teacher, a neural network can not learn new strategies for particular situation that are not covered by the set of examples used to train the network. This limitation can be over come, by the use of reinforcement learning.

3.5.2. Unsupervised Learning

In an unsupervised learning or self-organized learning there is no external teacher or critic to oversee the learning process, as indicated in the figure below. In other words, there are no specific examples of the function to be learned by the network. Rather, provision is made for a task-independent measure of the quality of representation that the network is required to learn, and the free parameters of the network are optimized with respect to that measure.

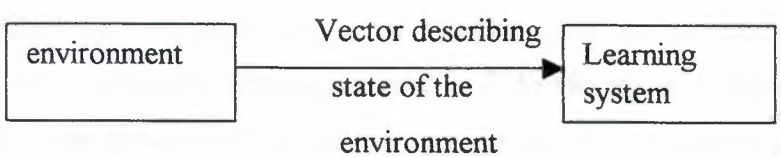


Figure 3.13 Block diagram of unsupervised learning.

Once the network has become tuned to the statistical regularities of the input data, it develops the ability to form internal representations for encoding features of the input by creating new classes automatically [28]. To perform unsupervised leaning, a competitive leaning rule may be used, for example, a N.N that consists of two layers may be used, namely an input layer and a competitive layer. The input layer receives the available data. The competitive layer consists of neurons that compete with each other for the opportunity to respond to features contained in the input data. In its simplest form, the network operates in accordance with a "winner-takes-all" strategy. In such a strategy the neurons with greatest total input wins the competition and turns on; all the other neurons then switch off.

3.5.3. Supervised Verses Unsupervised Learning

Among the algorithm used to perform supervised learning, the back-propagation algorithm has emerged as the most widely used and successful algorithm for the design to multilayer feedforward networks. There are two distinct phases to the operation of back-propagation learning: the forward phase and the backward phase. In the forward phase the input signals propagate through the network layer-by-layer, eventually proceeding some response of the output of the network. The actual response so produced is compared with a desired (target) response, generating error signals that are then propagated in a backward direction through the network. In this backward phase of operation, the free parameters of the network are adjusted so as to minimize the sum of squared errors. Back-propagation learning has been applied successfully to solve some difficult problems such as speech recognition from text, handwriting digit recognition, and adaptive control. Unfortunately, back-propagation and other supervised learning algorithms may be limited by their poor scaling behavior. To understand this limitation, an example of multilayer feedforward network consisting of L computation layer is considered. The effect of a synaptic weight in the first layer on the output of the network depends on its interaction with approximately F_i to the power L other synaptic weights, where F_i is the fan-in, defined as the average number of incoming links of neurons in the network. Hence, as the size of the network increases, the network becomes more computationally intensive, and so the time required to train the network grows exponentially and the learning process becomes unacceptable slow.

One possible solution to the scaling problem described here is to use an unsupervised learning procedure. In particular, if a self-organizing process is able to be apply in a sequential manner, one layer of a time; it is feasible to train deep networks in time that is linear in the number of layers. Moreover, with the ability of the self-organizing network to form internal representations that model the underlying structure of the input data in a more explicit or simple form, it is hoped that the transformed version of the sensory input would be easier to interpret, so that correct responses could be associated with the network's internal representations of the environment more quickly. In other words, the use of supervised learning procedures may provide a more acceptable solution than unsupervised learning alone [29].

3.6. Back-Propagation Learning Algorithm

The most common supervised learning algorithm is the back-propagation (BP) algorithm, which is also called generalized *delta rule*. It is a gradient descent algorithm that is normally used to train the Multilayer Perceptron (MLP) network.

Basically the error back-propagation process consists of two passes through the different layers of the network: a forward pass and a backward pass. In the for*ward pass* an activity pattern (input vector) is applied to the sensory nodes of the network, and its effect propagates layer by layer. Finally a set of outputs is produced as the actual response of the network. During the forward pass the synaptic weights of the network are all fixed. During the backward pass on the other hand the synaptic weights are all adjusted in accordance with the error-correction rule. Specifically, the actual response of the network is subtracted from a desired (target) response to produce an error signal. This error signal is then propagated backward through the network against the direction of synaptic connections, hence the name "error back-propagation". The synaptic weights are adjusted so as to make the actual response of the network move closer to the desired response.

The error signal at the output of neuron j at iteration n (i.e. presentation of the n^{th} training pattern) is defined by [21]

$$e_j(n) = d_j(n) - y_j(n) \qquad \text{neuron } j \text{ is an output node.} \qquad (3.6)$$

The instantaneous value of the squared error for neuron j as $\frac{1}{2}e^2(n)$. Correspondingly, the instantaneous value $\xi(n)$ of the sum of squared errors is obtained by summing

$\frac{1}{2}e^2(n)$ over all neurons in the output layer; these are the only "visible" neurons for which error signals can be calculated. The instantaneous sum of squared errors of the network is thus written as

$$\xi(n) = \frac{1}{2}\sum_{j \in c} e^2(n) \tag{3.7}$$

where the set c includes all the neurons in the output layer of the network. Let N denote the total number of patterns (examples) contained in the training set. The *average squared error* is obtained by summing $\xi(n)$ over all n and then normalizing with respect to the set size N, as shown by

$$\xi(n) = \frac{1}{N}\sum_{n=1} \xi(n) \tag{3.8}$$

The instantaneous sum of error squares $\xi(n)$, and therefore the average squared error ξ_{av}, is a function of all the free parameters (i.e. synaptic weights and thresholds) of the network. For a given training set, ξ_{av} represents the *cost function* as the measure of training set learning performance. The objective of the learning process is to adjust the free parameters of the network so as to minimize ξ_{av}, to do this a simple method of training is considered in which the weights are updated on a pattern-by-pattern basis. The adjustments to the weights are made in accordance with the respective errors computed for each pattern presented to the network. The arithmetic average of these individual weight changes over the training set is therefore an estimate of the true change that would result from modifying the weights based on minimizing the cost function ξ_{av} over the entire training set.

Then figure 3.14 can be considered which depicts neuron j being fed by a set of function signals produced by a layer of neurons to its left. The net internal activity level $v_j(n)$ produced at the input of the non-linearity associated with neuron j is therefore

$$v_j(n) = \sum_{i=0}^{p} w_{ji}(n) y_i(n) \tag{3.9}$$

where p is the total number of inputs (excluding the threshold) applied to neuron j. The synaptic weight w_{j0} (corresponding to the fixed input $y_0 = -1$) equals the threshold θ_j applied to neuron j. Hence the function signal $y_j(n)$ appearing at the output of neuron j at iteration n is

$$y_j(n) = p_j\big(v_j(n)\big) \tag{3.10}$$

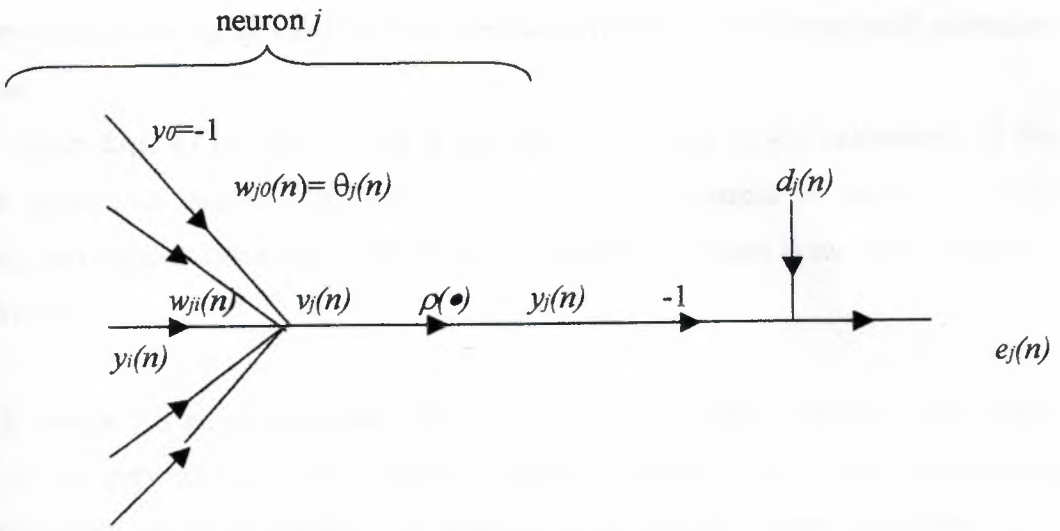


Figure 3.14 Signal-flow graph, highlighting the details of output neuron j.

The back-propagation algorithm applies a correction $\Delta w_{ji}(n)$ to the synaptic weight $w_{ji}(n)$, which is proportional to the instantaneous gradient $\partial\xi(n)/\partial w_{ji}(n)$. According to the chain rule, this gradient can be expressed as follows:

$$\frac{\partial \xi(n)}{\partial w_{ji}(n)} = \frac{\partial \xi(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} \tag{3.11}$$

The gradient $\partial\xi(n)/\partial w_{ji}(n)$ represents a sensitivity factor, determining the direction of search in weight space for the synaptic weight w_{ji}. The correction $\Delta w_{ji}(n)$ applied to $w_{ji}(n)$ is defined by the *delta rule* [21]:

$$W_{ji}(n) = -\eta \frac{\partial \xi(n)}{\partial w_{ji}(n)} \tag{3.12}$$

where η, is a constant that determines the rate of learning; it is called the *learning-rate parameter* of the back-propagation algorithm.

The use of the minus sign in Eq. (3.12) accounts for gradient descent in weight space. The use of Eq. (3.11) in (3.12) yields

$$\Delta w_{ji}(n) = \eta\, \delta_j(n) y_i(n) \tag{3.13}$$

where the *local gradient* $\delta_j(n)$ is defined by [30]

$$\delta_j(n) = e_j(n)\, \rho'(v(n)) \tag{3.14}$$

The local gradient points to required changes in synaptic weights. According to Eq. (3.14), the local gradient $\delta_j(n)$ for output neuron j is equal to the product of the

corresponding error signal $e_j(n)$ and the derivative $\rho'(v(n))$ *of* the associated activation function.

From Eqs. (3.13) and (3.14) a key factor involved in the calculation of the weight adjustment $\Delta w_{ji}(n)$ is the error signal $e_j(n)$ at the output of neuron j. In this context, two distinct cases can be identified, depending on where is neuron j located in the network.

Case I; neuron j is an output node. This case is simple to handle, because each output node of the network is supplied with a desired response of its own, making it a straightforward matter to calculate the associated error signal. Having determined $e_j(n)$, it is a straightforward matter to compute the local gradient $\delta_j(n)$ using Eq. (3.14).

Case II; neuron j is a hidden node. Even though hidden neurons are not directly accessible, they share responsibility for any error made at the output of the network. The question is to know how to penalize or reward hidden neurons for their share of the responsibility. This problem is indeed the *credit-assignment problem* [21]. It is solved in an elegant fashion by back-propagating the error signals through the network. i.e., when neuron j is located in a hidden layer of the network, there is no specified desired response for that neuron. Accordingly, the error signal for a hidden neuron would have to be determined recursively in terms of the error signals of all the neurons to which that hidden neuron is directly connected; this is where the development of the back-propagation algorithm gets complicated. The figure below depicts neuron j as a hidden node of the network. According to Eq. (3.14), the local gradient $\delta_j(n)$ for hidden neuron j may be redefined as

$$\delta_j(n) = -\frac{\partial \xi(n)}{\partial y_j(n)} \rho_j'\left(v_j(n)\right) \qquad \text{neuron } j \text{ is hidden} \qquad (3.15)$$

the following procedure is used to calculate the partial derivative

$$\xi(n) = \tfrac{1}{2} \sum_{k \in C} e_k^2(n) \qquad \text{neuron } k \text{ is an output node} \qquad (3.16)$$

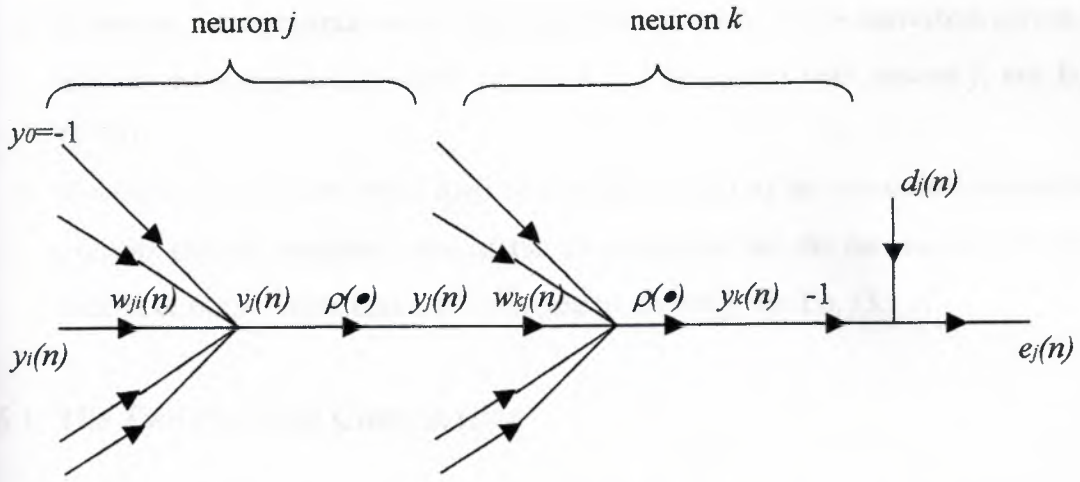


Figure 3.15 Signal-flow graph, highlighting the details of output neuron k connected to hidden neuron j.

Finally, the local gradient $\delta_j(n)$ for hidden neuron j is given by:

$$\delta_j(n) = \rho'\big(v_j(n)\big)\sum \delta_k(n)\, w_{kj}(n) \qquad \text{neuron } j \text{ is hidden} \qquad (3.17)$$

The factor $\rho'_j(v_j(n))$ involved in the computation of the local gradient $\delta_j(n)$ in Eq.(3.17) depends solely on the activation function associated with hidden neuron j. The remaining factor involved in this computation, namely, the summation over k, depends on two sets of terms. The first set of terms, the $\delta_k(n)$, requires knowledge of the error signals $e_k(n)$, for all those neurons that lie in the layer to the immediate right of hidden neuron j, and that are directly connected to neuron j. The second set of terms, the $w_{kj}(n)$, consists of the synaptic weights associated with these connections.

Now the relations that have been derived for the back-propagation algorithm can be summarized.

First, the correction $\Delta w_{ji}(n)$ applied to the synaptic weight connecting neuron i to neuron j is defined by the delta rule [30]:

$$\begin{pmatrix} \text{weight} \\ \text{correction} \\ \Delta w_{ji}(n) \end{pmatrix} = \begin{pmatrix} \text{learning} \\ \text{rate parameter} \\ \eta \end{pmatrix} \bullet \begin{pmatrix} \text{local} \\ \text{gradient} \\ \delta_j(n) \end{pmatrix} \bullet \begin{pmatrix} \text{input signal} \\ \text{of neuron } j \\ y_j(n) \end{pmatrix} \qquad (3.18)$$

Second, the local gradient $\delta_j(n)$ depends on whether neuron j is an output node or a hidden node:

1. If neuron j is an output node, $\delta_j(n)$ equals the product of the derivative $\rho'_j(v_j(n))$ and the error signal $e_j(n)$, both of which are associated with neuron j; see Eq. (3.14).
2. If neuron j is a hidden node, $\delta_j(n)$ equals the product of the associated derivative $\rho'_j(v_j(n)$ and the weighted sum of the δ's computed for the neurons in the next hidden or output layer that are connected to neuron j; see Eq. (3.17).

3.6.1. The Two Passes of Computation

In the application of the back-propagation algorithm, two distinct passes of computation may be distinguished. The first pass is referred to as the forward pass, and the second one as the backward pass.

In the *forward pass* the synaptic weights remain unaltered throughout the network, and the function signals of the network are computed on a neuron-by-neuron basis. Specifically, the function signal appearing at the output of neuron j is computed as [29]

$$y_j(n) = \rho'\left(v_j(n)\right) \tag{3.19}$$

where $v_j(n)$ is the net internal activity level of neuron j, defined by

$$v_j(n) = \sum_{i=0}^{p} w_{ji}(n)\, y_i(n) \tag{3.20}$$

where p is the total number of inputs (excluding the threshold) applied to neuron j, w_{ji} and is the synaptic weight connecting neuron i to neuron j, and $y_i(n)$ is the input signal of neuron j or, equivalently, the function signal appearing at the output of neuron i. If neuron j is in the first hidden layer of the network, then the index i refers to the i^{th} input terminal of the network, for which the following equation is written

$$y_i(n) = x_i(n) \tag{3.21}$$

where $x_i(n)$ is the i^{th} element of the input vector (pattern). If, on the other hand, neuron j is in the output layer of the network, the index j refers to the j^{th} output terminal of the network, for which

$$y_j(n) = o_j(n) \tag{3.22}$$

where $o_j(n)$ is the j^{th} element of the output vector (pattern). This output is compared with the desired response $d_j(n)$ obtaining the error signal $e_j(n)$ for the j^{th} output neuron. Thus

the forward phase of computation begins at the first hidden layer by presenting it with the input vector, and terminates at the output layer by computing the error signal for each neuron of this layer.

The backward pass, on the other hand, starts at the output layer by passing the error signals leftward through the network, layer-by-layer, and recursively computing the δ (i.e. the local gradient) for each neuron. This recursive process permits the synaptic weights of the network to undergo changes in accordance with the delta rule. For a neuron located in the output layer, δ is simply equal to the error signal of that neuron multiplied by the first derivative of its non-linearity. Hence Eq.(3.18) was used to compute the changes to the weights of all the connections feeding into the output layer. Given the δ's for the neurons of the output layer, next Eq.(3.17) is used to compute the δ's for all the neurons in the penultimate layer and therefore the changes to the weights of all connections feeding into it. The recursive computation is continued, layer by layer, by propagating the changes to all synaptic weights made. Note that for the presentation of each training example, the input pattern is fixed throughout the round-trip process, encompassing the forward pass followed by the backward pass.

3.6.2. Rate Of Learning

The back-propagation algorithm provides an approximation to the trajectory in weight space computed by the method of steepest descent. The smaller we make the learning-rate parameter η, the smaller will the changes to the synaptic weights in the network be from one iteration to the next and the smoother will be the trajectory in weight space. This improvement, however, is attained at the cost of a slower rate of learning. If, on the other hand, the learning-rate parameter η is made too large so as to speed up the rate of learning, the resulting large changes in the synaptic weights assume such a form that the network may become unstable (i.e. oscillatory). A simple method of increasing the rate of learning and yet avoiding the danger of instability is to modify the delta rule of Eq.(3.13) by including a *momentum* term [29], as shown by

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_j(n) \tag{3.23}$$

where α is usually a positive number called the *momentum constant*. It controls the feedback loop acting around $\Delta w_{ji}(n)$. In deriving the back-propagation algorithm, it was

assumed that the learning-rate parameter is a constant denoted by η. In reality, it should be defined as η_{ji}; that is, the learning-rate parameter should be *connection-dependent.*

It is also noteworthy that in the application of the back-propagation algorithm all the synaptic weights in the network may be chosen to be adjustable, or any number of weights in the network may be constrained to remain fixed during the adaptation process. In the latter case, the error signals are back propagated through the network in the usual manner; however, the fixed synaptic weights are left unaltered. This can be done simply by making the learning-rate parameter η_{ji} for synaptic weight w_{ji} equal to zero.

Another point of interest is the manner in which the various layers of the back-propagation network are interconnected. In the development of the back-propagation algorithm presented here, the neurons in each layer of the network receive their inputs from other units in the previous layer. In fact, there is no reason why a neuron in a certain layer may not receive inputs from other units in earlier layers of the network. In handling such a neuron, there are two kinds of error signals to be considered:

1. An error signal that results from the direct comparison of the output signal of that neuron with a desired response;
2. An error signal that is passed through the other units whose activating it affects. In this situation, the correct procedure to deal with the network is simply to add the changes in synaptic weights dictated by the direct comparison to those propagated back from the other units.

In a practical application of the back-propagation algorithm, one complete presentation of the entire training set during the learning process is called an *epoch* [30].

The learning process is maintained on an epoch-by-epoch basis until the synaptic weights and threshold levels of the network stabilize and the average squared error over the entire training set converges to some minimum value. It is good practice to randomize the order of presentation of training examples from one epoch to the next. This randomization tends to make the search in weight space stochastic over the learning cycles, thus avoiding the possibility of limit cycles in the evolution of the synaptic weight vectors. For a given training set, back-propagation learning may thus proceed in one of two basic ways; pattern mode or batch mode.

3.6.3. Stopping Criteria And Convergence

The back-propagation algorithm, in general, has some reasonable criteria, each with its own practical merit, which may be used to terminate the weight adjustments. To formulate such a criterion, the logical thing to do is to think in terms of the unique properties of a local or global minimum of the error surface. Let the weight vector w^* denote a minimum, be it local or global. A necessary condition for w^* to be a minimum is that the gradient vector $g(w)$ (i.e. first-order partial derivative) of the error surface with respect to the weight vector w be zero at $w = w^*$. Accordingly, a sensible convergence criterion for back-propagation learning may be formulated as follow [29]:

- The back-propagation algorithm is considered to have converged when the Euclidean norm of the gradient vector reaches a sufficiently small gradient threshold.

The drawback of this convergence criterion is that, for successful trials, learning times may be long. Also, it requires the computation of the gradient vector $g(w)$.
Another unique property of a minimum that can be used is the fact that the cost function or error measure $\xi_{av}(w)$ is stationary at the point $w = w^*$. Therefore, a different criterion of convergence can be suggested:

- The back-propagation algorithm is considered to have converged when the absolute rate of change in the average squared error per epoch is sufficiently small.

Typically, the rate of change in the average squared error is considered to be small enough if it lies in the range of 0.1 to 1 percent per epoch; sometimes, a value as small as 0.01 percent per epoch is used.

A variation of this second criterion for convergence of the algorithm is to require that the maximum value of the average squared error $\xi_{av}(w)$ be equal to or less than a sufficiently small threshold. Kramer and Sangiovanni-Vincentelli suggest a hybrid criterion of convergence consisting of this latter threshold and a gradient threshold, as stated here[20]:

- The back-propagation algorithm is terminated at the weight vector w_{final} when $\| g(w_{final}) \| \leq \varepsilon$, where ε is a sufficiently small gradient threshold, or $\xi_{av}(w_{final}) \leq \tau$ where τ is a sufficiently small error energy threshold.

Another useful criterion for convergence is as follows:

- After each learning iteration, the network is tested for its generalization performance. The learning process is stopped when the generalization performance is adequate, or when it is apparent that the generalization performance has peaked.

The back-propagation algorithm is first-order approximation of the steepest-descent technique in the sense that it depends on the gradient of instantaneous error surface in weight space. The algorithm is therefore stochastic in nature. Indeed, the back-propagation algorithm is an application of a statistical method known as *stochastic approximation* [30]. Consequently, it suffer from a slow convergence property, where two fundamental causes of this property can be defined:

1. The error surface is fairly flat along the weight dimension, which means that the derivative of the error surface with respect to that weight is small in magnitude. In such a situation, the adjustment applied to the weight is small and many iteration for the algorithm may be required to produce a significance reduction in the error performance of the network. The error surface is highly curved along a weight dimension, where the derivative of the error surface with respect to the weight is large in magnitude. In this second situation, the adjustment applied to the weight is large, which may cause the algorithm to overshoot the minimum of the error surface.

2. The direction of the negative gradient vector may point away from the minimum of the error surface; hence the adjustment applied to the algorithm may induce the algorithm to move in the wrong direction.

3.6.4. Disadvantages Of Back-Propagation Learning

Consequently, the rate of convergence in back-propagation algorithm tends to be relatively slow, which in turn makes it computationally expensive. The local convergence rate of the back-propagation algorithm is linear, which justify the rank-deficient in the Jacobian and Hessian matrices; these are the consequences of the intrinsically ill-conditioned nature of neural network training problems. Saarinen

interpret the linear local convergence rates of the back-propagation learning on two ways:

1. It is a vindication of back-propagation in the sense that higher order methods may not converge much faster while requiring more computational effort.
2. Large-scale neural network training problems are so inherently difficult to perform that no supervised learning strategy is feasible, and other approaches such as the use of processing may be necessary.

Another peculiarity of the error surface that impacts the performance of back-propagation algorithm is the presence of local minima in addition of global minima [9]. Since the back-propagation is a hill-climbing technique, it runs the risk of being trapped in a local minimum. It is undesirable to have the learning process terminate at a local minimum, especially if it is located far above a global minimum. But stacking in a local minimum is rarely a practical problem for back-propagation learning.

Another problem of learning by back-propagation that has to be overcome is scaling, which addresses the issue of how well the network behaves as the computational task increase in size and complexity. One way of alleviating the scaling problem is to develop insight into the problem at hand and use it to put ingenuity into the architectural design. Another way to deal with scaling problem is to reformulate the back-propagation learning process with modularity built into the network architecture. A computational system is said to have modular architecture if it can be broken down into two or more subsystems that perform computation on distinct input in the absence of communication with each other.

3.6.5. Advantages Of Back-Propagation Learning

The back-propagation algorithm is an example of a connectionist paradigm that relies on local computations to discover the information-processing capabilities of neural networks. This form of computational restriction is referred to as the locality constraint, in this sense the computation performed by a neuron is influenced solely by those neurons that are in physical contact with it. The use of local computations in the design of artificial neural networks is usually advocated for three principal reasons:

1. Artificial neural networks that perform local computations are often held up as metaphors for biological neural networks
2. The use of local computations permits a graceful degradation in performance due to hardware errors, and therefore a fault-tolerant network design.
3. Local computations favour the use of parallel architectures as an efficient method for the implementation of artificial neural networks.

Taking a lock at these points it is obvious that, the third point is perfectly justified in the case of back-propagation learning, where it has successfully implemented on parallel computers by many investigators, and VLSI architecture has been for the hardware realization of multilayer perceptrons. The second point is justified so as certain precaution are taken in the application of back-propagation algorithm, such as injecting small numbers of "transient faults" at each step. For the first point, relating to the biological plausibility of back-propagation learning, it has indeed questioned as follow:

1. The reciprocal synaptic connections between neurons of a multilayer perceptron may assume weights that are excitatory or inhibitory.
2. In a multilayer perceptron, hormonal and other types of global communication are ignored.
3. In back propagation learning, a synaptic weight is modified by a pre-synaptic activity and an error signal independent of postsynaptic activity.
4. The implementation of back-propagation learning requires the rapid transmission of information backward along the axon.
5. Back-propagation learning implies the existence of a "teacher," which in the context of the brain would presumably be another set of neurons with novel properties.

3.6.6. Accelerating Convergence Of Back-Propagation Through Learning Rate Adaptation

The procedures for increasing the rate of convergence while maintaining the locality constrains that is inherent characteristic of back-propagation learning are described here:

1. Every adjustable network parameter should have its own individual learning rate parameter.
2. Every learning rate parameter should be allowed to vary from one iteration to the next.

- When the derivative of the cost function with respect to the synaptic weight has the same algebraic sign for several consecutive iterations of the algorithm, the learning parameter for that particular weight should be increased.
- When the algebraic sign of the derivative of the cost function with respect to particular synaptic weight alternates for several consecutive iterations of the algorithm, the learning rate for that weight should be decreased.

3.7. Summary

In summary of this chapter we can say that neural networks consist of set of neurons or processing units, and are suitable for many tasks. NN can modify its behavior in response to their environments. In other words they are capable of learning, and once they are trained they can make their decisions. The learning can be supervised or unsupervised. In supervised learning input-output vectors are presented to the network, whereas in unsupervised learning the network detects similarities between inputs and group them accordingly. This chapter has presented an over view of NN, well known NN architecture, their operational principles, and their learning algorithms such as Back-propagation which is the most commonly used, its advantages and disadvantages are listed. Different mathematical models of neurons, neural networks structures are considered.

Chapter 4: NEURAL NETWORK-BASED FUZZY INFERENCE SYSTEM

4.1. Overview

In this chapter the power of learning and parallelism of neural network is combined with the reasoning mechanism of fuzzy logic to build up a more powerful system referred in the literature as neuro-fuzzy system. So here we will take about the combination models, learning mechanism, hybrid systems architecture, then the discussion is expanded to cover ANFIS, its learning algorithm, and its structure for the different types of fuzzy, i.e. Mamadani and Takagi-Sugeno types of fuzzy.

4.2. Models of Neuro-Fuzzy System

There are many ways to synthesize neural networks and fuzzy logic. Neural networks provide algorithms for numeric classification, optimization, and associative storage and recall. Working at the semantic level, fuzzy logic provides a tool to process inexact or approximate data. By incorporating fuzzy logic techniques into a neural network, we can obtain more flexibility. Fuzzy neural networks provide greater representation power, have higher processing speeds, and are more robust than conventional neural networks. Fuzzy neural networks are in fact "fuzzified" neural networks.

4.2.1. Input-Output Approach

The first approach is to use input-output signals or weights in neural networks as fuzzy sets along with fuzzy neurons. Several authors have proposed models for Neuro-Fuzzy. Gupta presented two possible models for fuzzy neural systems [22]. The first model, shown in Figure 4.1, consists of a fuzzy inference block followed by a neural network block. The neural network block represents a multilayer feed-forward neural network. The fuzzy inference block provides input to the neural network. The neural network can be adapted to yield desired outputs or decisions. The model can be trained

with the training samples.

In the second model, shown in Figure 4.2, the neural network block drives the fuzzy inference system. The neural network can be adapted to produce desired outputs or decisions. The first model takes linguistic inputs, whereas the second model takes numeric inputs. The computational process for these models includes the development of fuzzy neural models motivated by biological neurons, models of synaptic connections, which incorporate fuzziness into a neural network, and learning algorithms for these models.

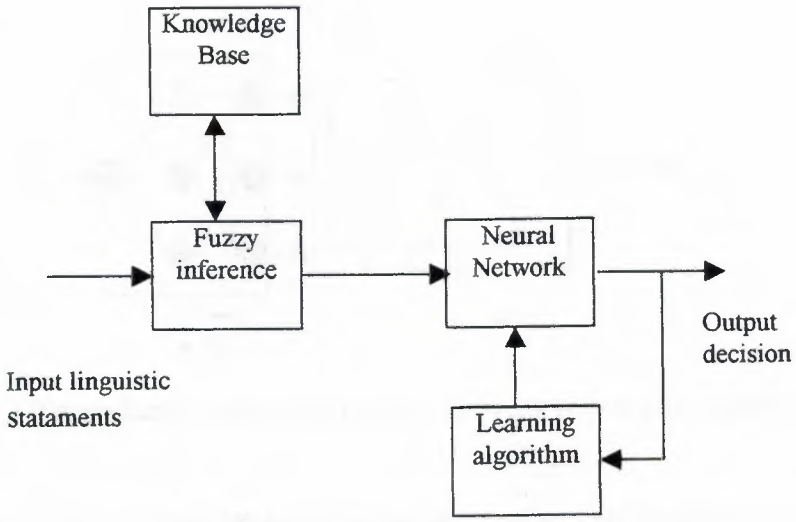


Figure 4.1 Input/Output model of Neuro-Fuzzy System

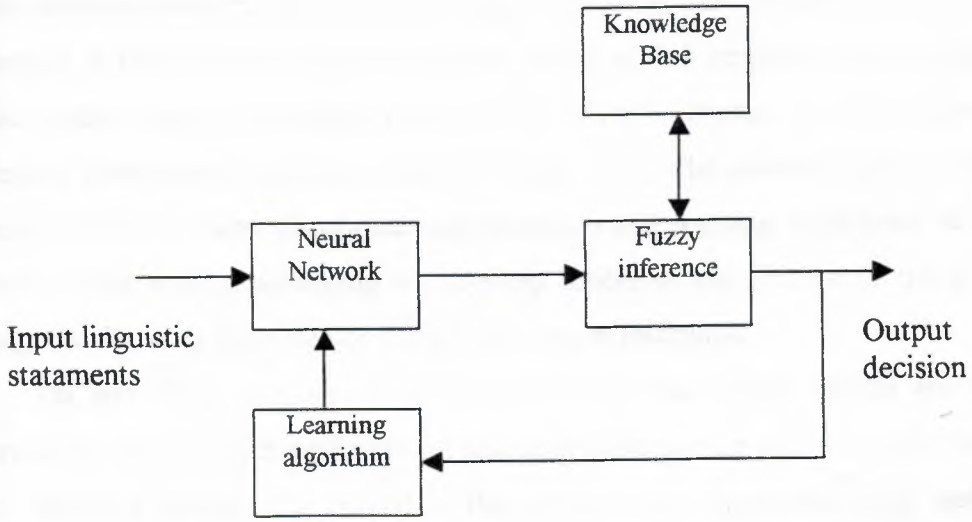


Figure 4.2 Input/Output Model of Neuro-fuzzy System

4.2.2. Preprocess/postprocess Approach

The second approach is to use fuzzy membership functions to preprocess or postprocess signals with neural networks. Figures 4.3 shows these models. A fuzzy inference system can encode an expert's knowledge directly and easily using rules with linguistic labels. It usually takes a lot of time to design and tune membership functions, which quantitatively define linguistic labels. Neural network learning techniques can automate this process and subsequently reduce the development time and cost, at the same time improving performance.

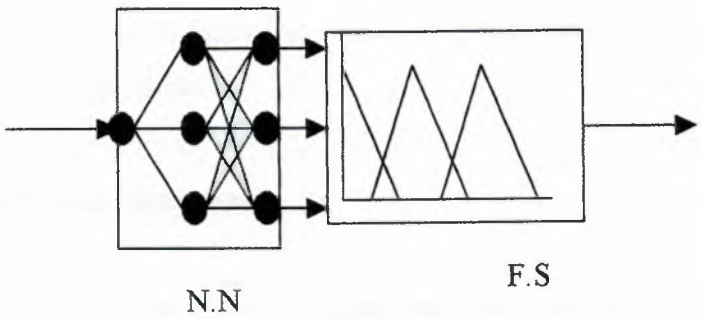


Figure 4.3 Neuro fuzzy system with neural network in pre-processing role.

The idea of using neural networks to design membership functions was proposed by Takagi and Hayashi [6], and it involved the design of multidimensional membership functions. Many models with this approach have been used. Lin and Lee proposed a neural-network-based model for fuzzy logic control/decision systems [12]. The model represents a feed-forward neural network. Input nodes represent input signals and output nodes represent output decisions or signals. Nodes in the hidden layers implement membership functions and fuzzy logic rules. The system is a fuzzy inference system; however, it uses distributed representation and learning algorithms of a neural network. Parameters representing membership functions are determined using a back-propagation learning algorithm or a gradient-descent technique.

Pal and Mitra proposed a similar model. In their model, inputs are fed to a preprocessor block, which performs the same functions as that of the fuzzifier block in a fuzzy inference system. The output of the preprocessor represents fuzzy membership values. For each input variable term, variables such as *low*, *medium*, and *high* are used. If input consists of n variables, then the preprocessor block yields m x n outputs, where

m represents the number of term values used in the model. The output of the preprocessor block is then fed to a multilayer perceptron model. The perceptron model implements the inference engine. Pal and Mitra have used the model for classifying vowels [16]. Kulkarni has developed a similar model and has used it for multispectral image analysis [16].

4.2.3. Fuzzy Associative Memories (FAMs)

In a simple implementation, a Fuzzy Associative Memory (FAM) is a fuzzy logic rule with an associated weight. A mathematical framework exists that can map a FAM to a neural network; a fuzzy logic decision system can then be built using a back-propagation learning algorithm.

4.2.4. Hybrid System Approach

Yet another way to combine neural networks with fuzzy logic is to design a hybrid system wherein some processing stages are implemented with neural networks and some with a fuzzy inference system [24]. An example of such a system would be a tree classifier in which classification at some node can be carried out with a fuzzy inference system and classification at some other node can be performed using a neural network. The main advantage of such a hybrid system is that when the classification is based on experts' rules we can use the fuzzy inference system, and when the classification is based on training samples we can use a neural network.

Hybrid systems are defined in many different ways. In a simple way, hybrid systems are those composed by more than one intelligent system. Hybrid systems are expected to be more powerful due to the combining advantages of different intelligent techniques. Two or more intelligent systems can be combined to create a unique hybrid system. The most popular hybrid systems are:

- Sequential hybrid system.
- Auxiliary hybrid system.
- Incorporated hybrid system.

4.2.4.1. Sequential hybrid system

This model represents the weakest degree of integration and it is composed of two intelligent systems connected in serial (figure 4.4). One example of this type of system may be a pre-processor Fuzzy System activating a Neural Net

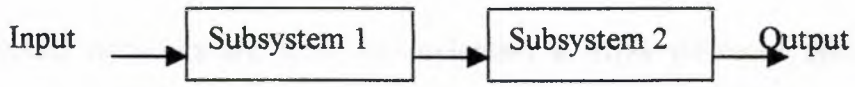


Figure 4.4 Sequential Hybrid System

4.2.4.2. Auxiliary hybrid system

This model is composed of a sub-system added by another intelligent sub-system. The integration degree is greater than in the previous case (figure 4.5). An example of this kind of systems is a Genetic Algorithm used to determine the weights of a Neural Network.

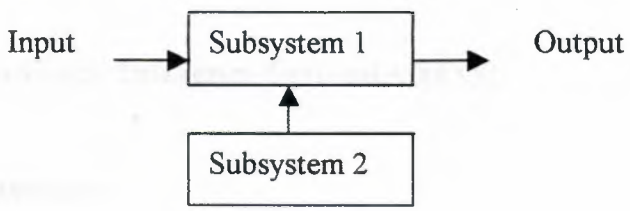


Figure 4.5 Auxiliary Hybrid System

4.2.4.3. Incorporated hybrid system

Incorporated hybrid systems represent the greatest degree of integration (figure 4.6). There is no possible differentiation between the different intelligent systems. We can say that the first system contains the second one or vice-versa. An example is a Neuro-Fuzzy system, where a Fuzzy inference system is implemented using a Neural Network Structure.

Among the most popular hybrid models are the Neuro-Fuzzy systems, Neuro-Genetic systems, Neural-Statistic systems and Fuzzy-Genetic systems.

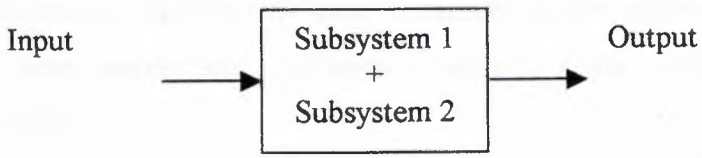


Figure 4.6 Incorporated Hybrid System

Neural networks are used to implement a fuzzy inference system. A fuzzy inference system consists of three components. Firstly, a rule base contains a selection of fuzzy rules. Secondly, a database defines the membership functions used in the rules and, finally, a reasoning mechanism carries out the inference procedure on the rules and given facts.

Jang and Sun[11] presented an adaptive network model for a fuzzy inference system. The model represents adaptive network-based fuzzy inference systems (ANFISs). The ANFIS model is a generic model, and neural networks and fuzzy inference systems can be considered as special instances of an adaptive network when proper node functions are assigned.

4.3. Adaptive Neuro-Fuzzy Inference System(ANFIS)

4.3.1. ANFIS Architecture

In this section, a class of adaptive networks which are functionally equivalent to fuzzy inference systems is proposed. The proposed architecture is referred to as ANFIS, standing for *Adaptive-Network-Based Fuzzy Inference System*. It can use as back-propagation as the hybrid learning rule.

The ANFIS can construct input-output mapping based on both human knowledge (in the form of fuzzy if-then rules) and stipulated input-output data pairs. The ANFIS architecture is employed to modal nonlinear functions, identify nonlinear components on-linely i.e., a control system, and predict a chaotic time series [7].

When the design of fuzzy controller is considered, the main inconvenient of Fuzzy Controller Systems is to determine the rule base, due to the complexity of the plant. Sometimes it is also very difficult to create an appropriate rule base when using

the aid of an expert-man. The Neuro-Fuzzy Controller system offers the possibility to create this rule base automatically through a learning phase, evaluating the error response of the system.

For simplicity, assume the fuzzy inference system under consideration has two inputs x and y and one output z. suppose that the rule base contains two fuzzy if-then rules of Takagi and Sugeno's type [11].

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

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

Then type-3 fuzzy reasoning is illustrated in figure 4.7.(a), and the corresponding equivalent ANFIS architecture (type-3 ANFIS) is shown in figure 4.7.(b). The node functions in the same layer are of the same function family as described below:

Layer 1: every lump i in this layer is a square node with a node function

$$O_i^1 = \mu A_i(x) \tag{4.1}$$

Where x is the input to node i, and A_i is the linguistic label (small, large, etc.) associated with this node function. In other words, O_i^1 is the membership function of A_i and it specifies the degree to which the given x satisfies the quantifier A_i. Usually we choose $\mu A_i(x)$ to be bell-shaped with maximum equal to 1 and minimum equal to 0, such as

$$\mu A_i(x) = \frac{1}{1+(\frac{x-c_i}{a_i})^2 b_i} \tag{4.2}$$

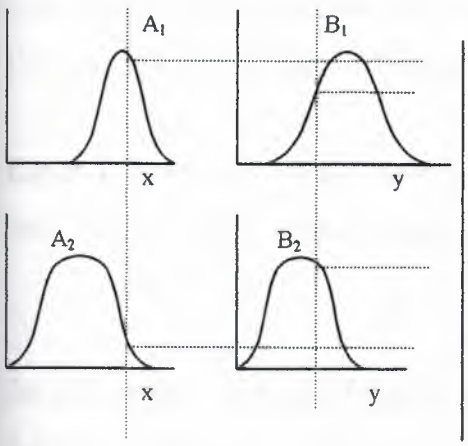
or

$$\mu A_i(x) = \exp[-(\frac{x-c_i}{a_i})^2] \tag{4.3}$$

where $\{a_i, b_i, c_i\}$ is the parameter set. As the values of these parameters change, the bell-shaped functions vary accordingly, thus exhibiting various forms of membership functions on linguistic label A_i. In fact, any continuous and piecewise differentiable functions, such as commonly used trapezoidal or triangular-shaped membership functions, are also qualified candidates from node functions in this layer. Parameters in this layer are referred to as premise parameters.

Layer 2: every node in this layer a circle node labeled II which multiplies the incoming signals and sends the product out. For instance,

$$w_i = \mu a_i(x) * \mu b_i(y), \quad i = 1,2 \tag{4.4}$$



w_1

$$f_1 = p_1 x + q_1 y + r_1$$

$$\Rightarrow \quad f = \frac{w_1 f_1 + w_2 f_2}{w_1 + w_2} = \hat{w}_1 f_1 + \hat{w}_2 f_2$$

w_2

$$f_2 = p_2 x + q_2 y + r_2$$

Figure 4.7.(a) type 3 Fuzzy reasoning

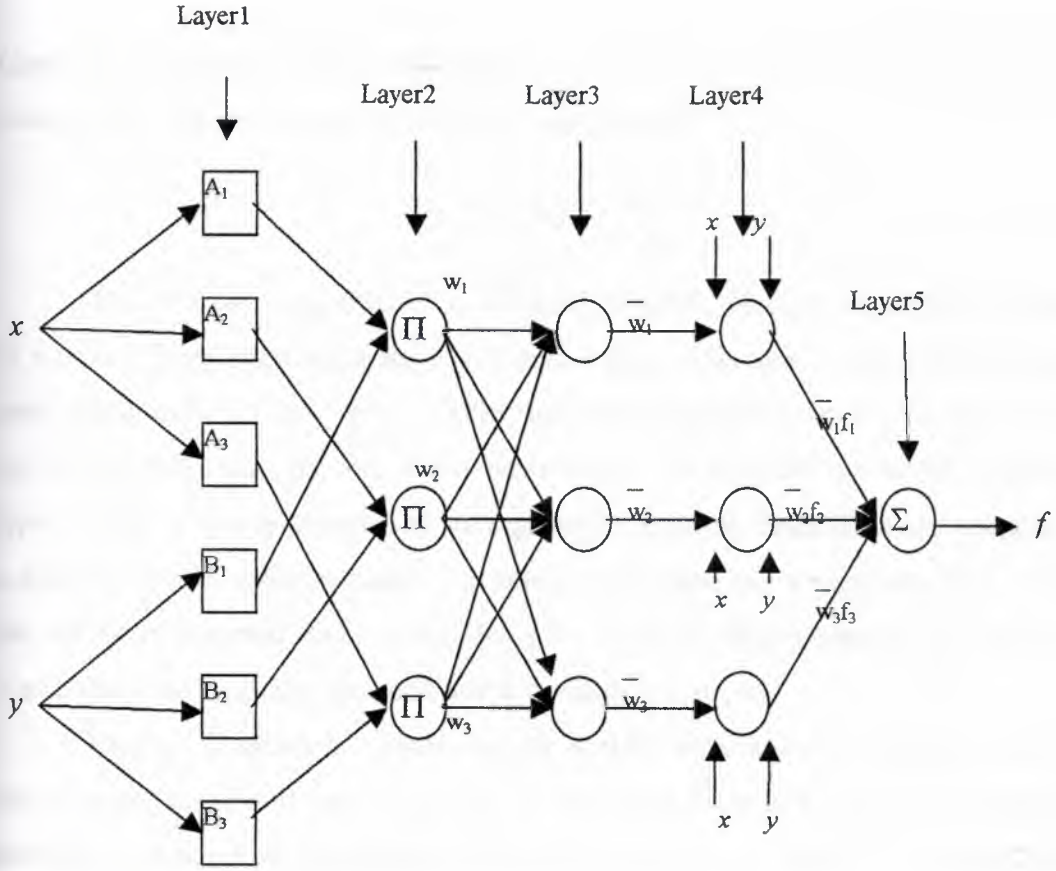


Figure.4.7. (b) Equivalent ANFIS (type 3)

Each node output represents the firing strength rule. (in fact, other T-norm operators that parameter generalized AND can be used as the node function in this layer).

Layer 3: every node in this layer is a circle node labelled *N*. the i^{th} node calculates the ratio of the i^{th} rule's firing strength to the sum of all rules firing strengths:

$$\overline{w}_i = \frac{w_i}{w_1 + w_2}, \quad i = 1,2 \tag{4.5}$$

for convenience, outputs of this layer will be called *normalized firing strengths*.

Layer 4: Every node i in this layer is a square node with a node function

$$O_i^4 = \overline{w}_i f_i = \overline{w}(p_i x + q_i y + r_i) \tag{4.6}$$

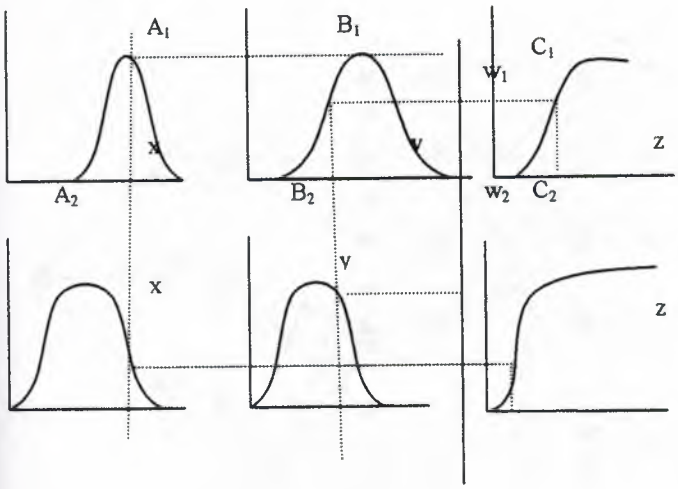
where $\overline{w}_i$ is the output of layer 3, and $\{p_i, q_i, r_i\}$ is the parameter set. Parameters in this layer will be referred to as *consequent parameters*.

Layer 5: The single node in this layer is a circle node labeled Σ that computes the overall output as the summation of all incoming signals, i.e.,

$$O_i^5 = \sum_i \overline{w}_i f_i = \frac{\sum_i w_i f_i}{\sum_i w_i} \tag{4.7}$$

Thus we have constructed an adaptive network which is functionally equivalent to a type-3 fuzzy inference system. For type-1 fuzzy inference systems, the extension is quite straightforward and type-1 ANFIS is shown in figure 4.8, where the output of each rule is induced jointly by the output membership function and the firing strength. For type-2 fuzzy inference systems. If we replace the centroid defuzzification operator with a discrete version which calculates the approximate centroid of area, then type-3 ANFIS can still be constructed accordingly. However, it will be more complicated than its type-3 and type-1 versions and thus not worth the efforts to do so.

Figure 4.9 shows a 2-input, type-3 ANFIS with nine rules. Three membership functions are associated with each input, so the input space is partitioned into nine fuzzy subspaces, each of which governed by a fuzzy subspace, while the consequent part specifies the output within this fuzzy subspace.



$$f = \frac{w_1 f_1 + w_2 f_2}{w_1 + w_2} = \hat{w}_1 f_1 + \hat{w}_2 f_2$$

Figure 4.8 (a) type 1 fuzzy reasoning

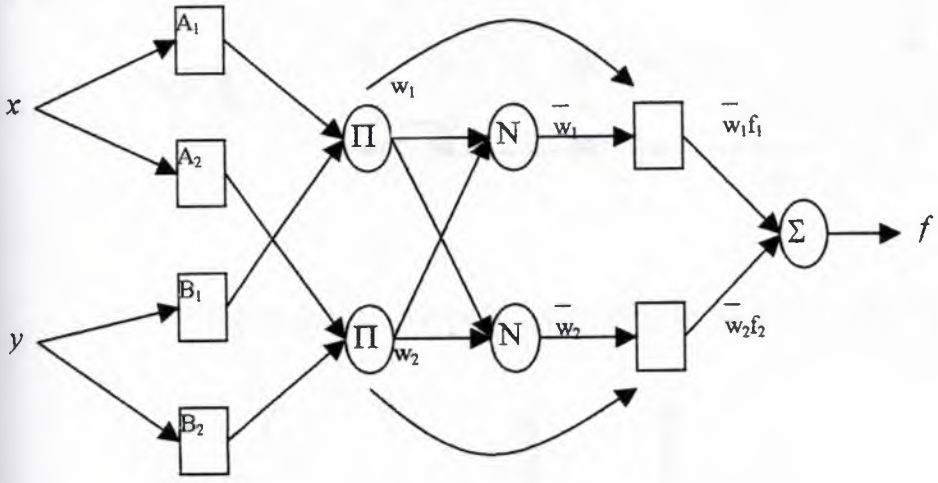


Figure 4.8 (b) Equivalent ANFIS (type 1 ANFIS)

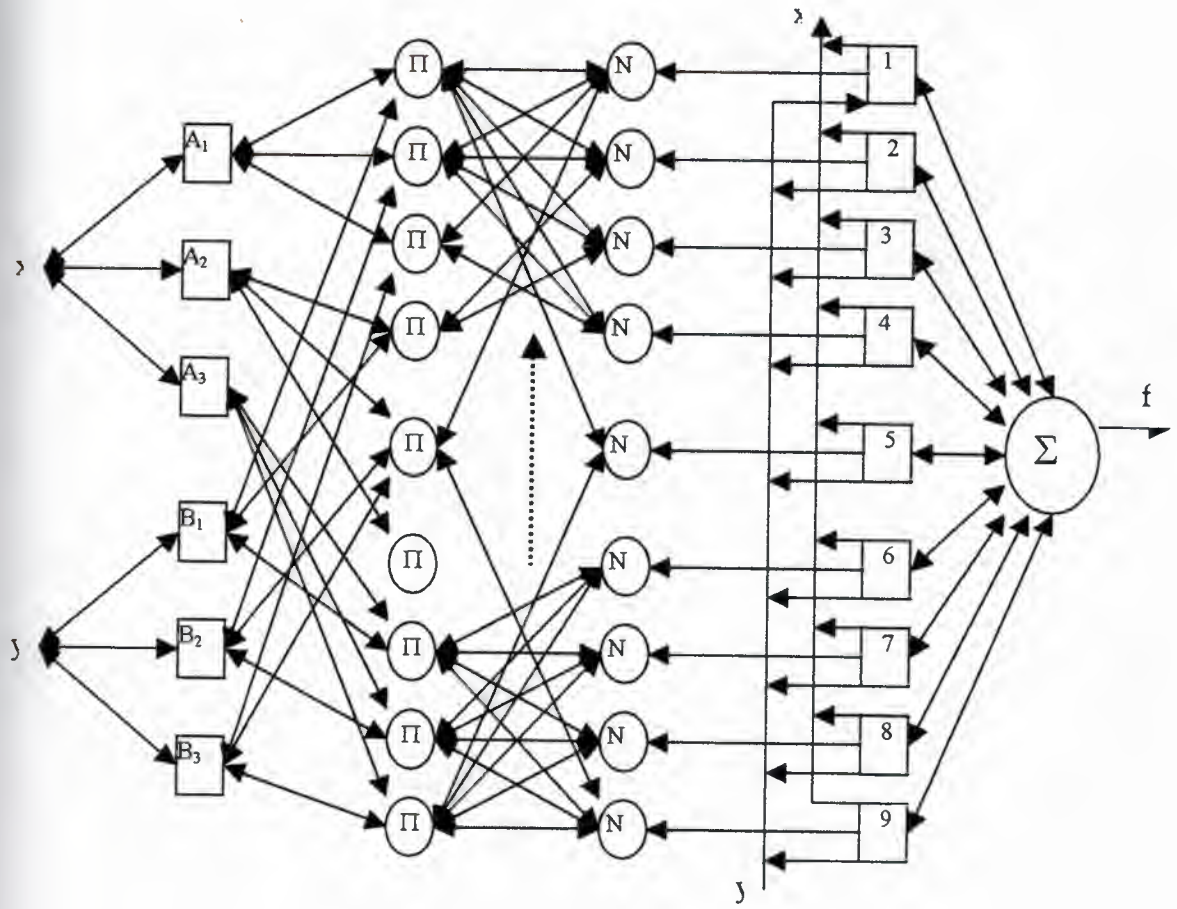


Figure 4.9 (a) two-input type 3 ANFIS with nine rules.

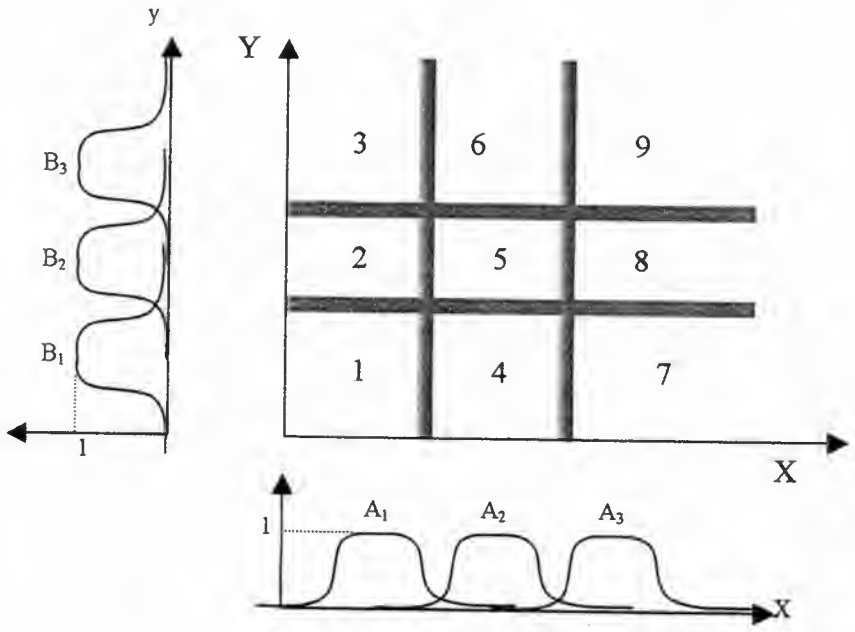


Figure 4.9 (b) corresponding subspaces

4.4 Hybrid Learning Rule: Batch (Off-Line) Learning

Though we can apply the gradient method [25] to identify the parameters in an adaptive network, the method is generally slow and likely to become trapped in local minima. Here a hybrid learning rule is proposed, which combines the gradient method and the least squares estimate (LSE) [23] to identify parameters.

For simplicity, assume that the adaptive network under consideration has only one output

$$output = F(\vec{I}, S) \tag{4.8}$$

Where $\vec{I}$ is the set of input variables, and S is the set of parameters. If there exists a function H such that the compose function H *o* F is linear in some of the elements of S, these elements can be identified by the least squares mean. More formally, if the parameter set S can be decomposed into two sets

$$S = S_1 \oplus S_2 \tag{4.9}$$

(where $\oplus$ represents direct sum) such that H *o* F is linear The elements of S_2, then upon applying H to (4.8), we have

$$H(output) = H \circ F(\vec{I}, S) \tag{4.10}$$

Which is linear in the elements of S_2. Now given values of elements of S_1, we can plug P training data into (4.10) to obtain a matrix equation:

$$AX = B \tag{4.11}$$

Where X is unknown vector whose elements are parameters in S_2. Let $|S_2| = M$, then the dimensions of A, X and B are $P*M$, $M*1$, and $P*1$, respectively. Since P (the number of training data pairs) is usually greater than M(number of linear parameters), this is an over determined problem [8] generally there is no exact solution to (4.11). Instead, least squares estimate (LSE) of X, X^*, is sought to minimum least squared error $||AX - B||^2$. This is a standard problem that forms the grounds for linear regression, adaptive filtering and signal processing. The most well-known formula for X^* uses the pseudo-inverse of X [20]:

$$X^* = (A^T A)^{-1} A^T B \tag{4.12}$$

Where A^T is the transpose of A, and $(A^T A)^{-1} A^T$ is the pseudo-inverse of A if $A^T A$ is non singular. While (4.12) is concise in notation, its expensive in computation when dealing

with the matrix inverse and, moreover, it becomes ill-defined if $A^T A$ is singular. As a result, sequential formulas are employed to compute the LSE of X. This sequential method of LSE is more efficient (especially when M is small) and can be easily modified to an on-line version for systems with changing characteristics. Specifically, let the i^{th} row vector of matrix A defined in (4.11) be a_i^T, and the i^{th} elements of B be b_i^T, then X can be calculated iteratively using the sequential formulas widely adopted in the literature[6],

$$\begin{aligned} X_{i+1} &= X_i + S_{i+1} a_{i+1} (b_{i+1}^T - a_{i+1}^T Xi) \\ S_{i+1} &= S_i - \frac{S_1 a_{i+1} a_{i+1}^T S_i}{1 + a_{i+1}^T S_i a_{i+1}}, \qquad i = 0,1,.....,P\text{-}1 \end{aligned} \tag{4.13}$$

where S_i is often called the covariance matrix and the least squares estimate X^* is equal to X_p. The initial conditions to bootstrap (4.13) are $X_0 = 0$ and $S_0 = \gamma I$, where γ is a positive large number and I is the identity matrix of dimension $M \times M$. When dealing with multi-output adaptive networks (output in (4.8) is column vector), (4.13) still applies except that b_i^T is the i^{th} rows of matrix B.

Now the gradient method and the least squares estimate can be combined to update the parameters in an adaptive network. Each epoch of this hybrid learning procedure is composed of a forward pass and a backward pass. in the forward pass, we supply input data and functional signals go forward to calculate each node output until the matrices A and B in (4.11) are obtained, and the parameters in S_2 are identified by the sequential least squares formulas in(4.13). After identifying parameters in S_2, the functional signals keep going forward till the error measure is calculated. In the backward pass, the error rates (the derivative of the error measure with respect to each node output, -as explained in chapter3- propagate from the output end toward the input end, and the parameters in S_1 are updated by the gradient method using the following equation [29]:

$$\Delta\alpha = -\eta \frac{\partial E}{\partial \alpha} \tag{4.14}$$

For given fixed values of parameters in S_1, the parameters in S_2 thus found are guaranteed to be the global optimum point in the S_2 parameter space due to the choice of the squared error measure. Not only can this hybrid learning rule decrease the

dimension of the search space in the gradient method, but, in general, it will also cut down substantially the convergence time.

Take for example an one-hidden-layer back-propagation neural network with sigmoid activation functions. If this neural network has p output units, then the output in (4.8) is a column vector. Let H(·) be the inverse sigmoid function

$$H(x) = \ln(\frac{x}{1-x}) \tag{4.15}$$

then (4.10) becomes a linear (vector) function such element of *H(output)* is linear combination of the parameters (weights and thresholds) pertaining to layer 2. in other words,

S_1 = weights and thresholds of hidden layer

S_2 = weights and thresholds of output layer.

Therefore we can apply the back-propagation learning rule to tune the parameters in the hidden layer, and the parameters in the output layer can be identified by the least squares method. However, it should be kept in mind that by using the least squares method on the data transformed by H (·), the obtained parameters are optimal in terms of the transformed squared error measure instead of the original one. Usually this will not cause practical problem as long as H(·) is monotonically increasing.

4.5 Hybrid Learning Rule: Pattern (On-Line) Learning

If the parameters are updated after each data presentation, we have the pattern learning or on-line learning paradigm. This learning paradigm is vital to the on-line parameter identification for systems with changing characteristics. To modify the batch learning rule to its on-line version, it is obvious that the gradient descent should be based on E_p instead of E [31].

$$\frac{\partial E_p}{\partial \alpha} = \sum_{o^* \in S} \frac{\partial E_p}{\partial O^*} \frac{\partial O^*}{\partial \alpha} \tag{4.16}$$

strictly speaking, this is not a truly gradient search procedure to minimize E, yet it will approximate to one if learning rate is small.

For the sequential least squares formulas to account for the time-varying characteristics of the incoming data, we need to decay the effects of old data pairs as new data pairs become available. Again, this problem is well studied in the adaptive

control and system identification literature and a number of solutions are available [4]. One simple method is to formulate the squared error measure as a weighted version that gives higher weighting factors to more recent data pairs. The amounts to addition of a *forgetting factor* λ to the original sequential formula [11]:

$$X_{i+1} = X_i + S_{i+1} a_{i+1} (b_{i+1}^T - a_{i+1}^T Xi)$$
$$S_{i+1} = \frac{1}{\lambda}\left[S_i - \frac{S_1 a_{i+1} a_{i+1}^T S_i}{1 + a_{i+1}^T S_i a_{i+1}} \right] \tag{4.17}$$

Where the value of λ is between 0 and 1. The smaller λ is, the faster the effects of old data decay. But a small λ sometimes causes numerical instability and should be avoided.

4.6. Hybrid Learning Algorithm for ANFIS structure

From the proposed type-3 ANFIS architecture (Figure 4.9), it is observed that given the values of premise parameters, the overall output can be expressed as linear combinations of the consequent parameters. More precisely, the output f in figure 4.9. can be rewritten as

$$f = \frac{w_1}{w_1 + w_2} f_1 + \frac{w_2}{w_1 + w_2} f_2$$
$$f = \overline{w_1} f_1 + \overline{w_2} f_2 \tag{4.18}$$
$$f = (\overline{w_1} x) p_1 + (\overline{w_1} y) q_1 + (\overline{w_1}) r_1 + (\overline{w_2} x) p_2 + (\overline{w_2} x) q_2 + (\overline{w_2}) r_2$$

which is linear in the consequent parameters (p_1, q_1, r_1, p_2, q_2 *and* r_2). As a result we have

S = set of total parameters

S_1 = set of premise parameters

S_2 = set of consequent parameters

in (4.9); $H(\cdot)$ and $F(\cdot,\cdot)$ are the identity function and the function of the fuzzy inference system, respectively.

Therefore the hybrid learning algorithm developed previously can be applied directly. More specifically, in the forward pass of the hybrid learning algorithm, functional signals go forward till layer 4 and the consequent parameters are identified by the least squares estimate. In the backward pass, the error rates propagate backward

and the premise parameters are updated by the gradient descent. Table 4.1 summarizes the activities in each pass.

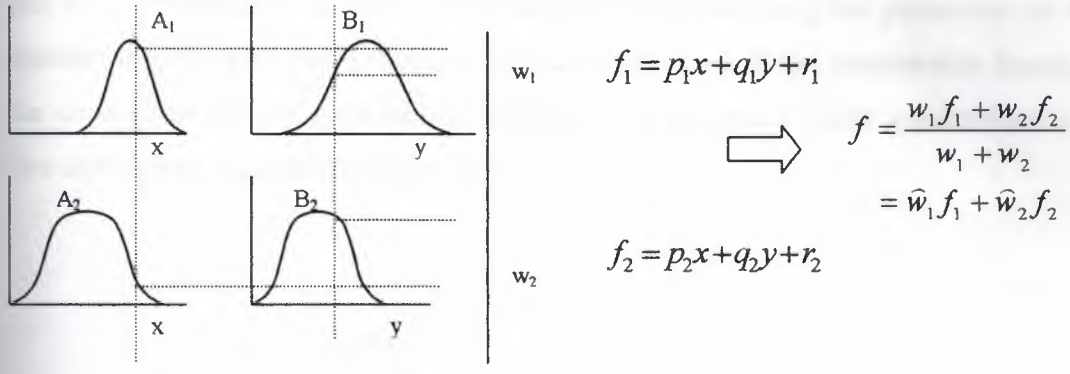


Figure 4.10 (a) type 3 Fuzzy reasoning

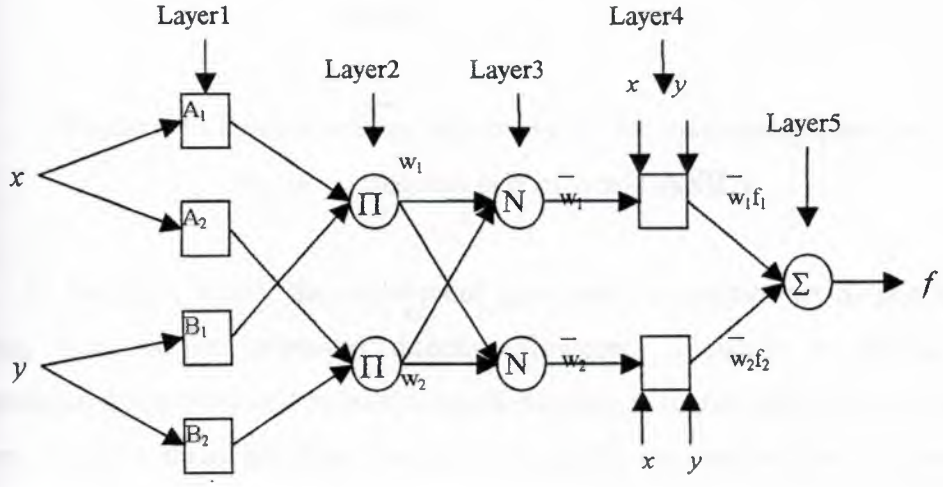


Figure 4.10 (b) Equivalent ANFIS (type 3)

	Forward pass	Backward pass
Premise parameters	Fixed	Gradient descent
Consequent parameters	Least Square Estimate	Fixed
signals	Node output	Error rates

Table 4.1 Two passes in the hybrid learning for ANFIS

As mentioned earlier, the consequent parameters thus identified are optimal (in the consequent parameter space) under the condition that the premise parameters are fixed. Accordingly the hybrid approach is much faster than the strict gradient descent and it is worthwhile to look for the possibility of decomposing the parameter set in the manner of (4.9). For type-1 ANFIS, this can be achieved if the membership function on the consequent part of each rule is replaced by a piecewise linear approximation with two consequent parameters (figure. 4.11).

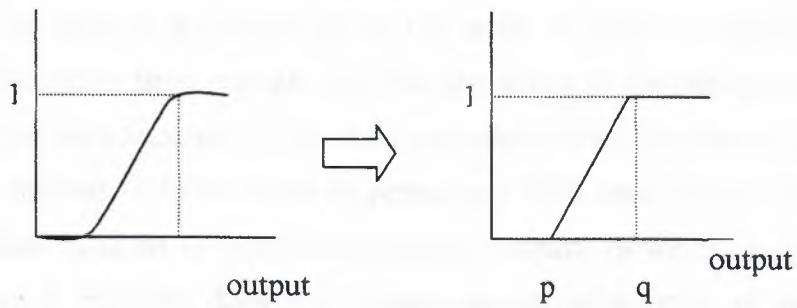


Figure 4.11 Piecewise linear approximation of membership functions on the consequent part of type-1 ANIFS.

In this case, again, the consequent parameters constitute set S_2 and the hybrid learning rule can be employed directly. However, it should be noted that the computation complexity of the least squares estimate is higher than that of the gradient descent. In fact there are four methods to update the parameters, as listed below according to their computation complexities [8]:

1. Gradient Descent Only: all parameters are updated by the gradient descent.
2. Gradient Descent and One Pass of LSE: The LSE is applied only once at every beginning to get the initial values of the consequent parameters and then the gradient descent, takes over to update all parameters.
3. Gradient descent and LSE: This is the proposed hybrid learning rule.
4. Sequential (Approximate) LSE Only: The ANFIS is linearized with respect to the premise parameters and the extended Kalman filter algorithm is employed to update all parameters. This has been proposed in the neural network literature [25].

The choice of above methods should be based on the trade-off between computation complexity and resulting performance. Note that the consequent parameters can also be updated by the Widrow-Hoff LMS algorithm [30]. The widrow-Hoff algorithm requires less computation and favors parallel hardware implementation, but it converges relatively slowly when compared to the least square estimate.

As pointed out by one of the reviewers [15], the learning mechanisms should not be applied to the determination on membership functions since they convey linguistic and sub ejective description of ill-defined concepts. We think this is case-by-case situation and the decision should be left to the users. In principle, if size of available input-output data set is large enough, then the fine-tuning of the membership functions are applicable (or even necessary) since the human-determined membership function are subject to the differences from person to person and from time to time; therefore they are rarely optimal in terms of reproducing desired outputs. However, if the data set is too small, then it probably does not contain enough information of system under consideration. In this situation, the human-determined membership functions represent important knowledge obtained through human experts experiences and it might not be reflected in the data set; therefore the membership functions should be kept fixed throughout the learning process.

Interestingly enough, if the membership functions are fixed and only the consequent part is adjusted, the ANFIS can be viewed as a functional-link network [24] where the "enhanced representation" of the input variables is achieved by the membership functions. This "enhanced representation" which takes advantage of human knowledge is apparently more insight-revealing than the functional expansion and the tensor (outer product) models. By fine-tuning the membership functions, we actually make this "enhanced representation" also adaptive.

Because the update formulas of the premise and consequent parameters are decoupled in the hybrid learning rule (table 1), further speedup of learning is possible by using other versions of the gradient method on the premise parameters, such as conjugate descent, second-order back-propagation , quick-propagation, nonlinear optimization [11].

4.7. Fuzzy Inference Systems with Simplified Fuzzy If-Then Rules

The reasoning mechanisms (figure 4.10) introduced earlier are commonly used, each of them has inherent drawbacks. For type-1 reasoning (figure 4.8. or 4.11), the membership functions on the consequence part are restricted to monotonic functions which are not compatible with linguistic terms such as "medium" whose membership function should be bell-shaped. For type-2 (figure 4.8) reasoning the defuzzification process is time-consuming and systematic fine-tuning of the parameters is not easy. For type-3 reasoning (figure 4.7), it is just hard to assign any appropriate linguistic terms to the consequence part which is not a fuzzy function of the input variables to cope with these disadvantages, simplified fuzzy if-then rules of the following form are introduced:

If x is *big* and y is *small*, then z is d.

Where z is described by a crisp value (or equivalently, a singular membership function), this class of simplified fuzzy if-then rules can employ all three types of reasoning mechanisms. More specially, the consequent part of this simplified fuzzy if-then rule is represented by a step function (centered at $z = d$) in type 1, a singular membership function (at $z = d$) in type 2, and a constant output function in type 3, respectively. Thus the three reasoning mechanisms are unified under this simplified fuzzy if-then rules.

Most off all, with this simplified fuzzy if-then rule, it is possible to prove that under certain circumstance, the resulting fuzzy inference system has unlimited approximation power to match any nonlinear functions arbitrarily well on a compact set.

In application of fuzzy inference systems, the domain in which we operate is almost always closed and bounded and therefore it is compact. For the first and second criteria, it is trivial to find simplified fuzzy inference systems that satisfy them. Now all we need to do is examine the algebraic closure under addition and multiplication. Suppose we have two fuzzy inference systems S and $\overline{S}$; each has two rules and the output of each system can be expressed as

$$S : z = \frac{w_1 f_1 + w_2 f_2}{w_1 + w_2} \tag{4.19}$$

$$\widehat{S} : \widehat{z} = \frac{\widehat{w}_1 \widehat{f}_1 + \widehat{w}_2 \widehat{f}_2}{\widehat{w}_1 + \widehat{w}_2} \tag{4.20}$$

where $f_1, f_2, \hat{f}_1$, and $\hat{f}_2$ are constant output of each rule. Then $az + b\hat{z}$ and $z\hat{z}$ can be calculated as follows [11]:

$$az + b\hat{z} = a\frac{w_1 f_1 + w_2 f_2}{w_1 + w_2} + b\frac{\hat{w}_1 \hat{f}_1 + \hat{w}_2 \hat{f}_2}{\hat{w}_1 + \hat{w}_2}$$

$$= \frac{w_1\hat{w}_1(af_1 + b\hat{f}_1) + w_1\hat{w}_2(af_1 + b\hat{f}_2) + w_2\hat{w}_1(af_2 + b\hat{f}_1) + w_2\hat{w}_2(af_2 + b\hat{f}_2)}{w_1\hat{w}_1 + w_1\hat{w}_2 + w_2\hat{w}_1 + w_2\hat{w}_2}$$

$$z\hat{z} = \frac{w_1\hat{w}_1 f_1\hat{f}_1 + w_1\hat{w}_2 f_1\hat{f}_2 + w_2\hat{w}_1 f_2\hat{f}_1 + w_2\hat{w}_2 f_2\hat{f}_2}{w_1\hat{w}_1 + w_1\hat{w}_2 + w_2\hat{w}_1 + w_2\hat{w}_2} \qquad (4.21)$$

which are of the same form as (4.19) and (4.20). Apparently the ANFIS architectures that compute $az + b\hat{z}$ and $z\hat{z}$ are of the same class of S and $\hat{S}$ if and only if the class of membership functions is invariant under multiplication. This is loosely true if the class of membership functions is the set of all bell-shaped functions, since the multiplication of two bell-shaped function is almost always still bell-shaped. Another more tightly defined class of membership functions. Satisfying this criterion, as pointed out in [23], is the scaled Gaussian membership function:

$$\mu A_i(x) = a_i \exp[-(\frac{x - c_i}{a_i})^2] \qquad (4.22)$$

Therefore by choosing an appropriate class of membership functions, we can conclude that the ANFIS with simplified fuzzy if-then rules satisfy the criteria of the Stone-Weierstrass theorem [8]. Consequently, for any given $\varepsilon > 0$, and any real-valued function g, there is a fuzzy inference system S such that $|g(\bar{x}) - S(\bar{x})| < \varepsilon$ for all $\bar{x}$ in the underlying compact set. Moreover, since the simplified ANFIS is a proper subset of all three types of ANFIS in figure.4.7 or 4.8, we can draw the conclusion that all the three types of ANFIS have unlimited approximation power to match any given data set. However, caution has to be taken in accepting this claim since there is no mention about how to construct the ANFIS according to the given data set. That is why learning plays a role in this context.

4.8. Summary

The Neural network and Fuzzy systems are complementary rather than competitive. NN provide algorithms for classification and optimization, and they work

at numerical level. Fuzzy logic offers a tool to process inaccurate and approximate information, as well as mechanism for implementing rules. This chapter presented neural fuzzy systems which combine both features, provide more flexibility, faster, and more robust than NN alone. There are number of ways to combine neuro-fuzzy systems, in this chapter we just go through over these methods, their architectures, and the operation principles of them. ANFIS architecture and its operational principle are described as well. The learning algorithm ANFIS uses back-propagation or hybrid learning rules. Learning algorithms for TKS type neuro-fuzzy system are given too.

CHAPTER 5: DEVELOPMENT OF NEURO-FUZZY SYSTEM FOR MODELING OF NONLINEAR PROCESSES

5.1. Overview

The material studied so far was carrying a theoretical explanation of neuro-fuzzy system approaches and its features starting form the appearance of fuzzy theory; its advantages over the standard control theories or in other words the mathematical modeling method were any plant or problem can be modeled or described using the formal way of description equations. ect., on the other hand fuzzy theory allowed us to use a human verbal description, and this helped to overcome most of the uncertainties, unstabilities that system could have.

In this part of the thesis a modeling of nonlinear dynamics objects and predicting exchange rate are considered and manipulated using a simulation program of a neural network trained via fuzzy logic approach.

The realization of modeling of nonlinear dynamics objects and predicting exchange rate using MATLAB tools is considered.

The ANFIS can construct input-output mapping based on both human knowledge (in the form of fuzzy if-then rules) and stipulated input-output data pairs. The ANFIS architecture is employed to modal nonlinear functions, identify nonlinear components on-linely i.e., a control system, and predict a chaotic time series [7].

5.2. Modeling of dynamic objects by Neuro-Fuzzy system

5.2.1. Direct and inverse schemes for modeling dynamic objects.

In this part modeling of the dynamics of nonlinear plant using neuro-fuzzy system is considered.

Assume that the plant is described by the following differential equation

$$\sum_{i=1}^{n} a_{n-i} y^{(i)}(t) + c\varphi(y(t)) = \sum_{j=1}^{m} b_{m-j} u^{(j)}(t) \qquad (5.1)$$

where a_i ($i=1,\ldots,n$) and b_j ($b=1,\ldots,m$) are known parameters of control object, c is

unknown nonlinear parameter, $m<n$.

Problem consists of modeling of dynamic plant (1) by using Neuro-Fuzzy system. This is implemented by learning unknown coefficients of Neuro-Fuzzy system, -i.e., a, b, and c which are the values of membership function, some of the successive actual value for a,b, and c respectively, the rest of value will be found in appendix2.

37.4084	5.3988	5.3988
29.6976	15.5695	15.5695
20.6903	14.6254	14.6254
10.7795	6.8250	6.8250
0.3982	2.4119	2.4119

In figure (5.1) the structure of the system is shown.

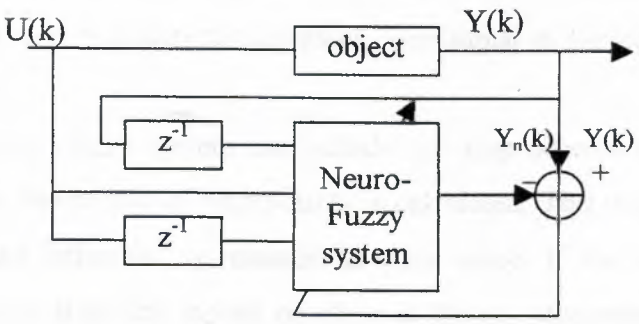


Figure 5.1 Direct modeling of dynamic object.

In figure 5.2 the structure of the system for obtaining inverse model of plant is shown.

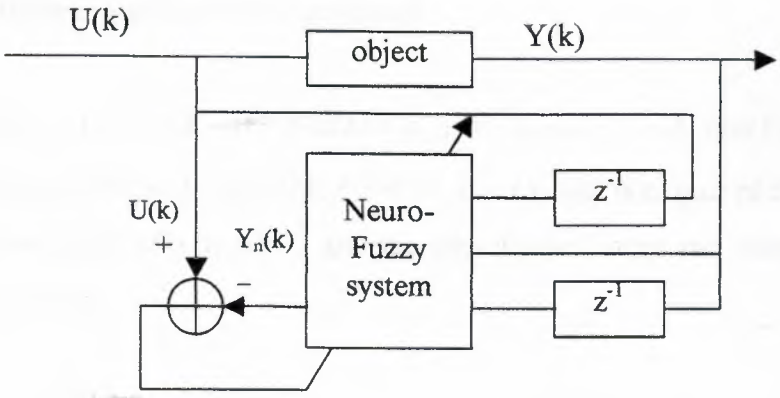


Figure 5.2 Inverse modeling of dynamic object.

Here, inputs of neuro-fuzzy system are output signals and one-step delay of input and output. Output of neuro-fuzzy system is compared with the input of plant. As a result of comparison the value of error is determined. Using this value the unknown parameters of the neuro-fuzzy system are trained. The learning is continued until the value of error become acceptable small.

5.2.2. Simulation of nonlinear dynamics of plant

The simulation was done using MATLAB, the program was written for direct and inverse modeling of plant. In the figure 5.3 the flowchart of the algorithm described, program listing can be found in appendix1.

In the first stage the random values of parameters of neuro-fuzzy system are generated. Then the value of input signal for the plant is calculated; we use here sinusoidal input signal $u(t) = sin(\pi t)$ for the plant. This signal at the same time is given to neuro-fuzzy system.

The input of neuro-fuzzy system can include one-step delayed input and output of the plant. After that, the output of neuro-fuzzy is calculated. This output is compared with plant's output and difference is retained as error value. If the value of error is acceptable small or zero then the model of plant is found, otherwise using learning algorithm described in chapter four the creation of parameters of neuro-fuzzy system is carried out, new value of output signal neuro-fuzzy system is determined and compared again with the plant's output. This procedure is continued until the value of error become zero or acceptable small.

For modeling nonlinear dynamics, the following control object that is described by nonlinear difference equation was considered

$$y(k) = a_1.y(k-1) + a_2.y(k-1) + b_1.u(k) + b_2.y(k-1).u(k-1) + b_3.\cos(k) \qquad (5.2)$$

here $a_1=1.7938$, $a_2=0.8066$, $b_1=1.0224$, $b_2=0.01$, $b_3=10$. $y(k)$ is output of the plant, $u(k)$ is input reference signal. $u(k-1)$, $y(k-1)$ are one-step delayed input and output signals of the plant, respectively.

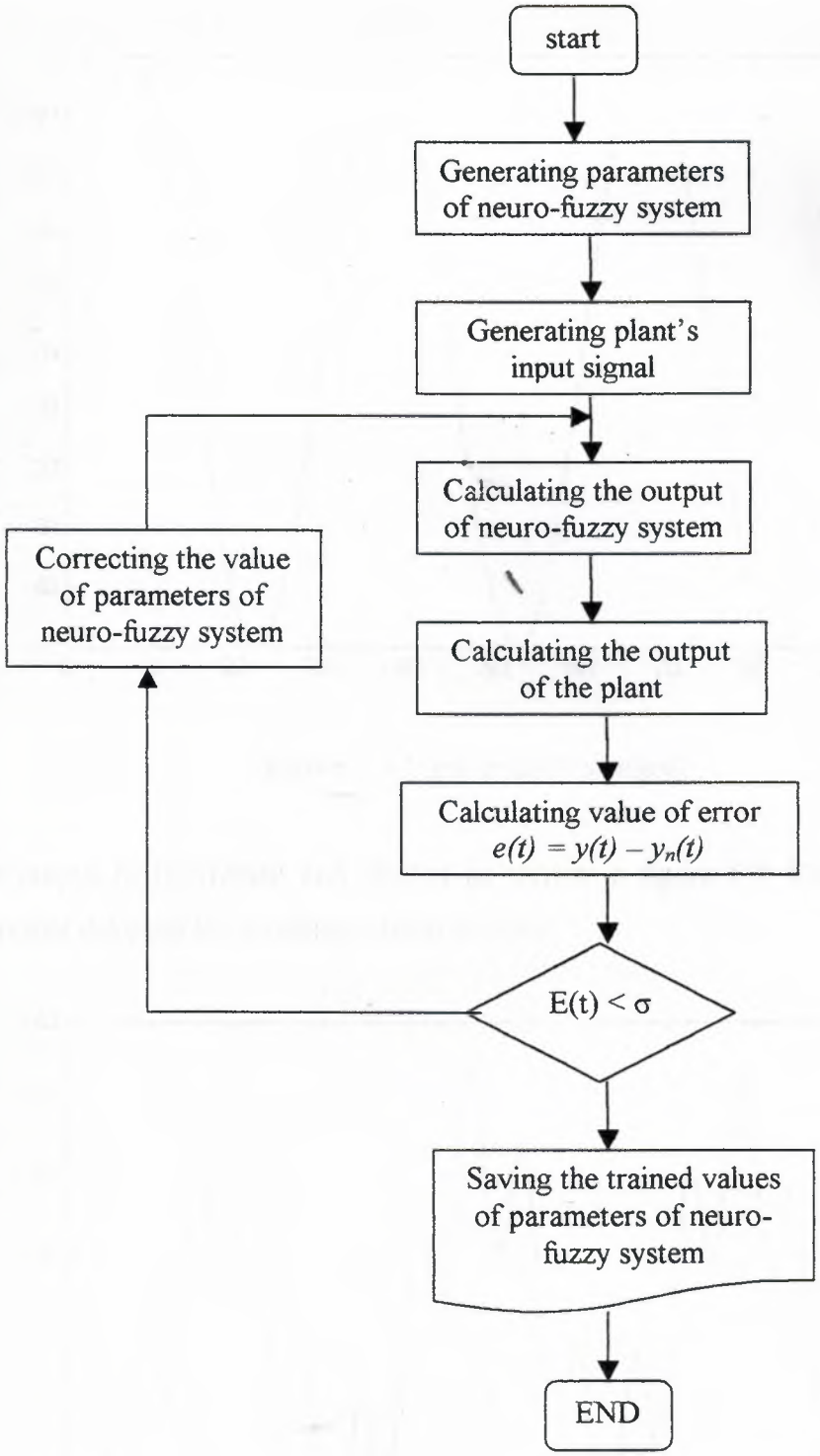


Figure 5.3 Flowchart of modeling nonlinear plant

Sinusoidal signal is given to the plant input, as shown in the figure 5.4.

$$u(e) = \sin(\pi/15) \qquad 5.3)$$

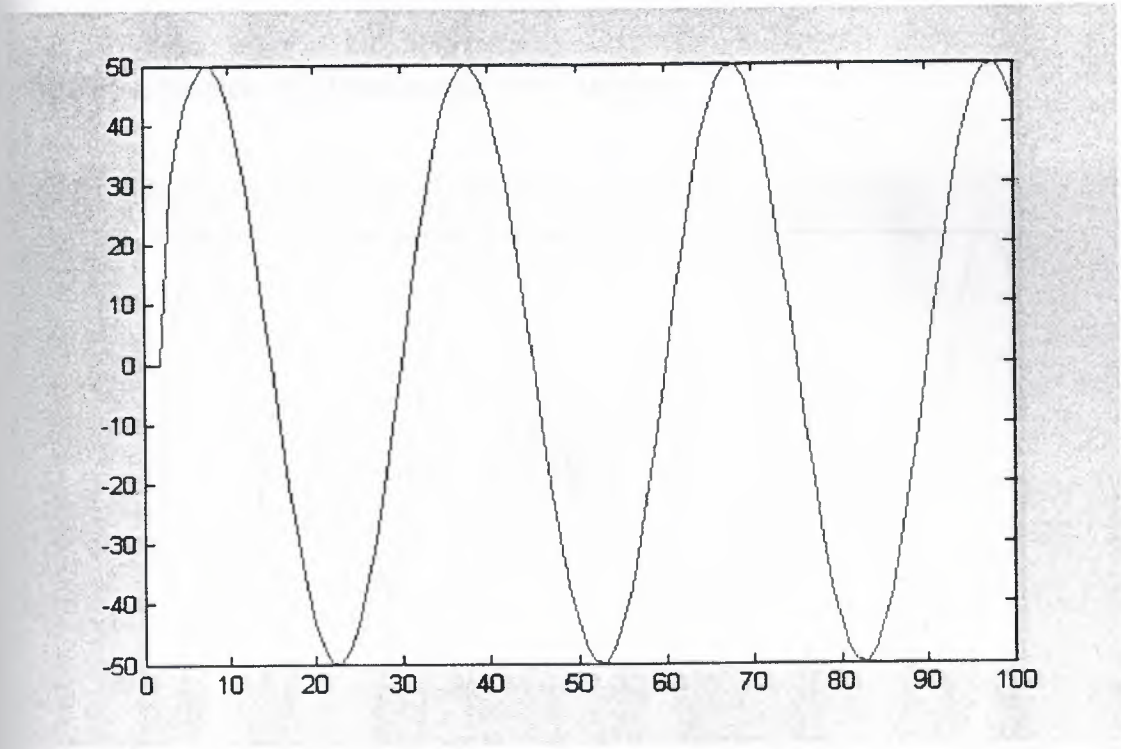


Figure 5.4 Input sinusoidal signal

Plant's output is determined and plotted as shown in figure 5.5. From the plot it is obvious that the plant has nonlinear characteristics.

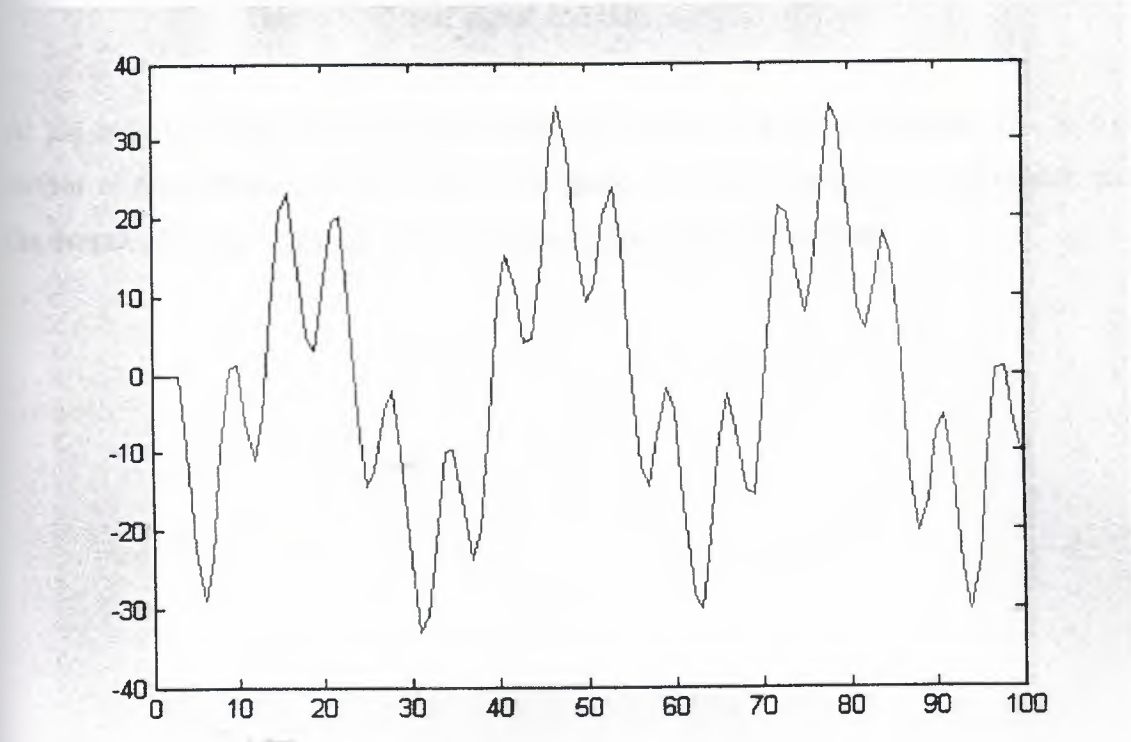


Figure 5.5 Plant output

In figure 5.6 the plant input and output plots are given.

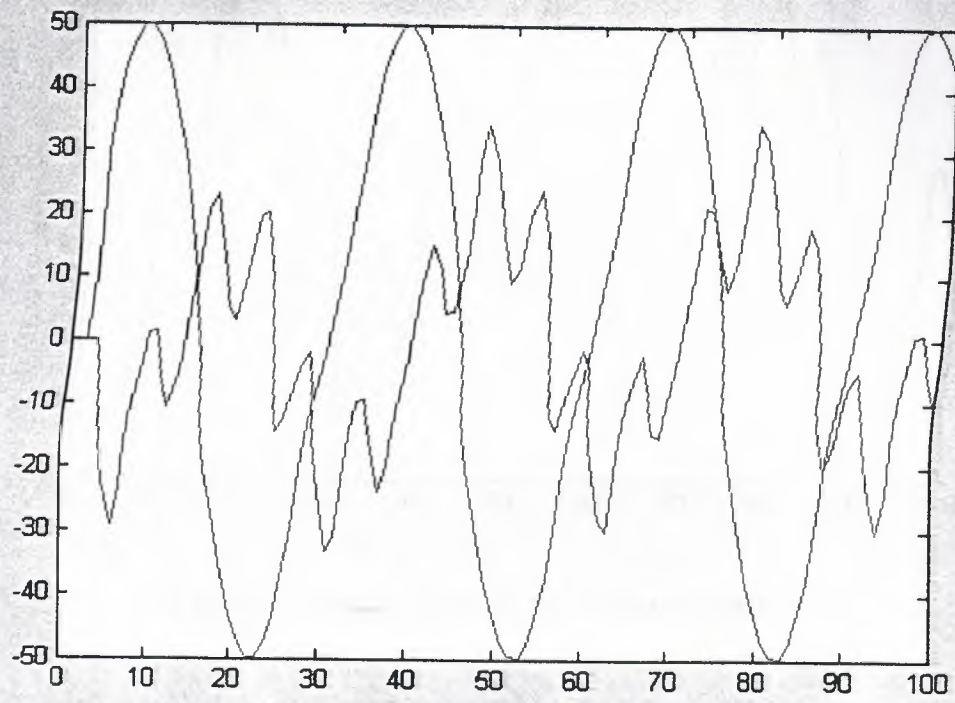


Figure 5.6 Input signal and output signal of object

At the end of training of neuro-fuzzy system the model of plant is obtained. The plot of output of neuro-fuzzy system is shown in figure 5.7. From the plot we can realize that the output of neuro-fuzzy system converges to the output of the plant.

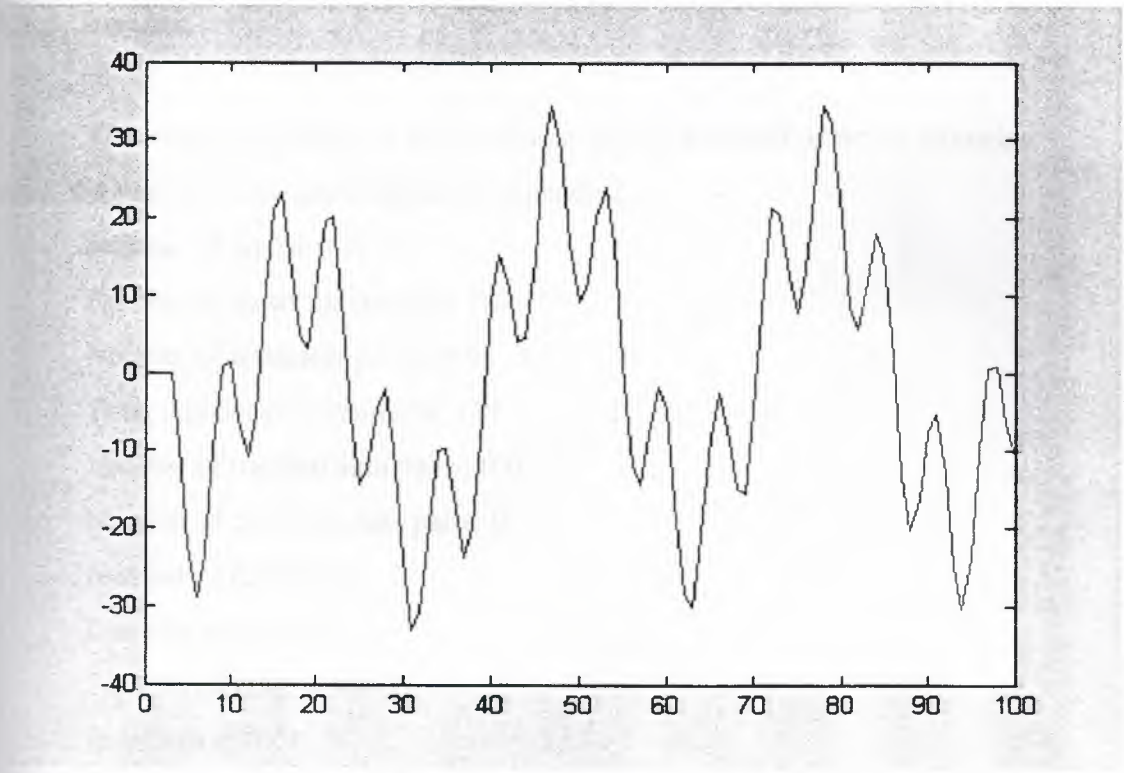


Figure 5.7 Output signal of Neuro-fuzzy system

Also the simulation of inverse neuro-fuzzy modeling of dynamic plant is carried out. In figure 5.8 the result of simulation of is given.

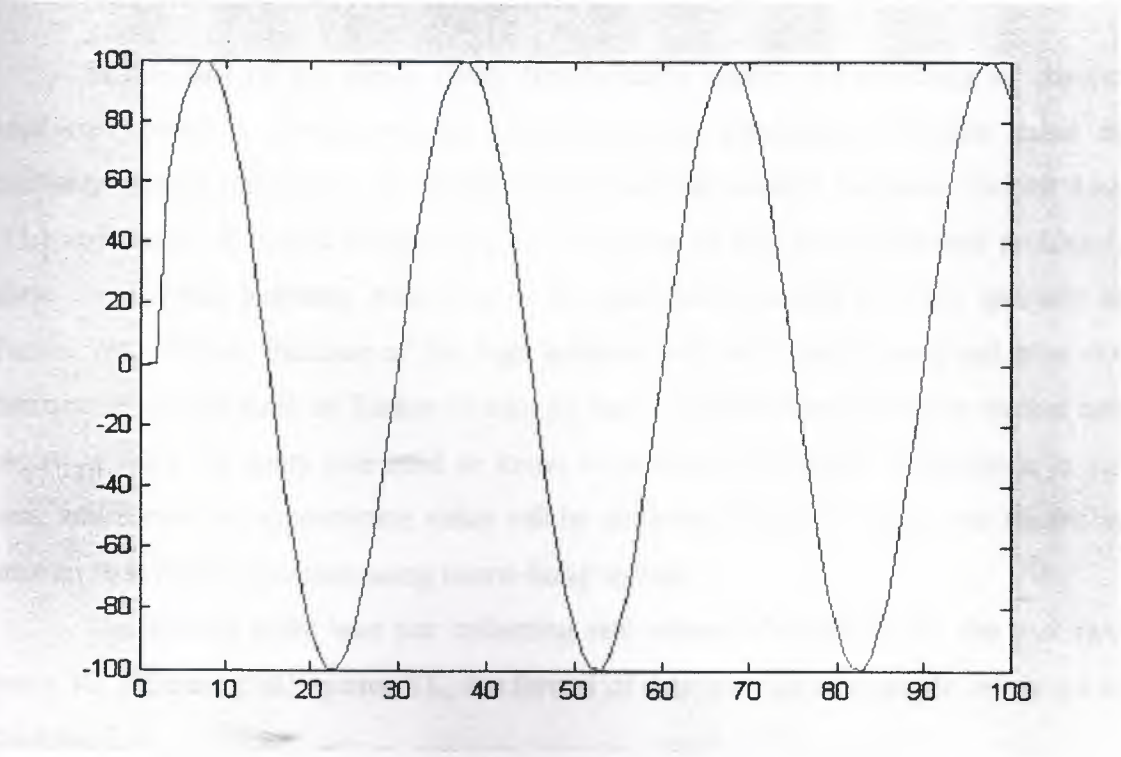


Figure 5.8 The curve for inverse model of the plant

5.2.3. Results

The results of training of the learning system are described in the following table, the rest of values can be found in appendix2.

Number of nodes: 131

Number of linear parameters: 147

Number of nonlinear parameters: 42

Total number of parameters: 189

Number of training data pairs: 100

Number of checking data pairs: 0

Number of fuzzy rules: 49

Learning rate: 0.08

Time of training: 3 min

Error value: 0.01

Step size increases to 0.011000 after epoch 6.

Designated epoch number: 20

5.3. Neuro-Fuzzy system for predicting Exchange Rate

In this part of the thesis, using Neuro-Fuzzy system the modeling of chaotic nonlinear system is considered. As an example the prediction of future value of exchange rate is considered. In specific we chose the relation between Turkish Lira (TL) and dollar of United Sates (USD). This is one of the most important problems; since it deals with business, marketing or in other wards people daily life specially in Turkey and Cyprus. Because of the high inflation and devaluation rates and after the decision of Central Bank of Turkey to take its hand of controlling the money market and exchange rates. So every one need to know what will be the value of exchange in the near future even an approximate value will be sufficient. From this point we started in attempt to solve this problem using neuro-fuzzy system.

The starting point was just collecting real values of exchange for the past two years, for different USD against TL, the format of data was; as an example, as shown in the table 5.1;

Date	USD
03.01.2000	543,401 TL
04.01.2000	538,399 TL
05.01.2000	537,722 TL
06.01.2000	537,738 TL
11.01.2000	542,212 TL
12.01.2000	541,876 TL
13.01.2000	542,939 TL
14.01.2000	545,344 TL
17.01.2000	548,869 TL
18.01.2000	549,036 TL
19.01.2000	548,880 TL

Table 5.1 Fragment of data for exchange rate between TL and USD.
The rest of data is listed in appendix 5 [33].

These data are taken as value of time series. To do so we utilize from MATLAB accessories and that by providing the ANFIS with time series prediction ability. In plotting the input data we got the following graph, where x-axis shows the number of iterations, and y-axis shows the corresponding value of Turkish Lira.

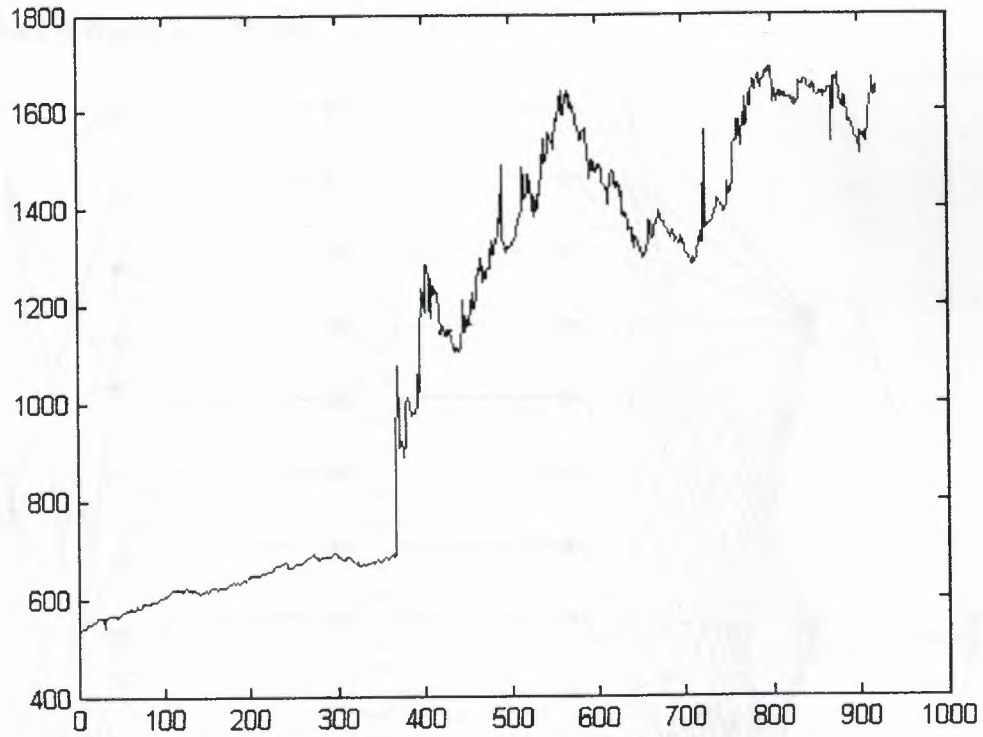


Figure 5.9 Plot of input data sample

In exchange rate prediction we want to use known values of the time up to the point in time, say t, to predict the value at some point of the future, say $t+P$, the standard method of this type of prediction is to create a mapping from D sample data points, which is the data we collected, (as shown in figure 5.8) sampled every Δ units in time, $(x(t-(D-1)\ \Delta), \ldots\ldots, x(t-\ \Delta), \ldots\ldots, x(t)$, to predict future value $x(t+P)$. Following the conventional settings for time series we set $D=4$ and $P=6$. For each t, the point training data for ANFIS is a four dimensional vector of the following form:

$$w(t) = [x(t-18)\ x(t-12)\ x(t-6)\ x(t)] \tag{5.4}$$

In other words we attempt to find an approximate relation that illustrates the change of exchange value after specific period of time, and since the exchange value is taken daily then the value that is to be predicted is after 6 days

the output training data correspond to

$$s(t) = x(t+6) \tag{5.5}$$

the training input/output data will be a structure whose first component is the four dimension input vector w, and second component is the output s.

To be more specific the general structure for the neuro-fuzzy system we applied is shown in figure 5.10 below.

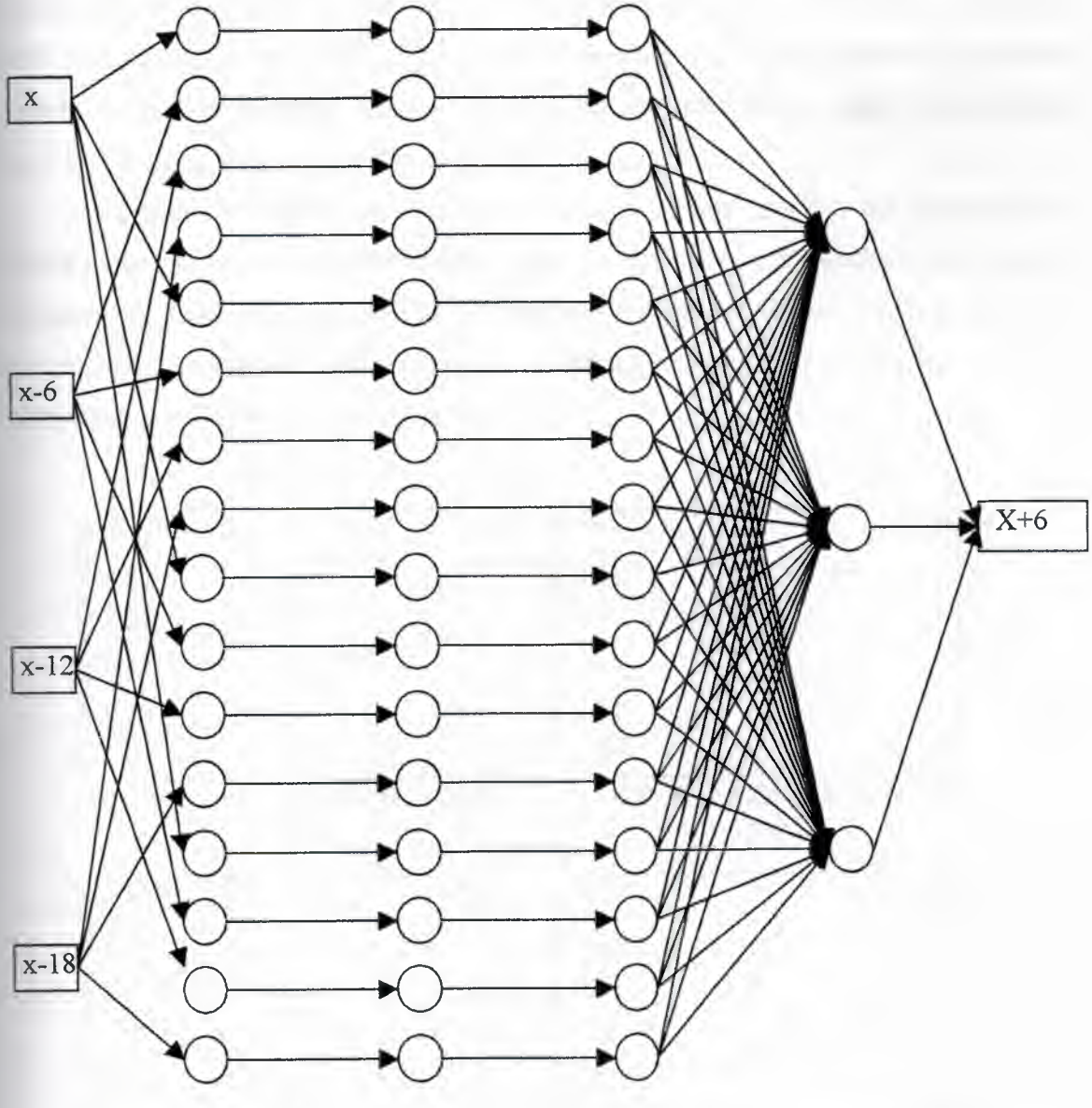


Figure 5.10 general structure of the applied neuro-fuzzy system.

where we have five layer network with 16 fuzzy rule with in the hidden layer, the first layer is input layer where it has four input nodes getting their values from the input vector, which is the value of TL exchange rate at specific time t. the second layer is the fuzzification layer, third layer consists of normalization nodes, fourth layer is the defuzzification layer, and the fifth layer is the output layer. The final output node of the system should have predicted value i.e. x+6 of the entered value x for a designated time t. note that here we kept the value of USD as fixed or in other world as a unit.

For each t, ranging in values from 25 to 900, the training input/output data will be a structure whose first component is the four-dimensional input *w*, and whose second component is the output s. There will be 922 input/output data values. We use the first 460 data values for the ANFIS training (these become the training data set), while the others are used as checking data for validating the identified fuzzy model. This results a two 460-point data structures: trnData and chkData.

To start the training, we need an FIS structure that specifies the structure and initial parameters of the FIS for learning. Since we did not specify numbers and types of membership functions used in the FIS, default values are assumed. These defaults provide two generalized bell membership functions on each of the four inputs, eight altogether. As shown in the figure below

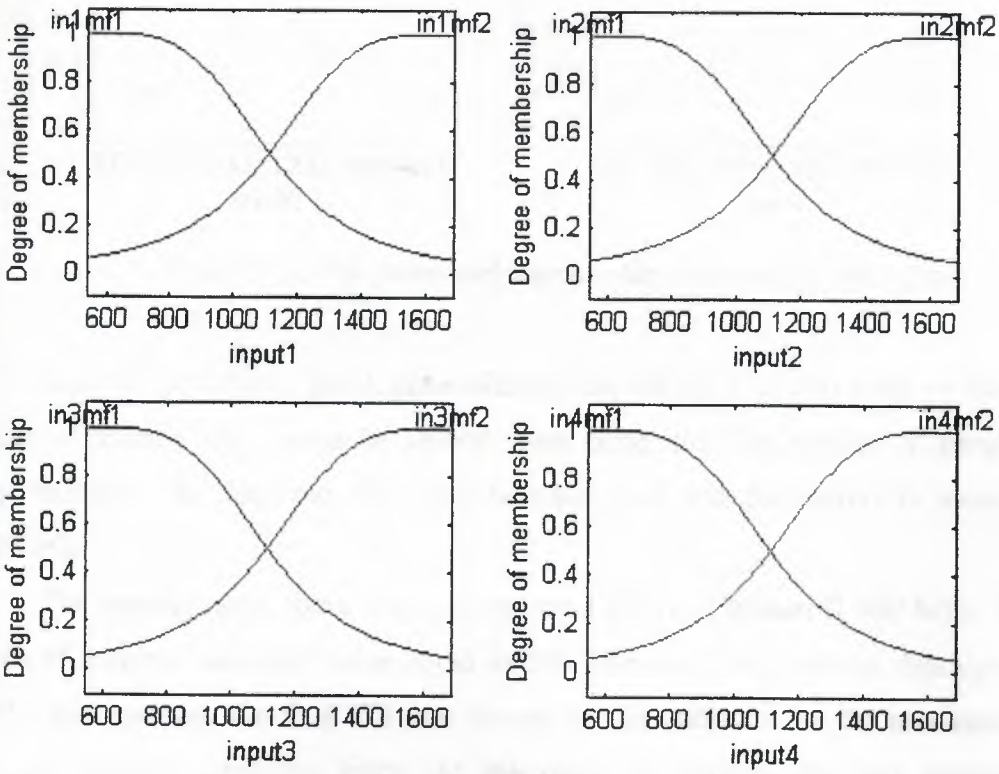


Figure 5.11 Two generalized bell membership functions on each of the four inputs.

The generated FIS structure contains 16 fuzzy rules with 104 parameters, the program generates initial membership functions that are equally spaced and cover the whole input space. Plot the input membership functions are given in figure 5.12.

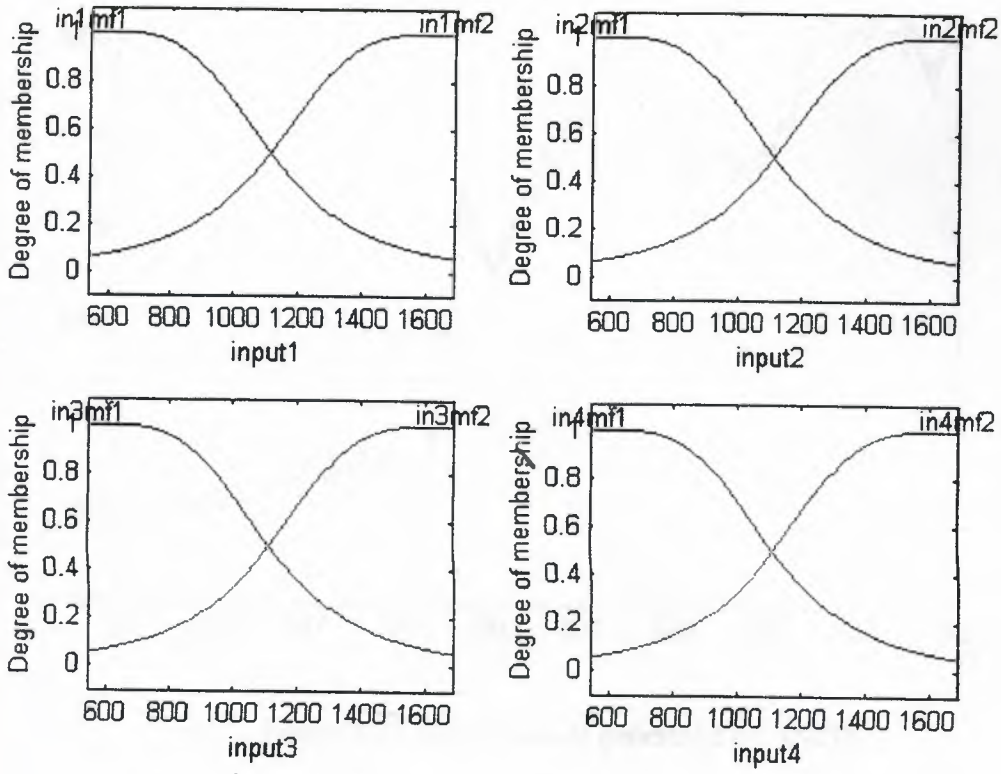


Figure 5.12 The generated membership function for input data

In order to achieve good generalization capability, it is important to have the number of training data points be several times larger than the number of parameters being estimated. In this case, the ratio between data and parameters is about four (460/104).

The training takes about four minutes on a PC AT Pentium II 300 MHz, for 10 epochs of training, and error value equal to 0.01. Because the checking data option of ANFIS was invoked, the final FIS you choose would ordinarily be the one associated with the minimum checking error. At the result of learning the new membership functions are obtained.

The comparison of the original exchange rate time series and the ANFIS prediction is given below, in figure 5.13. Note that the difference between the original exchange rate time series and the anfis estimated values is very small. This is why you can only see one curve in the first plot.

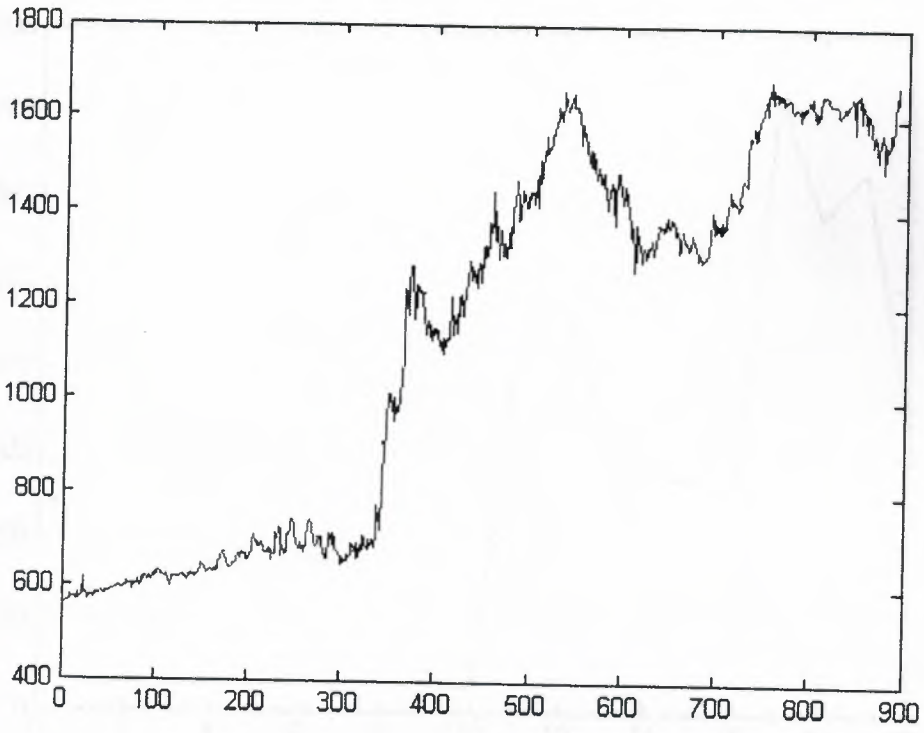


Figure 5.13 Output signal generated by ANFIS

The prediction error is shown in the second plot with a much finer scale (figure 5.14). Note that we have only trained for 10 epochs. Better performance is expected if we apply more extensive training.

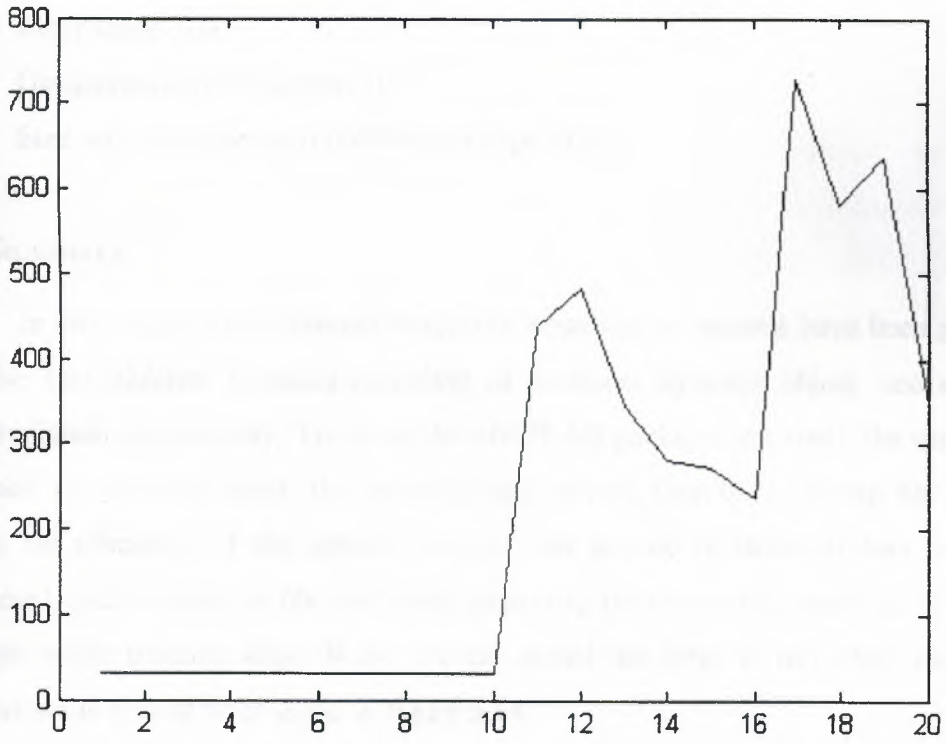


Figure 5.14 The error signal of plotting

5.3.2. Results

The results of training of the learning system are described in the following table

Number of nodes: 55

Number of hidden neurons: 48

Number of linear parameters: 80

Number of nonlinear parameters: 24

Total number of parameters: 104

Number of training data pairs: 460

Number of checking data pairs: 423

Number of fuzzy rules: 16

Learning rate: 0.08

Time of training: 4 min

Error value: 0.01

Designated epoch number: 10.

Step size decreases to 0.009000 after epoch 10.

5.4. Summary

In this chapter two different models of neuro-fuzzy systems have been produced to solve two different practical modeling of nonlinear dynamic object, and exchange rate prediction, respectively. To do so the MATLAB package was used, the systems are designed, the date are input, the networks are trained, then by analyzing the obtained results the efficiency of the applied method was proved in terms of time and labor consumed, and accuracy of obtained with respect to the error value specified to 0.01 for example while training stage of the second model the error is too small and that is obvious in the plot of error signal in figure 5.14.

CONCLUSION

The analysis of industrial and non-industrial processes show that they are characterized with uncertainty of environment, fuzziness of information, flexible of functionality. For these reasons the development of intellectual systems based on fuzzy and neural network technologies is very important. Because fuzzy system can over come the uncertainty via its linguistic expression and reasoning mechanism, while the neural network overcomes the slowness and changeability of functions via its parallel structure and ability of learning.

In the thesis the developments of neuro-fuzzy systems in general and modeling nonlinear plants, and predicting exchange rate for Turkish Lira against US Dollar in specific were considered. To solve these two problems this requires us to go through the antecedent chapters, in briefly description.

In the first chapter some of the previously done works using the neuro-fuzzy concepts, and ideas were introduced, were referenced, and their results were investigated briefly as well.

In the second chapter the general architecture and main elements of fuzzy rule based systems, their structures, and functions of main blocks. The different types of fuzzy rule based systems and their inference mechanism were described.

In the third chapter the common architectures of neural networks, their different types of learning algorithms were described, and one of their most efficient and most widely used learning algorithm which is back-propagation learning algorithm was introduced as well; where it has good learning speed. Its functions, advantages and disadvantages were explained.

In the forth chapter the different structures of neuro-fuzzy systems, their operation principles, the learning algorithms which are used for training the neuro-fuzzy systems were explained. The structure and algorithms of adaptive neuro-fuzzy inference system for different type fuzzy IF-THEN rules were described.

In the fifth chapter in order to prove the efficiency of neuro-fuzzy systems we considered two different practical problems to be solved by neuro-fuzzy system concept, the first one is direct and inverse modeling of nonlinear dynamic plant, the second one is predicting of exchange rate of Turkish Lira against US Dollar. The both

problems were applied utilizing from the learning ability of ANFIS associated with MATLAB toolbox.

The applied method emphasized the power and efficiency of neuro-fuzzy system by watching the applied system results where it has small error ratio with respect to the data provided, i.e. accuracy. The system shows a less ability to be stacked in local or global minima according to the linguistic description of functions which helps to decrease any uncertainty or eliminate it totally some times could happen either while learning or before, this causes the system to be converged and trained in less time, and also labor consumed.

The aims of the presented work within this thesis were to obtain neural network learning of fuzzy systems to solve two different static and dynamic problems, modeling of nonlinear object, predicting, predicting exchange rate of Turkish Lira against US Dollar, and to prove the efficiency of the applied method to solve different problems.

These aims have all been achieved throughout the work that is described within this thesis. Obtained results and the analysis of them show the achievement of the work.

REFERENCES

[1] Zadeh L.A. Fuzzy Sets, Information and Control, vol.8, pp. 338-353, 1965.

[2] Zadeh, L.A., Fuzzy Sets and Applications: Selected Papers by L. A. Zadeh, John Wiley & Sons, New York, 1987.

[3] Yager R.R., Zadeh L.A.(Eds). Fuzzy sets, neural networks and soft computing, Van Nostrand Reinhold, New York, 1994.

[4] Kosko B. Neural networks and fuzzy systems. A dynamical system approach to machine intelligence. Prentice- Hall International Inc., 1993.

[5] Kosko B., Neural Networks and Fuzzy Systems. Englewood Cliffs, NJ: Prentice-Hall, 1992. Carnegie Mellon University Neural Network Benchmark Collection, Pittsburgh, PA, 1994.

[6] T. Takagi, M. Sugeno, Fuzzy identification of Systems and it Applications to Modeling and Control, IEEE Trans. Syst., Man. Cybern., vol. SMC-15, pp. 116-132, January 1985.

[7] R.A.Aliev, R.H.Abiyev, R.R.Aliev. Automatic control system synthesis with the learned neural network based fuzzy controller. Texnicheskaya kibernetika, Moscow, N2, pp. 192-197, Russian,1994,

[8] Rahib Abiyev. Controllers based on Soft-computing elements, Electrical Electronics and Computer Engineering Symposium NEU-CEE2001 & Exhibition. Nicosia, TRNC, Turkey, pp. 182-188, May 23-25, 2001.

[9] Rahib Abiyev. Fuzzy inference system based on neural network for technological processes control. Journal of Mathematical and Computational Applications. Turkey, pp.245-252, 2002.

[10] Fakhreddin Mamedov, Rahib Abiyev. Fuzzy Neural modeling of the nonlinear processes in condition of uncertainty // World Conference on Intelligent Systems for Industrial Automation, WCIS-2000, Tashkent, Uzbekistan, pp.310-315, September 14-16, 2000.

[11] Jyh-Shing Roger Jang. ANFIS: Adaptive-Network-Based Fuzzy Inference System. IEEE Transactions on Systems, Man and Cybernetics, Vol.23, No.3, pp. 665-683, May/June 1993.

[12] Lin, C.T. and Lee George C.S., Neural Network Based Fuzzy Logic Control and Decision System, IEEE Transactions on Computers, vol. 40, pp.1320-1360, 1991.

[13] Marek S. Onski, Applications of ANFIS Neuro-Fuzzy System for Prediction of Concrete Fatigue Durability. Institute of Computer Methods in Civil Engineering, Cracow University of Technology, Poland, pp17-20, 2000.

[14] Mamdani, E.H., Application of fuzzy algorithms for control of simple dynamic plant. Proc. IEEE (Control and Science), V. 121, pp.1585-1588, 1974.

[15] A. Iriarte Lanas, M.A. Pacheco, M.M. Vellasco, R. Tanscheit, Neuro-Fuzzy Control Of A Multivariable Nonlinear Process. Department of computer Engineering, Brazil, pp 1-15, 1999.

[16] Arun D. Kulkarni, Computer Vision and Fuzzy-Neural Systems, Prentice Hall, USA, Ch3, Ch4, 2001

[17] P.P.Groumpos, B.G.Ilyasov, L.A.Ismagilova, R.G.Valeeva. Intelligent Control Algorithms Of Dynamic Manufacturing Systems. Pros. Of ASI'98 Life Cycle Approaches To Production Systems: Management, Control And Supervision. Bremen, Germany, 1998.

[18] Bellei E., Petacchi R., Reyneri L., Neuro-Fuzzy Methodologies for the Clustering and the Reliability Estimation of Olive Fruit Fly Infestation. ESANN'2002 proceedings - European Symposium on Artificial Neural Networks, Bruges (Belgium), d-side publi., pp. 459-464, 24-26 April 2002.

[19] Deterding, D.H. Speaker Normalization for Automatic Speech Recognition, Ph.D. Dissertation, Cambridge, England, 1989.

[20] Detlef Nauck, and Rudolf Kruse. NEFCLASS . A Neuro-Fuzzy Approach For The Classification Of Data. In K. M.George, Janice H. Carrol, Ed Deaton, Dave Oppenheim, and Jim Hightower, ACM Symposium on Applied Computing, Nashville, pp. 461-465. ACM Press, NewYork, 26-28 February 1995.

[21] D. Psaltis, A Sideris, and A Yamamura, A Multilayered Neural Network Controller, IEEE Contr. Syst., vol. 8, pp. 17-21, April 1988.

[22] Gary R. George P. E., Frank Cardullo, Application of Neuro-Fuzzy Systems to Behavioral Representation in Computer Generated Forces. State University of New York at Binghamton, pp 1-11, N. Y. 2001.

[23] Gupta, M. M, Fuzzy Neural Network: Theory And Application, Proceedings of SPIE, vol 2353, pp.300-325, 1994

[24] Hamid Ghezelayagh Kwang Y. Lee, Application Of Neuro-Fuzzy Identifier In The Predictive Control Of A Power Plant, University Park, PA 16802, 15th Triennial World Congress, Barcelona, Spain, 2002

[25] Kasabov, N., Kim J S, Watts, M., Gray, A FuNN/2- A Fuzzy Neural Network Architecture for Adaptive Learning and Knowledge Acquisition, Information Sciences -Applications, 101(3-4): pp.155-175. 1997

[26] M. Onder Efe, Okyay Kaynak, "On stabilization of Gradient-Based Training Strategies for Computationally Intelligent Systems". IEEE Transactions on Fuzzy Systems, Vol.8, No.5, pp. 564-575, October, 2000.

[27] R. Aliev, K. W. Bonfig, F. Aliew, Neural Learning System for Technological Process Control, ICAFS'98, Third International Conference on Application Of Fuzzy Systems and Soft Computing, Wiesbaden, Germany, 1998.

[28] Reza Langari & Hamid R. Berenji, Fuzzy Logic in Control Engineering, pp 137-198, 1996

[29] S. Chen, C. F. N. Cowan, and P. M. Grant, Orthogonal Least Squares Learning Algorithm For Radial Basis Function Network, IEEE Trans. Neural Networks, vol. 2, no. 2, pp 302-309, March 1991.

[30] S. E. Fulham, Faster-Learning Variations on Back-Propagation: An Empirical Study, in Proc. 1988 Connectionist Models Summer School, D. Touretzky, G. Hinton, and T. Sejnowski, Eds., Carnegie Mollon Uni., pp. 38-51, 1988.

[31] Widrow, B., Winter, and Baxter, Learning Phenomena in Layered Neural Networks, Prentice-Hall, 1987.

[32] Şenol Bektaş, Fakhraddın Mamedov and Adnan Khashman, Graduate studies: A complete reference, Near East University. Nikosia.2001.

[33] Hppt://www.otoalsat.com/financeman/kurlar.

[34] MATLAB package 6.2.

APPENDIX 1

Program code for modeling

```
pi=3.14;
l=3;
k1=50;   k2=1;
a0=1.42;
a1=2.12;
a2=1;
b=80;
nc=0.01;
t=0.15;
aa1=(2*a0+a1*t)/(a0+a1*t+a2*t*t);
aa2=-a0/(a0+a1*t+a2*t*t);
bb1=(b*t*t)/(a0+a1*t+a2*t*t);
bb2=0;
for l=1:100
   out(l)=0;
   u1(l)=0;
   g(l)=0;
end

for l=3:100
  g(l)=(sin(pi*l/15))'
  out(l)=g(l)*bb1+aa1*out(l-1)+aa2*out(l-2)+nc*out(l-1)*g(l-1)+10*cos(l);
  x1(l)=k1*g(l);
  x2(l)=k2*out(l-1);
end;

plot(x1)
pause
plot(x2)
pause
plot(x1)
pause

mfType='gbellmf';
epoch_n=20;
  input=[x1' x2']
  trn_Data=[x1' x2' x2']

  in_fismat = genfis1(trn_Data);

  out_fismat=anfis(trn_Data,in_fismat,20);

  plot(evalfis(input,out_fismat));
```

APPENDIX 2

Program execution results for modeling

```
Columns 78 through 88
  -0.5811  -0.7375  -0.8617  -0.9484  -0.9936  -0.9954  -0.9538  -0.8705  -0.7492
-0.5952  -0.4153
 Columns 89 through 99
  -0.2171  -0.0096   0.1985   0.3978   0.5798   0.7364   0.8609   0.9479   0.9934
0.9956   0.9543
 Column 100
   0.8713
out =
 Columns 1 through 11
      0       0  -9.2992 -10.5473   2.0178  15.8131  15.8635   1.3967 -13.7452 -
15.1126  -1.4689
 Columns 12 through 22
  14.3108  17.4185   4.8127 -12.0693 -17.8972  -7.5361   9.2716  16.8887   8.1646
-8.9924 -18.8836
 Columns 23 through 33
 -12.4482   4.4097  16.2492  12.2855  -3.6993 -16.8393 -14.8601   0.6282
15.6022  16.5083   2.7149
 Columns 34 through 44
 -12.9158 -15.8612  -3.2969  13.3015  18.7057   7.9340  -9.1677 -16.9760  -
8.4495   8.4033  17.8945
 Columns 45 through 55
  11.0883  -5.9740 -17.8189 -13.7579   2.2945  15.4279  13.4506  -1.8956 -
16.5340 -16.9935  -2.7943
 Columns 56 through 66
  13.1085  16.2310   3.8716 -12.4132 -17.4417  -6.3745  10.8269  18.5493
9.8738  -7.0712 -16.5894
 Columns 67 through 77
  -9.8533   6.9768  18.4151  13.8883  -2.5409 -15.9045 -14.0849   1.0517  15.3793
15.5091   1.1079
 Columns 78 through 88
 -14.7857 -17.7403  -5.1920  11.2050  16.2985   5.3724 -11.5159 -18.7828  -
9.6482   7.6262  17.3271
 Columns 89 through 99
  10.7286  -5.8934 -17.0404 -12.2502   4.2751  17.5235  15.4674   0.1146 -
14.3422 -14.5781  -0.3878
 Column 100
  15.1259
g =
 Columns 1 through 11
      0       0   0.5875   0.7429   0.8658   0.9509   0.9944   0.9946   0.9514
0.8666   0.7439
 Columns 12 through 22
   0.5888   0.4080   0.2094   0.0016  -0.2062  -0.4051  -0.5862  -0.7418  -0.8650  -
0.9504  -0.9943
 Columns 23 through 33
```

```
-0.9948  -0.9518  -0.8673  -0.7450  -0.5901  -0.4095  -0.2109  -0.0032   0.2047
0.4036   0.5849
 Columns 34 through 44
   0.7407   0.8642   0.9499   0.9941   0.9949   0.9523   0.8681   0.7461   0.5914
0.4109   0.2125
 Columns 45 through 55
   0.0048  -0.2031  -0.4022  -0.5837  -0.7397  -0.8634  -0.9494  -0.9939  -0.9951
-0.9528  -0.8689
 Columns 56 through 66
  -0.7471  -0.5927  -0.4124  -0.2140  -0.0064   0.2016   0.4007   0.5824   0.7386
0.8626   0.9489
 Columns 67 through 77
   0.9938   0.9953   0.9533   0.8697   0.7482   0.5940   0.4138   0.2156   0.0080  -
0.2000  -0.3993
 Columns 78 through 88
  -0.5811  -0.7375  -0.8617  -0.9484  -0.9936  -0.9954  -0.9538  -0.8705  -0.7492
-0.5952  -0.4153
 Columns 89 through 99
  -0.2171  -0.0096   0.1985   0.3978   0.5798   0.7364   0.8609   0.9479   0.9934
0.9956   0.9543
 Column 100      0.8713
out =
 Columns 1 through 11
        0         0  -9.2992  -10.5473    2.0178   15.8131   15.8635    1.3967  -13.7452  -
15.1126   -1.4689
 Columns 12 through 22
   14.3108   17.4185    4.8127  -12.0693  -17.8972   -7.5361    9.2716   16.8887    8.1646
-8.9924  -18.8836
 Columns 23 through 33
  -12.4482    4.4097   16.2492   12.2855   -3.6993  -16.8393  -14.8601    0.6282
15.6022   16.5083    2.7149
 Columns 34 through 44
  -12.9158  -15.8612   -3.2969   13.3015   18.7057    7.9340   -9.1677  -16.9760   -
8.4495    8.4033   17.8945
 Columns 45 through 55
   11.0883   -5.9740  -17.8189  -13.7579    2.2945   15.4279   13.4506   -1.8956  -
16.5340  -16.9935   -2.7943
 Columns 56 through 66
   13.1085   16.2310    3.8716  -12.4132  -17.4417   -6.3745   10.8269   18.5493
9.8738   -7.0712  -16.5894
 Columns 67 through 77
   -9.8533    6.9768   18.4151   13.8883   -2.5409  -15.9045  -14.0849    1.0517   15.3793
15.5091    1.1079
 Columns 78 through 88
  -14.7857  -17.7403   -5.1920   11.2050   16.2985    5.3724  -11.5159  -18.7828   -
9.6482    7.6262   17.3271
 Columns 89 through 99
   10.7286   -5.8934  -17.0404  -12.2502    4.2751   17.5235   15.4674    0.1146  -
14.3422  -14.5781   -0.3878
```

```
Column 100    15.1259
g =
  Columns 1 through 11
       0       0  0.5875  0.7429  0.8658  0.9509  0.9944  0.9946  0.9514
0.8666  0.7439
  Columns 12 through 22
   0.5888  0.4080  0.2094  0.0016 -0.2062 -0.4051 -0.5862 -0.7418 -0.8650 -
0.9504 -0.9943
  Columns 23 through 33
  -0.9948 -0.9518 -0.8673 -0.7450 -0.5901 -0.4095 -0.2109 -0.0032  0.2047
0.4036  0.5849
  Columns 34 through 44
   0.7407  0.8642  0.9499  0.9941  0.9949  0.9523  0.8681  0.7461  0.5914
0.4109  0.2125
  Columns 45 through 55
   0.0048 -0.2031 -0.4022 -0.5837 -0.7397 -0.8634 -0.9494 -0.9939 -0.9951
-0.9528 -0.8689
  Columns 56 through 66
  -0.7471 -0.5927 -0.4124 -0.2140 -0.0064  0.2016  0.4007  0.5824  0.7386
0.8626  0.9489
  Columns 67 through 77
   0.9938  0.9953  0.9533  0.8697  0.7482  0.5940  0.4138  0.2156  0.0080 -
0.2000 -0.3993
  Columns 78 through 88
  -0.5811 -0.7375 -0.8617 -0.9484 -0.9936 -0.9954 -0.9538 -0.8705 -0.7492
-0.5952 -0.4153
  Columns 89 through 99
  -0.2171 -0.0096  0.1985  0.3978  0.5798  0.7364  0.8609  0.9479  0.9934
0.9956  0.9543
  Column 100
   0.8713
out =
  Columns 1 through 11
       0       0 -9.2992 -10.5473  2.0178  15.8131  15.8635  1.3967 -13.7452 -
15.1126 -1.4689
  Columns 12 through 22
  14.3108  17.4185  4.8127 -12.0693 -17.8972 -7.5361  9.2716  16.8887  8.1646
-8.9924 -18.8836
  Columns 23 through 33
 -12.4482  4.4097  16.2492  12.2855 -3.6993 -16.8393 -14.8601  0.6282
15.6022  16.5083  2.7149
  Columns 34 through 44
 -12.9158 -15.8612 -3.2969  13.3015  18.7057  7.9340 -9.1677 -16.9760 -
8.4495  8.4033  17.8945
  Columns 45 through 55
  11.0883 -5.9740 -17.8189 -13.7579  2.2945  15.4279  13.4506 -1.8956 -
16.5340 -16.9935 -2.7943
  Columns 56 through 66
```

```
  13.1085   16.2310    3.8716  -12.4132  -17.4417   -6.3745   10.8269   18.5493
9.8738   -7.0712  -16.5894
 Columns 67 through 77
  -9.8533    6.9768   18.4151   13.8883   -2.5409  -15.9045  -14.0849    1.0517   15.3793
15.5091    1.1079
 Columns 78 through 88
 -14.7857  -17.7403   -5.1920   11.2050   16.2985    5.3724  -11.5159  -18.7828   -
9.6482    7.6262   17.3271
 Columns 89 through 99
  10.7286   -5.8934  -17.0404  -12.2502    4.2751   17.5235   15.4674    0.1146   -
14.3422  -14.5781   -0.3878
 Column 100
  15.1259
g =
 Columns 1 through 11
         0         0    0.5875    0.7429    0.8658    0.9509    0.9944    0.9946    0.9514
0.8666    0.7439
 Columns 12 through 22
    0.5888    0.4080    0.2094    0.0016   -0.2062   -0.4051   -0.5862   -0.7418   -0.8650   -
0.9504   -0.9943
 Columns 23 through 33
   -0.9948   -0.9518   -0.8673   -0.7450   -0.5901   -0.4095   -0.2109   -0.0032    0.2047
0.4036    0.5849
 Columns 34 through 44
    0.7407    0.8642    0.9499    0.9941    0.9949    0.9523    0.8681    0.7461    0.5914
0.4109    0.2125
 Columns 45 through 55
    0.0048   -0.2031   -0.4022   -0.5837   -0.7397   -0.8634   -0.9494   -0.9939   -0.9951
-0.9528   -0.8689
 Columns 56 through 66
   -0.7471   -0.5927   -0.4124   -0.2140   -0.0064    0.2016    0.4007    0.5824    0.7386
0.8626    0.9489
 Columns 67 through 77
    0.9938    0.9953    0.9533    0.8697    0.7482    0.5940    0.4138    0.2156    0.0080   -
0.2000   -0.3993
 Columns 78 through 88
   -0.5811   -0.7375   -0.8617   -0.9484   -0.9936   -0.9954   -0.9538   -0.8705   -0.7492
-0.5952   -0.4153
 Columns 89 through 99
   -0.2171   -0.0096    0.1985    0.3978    0.5798    0.7364    0.8609    0.9479    0.9934
0.9956    0.9543
 Column 100
    0.8713
out =
 Columns 1 through 11
         0         0   -9.2992  -10.5473    2.0178   15.8131   15.8635    1.3967  -13.7452   -
15.1126   -1.4689
 Columns 12 through 22
```

14.3108 17.4185 4.8127 -12.0693 -17.8972 -7.5361 9.2716 16.8887 8.1646
-8.9924 -18.8836
Columns 23 through 33
-12.4482 4.4097 16.2492 12.2855 -3.6993 -16.8393 -14.8601 0.6282
15.6022 16.5083 2.7149
Columns 34 through 44
-12.9158 -15.8612 -3.2969 13.3015 18.7057 7.9340 -9.1677 -16.9760 -
8.4495 8.4033 17.8945
Columns 45 through 55
11.0883 -5.9740 -17.8189 -13.7579 2.2945 15.4279 13.4506 -1.8956 -
16.5340 -16.9935 -2.7943
Columns 56 through 66
13.1085 16.2310 3.8716 -12.4132 -17.4417 -6.3745 10.8269 18.5493
9.8738 -7.0712 -16.5894
Columns 67 through 77
-9.8533 6.9768 18.4151 13.8883 -2.5409 -15.9045 -14.0849 1.0517 15.3793
15.5091 1.1079
Columns 78 through 88
-14.7857 -17.7403 -5.1920 11.2050 16.2985 5.3724 -11.5159 -18.7828 -
9.6482 7.6262 17.3271
Columns 89 through 99
10.7286 -5.8934 -17.0404 -12.2502 4.2751 17.5235 15.4674 0.1146 -
14.3422 -14.5781 -0.3878
Column 100
15.1259
g =
Columns 1 through 11
0 0 0.5875 0.7429 0.8658 0.9509 0.9944 0.9946 0.9514
0.8666 0.7439
Columns 12 through 22
0.5888 0.4080 0.2094 0.0016 -0.2062 -0.4051 -0.5862 -0.7418 -0.8650 -
0.9504 -0.9943
Columns 23 through 33
-0.9948 -0.9518 -0.8673 -0.7450 -0.5901 -0.4095 -0.2109 -0.0032 0.2047
0.4036 0.5849
Columns 34 through 44
0.7407 0.8642 0.9499 0.9941 0.9949 0.9523 0.8681 0.7461 0.5914
0.4109 0.2125
Columns 45 through 55
0.0048 -0.2031 -0.4022 -0.5837 -0.7397 -0.8634 -0.9494 -0.9939 -0.9951
-0.9528 -0.8689
Columns 56 through 66
-0.7471 -0.5927 -0.4124 -0.2140 -0.0064 0.2016 0.4007 0.5824 0.7386
0.8626 0.9489
Columns 67 through 77
0.9938 0.9953 0.9533 0.8697 0.7482 0.5940 0.4138 0.2156 0.0080 -
0.2000 -0.3993
Columns 78 through 88

```
 -0.5811  -0.7375  -0.8617  -0.9484  -0.9936  -0.9954  -0.9538  -0.8705  -0.7492  -
0.5952  -0.4153
 Columns 89 through 99
 -0.2171  -0.0096   0.1985   0.3978   0.5798   0.7364   0.8609   0.9479   0.9934
0.9956   0.9543
 Column 100
  0.8713
out =
 Columns 1 through 11
      0       0  -9.2992 -10.5473   2.0178  15.8131  15.8635   1.3967 -13.7452 -
15.1126  -1.4689
 Columns 12 through 22
  14.3108  17.4185   4.8127 -12.0693 -17.8972  -7.5361   9.2716  16.8887   8.1646
-8.9924  -18.8836
 Columns 23 through 33
 -12.4482   4.4097  16.2492  12.2855  -3.6993 -16.8393 -14.8601   0.6282
15.6022  16.5083   2.7149
 Columns 34 through 44
 -12.9158 -15.8612  -3.2969  13.3015  18.7057   7.9340  -9.1677 -16.9760  -
8.4495   8.4033  17.8945
 Columns 45 through 55
  11.0883  -5.9740 -17.8189 -13.7579   2.2945  15.4279  13.4506  -1.8956  -
16.5340 -16.9935  -2.7943
 Columns 56 through 66
  13.1085  16.2310   3.8716 -12.4132 -17.4417  -6.3745  10.8269  18.5493
9.8738  -7.0712 -16.5894
 Columns 67 through 77
  -9.8533   6.9768  18.4151  13.8883  -2.5409 -15.9045 -14.0849   1.0517  15.3793
15.5091   1.1079
 Columns 78 through 88
 -14.7857 -17.7403  -5.1920  11.2050  16.2985   5.3724 -11.5159 -18.7828  -
9.6482   7.6262  17.3271
 Columns 89 through 99
  10.7286  -5.8934 -17.0404 -12.2502   4.2751  17.5235  15.4674   0.1146  -
14.3422 -14.5781  -0.3878
 Column 100
  15.1259
g =
 Columns 1 through 11
      0       0   0.5875   0.7429   0.8658   0.9509   0.9944   0.9946   0.9514
0.8666   0.7439
 Columns 12 through 22
  0.5888   0.4080   0.2094   0.0016  -0.2062  -0.4051  -0.5862  -0.7418  -0.8650  -
0.9504  -0.9943
 Columns 23 through 33
 -0.9948  -0.9518  -0.8673  -0.7450  -0.5901  -0.4095  -0.2109  -0.0032   0.2047
0.4036   0.5849
 Columns 34 through 44
```

```
  0.7407   0.8642   0.9499   0.9941   0.9949   0.9523   0.8681   0.7461   0.5914
0.4109   0.2125
 Columns 45 through 55
  0.0048  -0.2031  -0.4022  -0.5837  -0.7397  -0.8634  -0.9494  -0.9939  -0.9951
-0.9528  -0.8689
 Columns 56 through 66
 -0.7471  -0.5927  -0.4124  -0.2140  -0.0064   0.2016   0.4007   0.5824   0.7386
0.8626   0.9489
 Columns 67 through 77
  0.9938   0.9953   0.9533   0.8697   0.7482   0.5940   0.4138   0.2156   0.0080  -
0.2000  -0.3993
 Columns 78 through 88
 -0.5811  -0.7375  -0.8617  -0.9484  -0.9936  -0.9954  -0.9538  -0.8705  -0.7492
-0.5952  -0.4153
 Columns 89 through 99
 -0.2171  -0.0096   0.1985   0.3978   0.5798   0.7364   0.8609   0.9479   0.9934
0.9956   0.9543
 Column 100
  0.8713
out =
 Columns 1 through 11
       0       0  -9.2992 -10.5473   2.0178  15.8131  15.8635   1.3967 -13.7452 -
15.1126  -1.4689
 Columns 12 through 22
  14.3108  17.4185   4.8127 -12.0693 -17.8972  -7.5361   9.2716  16.8887   8.1646
-8.9924 -18.8836
 Columns 23 through 33
 -12.4482   4.4097  16.2492  12.2855  -3.6993 -16.8393 -14.8601   0.6282
15.6022  16.5083   2.7149
 Columns 34 through 44
 -12.9158 -15.8612  -3.2969  13.3015  18.7057   7.9340  -9.1677 -16.9760  -
8.4495   8.4033  17.8945
 Columns 45 through 55
  11.0883  -5.9740 -17.8189 -13.7579   2.2945  15.4279  13.4506  -1.8956 -
16.5340 -16.9935  -2.7943
 Columns 56 through 66
  13.1085  16.2310   3.8716 -12.4132 -17.4417  -6.3745  10.8269  18.5493
9.8738  -7.0712 -16.5894
 Columns 67 through 77
  -9.8533   6.9768  18.4151  13.8883  -2.5409 -15.9045 -14.0849   1.0517  15.3793
15.5091   1.1079
 Columns 78 through 88
 -14.7857 -17.7403  -5.1920  11.2050  16.2985   5.3724 -11.5159 -18.7828  -
9.6482   7.6262  17.3271
 Columns 89 through 99
  10.7286  -5.8934 -17.0404 -12.2502   4.2751  17.5235  15.4674   0.1146 -
14.3422 -14.5781  -0.3878
 Column 100
  15.1259
```

```
g =
  Columns 1 through 11
       0       0  0.5875  0.7429  0.8658  0.9509  0.9944  0.9946  0.9514
0.8666  0.7439
  Columns 12 through 22
   0.5888  0.4080  0.2094  0.0016 -0.2062 -0.4051 -0.5862 -0.7418 -0.8650 -
0.9504 -0.9943
  Columns 23 through 33
  -0.9948 -0.9518 -0.8673 -0.7450 -0.5901 -0.4095 -0.2109 -0.0032  0.2047
0.4036  0.5849
  Columns 34 through 44
   0.7407  0.8642  0.9499  0.9941  0.9949  0.9523  0.8681  0.7461  0.5914
0.4109  0.2125
  Columns 45 through 55
   0.0048 -0.2031 -0.4022 -0.5837 -0.7397 -0.8634 -0.9494 -0.9939 -0.9951
-0.9528 -0.8689
  Columns 56 through 66
  -0.7471 -0.5927 -0.4124 -0.2140 -0.0064  0.2016  0.4007  0.5824  0.7386
0.8626  0.9489
  Columns 67 through 77
   0.9938  0.9953  0.9533  0.8697  0.7482  0.5940  0.4138  0.2156  0.0080 -
0.2000 -0.3993
  Columns 78 through 88
  -0.5811 -0.7375 -0.8617 -0.9484 -0.9936 -0.9954 -0.9538 -0.8705 -0.7492
-0.5952 -0.4153
  Columns 89 through 99
  -0.2171 -0.0096  0.1985  0.3978  0.5798  0.7364  0.8609  0.9479  0.9934
0.9956  0.9543
  Column 100
   0.8713
out =
  Columns 1 through 11
       0       0 -9.2992 -10.5473  2.0178  15.8131  15.8635  1.3967 -13.7452 -
15.1126 -1.4689
  Columns 12 through 22
   14.3108  17.4185  4.8127 -12.0693 -17.8972 -7.5361  9.2716  16.8887  8.1646
-8.9924 -18.8836
  Columns 23 through 33
  -12.4482  4.4097  16.2492  12.2855 -3.6993 -16.8393 -14.8601  0.6282
15.6022  16.5083  2.7149
  Columns 34 through 44
  -12.9158 -15.8612 -3.2969  13.3015  18.7057  7.9340 -9.1677 -16.9760 -
8.4495  8.4033  17.8945
  Columns 45 through 55
   11.0883 -5.9740 -17.8189 -13.7579  2.2945  15.4279  13.4506 -1.8956 -
16.5340 -16.9935 -2.7943
  Columns 56 through 66
   13.1085  16.2310  3.8716 -12.4132 -17.4417 -6.3745  10.8269  18.5493
9.8738 -7.0712 -16.5894
```

```
Columns 67 through 77
  -9.8533   6.9768  18.4151  13.8883  -2.5409 -15.9045 -14.0849   1.0517  15.3793
15.5091   1.1079
 Columns 78 through 88
 -14.7857 -17.7403  -5.1920  11.2050  16.2985   5.3724 -11.5159 -18.7828  -
9.6482   7.6262  17.3271
 Columns 89 through 99
  10.7286  -5.8934 -17.0404 -12.2502   4.2751  17.5235  15.4674   0.1146  -
14.3422 -14.5781  -0.3878
 Column 100
  15.1259
g =
 Columns 1 through 11
       0       0  0.5875  0.7429  0.8658  0.9509  0.9944  0.9946  0.9514
0.8666   0.7439
 Columns 12 through 22
  0.5888   0.4080   0.2094   0.0016  -0.2062  -0.4051  -0.5862  -0.7418  -0.8650  -
0.9504  -0.9943
 Columns 23 through 33
  -0.9948  -0.9518  -0.8673  -0.7450  -0.5901  -0.4095  -0.2109  -0.0032   0.2047
0.4036   0.5849
 Columns 34 through 44
  0.7407   0.8642   0.9499   0.9941   0.9949   0.9523   0.8681   0.7461   0.5914
0.4109   0.2125
 Columns 45 through 55
  0.0048  -0.2031  -0.4022  -0.5837  -0.7397  -0.8634  -0.9494  -0.9939  -0.9951
-0.9528  -0.8689
 Columns 56 through 66
  -0.7471  -0.5927  -0.4124  -0.2140  -0.0064   0.2016   0.4007   0.5824   0.7386
0.8626   0.9489
 Columns 67 through 77
   0.9938   0.9953   0.9533   0.8697   0.7482   0.5940   0.4138   0.2156   0.0080  -
0.2000  -0.3993
 Columns 78 through 88
  -0.5811  -0.7375  -0.8617  -0.9484  -0.9936  -0.9954  -0.9538  -0.8705  -0.7492
-0.5952  -0.4153
 Columns 89 through 99
  -0.2171  -0.0096   0.1985   0.3978   0.5798   0.7364   0.8609   0.9479   0.9934
0.9956   0.9543
 Column 100
   0.8713
out =
 Columns 1 through 11
       0       0  -9.2992 -10.5473   2.0178  15.8131  15.8635   1.3967 -13.7452  -
15.1126  -1.4689
 Columns 12 through 22
  14.3108  17.4185   4.8127 -12.0693 -17.8972  -7.5361   9.2716  16.8887   8.1646
-8.9924 -18.8836
 Columns 23 through 33
```

```
-12.4482   4.4097  16.2492  12.2855  -3.6993 -16.8393 -14.8601   0.6282  15.6022
16.5083   2.7149
 Columns 34 through 44
 -12.9158 -15.8612  -3.2969  13.3015  18.7057   7.9340  -9.1677 -16.9760  -
8.4495   8.4033  17.8945
 Columns 45 through 55
  11.0883  -5.9740 -17.8189 -13.7579   2.2945  15.4279  13.4506  -1.8956  -
16.5340 -16.9935  -2.7943
 Columns 56 through 66
  13.1085  16.2310   3.8716 -12.4132 -17.4417  -6.3745  10.8269  18.5493
9.8738  -7.0712 -16.5894
 Columns 67 through 77
  -9.8533   6.9768  18.4151  13.8883  -2.5409 -15.9045 -14.0849   1.0517  15.3793
15.5091   1.1079
 Columns 78 through 88
 -14.7857 -17.7403  -5.1920  11.2050  16.2985   5.3724 -11.5159 -18.7828  -
9.6482   7.6262  17.3271
 Columns 89 through 99
  10.7286  -5.8934 -17.0404 -12.2502   4.2751  17.5235  15.4674   0.1146  -
14.3422 -14.5781  -0.3878
 Column 100
  15.1259
g =
 Columns 1 through 11
      0       0   0.5875   0.7429   0.8658   0.9509   0.9944   0.9946   0.9514
0.8666   0.7439
 Columns 12 through 22
   0.5888   0.4080   0.2094   0.0016  -0.2062  -0.4051  -0.5862  -0.7418  -0.8650  -
0.9504  -0.9943
 Columns 23 through 33
  -0.9948  -0.9518  -0.8673  -0.7450  -0.5901  -0.4095  -0.2109  -0.0032   0.2047
0.4036   0.5849
 Columns 34 through 44
   0.7407   0.8642   0.9499   0.9941   0.9949   0.9523   0.8681   0.7461   0.5914
0.4109   0.2125
 Columns 45 through 55
   0.0048  -0.2031  -0.4022  -0.5837  -0.7397  -0.8634  -0.9494  -0.9939  -0.9951
-0.9528  -0.8689
 Columns 56 through 66
  -0.7471  -0.5927  -0.4124  -0.2140  -0.0064   0.2016   0.4007   0.5824   0.7386
0.8626   0.9489
 Columns 67 through 77
   0.9938   0.9953   0.9533   0.8697   0.7482   0.5940   0.4138   0.2156   0.0080  -
0.2000  -0.3993
 Columns 78 through 88
  -0.5811  -0.7375  -0.8617  -0.9484  -0.9936  -0.9954  -0.9538  -0.8705  -0.7492
-0.5952  -0.4153
 Columns 89 through 99
```

-0.2171 -0.0096 0.1985 0.3978 0.5798 0.7364 0.8609 0.9479 0.9934
0.9956 0.9543

Column 100

0.8713

out =

Columns 1 through 11

0 0 -9.2992 -10.5473 2.0178 15.8131 15.8635 1.3967 -13.7452 -
15.1126 -1.4689

Columns 12 through 22

14.3108 17.4185 4.8127 -12.0693 -17.8972 -7.5361 9.2716 16.8887 8.1646
-8.9924 -18.8836

Columns 23 through 33

-12.4482 4.4097 16.2492 12.2855 -3.6993 -16.8393 -14.8601 0.6282
15.6022 16.5083 2.7149

Columns 34 through 44

-12.9158 -15.8612 -3.2969 13.3015 18.7057 7.9340 -9.1677 -16.9760 -
8.4495 8.4033 17.8945

Columns 45 through 55

11.0883 -5.9740 -17.8189 -13.7579 2.2945 15.4279 13.4506 -1.8956 -
16.5340 -16.9935 -2.7943

Columns 56 through 66

13.1085 16.2310 3.8716 -12.4132 -17.4417 -6.3745 10.8269 18.5493
9.8738 -7.0712 -16.5894

Columns 67 through 77

-9.8533 6.9768 18.4151 13.8883 -2.5409 -15.9045 -14.0849 1.0517 15.3793
15.5091 1.1079

Columns 78 through 88

-14.7857 -17.7403 -5.1920 11.2050 16.2985 5.3724 -11.5159 -18.7828 -
9.6482 7.6262 17.3271

Columns 89 through 99

10.7286 -5.8934 -17.0404 -12.2502 4.2751 17.5235 15.4674 0.1146 -
14.3422 -14.5781 -0.3878

Column 100

15.1259

g =

Columns 1 through 11

0 0 0.5875 0.7429 0.8658 0.9509 0.9944 0.9946 0.9514
0.8666 0.7439

Columns 12 through 22

0.5888 0.4080 0.2094 0.0016 -0.2062 -0.4051 -0.5862 -0.7418 -0.8650 -
0.9504 -0.9943

Columns 23 through 33

-0.9948 -0.9518 -0.8673 -0.7450 -0.5901 -0.4095 -0.2109 -0.0032 0.2047
0.4036 0.5849

Columns 34 through 44

0.7407 0.8642 0.9499 0.9941 0.9949 0.9523 0.8681 0.7461 0.5914
0.4109 0.2125

Columns 45 through 55

```
  0.0048  -0.2031  -0.4022  -0.5837  -0.7397  -0.8634  -0.9494  -0.9939  -0.9951  -
0.9528  -0.8689
  Columns 56 through 66
  -0.7471  -0.5927  -0.4124  -0.2140  -0.0064   0.2016   0.4007   0.5824   0.7386
0.8626   0.9489
  Columns 67 through 77
   0.9938   0.9953   0.9533   0.8697   0.7482   0.5940   0.4138   0.2156   0.0080  -
0.2000  -0.3993
  Columns 78 through 88
  -0.5811  -0.7375  -0.8617  -0.9484  -0.9936  -0.9954  -0.9538  -0.8705  -0.7492
-0.5952  -0.4153
  Columns 89 through 99
  -0.2171  -0.0096   0.1985   0.3978   0.5798   0.7364   0.8609   0.9479   0.9934
0.9956   0.9543
  Column 100
   0.8713
out =
  Columns 1 through 11
       0       0  -9.2992 -10.5473   2.0178  15.8131  15.8635   1.3967 -13.7452 -
15.1126  -1.4689
  Columns 12 through 22
  14.3108  17.4185   4.8127 -12.0693 -17.8972  -7.5361   9.2716  16.8887   8.1646
-8.9924 -18.8836
  Columns 23 through 33
 -12.4482   4.4097  16.2492  12.2855  -3.6993 -16.8393 -14.8601   0.6282
15.6022  16.5083   2.7149
  Columns 34 through 44
 -12.9158 -15.8612  -3.2969  13.3015  18.7057   7.9340  -9.1677 -16.9760  -
8.4495   8.4033  17.8945
  Columns 45 through 55
  11.0883  -5.9740 -17.8189 -13.7579   2.2945  15.4279  13.4506  -1.8956 -
16.5340 -16.9935  -2.7943
  Columns 56 through 66
  13.1085  16.2310   3.8716 -12.4132 -17.4417  -6.3745  10.8269  18.5493
9.8738  -7.0712 -16.5894
  Columns 67 through 77
  -9.8533   6.9768  18.4151  13.8883  -2.5409 -15.9045 -14.0849   1.0517  15.3793
15.5091   1.1079
  Columns 78 through 88
 -14.7857 -17.7403  -5.1920  11.2050  16.2985   5.3724 -11.5159 -18.7828  -
9.6482   7.6262  17.3271
  Columns 89 through 99
  10.7286  -5.8934 -17.0404 -12.2502   4.2751  17.5235  15.4674   0.1146 -
14.3422 -14.5781  -0.3878
  Column 100
  15.1259
>>
input =
```

0	0
0	0
29.3764	0
37.1430	-9.2992
43.2880	-22.5126
47.5430	-29.3280
49.7222	-24.1305
49.7305	-11.3034
47.5676	-1.3633
43.3278	-1.4805
37.1963	-9.0749
29.4408	-14.3582
20.3999	-9.5024
10.4683	3.9713
0.0796	16.3860
-10.3125	18.6291
-20.2544	10.4132
-29.3119	0.4659
-37.0897	-1.5615
-43.2481	5.9609
-47.5183	15.1045
-49.7138	15.7069
-49.7387	4.8266
-47.5920	-10.4045
-43.3675	-19.1846
-37.2495	-16.8135
-29.5051	-8.8328
-20.4725	-5.7417
-10.5461	-13.1859
-0.1593	-26.6640
10.2346	-35.5991
20.1815	-32.9932
29.2473	-21.7822
37.0362	-12.0838
43.2081	-11.9058
47.4934	-19.8516
49.7052	-26.4869
49.7468	-23.0814
47.6164	-9.7010
43.4071	4.7591
37.3025	10.6252
29.5694	6.2025
20.5452	-0.7930
10.6240	-0.4587
0.2389	10.0305
-10.1566	23.6423
-20.1086	29.7909
-29.1827	23.9744
-36.9827	11.8819

-43.1679	4.1567
-47.4685	6.6088
-49.6965	14.8959
-49.7547	18.6022
-47.6406	10.9688
-43.4465	-4.7048
-37.3555	-17.9092
-29.6335	-20.4068
-20.6177	-13.6154
-10.7018	-7.1127
-0.3185	-9.6771
10.0786	-21.1317
20.0357	-32.4744
29.1180	-34.1289
36.9290	-24.7096
43.1277	-12.2665
47.4434	-6.9065
49.6877	-11.5808
49.7625	-19.4745
47.6647	-20.3672
43.4859	-10.1210
37.4084	5.3988
29.6976	15.5695
20.6903	14.6254
10.7795	6.8250
0.3982	2.4119
-10.0006	8.0524
-19.9627	20.5385
-29.0533	29.5881
-36.8753	27.2190
-43.0874	15.0883
-47.4182	3.0148
-49.6787	0.0091
-49.7702	6.0654
-47.6887	12.2903
-43.5251	9.2564
-37.4612	-4.1311
-29.7617	-19.4436
-20.7627	-26.3114
-10.8573	-21.7888
-0.4778	-12.8922
9.9226	-10.0141
19.8897	-17.3045
28.9884	-28.8557
36.8215	-34.1518
43.0469	-27.7375
47.3929	-14.2315
49.6696	-4.1136
49.7778	-4.1749

```
  47.7126  -11.3873
  43.5643  -15.7988
```

trn_Data =

```
         0         0         0
         0         0         0
   29.3764         0         0
   37.1430   -9.2992   -9.2992
   43.2880  -22.5126  -22.5126
   47.5430  -29.3280  -29.3280
   49.7222  -24.1305  -24.1305
   49.7305  -11.3034  -11.3034
   47.5676   -1.3633   -1.3633
   43.3278   -1.4805   -1.4805
   37.1963   -9.0749   -9.0749
   29.4408  -14.3582  -14.3582
   20.3999   -9.5024   -9.5024
   10.4683    3.9713    3.9713
    0.0796   16.3860   16.3860
  -10.3125   18.6291   18.6291
  -20.2544   10.4132   10.4132
  -29.3119    0.4659    0.4659
  -37.0897   -1.5615   -1.5615
  -43.2481    5.9609    5.9609
  -47.5183   15.1045   15.1045
  -49.7138   15.7069   15.7069
  -49.7387    4.8266    4.8266
  -47.5920  -10.4045  -10.4045
  -43.3675  -19.1846  -19.1846
  -37.2495  -16.8135  -16.8135
  -29.5051   -8.8328   -8.8328
  -20.4725   -5.7417   -5.7417
  -10.5461  -13.1859  -13.1859
   -0.1593  -26.6640  -26.6640
   10.2346  -35.5991  -35.5991
   20.1815  -32.9932  -32.9932
   29.2473  -21.7822  -21.7822
   37.0362  -12.0838  -12.0838
   43.2081  -11.9058  -11.9058
   47.4934  -19.8516  -19.8516
   49.7052  -26.4869  -26.4869
   49.7468  -23.0814  -23.0814
   47.6164   -9.7010   -9.7010
   43.4071    4.7591    4.7591
   37.3025   10.6252   10.6252
   29.5694    6.2025    6.2025
   20.5452   -0.7930   -0.7930
```

10.6240	-0.4587	-0.4587
0.2389	10.0305	10.0305
-10.1566	23.6423	23.6423
-20.1086	29.7909	29.7909
-29.1827	23.9744	23.9744
-36.9827	11.8819	11.8819
-43.1679	4.1567	4.1567
-47.4685	6.6088	6.6088
-49.6965	14.8959	14.8959
-49.7547	18.6022	18.6022
-47.6406	10.9688	10.9688
-43.4465	-4.7048	-4.7048
-37.3555	-17.9092	-17.9092
-29.6335	-20.4068	-20.4068
-20.6177	-13.6154	-13.6154
-10.7018	-7.1127	-7.1127
-0.3185	-9.6771	-9.6771
10.0786	-21.1317	-21.1317
20.0357	-32.4744	-32.4744
29.1180	-34.1289	-34.1289
36.9290	-24.7096	-24.7096
43.1277	-12.2665	-12.2665
47.4434	-6.9065	-6.9065
49.6877	-11.5808	-11.5808
49.7625	-19.4745	-19.4745
47.6647	-20.3672	-20.3672
43.4859	-10.1210	-10.1210
37.4084	5.3988	5.3988
29.6976	15.5695	15.5695
20.6903	14.6254	14.6254
10.7795	6.8250	6.8250
0.3982	2.4119	2.4119
-10.0006	8.0524	8.0524
-19.9627	20.5385	20.5385
-29.0533	29.5881	29.5881
-36.8753	27.2190	27.2190
-43.0874	15.0883	15.0883
-47.4182	3.0148	3.0148
-49.6787	0.0091	0.0091
-49.7702	6.0654	6.0654
-47.6887	12.2903	12.2903
-43.5251	9.2564	9.2564
-37.4612	-4.1311	-4.1311
-29.7617	-19.4436	-19.4436
-20.7627	-26.3114	-26.3114
-10.8573	-21.7888	-21.7888
-0.4778	-12.8922	-12.8922
9.9226	-10.0141	-10.0141
19.8897	-17.3045	-17.3045

```
28.9884  -28.8557  -28.8557
36.8215  -34.1518  -34.1518
43.0469  -27.7375  -27.7375
47.3929  -14.2315  -14.2315
49.6696   -4.1136   -4.1136
49.7778   -4.1749   -4.1749
47.7126  -11.3873  -11.3873
43.5643  -15.7988  -15.7988
```

ANFIS info:

Number of nodes: 131
Number of linear parameters: 147
Number of nonlinear parameters: 42
Total number of parameters: 189
Number of training data pairs: 100
Number of checking data pairs: 0
Number of fuzzy rules: 49

Warning: number of data is smaller than number of modifiable parameters

Start training ANFIS ...

```
1    5.30644e-005
2    5.31178e-005
3    5.29964e-005
4    5.29912e-005
5    5.28154e-005
6    5.26248e-005
```

Step size increases to 0.011000 after epoch 6.

```
7    5.25697e-005
8    5.25457e-005
9    5.25443e-005
10   5.26051e-005
11   5.24986e-005
12   5.24062e-005
13   5.22334e-005
14   5.22664e-005
15   5.21349e-005
16   5.193e-005
17   5.20346e-005
18   5.19601e-005
19   5.16698e-005
20   5.17142e-005
```

Designated epoch number reached --> ANFIS training completed at epoch 20.

APPENDIX 3

Program code for predicting exchange rate

```
load usd2002.txt

t = usd2002(:, 1);
x = usd2002(:, 2);

plot(t, x);
w(t) = [x(t-18) x(t-12) x(t-6) x(t)]
s(t) = x(t+6)

for t=20:902
Data(t-19,:)=[x(t-18) x(t-12) x(t-6) x(t) x(t+6)];
end

trnData=Data(1:460, :);
chkData=Data(461:end, :);
fismat = genfis1(trnData);

subplot(2,2,1)
plotmf(fismat, 'input', 1)
subplot(2,2,2)
plotmf(fismat, 'input', 2)
subplot(2,2,3)
plotmf(fismat, 'input', 3)
subplot(2,2,4)
plotmf(fismat, 'input', 4)

fismat2=anfis(trnData,fismat,[],[],chkData);
[fismat,error1,ss,fismat2,error2] = anfis(trnData,fismat,[],[],chkData);

subplot(2,2,1)
plotmf(fismat2, 'input', 1)
subplot(2,2,2)
plotmf(fismat2, 'input', 2)
subplot(2,2,3)
plotmf(fismat2, 'input', 3)
subplot(2,2,4)
plotmf(fismat2, 'input', 4)

plot([error1; error2]);

anfis_output = evalfis([trnData, chkData],fismat2);
plot(anfis_output);
```

APPENDIX 4

Program execution results for predicting exchange rate

ANFIS info:

Number of nodes: 55
Number of linear parameters: 80
Number of nonlinear parameters: 24
Total number of parameters: 104
Number of training data pairs: 460
Number of checking data pairs: 0
Number of fuzzy rules: 16

Start training ANFIS ...

1	31.4343
2	31.6357
3	32.4524
4	31.7551
5	31.5946
6	31.9037
7	32.5328
8	32.3151
9	32.358
10	31.4373

Step size decreases to 0.009000 after epoch 10.

Designated epoch number reached --> ANFIS training completed at epoch 10.

ANFIS info:

Number of nodes: 55
Number of linear parameters: 80
Number of nonlinear parameters: 24
Total number of parameters: 104
Number of training data pairs: 460
Number of checking data pairs: 423
Number of fuzzy rules: 16

Start training ANFIS ...

1	31.4343	439.618
2	31.6357	484.197
3	32.4524	345.91
4	31.7551	281.453
5	31.5946	272.165
6	31.9037	236.305

7	32.5328	728.101
8	32.3151	580.387
9	32.358	637.682
10	31.4373	348.04

Step size decreases to 0.009000 after epoch 10.

Designated epoch number reached --> ANFIS training completed at epoch 10.

APPENDIX 5

Input data for predicting exchange rate

1 543,401	46 563,657	91 595,334
2 538,399	47 567,732	92 593,798
3 537,722	48 566,698	93 595,907
4 537,738	49 568,865	94 598,363
5 542,212	50 570,505	95 599,552
6 541,876	51 575,799	96 601,865
7 542,939	52 574,816	97 602,256
8 543,012	53 576,598	98 602,653
9 543,354	54 575,055	99 602,641
10 545,344	55 577,193	100 602,124
11 548,869	56 577,863	101 607,010
12 549,036	57 578,357	102 607,857
13 548,880	58 579,558	103 608,356
14 549,510	59 580,949	104 608,773
15 549,855	60 581,854	105 609,226
16 550,266	61 580,570	106 612,243
17 550,926	62 581,089	107 614,154
18 552,974	63 579,822	108 613,515
19 552,985	64 580,547	109 619,285
20 553,451	65 581,882	110 619,035
21 555,949	66 582,185	111 618,992
22 558,441	67 582,651	112 618,859
23 560,572	68 583,273	113 618,040
24 561,067	69 586,145	114 620,122
25 561,541	70 584,726	115 619,725
26 562,503	71 584,216	116 616,246
27 563,577	72 584,468	117 617,279
28 563,063	73 584,762	118 617,774
29 562,082	74 584,902	119 618,523
30 542,212	75 586,896	120 618,990
31 561,553	76 589,712	121 616,863
32 560,587	77 591,155	122 620,224
33 560,367	78 590,896	123 623,533
34 561,171	79 591,446	124 623,929
35 562,924	80 591,875	125 623,553
36 564,686	81 592,357	126 622,254
37 564,787	82 592,698	127 621,698
38 566,680	83 588,901	128 620,758
39 567,012	84 591,527	129 621,658
40 566,894	85 593,198	130 623,975
41 566,325	86 592,963	131 620,733
42 565,137	87 593,585	132 619,008
43 567,277	88 594,015	133 616,624
44 568,035	89 594,654	134 617,025
45 564,085	90 594,972	135 617,542

136 617,706
137 617,665
138 615,009
139 616,294
140 613,765
141 613,838
142 611,064
143 612,587
144 614,877
145 616,388
146 615,948
147 614,004
148 614,267
149 617,426
150 616,321
151 616,144
152 615,489
153 615,723
154 618,064
155 620,973
156 622,459
157 622,976
158 625,927
159 625,532
160 625,214
161 624,991
162 624,784
163 621,970
164 621,079
165 624,013
166 624,810
167 624,632
168 624,145
169 623,313
170 623,407
171 625,777
172 625,112
173 624,956
174 626,440
175 628,478
176 629,998
177 630,319
178 631,288
179 631,596
180 631,803
181 635,823
182 635,687
183 633,898
184 633,369
185 632,990
186 634,122
187 632,234
188 632,808
189 633,496
190 637,265
191 637,866
192 638,224
193 638,579
194 637,812
195 641,913
196 645,329
197 644,486
198 644,824
199 645,869
200 646,589
201 647,086
202 648,505
203 646,950
204 645,111
205 648,441
206 646,392
207 646,536
208 646,836
209 647,335
210 645,528
211 648,022
212 650,276
213 652,711
214 653,459
215 652,754
216 652,115
217 651,178
218 651,479
219 652,946
220 654,469
221 656,599
222 656,282
223 655,695
224 655,235
225 655,142
226 659,098
227 662,199
228 664,499
229 665,214
230 669,036
231 668,965
232 668,364
233 668,549
234 668,126
235 668,229
236 669,413
237 673,206
238 672,870
239 673,568
240 674,368
241 675,795
242 674,358
243 664,304
244 668,621
245 667,803
246 666,407
247 666,489
248 666,563
249 666,633
250 669,096
251 668,922
252 670,307
253 671,046
254 671,157
255 671,261
256 672,809
257 673,952
258 673,486
259 673,206
260 674,647
261 675,850
262 676,411
263 681,063
264 681,536
265 681,057
266 682,865
267 684,650
268 684,646
269 686,620
270 686,744
271 689,772
272 690,886
273 691,260
274 689,082
275 685,342
276 685,978
277 683,242
278 682,891
279 681,318
280 679,186
281 680,710
282 683,524

283 684,131
284 684,637
285 685,248
286 681,676
287 683,934
288 685,399
289 685,087
290 686,822
291 686,235
292 686,942
293 689,771
294 690,322
295 691,679
296 691,746
297 692,171
298 692,365
299 692,536
300 689,069
301 686,176
302 685,669
303 685,390
304 684,316
305 679,500
306 682,680
307 682,149
308 680,156
309 678,636
310 680,373
311 684,404
312 685,285
313 686,105
314 684,367
315 683,871
316 679,359
317 680,943
318 682,904
319 678,074
320 676,435
321 675,474
322 673,198
323 675,004
324 669,989
325 667,028
326 668,532
327 669,749
328 666,940
329 669,092
330 671,705
331 671,990
332 669,365
333 668,899
334 668,706
335 672,556
336 673,006
337 673,772
338 673,425
339 673,921
340 674,071
341 678,201
342 674,851
343 678,319
344 682,528
345 681,463
346 680,514
347 682,086
348 682,437
349 679,693
350 676,298
351 676,359
352 676,453
353 676,620
354 679,778
355 679,965
356 683,952
357 683,511
358 683,433
359 682,277
360 682,833
361 685,978
362 689,306
363 688,754
364 688,342
365 686,368
366 691,319
367 688,696
368 962,499
369 985,237
370 1078,163
371 950,870
372 910,535
373 925,119
374 920,125
375 913,265
376 908,080
377 891,886
378 927,802
379 946,060
380 1004,610
381 1008,365
382 1011,891
383 998,826
384 991,675
385 976,046
386 976,687
387 980,125
388 984,275
389 981,075
390 981,668
391 986,706
392 997,140
393 999,235
394 1061,640
395 1025,482
396 1120,115
397 1238,549
398 1234,287
399 1225,367
400 1204,111
401 1187,780
402 1242,267
403 1285,000
404 1285,029
405 1273,267
406 1261,905
407 1202,316
408 1174,055
409 1221,681
410 1258,961
411 1225,967
412 1230,227
413 1234,280
414 1233,550
415 1219,040
416 1230,714
417 1228,369
418 1166,164
419 1143,180
420 1143,108
421 1163,683
422 1165,563
423 1158,379
424 1140,763
425 1138,453
426 1145,519
427 1151,929
428 1140,438
429 1146,931

430 1150,881
431 1151,087
432 1145,376
433 1130,931
434 1113,130
435 1111,398
436 1108,380
437 1115,450
438 1109,597
439 1115,881
440 1116,972
441 1112,934
442 1108,908
443 1121,917
444 1127,140
445 1162,108
446 1211,964
447 1210,364
448 1150,289
449 1148,218
450 1171,484
451 1172,322
452 1168,170
453 1165,324
454 1164,673
455 1197,645
456 1202,064
457 1229,310
458 1196,190
459 1195,532
460 1195,919
461 1224,471
462 1242,264
463 1254,923
464 1280,046
465 1278,231
466 1276,285
467 1299,871
468 1271,315
469 1251,561
470 1273,528
471 1265,109
472 1258,815
473 1257,573
474 1276,020
475 1278,340
476 1280,137
477 1280,716
478 1280,900
479 1311,545
480 1335,791
481 1307,311
482 1308,675
483 1327,364
484 1342,443
485 1334,059
486 1331,844
487 1350,452
488 1380,654
489 1377,153
490 1373,239
491 1489,722
492 1372,145
493 1335,240
494 1315,386
495 1319,325
496 1321,746
497 1311,868
498 1322,780
499 1318,452
500 1325,085
501 1325,725
502 1326,247
503 1329,490
504 1332,716
505 1340,562
506 1340,750
507 1346,237
508 1352,395
509 1359,541
510 1374,437
511 1382,951
512 1385,669
513 1392,125
514 1400,569
515 1472,125
516 1485,666
517 1460,982
518 1423,666
519 1424,512
520 1425,306
521 1439,136
522 1460,233
523 1472,546
524 1448,375
525 1437,125
526 1425,056
527 1407,650
528 1384,508
529 1426,491
530 1414,821
531 1402,036
532 1396,719
533 1407,684
534 1428,667
535 1428,868
536 1452,115
537 1466,003
538 1472,809
539 1488,685
540 1543,265
541 1497,273
542 1498,214
543 1510,236
544 1521,963
545 1534,343
546 1557,836
547 1554,517
548 1546,960
549 1543,221
550 1540,363
551 1525,991
552 1550,160
553 1567,000
554 1581,698
555 1587,206
556 1592,750
557 1582,894
558 1619,750
559 1628,391
560 1644,826
561 1597,894
562 1595,235
563 1592,360
564 1603,894
565 1630,096
566 1635,306
567 1630,575
568 1638,253
569 1644,837
570 1629,555
571 1611,111
572 1605,836
573 1616,500
574 1605,117
575 1599,526
576 1597,394

577 1595,060
578 1573,196
579 1568,130
580 1560,339
581 1557,818
582 1541,388
583 1554,482
584 1545,285
585 1557,250
586 1559,332
587 1561,651
588 1563,640
589 1540,375
590 1522,657
591 1510,802
592 1510,723
593 1478,375
594 1464,400
595 1491,934
596 1499,574
597 1491,875
598 1486,123
599 1479,102
600 1476,394
601 1481,989
602 1491,705
603 1481,078
604 1485,334
605 1489,445
606 1477,817
607 1456,673
608 1453,381
609 1451,654
610 1448,102
611 1445,375
612 1436,956
613 1430,069
614 1408,944
615 1434,750
616 1455,229
617 1468,466
618 1480,675
619 1477,739
620 1471,750
621 1450,945
622 1453,224
623 1455,804
624 1440,112
625 1446,510
626 1453,615
627 1440,765
628 1432,254
629 1425,409
630 1433,811
631 1390,232
632 1385,565
633 1393,429
634 1383,215
635 1381,578
636 1372,769
637 1356,638
638 1361,291
639 1364,073
640 1360,568
641 1358,668
642 1339,245
643 1318,841
644 1328,088
645 1343,596
646 1341,225
647 1340,878
648 1324,602
649 1323,251
650 1313,626
651 1319,914
652 1310,009
653 1304,426
654 1298,428
655 1306,493
656 1301,531
657 1321,944
658 1325,254
659 1327,444
660 1368,243
661 1375,066
662 1368,280
663 1348,944
664 1340,226
665 1338,274
666 1345,473
667 1347,274
668 1362,157
669 1367,442
670 1375,113
671 1386,926
672 1398,336
673 1385,081
674 1382,002
675 1375,490
676 1374,912
677 1374,476
678 1372,226
679 1362,198
680 1354,969
681 1358,364
682 1355,125
683 1354,891
684 1348,950
685 1339,325
686 1343,246
687 1356,732
688 1350,225
689 1340,871
690 1339,500
691 1331,653
692 1338,716
693 1335,466
694 1331,254
695 1327,304
696 1331,556
697 1336,858
698 1342,108
699 1329,304
700 1328,125
701 1327,729
702 1312,019
703 1304,412
704 1305,440
705 1303,896
706 1301,254
707 1298,487
708 1289,301
709 1285,642
710 1297,629
711 1287,062
712 1290,562
713 1297,697
714 1297,573
715 1298,143
716 1311,450
717 1330,488
718 1338,214
719 1348,952
720 1351,892
721 1330,530
722 1330,952
723 1349,780

724 1560,235
725 1361,140
726 1355,733
727 1370,051
728 1365,905
729 1366,977
730 1369,254
731 1372,539
732 1373,129
733 1377,243
734 1386,393
735 1384,448
736 1389,968
737 1392,524
738 1406,327
739 1423,232
740 1412,184
741 1404,483
742 1407,125
743 1409,200
744 1403,817
745 1398,823
746 1394,774
747 1409,001
748 1410,236
749 1411,904
750 1437,003
751 1454,341
752 1435,307
753 1431,420
754 1432,532
755 1432,968
756 1463,240
757 1486,916
758 1528,800
759 1541,562
760 1541,932
761 1542,768
762 1580,113
763 1580,138
764 1562,306
765 1529,725
766 1544,235
767 1564,821
768 1569,237
769 1570,764
770 1626,946
771 1568,045
772 1570,235
773 1572,051
774 1583,055
775 1633,240
776 1612,775
777 1611,875
778 1638,235
779 1653,231
780 1663,475
781 1654,240
782 1643,216
783 1654,000
784 1667,256
785 1670,235
786 1675,982
787 1677,583
788 1658,637
789 1646,313
790 1659,858
791 1662,124
792 1668,939
793 1667,854
794 1678,492
795 1676,905
796 1675,558
797 1682,124
798 1687,228
799 1680,690
800 1687,012
801 1674,611
802 1645,701
803 1625,228
804 1619,936
805 1642,103
806 1629,037
807 1618,360
808 1632,896
809 1635,254
810 1639,263
811 1630,843
812 1628,532
813 1639,432
814 1630,036
815 1632,029
816 1634,846
817 1625,233
818 1621,634
819 1621,965
820 1628,960
821 1626,225
822 1624,840
823 1622,647
824 1624,680
825 1620,212
826 1618,188
827 1615,223
828 1611,923
829 1620,495
830 1617,705
831 1614,368
832 1627,738
833 1635,214
834 1657,662
835 1654,375
836 1653,006
837 1654,486
838 1661,493
839 1661,865
840 1662,005
841 1662,005
842 1658,848
843 1650,636
844 1647,908
845 1648,963
846 1651202
847 1644,439
848 1647,106
849 1647,515
850 1654,981
851 1645,235
852 1636,022
853 1629,348
854 1629,578
855 1638,146
856 1640,427
857 1639,564
858 1638,102
859 1637,792
860 1626,368
861 1639,828
862 1637,487
863 1636,945
864 1636,447
865 1641,156
866 1646,201
867 1653,949
868 1648,601
869 1648,521
870 1535,246