Neuro-Fuzzy Modeling:
Architecture, Analyses and Applications

by

Jyh-Shing Roger Jang

Neuro-Fuzzy Modeling:
Architectures, Analyses and Applications

Jyh-Shing Roger Jang

Department of Electrical Engineering and Computer Science

University of California

Berkeley, CA 94720

July 1992

A dissertation

submitted in partial satisfaction of

the requirements for the degree of

Doctor of Philosophy in Electrical Engineering and Computer Science in

the Graduate Division of

the University of California, Berkeley.

Abstract

Fuzzy inference systems have been successfully applied in various areas ranging from automatic control to expert systems. This success is mostly due to the fact that fuzzy if-then rules are well-suited for capturing the imprecise nature of human knowledge and reasoning processes. On the other hand, neural networks tackle the same problems with a different strategy; they are equipped with a remarkable learning capability such that a desired input-output mapping can be discovered through learning by examples. These two modeling approaches differ completely in the way the knowledge is acquired and encoded internally; thus their advantages and disadvantages are complementary.

The aim of this dissertation is to construct a generalized framework that integrates both neural networks and fuzzy inference systems. Models pertaining to this framework possess both the learning capability of neural networks and the structured knowledge representation employed in fuzzy inference systems. In particular, we propose two integrated models: ANFIS (Adaptive-Network-based Fuzzy Inference System) and NFC (Neuro-Fuzzy Classifier). The former is appropriate for nonlinear modeling, time series prediction and intelligent control; the latter is devised for pattern classification.

ANFIS (Adaptive-Network-based Fuzzy Inference System) is a fuzzy inference system with a hybrid learning strategy combining gradient descent and the Kalman filter algorithm. We prove that ANFIS is a universal approximator and its superior representational power is demonstrated through the applications to nonlinear function modeling and time series prediction. Furthermore, we propose a self-learning control strategy, called TBP (temporal back-propagation), which utilizes ANFIS as a controller. Simulation shows

the effectiveness of the proposed control scheme and the robustness of the acquired fuzzy controller in balancing an inverted pendulum.

NFC (Neuro-Fuzzy Classifier) is a distribution-free classifier which can encode prior knowledge into its parameters and fine-tune the parameters to get a better performance. The experiments of NFC on the two-spiral problem and the Iris data classification exhibit its advantages over pure neural network approaches in terms of structure transparency and model performance.

Besides the applications mentioned above, the proposed integrated models have various other promising applications encompassing both the application domains of neural networks and fuzzy inference systems. With the advance of VLSI technologies, we expect that neuro-fuzzy chips will be realized in the near future and play an increasingly important role in automation.

To Sui-Hwai, Ting-Shouh,

and our parents.

Acknowledgments

I would like to thank my research advisor, Professor Lotfi A. Zadeh, for being a consistent source of support and encouragement. His guidance and help have made my Ph.D. program a smooth and enjoyable one.

Several other professors have contributed graciously their time on my behalf, and I would like to express my gratitude. I would like to thank Professor Ron Fearing and Professor Masayoshi Tomizuka for their careful reading of this dissertation, and Professor Stuart Dreyfus for serving on my qualifying committee. Many discussions with them have generated enlightening ideas and have kept my work on track.

I gratefully acknowledge crucial contributions to this project from other members in the *fuzzy* group. They are: Albert Chen, Yung-Yaw Chen, Soumitra Dutta, Pratap Khedkar, Chuen-Chien Lee, Chuen-Tsai Sun, Hideyuki Takagi, and Lixin Wang. Special thanks must go to Yung-Yaw Chen, who introduced me to this exciting research area. Moreover, I am deeply appreciative of the constructive comments of Chuen-Tsai Sun and Chuen-Chien Lee on my qualifying exam. I also want to thank Pratap Khedkar for devoting his time in proofreading my papers.

I wish to express my appreciation to Dr. Hamid Berenji who let me join the project in NASA, which sparked my interest in the fuzzy control field. I also would like to express my gratitude to Dr. Chi Yung Fu, my supervisor at the Lawrence Livermore National Laboratory, from whom I learned a lot in both neural network technology and general research methodology.

I am deeply indebted to my mother, Ru-Jiuann Shih, and my uncles, grandfather and grandmother. Without their encouragement and financial support, I could have never come to the States to pursue a higher degree.

Finally, I would like to dedicate this work to my wife, Sui-Hwang Rosa Hwang, and my son, Ting-Shouh Tim Jang. Without Rosa's constant support and understanding and Tim's cooperation, I would not have had the persistence to finish this work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

System modeling has been an important issue in both engineering and non-engineering areas. Conventional approaches to system modeling rely heavily on mathematical tools which emphasize the precision and exact description of each quantity involved. The use of these mathematical tools (such as differential or difference equations, transfer functions, etc.) is appropriate and well-justified when the system is simple and/or well-defined. However, as the system under consideration (which could range from a mechanical system to a huge economic system) grows larger and more complicated, mathematical tools become less effective or even inappropriate. This is due to the fact that either the mathematical expressions themselves are too complicated to be tractable, or the relationships between variables in the system are unclear or known with uncertainty.

To overcome the problems confronted by conventional modeling methods, *fuzzy logic modeling* and *neural network modeling* have been proposed as viable alternatives and

successfully employed in various areas where conventional approaches fail to provide satisfactory solutions. These two innovative modeling approaches share some common characteristics: they assume parallel operations; they are well-known for their fault tolerance capabilities; and they are often called model-free modeling approaches. Despite these similarities, they stem from very different origins. Fuzzy logic modeling is primarily based on fuzzy sets and fuzzy if-then rules proposed by Zadeh [109], which are closely related to psychology and cognitive science. On the other hand, neural network modeling is based on artificial neural networks which are motivated by biological neural systems. Because of their very origins, the respective philosophies and methodologies underlying their problem-solving approaches are quite different and, in general, complementary. As a result, many researchers are trying to integrated these two schemes to generate hybrid models that can take advantage of the strong points of both. This is also the motivation for our research, which aims at providing an integrate framework capable of subsuming both neural networks and fuzzy inference systems.

1.2 Contribution of Dissertation

In this dissertation, we propose two hybrid architectures, ANFIS (Adaptive-Network-based Fuzzy Inference System) and NFC (Neuro-Fuzzy Classifier), that can encode *a priori* knowledge (which can assume various forms of fuzzy if-then rules) into their structures and utilize a fast hybrid learning rule to update their parameters based on a desired input-output data set. To be more specific, we have accomplished the following:

- Introduced the neural-network-type learning rule (the so-called *back propagation*) into different types of fuzzy inference systems, which results in the ANFIS architecture.
- Derived a fast learning algorithm that combines the back-propagation learning rule and the Kalman filter algorithm, which can speed up the learning processes of ANFIS's, CFN's and various types of neural networks.
- Extended BP (back propagation) to TBP (temporal back propagation), which gives fuzzy controllers self-learning capability.
- Proposed the NFC architecture which performs pattern classification based on fuzzy if-then rules with modifiable parameters.

Moreover, we have conducted extensive computer simulations to verify the proposed architectures and learning algorithms. We successfully employ the ANFIS architecture to model nonlinear functions and predict both chaotic and real-world time series. The effectiveness of TBP is confirmed by its ability to construct a self-learning fuzzy controller for balancing an inverted pendulum. We also demonstrate how the NFC architecture is suitable for solving two benchmark problems in classification.

1.3 Organization

Chapter 2 presents the notations, structures, operations and previous work on fuzzy systems and neural networks. After indicating the strong and weak points of both modeling approaches, we explain how an integrated framework will benefit from the strong points of both, which led to the pursuit of this research.

Chapter 3 formally defines the concept of the adaptive network and the formulas of its back-propagation-type learning rule. To speed up the learning process, we propose a learning rule combining the gradient descent and the Kalman filter algorithm. Besides the usual batch learning, we also develop a pattern learning paradigm that updates parameters after each data presentation.

In chapter 4, we transform different types of fuzzy inference systems into their equivalent adaptive network architectures, generically called ANFIS (Adaptive-Network-based Fuzzy Inference System), thus introducing the learning capability into fuzzy inference systems. After noting the similarity between the configurations of ANFIS and radial basis function networks, we indicate their functional equivalence after some minor simplifications. And by the Stone-Weierstrass theorem, the ANFIS architecture, just like the back-propagation neural network, is shown to be a universal approximator that can approximate any nonlinear functions arbitrarily well on a compact set. The ANFIS architecture is then employed to model two highly nonlinear functions and predict both synthesis and real-world time series.

The concept of back propagation is further generalized to temporal back propagation (TBP) in Chapter 5, where TBP is used to generate a fuzzy controller from scratch for balancing an inverted pendulum. Various simulation settings are employed to show the effectiveness of this approach and the robustness of the resulting fuzzy controller.

An intelligent pattern classifier, called NFC (Neuro-Fuzzy Classifier), is proposed in chapter 6 to provide another paradigm of an integrated framework. It is actually an adaptive nonlinear discriminant function with fuzzy if-then rules embedded inside. The

discriminant power of NFC is verified through successful applications to two benchmark classification problems: the two-spiral problem and the Iris data classification.

Finally, in chapter 7, conclusions of this research and recommended directions for further investigation are discussed.

Chapter 2

Review and Motivation

2.1 Overview

This chapter describes the basic concepts, operations and structures of fuzzy systems and neural networks, and reviews previous work in the literature. The pros and cons of these two schemes are listed and compared, an effort which inevitably led to the motivation for this research.

More specifically, in section 2.2 we will introduce:

- Concepts, notations and operations on fuzzy sets.
- Fuzzy inference systems which employ fuzzy if-then rules and fuzzy reasoning.
- Previous work about fuzzy logic modeling.

Section 2.3 briefly explains the learning rules and structures of the two most commonly used neural networks in the literature, namely:

- Back-propagation neural networks (BPNN's)

- Radial function basis networks (RFBN's).

After reviewing the previous approaches to fuzzy logic modeling and neural network modeling, we indicate in section 2.4 that the advantages and disadvantages of both modeling schemes turn out to be complementary, which leads to our motivation for developing an integrated neuro-fuzzy modeling approach that can, we hope, inherit all the advantages and bypass all the disadvantages.

2.2 Fuzzy Logic Modeling

2.2.1 Fuzzy Sets

This section summarizes basic concepts and notations of fuzzy set theory and fuzzy logic which will be needed in this work. A detailed treatment of the subject may be found in Zadeh's pioneering work in 1965 [109] and other subsequent publications [15, 110, 35, 36, 112, 113, 111, 41].

Let X be a space of objects and x be a generic element of X. A classical set A is defined as a collection of elements or objects $x \in X$, such that each x can either belong to or not belong to the set A, $A \subseteq X$. By defining a characteristic function (or membership function) on each element x in X, a classical set A can be represented by a set of ordered pairs $(x, 0)$ or $(x, 1)$, where 1 indicates membership and 0 non-membership.

Unlike the conventional set mentioned above, a *fuzzy set* [109] expresses the "degree" to which an element belongs to a set. Hence the characteristic function of a fuzzy set is allowed to have value between 0 and 1, denoting the degree of membership of an element in a given set. If X is a collection of objects denoted generically by x, then a fuzzy set A

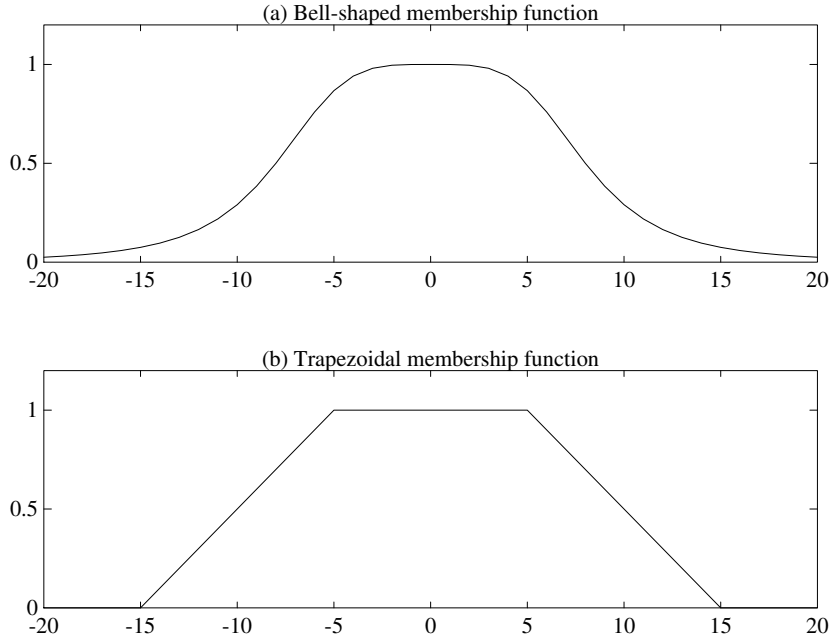


Figure 2.1: *Bell-shaped and trapezoidal membership functions.*

in X is defined as a set of ordered pairs:

$$A = \{(x, \mu_A(x)) \mid x \in X\} \tag{2.1}$$

$\mu_A(x)$ is called the *membership function* (MF) of x in A, which maps X to the membership space M, $M = [0, 1]$. When M contains only two points 0 and 1, A is nonfuzzy (crisp) and μ_A is identical to the characteristic function of a crisp set. For a convex fuzzy set (to be defined below), commonly used membership functions include trapezoidal (piecewise-linear) type and bell-shaped (smooth) type, as shown in Figure 2.1.

Suppose that X = "Age" then A can assume various linguistic terms such as "young" "middle-aged" and "old" which are characterized by membership functions $\mu_{old}(x)$, $\mu_{middle-aged}(x)$ and $\mu_{old}(x)$, respectively. This leads to the concept of *linguistic variables* [110] ("Age" in this case) whose values (*linguistic values*) are words or sentences

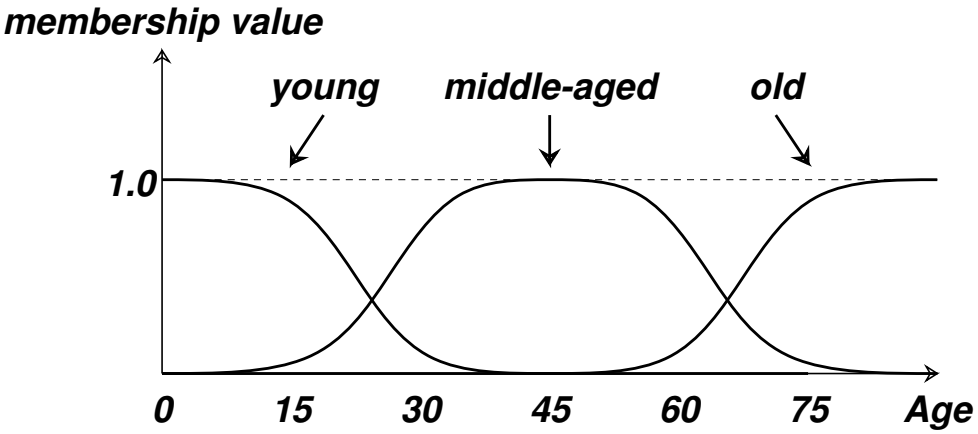


Figure 2.2: *Typical membership functions of linguistic values "young" "middle-aged" and "old".*

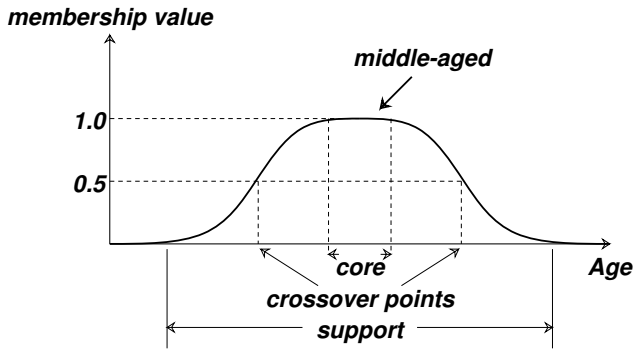
in a natural or synthetic language ("young" "middle-aged" and "old" in this case). Typical membership functions for these linguistic values are displayed in Figure 2.2.

The *core* of a fuzzy set A is the set of all points x in X such that $\mu_A(x) = 1$; the *support* of A is the set of all points x in X such that $\mu_A(x) > 0$. In particular, the element x in X at which $\mu_A(x) = 0.5$ is called the *crossover point*. Moreover, A is *normal* if its core is non-empty; A is *convex* if for any $x_1,\ x_2 \in X$ and any $\lambda \in [0, 1]$,

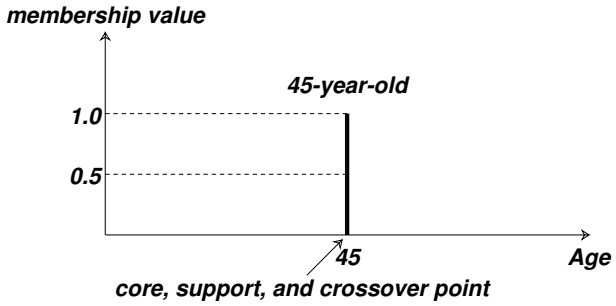
$$\mu_A(\lambda x_1 + (1 - \lambda)x_2) \geq min\{\mu_A(x_1), \mu_A(x_2)\}. \tag{2.2}$$

A fuzzy set whose support is a single point in X with $\mu_A(x) = 1$ is referred to as a *fuzzy singleton*. Figure 2.3 (a) and (b) illustrate the cores, supports and crossover points on the bell-shaped membership function of "middle-aged" and on the fuzzy singleton characterizing "45-year-old."

Corresponding to the ordinary logic operations, i.e., AND, OR and COMPLEMENT, fuzzy sets have similar operations which are called *conjunction*, *disjunction* and



(a)



(b)

Figure 2.3: *Crossover points and support of (a) "middle-aged" (b) "45-year-old" (a fuzzy singleton)* .

complement, respectively. Let

$$\begin{aligned} a &= \mu_A(x), \\ b &= \mu_B(x). \end{aligned} \tag{2.3}$$

Then the membership functions of $A \cap B$ and $A \cup B$ are specified by a conjunction operator $T(\cdot,\cdot)$ and a disjunction operator $S(\cdot,\cdot)$, respectively:

$$\begin{aligned} \mu_{A \cap B}(x) &= T(a,b), \\ \mu_{A \cup B}(x) &= S(a,b), \end{aligned} \tag{2.4}$$

where $T(\cdot,\cdot)$ satisfies the conditions of a *T-norm* [15, 112]:

$$\begin{aligned} &T(0,0) = 0 && (boundary) \\ &T(a,1) = T(1,a) = a && (boundary) \\ &T(a,b) \leq T(c,d)\ if\ a \leq c\ and\ b \leq d && (monotonicity) \\ &T(a,b) = T(b,a) && (commutativity) \\ &T(a,T(b,c)) = T(T(a,b),c) && (associativity) \end{aligned} \tag{2.5}$$

and $S(\cdot,\cdot)$ satisfies the conditions of a *T-conorm* [15, 112]:

$$\begin{aligned}
&S(1,1) = 1 && (boundary)\\
&S(a,0) = S(0,a) = a && (boundary)\\
&S(a,b) \leq S(c,d)\ if\ a \leq c\ and\ b \leq d && (monotonicity)\\
&S(a,b) = S(b,a) && (commutativity)\\
&S(a,S(b,c)) = S(S(a,b),c) && (associativity)
\end{aligned} \tag{2.6}$$

The membership functions of $\bar{A}$ (the negation of A) is usually specified by a negation operation $N(\cdot)$:

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x) = N(a). \tag{2.7}$$

As a matter of fact, T-norms $T(\cdot,\cdot)$ and T-conorms $S(\cdot,\cdot)$ are duals which support the generalization of DeMorgan's law:

$$\begin{aligned}
S(a,b) &= N(T(N(a),N(b))),\\
T(a,b) &= N(S(N(a),N(b))).
\end{aligned} \tag{2.8}$$

Several parameterized T-norms and dual T-conorms have been proposed in the literature, such as Yager [106], Dubois and Prade [16], Schweizer and Sklar [75], and Sugeno [82]. For instance, Schweizer and Sklar's T-norm operator can be expressed as

$$\begin{aligned}
T_{sc}(a,b,p) &= [MAX\{0,(a^{-p}+b^{-p}-1)\}]^{-\frac{1}{p}}\\
S_{sc}(a,b,p) &= 1-[MAX\{0,((1-a)^{-p}+(1-b)^{-p}-1)\}]^{-\frac{1}{p}}
\end{aligned} \tag{2.9}$$

It is observed that

$$\begin{aligned}
&\lim_{p\to 0} T_{sc}(a,b,p) = ab,\\
&\lim_{p\to \infty} T_{sc}(a,b,p) = min\{a,b\},
\end{aligned} \tag{2.10}$$

which correspond to two of the most frequently used T-norms in combining the membership values on the premise part of a fuzzy if-then rule. Actually the difference between these

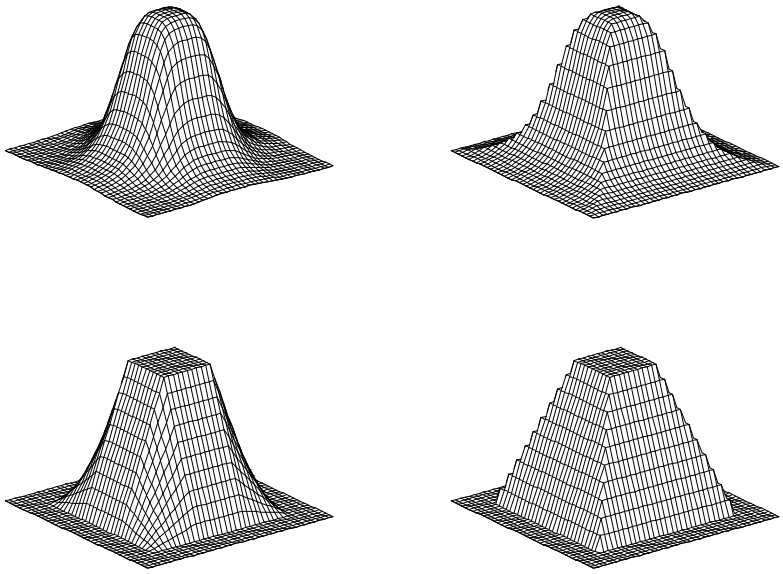


Figure 2.4: *Output surfaces of the multiplication operator (first column) and the min. operator (second column) on bell-shaped membership functions (first row) and trapezoidal membership functions (second row).*

two operators (multiplication and min.) is not too prominent, as is revealed in Figure 2.4 where the multiplication and min. operator are applied to both bell-shaped and trapezoidal membership functions on x and y. As a result, as long as we have a good method to fine-tune the membership functions, the choice of these two operators is not critical.

To give an general idea of how the parameter p affects the T-norm and T-conorm operator, Figure 2.5 (a) shows typical membership functions of fuzzy set A and B; (b) and (c) are $T_{sc}(a,b,p)$ and $S_{sc}(a,b,p)$, respectively, with $p = \infty$ (solid line), 1 (dashed line), 0 (dotted line), -1 (dash-dotted line). Note that the bell-shaped membership functions of A and B in Figure 2.5 (a) are defined as below

$$\mu_A(x) = \frac{1}{1 + [\frac{x+5}{7.5}]^4}, \tag{2.11}$$

$$\mu_B(x) = \frac{1}{1 + [\frac{x-5}{5}]^2}. \tag{2.12}$$

2.2.2 Fuzzy If-Then Rules

Fuzzy if-then rules or *fuzzy conditional statements* are expressions of the form *IF A THEN B*, where A and B are labels of fuzzy sets. 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. An example that describes a simple fact is

If pressure is high, then volume is small.

where *pressure* and *volume* are *linguistic variables* [110], *high* and *small* are *linguistic values* or *labels* that are characterized by membership functions.

Another form of fuzzy if-then rule, proposed by Takagi and Sugeno [90], has fuzzy sets involved only in the premise part. By using Takagi and Sugeno's fuzzy if-then rule, we can describe the resistant force on a moving object as follows:

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

where, again, *high* in the premise part is a 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 modeling 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 angle,

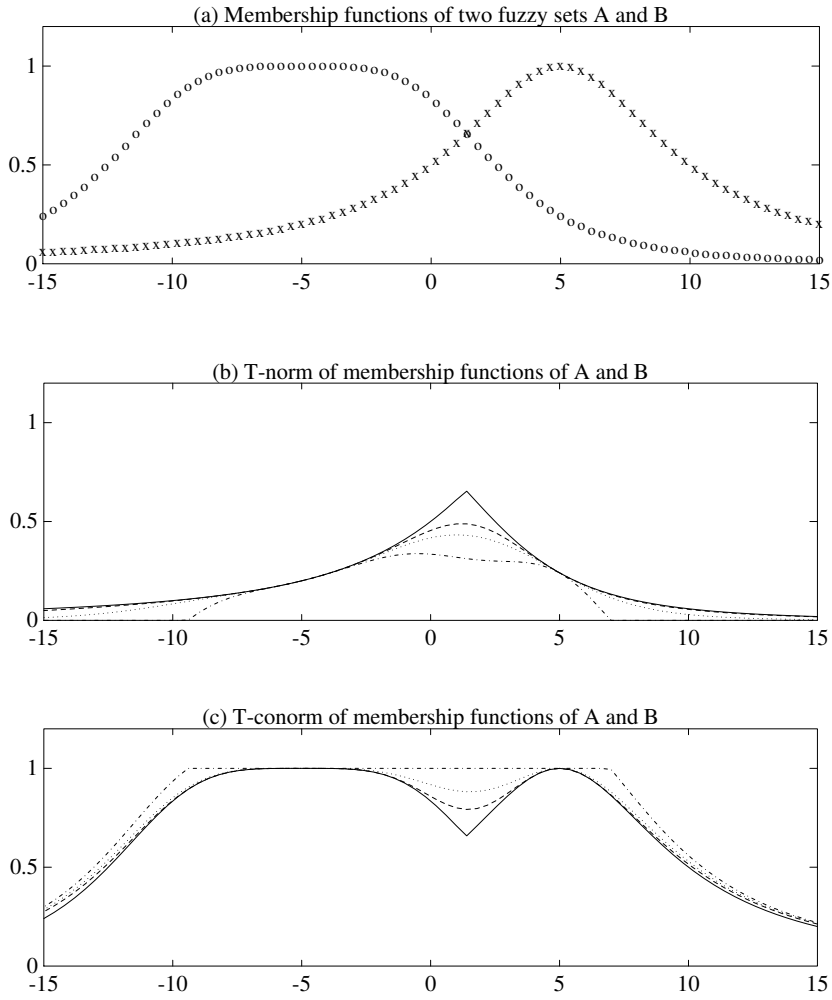


Figure 2.5: *Schweizer and Sklar's parameterized T-norms and T-conorms: (a) membership functions for fuzzy set A and B; (b)* $T_{sc}(a, b, p)$ *and (c)* $S_{sc}(a, b, p)$ *with* $p = \infty$ *(solid line), 1 (dashed line), 0 (dotted line), -1 (dash-dotted line).*

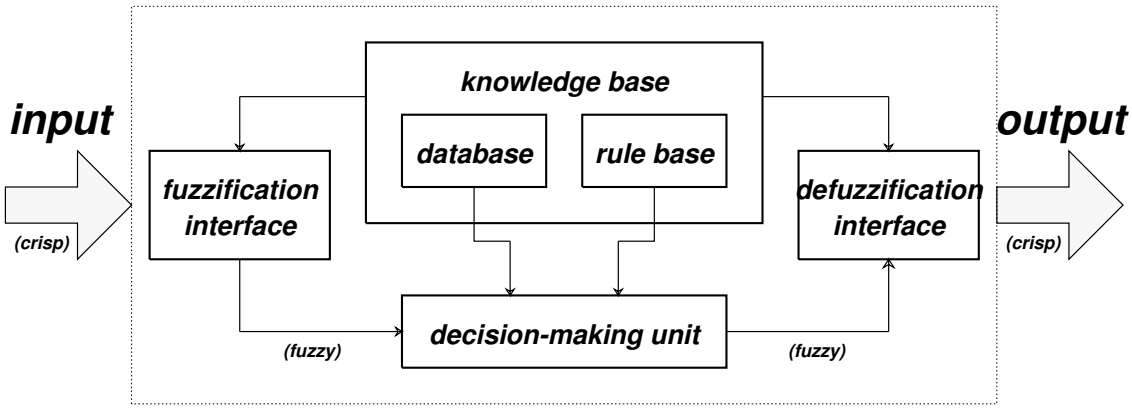


Figure 2.6: *Fuzzy inference system.*

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, a fuzzy-rule-based system widely used as a fuzzy controller, to be discussed in the next section.

2.2.3 Fuzzy Inference Systems

Fuzzy inference systems are also known as *fuzzy-rule-based systems*, *fuzzy models*, *fuzzy associative memories (FAM)*, or *fuzzy controllers* when used as controllers. Basically a fuzzy inference system is composed of five functional blocks (Figure 2.6):

- 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 interface** which transforms the crisp inputs into degrees of match with linguistic values;

- a **defuzzification interface** which transform the fuzzy results of the inference into a crisp output.

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

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

1. Compare the input variables 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, 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 consequents to produce a crisp output. (This step is called *defuzzification*.)

Several types of fuzzy reasoning [46, 47] have been proposed in past years. Depending on the types of fuzzy reasoning and fuzzy if-then rules employed, most fuzzy inference systems can be classified into three types (Figure 2.7):

Type 1: The overall output is the weighted average of each rule's crisp output induced by the rule's firing strength (the product or minimum of the degrees of match with the premise part) and output membership functions. The output membership functions used in this scheme must be monotonically non-decreasing [93].

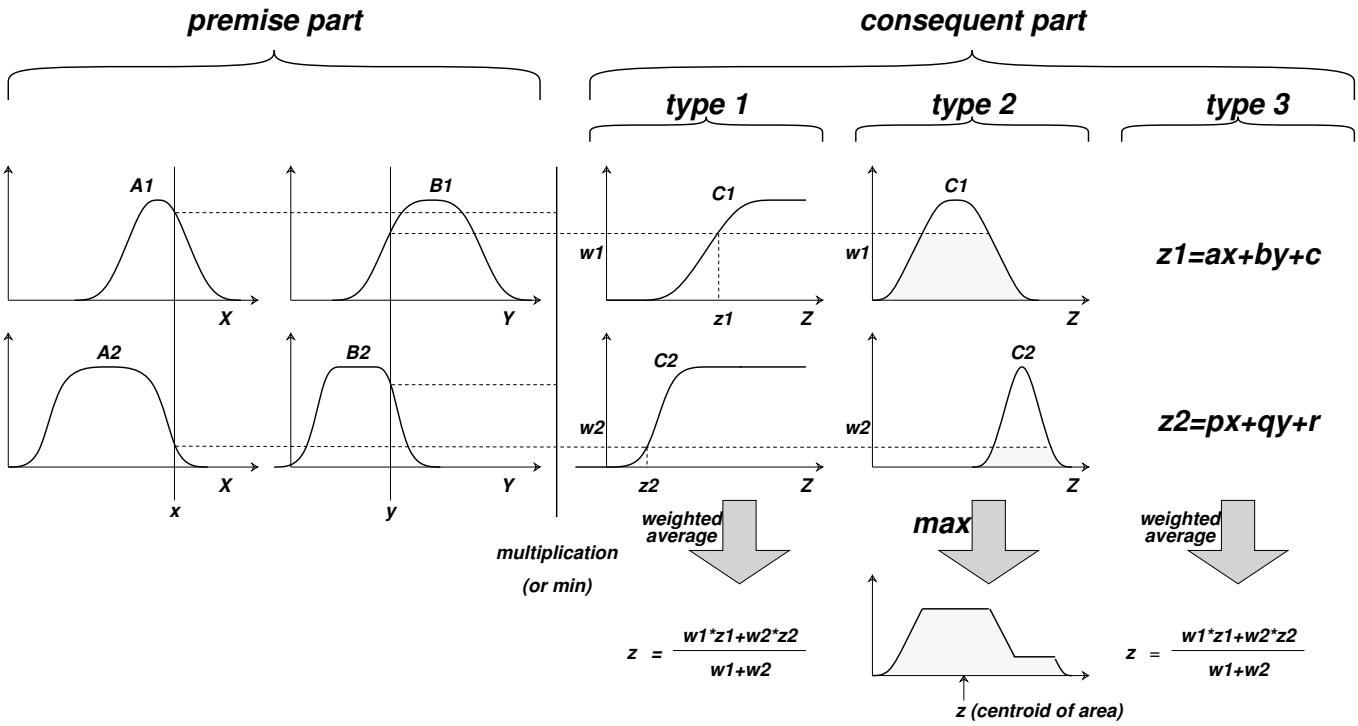
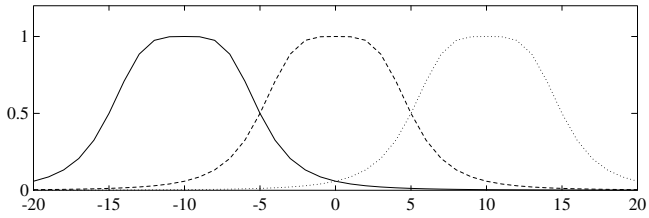


Figure 2.7: *Commonly used fuzzy if-then rules and fuzzy reasoning mechanisms.*

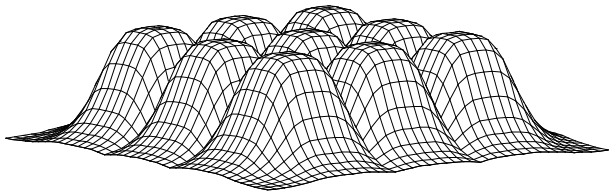
Type 2: The overall fuzzy output is derived by applying "max" operation to the qualified fuzzy outputs (each of which is equal to the minimum of firing strength and the output membership function of each rule). Various schemes have been proposed to choose the final crisp output based on the overall fuzzy output; some of them are center of area, bisector of area, mean of maxima, maximum criterion, etc [46, 47].

Type 3: Takagi and Sugeno's fuzzy if-then rules are used [90]. The output of each rule is a linear combination of input variables plus a constant term, and the final output is the weighted average of each rule's output.

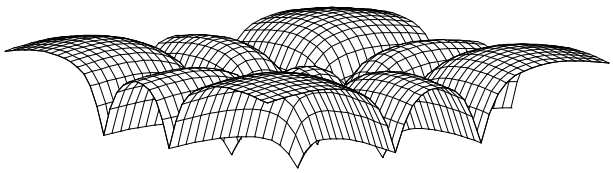
Figure 2.7 utilizes a two-rule two-input fuzzy inference system to show different types of fuzzy rules and fuzzy reasoning mentioned above. Be aware that most of the differences lie in the specification of the consequent part (monotonically non-decreasing or



(a)



(b)



(c)

Figure 2.8: *(a) Membership functions on x and y; (b) surface of dominant rule's firing strength; (b) surface of dominant rule's normalized firing strength.*

bell-shaped membership functions, or crisp function) and thus the defuzzification schemes (weighted average, centroid of area, etc) are also different.

We can define the *dominant rule* as the rule with the maximum firing strength, and the *normalized firing strength* as the ratio of a rule's firing strength with respect to the summation of all rules' firing strengths. Hence for type 1 and type 3 fuzzy inference systems, each rule's percentage of contribution to the overall output is specified exactly by

this rule's normalized firing strength. (This argument holds roughly for type 2 systems.) To demonstrate the statement that each rule is a local description of the underlying system, we depict several quantities of a 2-input 9-rule in Figure 2.8 where (a) shows the membership functions on two inputs x and y; (b) is the surface of the dominant rule's firing strength; and (c) is the surface of the dominant rule's normalized firing strength. It is obvious from (c) that the dominant rule's firing strength is large (close to one) only when the given input points are well inside of each fuzzy partition's interior, thus showing the locality of each rule.

Theoretically there is no evidence on which type of fuzzy inference system is better than the others; the choice of one type mainly depends on the user's preference and also on the application. For instance, when a fuzzy inference system is used as a fuzzy controller in real-time control, the computation speed is of major concern and this makes type 2 fuzzy inference systems a bad candidate since the calculation of the center of area (or bisector of area, mean of maxima, etc.) is time-consuming unless special hardware implementation (*fuzzy chip* [78, 38, 107, 108, 92]) is available. In fact, it will be proved in section 4.5 that these three types of fuzzy inference systems are universal approximators in the sense that they can approximate any nonlinear function arbitrarily well on a compact set, given that the number of rules is not restricted.

2.2.4 Fuzzy Logic Modeling

Fuzzy logic modeling (or *fuzzy modeling*) concerns the identification of the structure (number of rules, partition pattern, etc.) and parameters of fuzzy inference systems that can best describe a given input-output data set.

Takagi and Sugeno [90] were among the first researchers who recognized the importance of fuzzy logic modeling and developed a rule extraction algorithm by heuristic search and nonlinear optimization techniques [91]. In their scheme, all membership functions on the premise part have trapezoidal shapes and the consequent part is expressed as a linear combination of input variables plus a constant term. To find the optimal premise structure, they made a heuristic search by choosing the structure with the smallest unbiasedness criterion. Then the premise parameters were obtained by a complex nonlinear programming technique, and the consequent parameters were obtained by the stable-state Kalman filter. They also proposed a successive identification algorithm [83] to adjust the premise parameters, which involves a couple of heuristic terms and constraints. Though Sugeno's approach has been proven better than GMDH (Group Method of Data Handling) models [40], it is complex and not unified in its computation.

Since neural networks (NN's) are well known for their nonlinearity and adaptability, plenty of on-going research is focusing on building NN-based fuzzy inference systems. Horiawa et al. [23] employed a concise yet ingenious approach to synthesize a simplified fuzzy inference system from a back-propagation neural network with minor modifications on the learning rule and some of the node functions. However, due to their overemphasis on sticking to neural network paradigms, the resulting fuzzy inference system is a simplified version in the sense that only the *weighted sum* (instead of the weighted average) can be used in the fuzzy reasoning operation.

On the other hand, Takagi and Hayashi [88, 89] proposed an NN-driven fuzzy reasoning system which can generate an irregular partition of the input space through

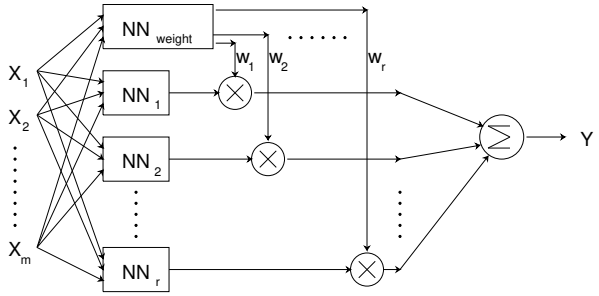


Figure 2.9: *Takagi's NN-driven fuzzy reasoning system.*

neural networks. Figure 2.9 is a schematic diagram of their network structure. Suppose there are r fuzzy rules in the fuzzy inference system, then NN_{weight} is used to decide the firing strength (w_i, $1 \leq i \leq r$) of each rule and NN_i ($0 \leq i \leq r$) is in charge of deriving the consequent equation F_i for the i-th rule. The system adopts a variant of type-3 fuzzy reasoning such that each rule can be expressed as

$$If\ \vec{x} \in A_i,\ then\ y = F_i(\vec{x}) \tag{2.13}$$

where $\vec{x}$ is a vector of input variables, A_i denotes one of the fuzzy subspaces derived by NN_{weight}, and $F_i(\cdot)$ is the consequent equation implemented by NN_i. The overall output y is calculated as the *weighted sum* of consequent equations, i.e.,

$$y = \sum_{i=1}^{r} w_i * F_i(\vec{x}) \tag{2.14}$$

It is observed that NN_{weight} can generate an irregular partition (Figure 2.10(b)), as compared to the conventional fuzzy partition (Figure 2.10(a)).

Basically, this approach can be divided into three steps:

1. Apply a clustering algorithm to the training data, and train NN_{weight} to perform classification on input variables according to the result of the clustering algorithm.

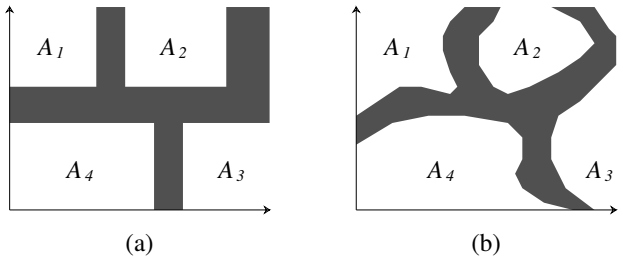


Figure 2.10: *Conventional and NN-based fuzzy partitions of input space* .

2. Train each NN_i based on both the input part of the training data and the estimated outputs obtained from checking data.

3. Use a back elimination method to simplify the consequent part and cut off irrelevant input variables.

Though Takagi and Hayashi's fuzzy partition is more flexible than the conventional one, the introduction of NN's as blocks in a fuzzy inference system causes problems such as how to estimate the desired output of each NN_i and how to embody human expertise into NN_{weight} and NN_i. Furthermore, their use of a weighted sum to calculate final output also results in a simplified version of fuzzy inference systems.

In order to keep the original spirit of fuzzy inference systems, we [28, 27, 30, 29] introduce the learning rule, instead of the structure, of back-propagation neural networks into fuzzy inference systems. We do so by first transforming the fuzzy inference system into an equivalent adaptive network, and then the learning proceeds by a combination of a back-propagation-type gradient descent and a sequential least squares estimate. More details are to be covered in the following chapters.

In fact, we are not the only researchers who have been working along these lines. Lin and Lee [51] proposed a very similar network structure with application to control and

scheduling problems. Wang and Mendel first came up with a table-lookup scheme [95] for generating fuzzy if-then rules from numerical data; then they also proposed a similar structure [96] with Gaussian functions as membership functions and the gradient descent as the learning rule.

2.3 Neural Network Modeling

Artificial neural networks, or simply neural networks (NN's), have been studied for a long time since Rosenblatt [71] first introduced single layer perceptrons. Because of the limitations of single-layer systems pointed out by Minsky and Papert [55] and their pessimistic views on multi-layer systems, the interest in NN's had been dwindling. Recent resurgence in the field of NN's had been encouraged by new learning algorithms [101, 65, 73, 18], analog VLSI techniques, and parallel processing [52].

Quite a few NN models have been proposed and investigated in recent years. These NN models can be classified into categories according to various criteria such as supervised or unsupervised learning rules, binary or continuous node values, feed-forward of recurrent network architectures, adjustable or hard-wired (fixed) parameters, uniform or hybrid node functions, biologically or psychologically motivated structures and operations, etc. However, in this thesis, we shall confine our scope to modeling problems with desired input-output data sets, so the relevant networks should have adjustable parameters which are updated by a supervised learning rule. In the following we introduce two network structures of this type that have been used frequently in the literature. Since the primary goal of all these three networks is to achieve a desired input-output mapping, they are often called *mapping*

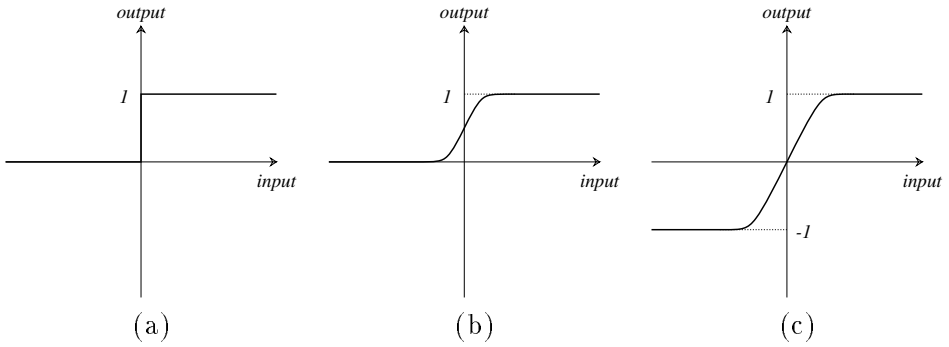


Figure 2.11: *Activation functions: (a) signum function; (b) sigmoid function; (c) hyper-tangent function.*

networks.

2.3.1 Back Propagation Neural Networks (BPNN's)

A back propagation neural network (BPNN) is composed of a number of interconnected computing units called *neurons* or *nodes*, each of which performs a simple nonlinear multi-input single-output function (*activation function* or *transfer function*) in a parallel manner. The activation functions are usually of a sigmoidal or hyper-tangent type which approximates the *signum function* (also know as *step function* or *hard-limiter*) and yet provides differentiability with respect to input signals. Figure 2.11 shows different types of activation function with definitions as follows:

$$\begin{aligned} &\text{signum function:} && sgn(x) = \begin{cases} 1 \text{ if x} \geq 0 \\ 0 \text{ if x} < 0 \end{cases} \\ &\text{sigmoid function:} && sig(x) = \frac{1}{1+exp(-x)} \\ &\text{hyper-tangent function:} && tanh(x) = \frac{exp(x)-exp(-x)}{exp(x)+exp(-x)} \end{aligned} \tag{2.15}$$

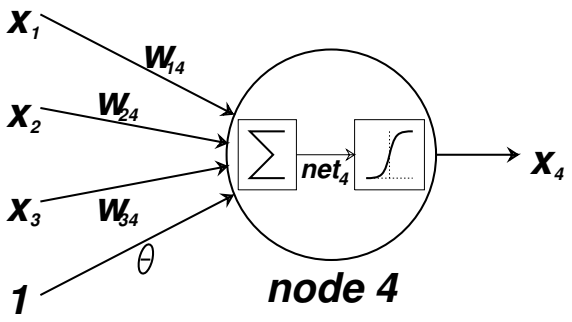


Figure 2.12: *A BPNN node.*

When the signum function (hard-limiter) is used as the activation function for a layered network, the network is often called *perceptron* [71, 63] and, as a matter of fact, it is one of the most important structures that lay the grounds for current understanding and thus applications of neural networks. However, due to the non-differentiability of the hard-limiter, the learning strategy of perceptrons is not obvious unless the other two activation functions are employed instead. Therefore throughout this dissertation, we will confine our discussion on neural networks with soft-limiters, that is, either sigmoid or hypertangent function.

The *net input* of a node is defined as the weighted sum of the incoming signals plus a threshold. For instance, the net input and output of node j in Figure 2.12 (where $j = 4$) are

$$net_j = \sum_i w_{ij} x_i + \theta_j, \tag{2.16}$$

$$\begin{aligned} x_j &= sig(net_j) = \tfrac{1}{1+exp(-net_j)}, \text{ (for sigmoid type)}, or \\ &= tanh(net_j) = \tfrac{exp(-net_j)-exp(-net_j)}{exp(-net_j)+exp(-net_j)}, \text{ (for hyper-tangent type)}, \end{aligned} \tag{2.17}$$

where x_i is the output of node i located in the previous layer, w_{ij} is the weight that associates with the link connecting node i and j, and θ_j is the threshold of node j. Since the weights

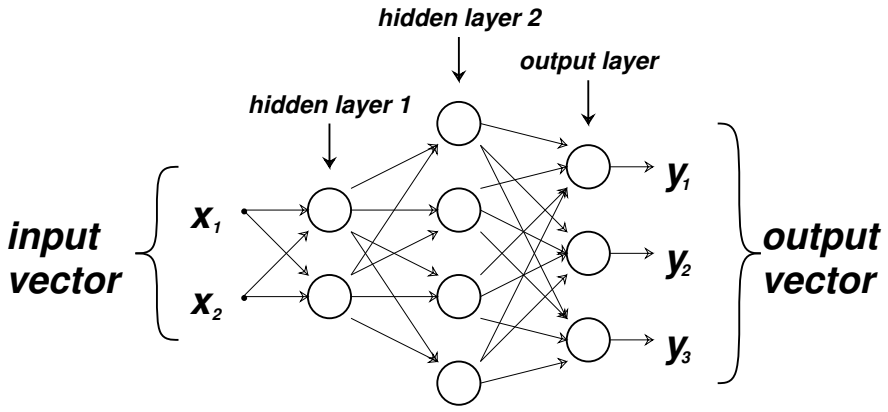


Figure 2.13: *A 2-2-4-3 BPNN.*

w_{ij}'s are actually internal parameters associated with each node j; changing the weights of a node will alter the behavior of the node and in turn alter the behavior of the whole BPNN. Figure 2.13 shows a 3 layer BPNN with 4 inputs, 2 hidden layers (each with 2 and 3 nodes) and 4 outputs. For simplicity, this BPNN will be referred to as a 2-2-4-3 structure according to its node number in each layer.

The *learning algorithm* or *adaptation rule* refers to the way the weights are changed in order to achieve a desired mapping between input and output. The *backward error propagation* [73] (also known as *back propagation* (*BP*) or *generalized delta rule* (*GDR*)) is an iterative gradient descent algorithm designed to minimize the mean squared error between the actual outputs of a BPNN and the desired outputs. Namely, this algorithm converges to a set of weights that minimizes the following error measure:

$$E = ||d(\vec{x}) - NN(\vec{x})||^2, \tag{2.18}$$

where $\vec{x}$ is the input vector, $d(\vec{x})$ is the desired output vector, and $NN(\vec{x})$ is the calculated output vector obtained as the output of the last layer in a BPNN. To update the parameters, we have to define an equivalent error δ_m for node m as follows (assuming the sigmoidal

function is used):

$$\delta_m = \begin{cases} -2(d_m(\vec{x}) - NN_m(\vec{x}))sig'(net_m) & \text{if node } m \text{ is in output layer,} \\ sig'(net_m)\sum_j \delta_j w_{mj} & \text{otherwise,} \end{cases} \tag{2.19}$$

where w_{mj} is the weight of the connection from node m to node j; $d_m(\vec{x})$ and $NN_m(\vec{x})$ are the m-th components of $d(\vec{x})$ and $NN(\vec{x})$, respectively. Then the update amount of weight w_{mi} is

$$\triangle w_{im} = -\eta \delta_m x_i \tag{2.20}$$

where η is a learning rate which affects the convergence speed and stability of the weights during the learning process.

The representational power or approximation power of the BPNN has been explored by some researchers. When used as a binary-valued NN where a hard limiter (step function) is used as the activation function, a three-layer BPNN can form arbitrary complex decision regions to separate different classes, as pointed out by Lippmann [52]. On the other hand, when a BPNN is employed to model a mapping with continuous outputs, Cybenko [14] showed that a continuous BPNN with one hidden layer and any fixed continuous sigmoidal nonlinear functions can approximate any continuous function arbitrarily well on a compact set.

BPNN's are by far the most often used NN structure for various applications in different areas such as speech recognition, pattern recognition, signal processing, data compression, automatic control, etc.

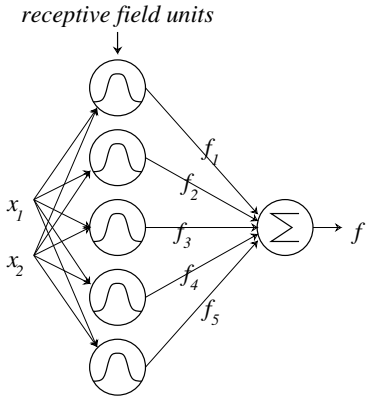


Figure 2.14: *A radial basis function network (RBFN)* .

2.3.2 Radial Basis Function Networks (RBFN's)

The locally-tuned and overlapping receptive field is a well-known structure that has been studied in regions of cerebral cortex, the visual cortex, etc. Based on the biological receptive fields, Moody and Darken [57, 58] proposed a network structure, *radial basis function network* (RBFN), that employs local receptive fields to perform function mappings. Figure 2.14 shows the schematic diagram of a RBFN with five receptive field units; the output of i-th receptive field unit (or hidden unit) is

$$w_i = R_i(\vec{x}) = R_i(\|\vec{x} - \vec{c_i}\|/\sigma_i),\ i = 1, 2, ..., H \tag{2.21}$$

where $\vec{x}$ is an N-dimensional input vector, $\vec{c_i}$ is a vector with the same dimension as $\vec{x}$, H is the number of receptive field units, and $R_i(\cdot)$ is the i-th receptive field response with a single maximum at the origin. Typically, $R_i(\cdot)$ is chosen as a Gaussian function

$$R_i(\vec{x}) = exp[-\frac{\|\vec{x} - \vec{c_i}\|^2}{\sigma_i^2}]. \tag{2.22}$$

or as a logistic function

$$R_i(\vec{x}) = \frac{1}{1 + exp[\|\vec{x} - \vec{c_i}\|^2/\sigma_i^2]} \tag{2.23}$$

Thus the radial basis function w_i computed by the i-th hidden units is maximum when the input vector $\vec{x}$ is near the center $\vec{c_i}$ of that unit.

The output of a radial basis function network can be computed in two ways. For the simpler one, as shown in Figure 2.14, the output is the weighted sum of the function value associated with each receptive field:

$$f(\vec{x}) = \sum_{i=1}^{H} f_i w_i = \sum_{i=1}^{H} f_i R_i(\vec{x}), \tag{2.24}$$

where f_i is the function value, or strength, of i-th receptive field. With the addition of lateral connections (not shown in Figure 2.14) between the receptive field units, the network can produce the normalized response function as the weighted average of the strengths [57]:

$$f(\vec{x}) = \frac{\sum_{i=1}^{H} f_i w_i}{\sum_{i=1}^{H} w_i} = \frac{\sum_{i=1}^{H} f_i R_i(\vec{x})}{\sum_{i=1}^{H} R_i(\vec{x})}. \tag{2.25}$$

To minimize the squared errors between desired output and model output, several learning algorithms have been proposed to identify the parameters ($\vec{c_i}$, σ_i and f_i) of a RBFN. Moody et al. [57] use self-organizing techniques to find the centers ($\vec{c_i}$) and widths (σ_i) of the receptive fields, and then employ the supervised Adaline or LMS learning rule to identify f_i. On the other hand, Chen et al. [9] apply the orthogonal least squares learning algorithm to determine those parameters.

2.4 Motivation for Neuro-Fuzzy modeling

A fuzzy inference system can utilize human expertise by storing its essential components in rule base and database, and perform fuzzy reasoning to infer the overall output value. The derivation of fuzzy if-then rules and corresponding membership functions depends heavily on the *a priori* knowledge about the system under consideration. However, there are still two basic but important problems concerning the preparation and manipulation of knowledge:

1. No systematic way exists to transform experiences or knowledge of human experts to the knowledge base of a fuzzy inference system.

2. There is still a need of adaptability or learning algorithms to tune the membership functions so as to minimize the discrepancy between model (calculated) output and desired output.

These two problems greatly restricted the application domains of fuzzy inference systems, either as a controller or as an expert system.

On the other hand, neural network modeling does not rely on human expertise. Instead, it employs a learning procedure and a given training data set to evolve a set of parameters (i.e., weights) such that the required functional behavior is achieved. Due to the homogeneous structure of NN, it is hard to extract structured knowledge from either the weights or the configuration of the NN in question. It should be emphasized that the weights in an NN with hard-limiter as its activation function do have physical meanings [52]: the weights of a given node represent the coefficients of the hyperplane (or discriminant function)

that partitions the input space into two regions with different output values. However, this interpretation of weights gets weaker and weaker under the following progressive conditions:

1. The NN's activation function is either sigmoid or hyperbolic tangent functions.
2. The outputs of each node are not binary. Namely, the node outputs are not one of the saturation values of the activation functions, which is $\{0, 1\}$ for sigmoid and $\{-1, 1\}$ for hyper-tangent activation function.
3. The given desired outputs are also continuous instead of binary.

Unfortunately, condition 1 and 2 hold almost all the time even when the given desired input-output pairs are binary. Furthermore, when NN's are used for modeling continuous systems, all the conditions mentioned above are always true. Therefore it is hard, if not impossible, to gain insight into the training data (or the original system) by examining the NN's weights or configuration after learning. The situation is much worse when we are using a large multi-layer NN with short-cut connections between non-adjacent layers to solve a real-world modeling problem; most of the time we are left with an NN with a bunch of connection weights that are just hard to have any easy and meaningful interpretations.

Viewing from another angle, one may wonder if it is possible to build the *a priori* knowledge about the training data (or the original system) into the weights or configuration of an NN such the efforts on the subsequent learning can be reduced. The answer is yes only when the knowledge is in the form of the hyperplane structure, which is rarely the case unless we can visualize the training data effectively. Generally speaking, the *a priori*

knowledge is usually obtained from human experts and it is most appropriate to express the knowledge as a set of fuzzy if-then rules. Under this conditions, the *a priori* knowledge would not be easily encoded into NNs' parameters and/or structures.

To sum up, NNs' drawbacks can be itemized as follows:

1. No effective methods have been proposed to determine the initial weight values and network's configurations (e.g., number of hidden layers and hidden nodes).

2. Even though human expertise about the system under consideration is available, it is not always usable to NN's unless the human knowledge is of the hyperplane format.

3. Most of the time, there are no easy ways to get functional insight into an trained NN with continuous outputs, even if it can perfectly reproduce the desired outputs.

To a large extent, the drawbacks pertaining to these two approaches seem complementary. Therefore it seems natural to consider building an integrated system combining the concepts of fuzzy logic modeling and and neural network modeling. In other words, the integrated approach, or *neuro-fuzzy modeling*, should incorporate the three most important features:

1. Meaningful and concise representation of structured knowledge.

2. Efficient learning capability to identify parameters.

3. Clear mapping between parameters and structured knowledge.

Quite a few methods for integrating fuzzy logic and neural networks are potentially feasible. However, we will use the above three criteria as the guidelines for finding a desired neuro-fuzzy modeling approach.

Chapter 3

Adaptive Networks: Architectures and Learning Algorithms

3.1 Introduction

This chapter introduces the architecture and learning procedure of the adaptive network which is in fact a superset of all kinds of feedforward neural networks with supervised learning capability. An adaptive network, as its name implies, is a network structure consisting of nodes and directional links through which the nodes are connected. Moreover, part or all of the nodes are adaptive, which means each output of these nodes depends on the parameter(s) pertaining to this node, and the learning rule specifies how these parameters should be changed to minimize a prescribed error measure.

The basic learning rule of adaptive networks is based on the gradient descent and the chain rule, which was proposed by by Werbos [101] in the 1970's. However, due to

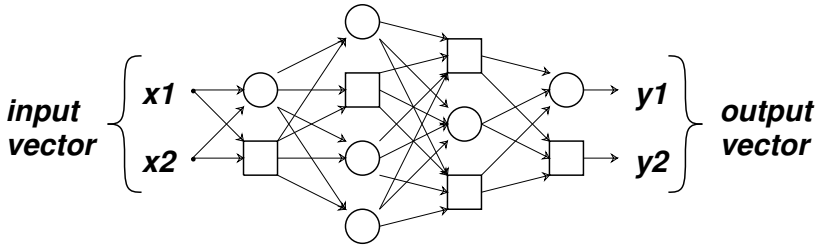


Figure 3.1: *An adaptive network.*

the state of artificial neural network research at that time, Werbos' early work failed to receive the attention it deserved. In the following presentation, the derivation is based on the author's work [28, 27] which generalizes the formulas in [73].

Since the basic learning rule is based the gradient method which is notorious for its slowness and tendency to become trapped in local minima, here we propose a hybrid learning rule which can speed up the learning process substantially Both the batch learning and the pattern learning of the proposed hybrid learning rule is discussed below.

3.2 Architecture and Learning Rule

An adaptive network (Figure 3.1) is a multi-layer feedforward network in which each node performs a particular function (*node function*) on incoming signals as well as a set of parameters pertaining to this node. The nature of the node functions may vary from node to node, and the choice of each node function depends on the overall input-output function which the adaptive network is required to carry out. Note that the links in an adaptive network only indicate the flow direction of signals between nodes; no weights are associated with the links.

To reflect different adaptive capabilities, we use both circle and square nodes in

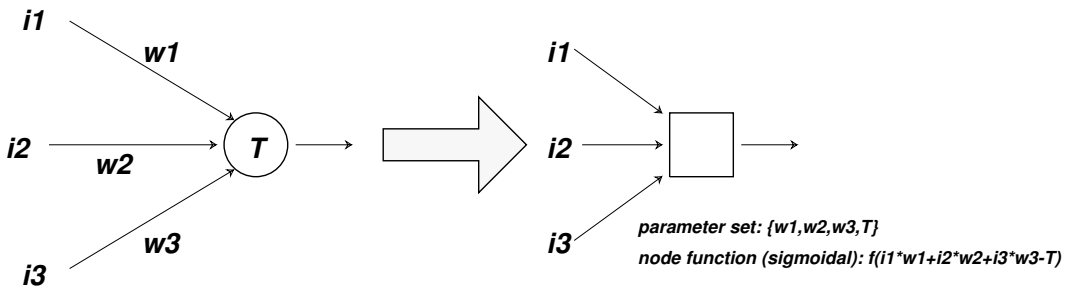


Figure 3.2: *An NN node and its equivalent adaptive network representation.*

an adaptive network. A square node (adaptive node) has parameters while a circle node (fixed node) has none. For instance, an ordinary NN node can be converted to our notation as shown in Figure 3.2. The parameter set of an adaptive network is the union of the parameter sets of each adaptive node. In order to achieve a desired input-output mapping, these parameters are updated according to given training data and a gradient-based learning procedure described below.

Suppose that a given adaptive network has L layers and the k-th layer has $\#(k)$ nodes. We can denote the node in the i-th position of the k-th layer by (k, i), and its node function (or node output) by O_i^k. Since a node output depends on its incoming signals and its parameter set, we have

$$O_i^k = O_i^k(O_1^{k-1}, \ldots O_{\#(k-1)}^{k-1}, a, b, c, \ldots), \tag{3.1}$$

where a, b, c, etc. are the parameters pertaining to this node. (Note that we use O_i^k as both the node output and node function.)

Assuming the given training data set has P entries, we can define the *error measure* (or *energy function*) for the p-th ($1 \leq p \leq P$) entry of training data entry as the sum of

squared errors:

$$E_p = \sum_{m=1}^{\#(L)} (T_{m,p} - O_{m,p}^L)^2, \tag{3.2}$$

where $T_{m,p}$ is the m-th component of p-th target output vector, and $O_{m,p}^L$ is the m-th component of actual output vector produced by the presentation of the p-th input vector. Hence the overall error measure is $E = \sum_{p=1}^{P} E_p$.

In order to develop a learning procedure that implements gradient descent in E over the parameter space, first we have to calculate the *error rate* $\frac{\partial E_p}{\partial O}$ for p-th training data and for each node output O. The error rate for the output node at (L, i) can be calculated readily from equation (3.2):

$$\frac{\partial E_p}{\partial O_{i,p}^L} = -2(T_{i,p} - O_{i,p}^L). \tag{3.3}$$

For the internal node at (k, i), the error rate can be derived by the chain rule:

$$\frac{\partial E_p}{\partial O_{i,p}^k} = \sum_{m=1}^{\#(k+1)} \frac{\partial E_p}{\partial O_{m,p}^{k+1}} \frac{\partial O_{m,p}^{k+1}}{\partial O_{i,p}^k}, \tag{3.4}$$

where $1 \leq k \leq L-1$. That is, the error rate of an internal node can be expressed as a linear combination of the error rates of the nodes in the next layer. Therefore for all $1 \leq k \leq L$ and $1 \leq i \leq \#(k)$, we can find $\frac{\partial E_p}{\partial O_{i,p}^k}$ by equation (3.3) and (3.4).

Now if α is a parameter of the given adaptive network, we have

$$\frac{\partial E_p}{\partial \alpha} = \sum_{O^* \in S} \frac{\partial E_p}{\partial O^*} \frac{\partial O^*}{\partial \alpha}, \tag{3.5}$$

where S is the set of nodes whose outputs depend on α. Then the derivative of the overall error measure E with respect to α is

$$\frac{\partial E}{\partial \alpha} = \sum_{p=1}^{P} \frac{\partial E_p}{\partial \alpha}. \tag{3.6}$$

Accordingly, the update formula for the generic parameter α is

$$\triangle\alpha = -\eta\frac{\partial E}{\partial\alpha}, \tag{3.7}$$

in which η is a learning rate which can be further expressed as

$$\eta = \frac{k}{\sqrt{\sum_\alpha(\frac{\partial E}{\partial\alpha})^2}}, \tag{3.8}$$

where k is the *step size*, the length of each gradient transition in the parameter space. Usually, we can change the value of k to vary the speed of convergence. The heuristic rules for changing k are discussed in section 4.6 where we report simulation results.

Actually, there are two learning paradigms for adaptive networks. With the *batch learning* (or *off-line learning*), the update formula for parameter α is based on equation (3.6) and the update action takes place only after the whole training data set has been presented, i.e., only after each *epoch* or *sweep*. On the other hand, if we want the parameters to be updated immediately after each input-output pair has been presented, then the update formula is based on equation (3.5) and it is referred to as the *pattern learning* (or *on-line learning*).

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

Though we can apply the gradient method to identify the parameters in an adaptive network, the method is generally slow and likely to become trapped in local minima. Here we propose a hybrid learning rule [27] which combines the gradient method and the least squares estimate (LSE) to identify parameters.

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

$$output = F(\vec{I}, S), \tag{3.9}$$

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 composite function $H \circ F$ is linear in some of the elements of S, then these elements can be identified by the least squares method. More formally, if the parameter set S can be decomposed into two sets

$$S = S_1 \oplus S_2, \tag{3.10}$$

(where $\oplus$ represents direct sum) such that $H \circ F$ is linear in the elements of S_2, then upon applying H to equation (3.9), we have

$$H(output) = H \circ F(\vec{I}, S), \tag{3.11}$$

which is linear in the elements of S_2. Now given values of elements of S_1, we can plug P training data into equation (3.11) and obtain a matrix equation:

$$AX = B \tag{3.12}$$

where X is an unknown vector whose elements are parameters in S_2. Let $|S_2| = M$, then the dimensions of A, X and B are $P \times M$, $M \times 1$ and $P \times 1$, respectively. Since P (number of training data pairs) is usually greater than M (number of linear parameters), this is an overdetermined problem and generally there is no exact solution to equation (3.12). Instead, a *least squares estimate* (LSE) of X, X^*, is sought to minimize the 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:

$$X^* = (A^T A)^{-1} A^T B, \tag{3.13}$$

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 equation (3.13) is concise in notation, it is expensive in computation when dealing with the matrix inverse and, moreover, it becomes ill-defined if $A^T A$ is singular. As a result, we employ sequential formulas 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 (see below) for systems with changing characteristics. Specifically, let the ith row vector of matrix A defined in equation (3.12) be a_i^T and the ith element of B be b_i^T, then X can be calculated iteratively using the sequential formulas widely adopted in the literature [2, 21, 53, 81]:

$$\left.\begin{array}{rcl} X_{i+1} & = & X_i + S_{i+1} a_{i+1} (b_{i+1}^T - a_{i+1}^T X_i) \\ S_{i+1} & = & S_i - \frac{S_i a_{i+1} a_{i+1}^T S_i}{1 + a_{i+1}^T S_i a_{i+1}}, \quad i = 0, 1, \cdots, P-1 \end{array}\right\}, \tag{3.14}$$

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 equation (3.14) 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 equation (3.9) is a column vector), equation (3.14) still applies except that b_i^T is the i-th rows of matrix B.

This sequential least squares estimate of X can be interpreted as a Kalman filter [33] for the process

$$X(k+1) = X(k), \tag{3.15}$$

$$Y(k) = A(k) X(k) + noise, \tag{3.16}$$

where $X(k) = X_k$, $Y(k) = b_k$ and $A(k) = a_k$. For this reason, equation (3.14) is sometimes loosely referred to as the Kalman filter algorithm. (Note that all our simulations described in later chapters are deterministic; there is no noise added in the simulation settings.)

Now we can combine the gradient method and the least squares estimate 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 equation (3.12) are obtained, and the parameters in S_2 are identified by the sequential least squares formulas in equation (3.14). 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 w.r.t. each node output, see equation(3.3) and (3.4)) propagate from the output end toward the input end, and the parameters in S_1 are updated by the gradient method in equation (3.7).

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 equation (3.9) is a column vector. Let $H(\cdot)$ be the inverse sigmoid function

$$H(x) = ln(\frac{x}{1-x}), \tag{3.17}$$

then equation (3.11) becomes a linear (vector) function such that each element of $H(output)$ is a 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 keep in mind that by using the least squares method on the data transformed by $H(\cdot)$, 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 $H(\cdot)$ is monotonically increasing.

3.4 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 (see equation (3.5)) instead of E. Strictly speaking, this is not a truly gradient search procedure to minimize E, yet it will approximate to one if the learning rate is small.

For the sequential least squares formulas to account for the time-varying charac-

teristics 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 [21]. One simple method is to formulate the squared error measure as a weighted version that gives higher weighting factors to more recent data pairs. This amounts to the addition of a *forgetting factor* λ to the original sequential formula:

$$\left.\begin{array}{rcl} X_{i+1} & = & X_i + S_{i+1}a_{i+1}(b_{i+1}^T - a_{i+1}^T X_i) \\ S_{i+1} & = & \frac{1}{\lambda}[S_i - \frac{S_i a_{i+1} a_{i+1}^T S_i}{\lambda + a_{i+1}^T S_i a_{i+1}}] \end{array}\right\}, \tag{3.18}$$

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

Chapter 4

ANFIS: Adaptive-Networks-based Fuzzy Inference Systems

4.1 Introduction

The architecture and learning rules of adaptive networks have been described in the previous chapter. Functionally, there are almost no constraints on the node functions of an adaptive network except piecewise differentiability. Structurally, the only limitation of network configuration is that it should be of feedforward type. Due to these minimal restrictions, the adaptive network's applications are immediate and immense in various areas.

In this chapter, we propose a class of adaptive networks which are functionally equivalent to fuzzy inference systems. The proposed architecture is referred to as *ANFIS*, standing for *Adaptive-Network-based Fuzzy Inference System*. We describe how to decom-

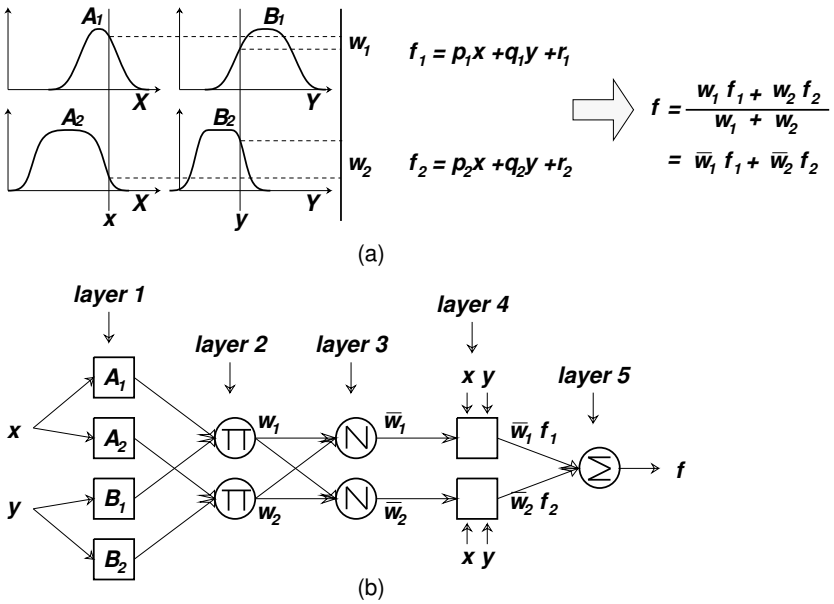


Figure 4.1: *(a) Type-3 fuzzy reasoning; (b) equivalent ANFIS (type-3 ANFIS).*

pose the parameter set such that the hybrid learning rule of adaptive networks can be applied to the ANFIS architecture for both type-1 and type-3 reasoning. Moreover, we demonstrate that radial basis function networks (RBFN's) are functionally equivalent to a simplified version of fuzzy inference systems. Finally, the effectiveness of the hybrid learning rule is tested through four simulation examples: examples 1 and 2 are the modeling of nonlinear functions; examples 3 shows how to identify nonlinear components on-linely in a control system; example 4 predicts the Mackey-Glass chaotic time series. Extensive comparisons with connectionist approaches and conventional statistical methods are conducted and discussed.

4.2 ANFIS Architecture

For simplicity, we 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 [90]:

Rule 1: If x is A_1 and y is B_1, then $f_1 = p_1 x + q_1 y + r_1$,

Rule 2: If x is A_2 and y is B_2, then $f_2 = p_2 x + q_2 y + r_2$.

then the type-3 fuzzy reasoning is illustrated in Figure 4.1(a), and the corresponding equivalent ANFIS architecture (*type-3 ANFIS*) is shown in Figure 4.1(b). The node functions in the same layer are of the same function family as described below:

Layer 1 Every node 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

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

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 for node functions in this layer. Parameters in this layer are referred to as *premise parameters*.

Layer 2 Every node in this layer is a circle node labeled Π which multiplies the incoming signals and sends the product out. For instance,

$$w_i = \mu_{A_i}(x) \times \mu_{B_i}(y), \ i = 1, 2. \tag{4.4}$$

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

Layer 3 Every node in this layer is a circle node labeled N. The i-th node calculates the ratio of the i-th rule's firing strength to the sum of all rules' firing strengths:

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

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

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

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

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*.

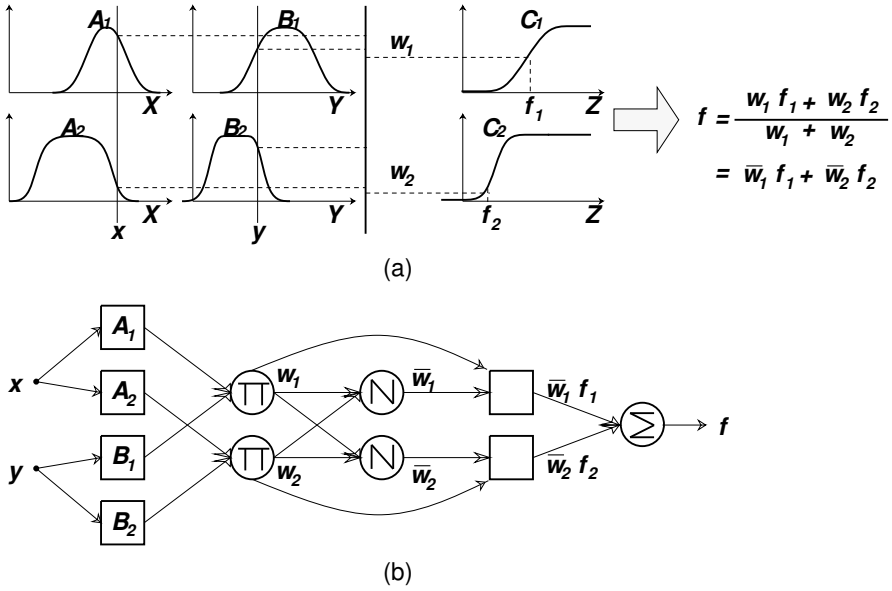


Figure 4.2: *(1) Type-1 fuzzy reasoning; (b) equivalent ANFIS (type-1 ANFIS).*

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.,

$$O_1^5 = overall\ output = \sum_i \overline{w}_i f_i = \frac{\sum_i w_i f_i}{\sum_i w_i} \tag{4.7}$$

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 the type-1 ANFIS is shown in Figure 4.2 where the output of each rule is induced jointly by the output membership funcion 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.

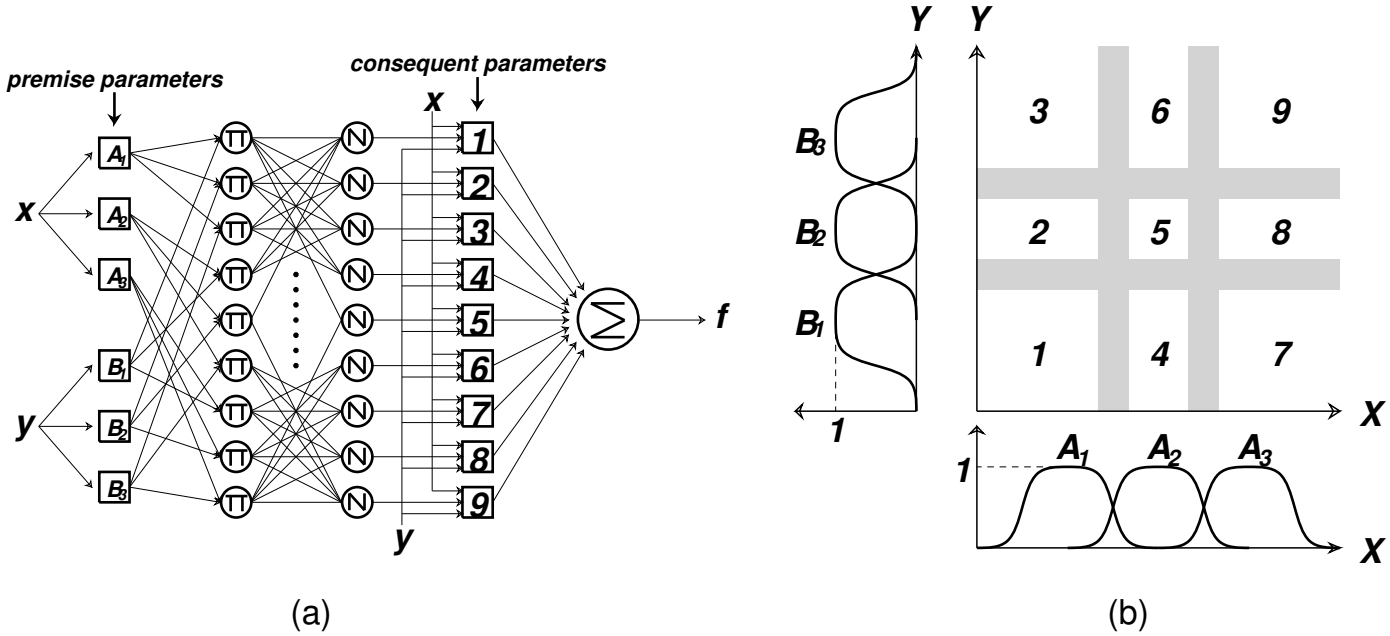


Figure 4.3: *(a) 2-input type-3 ANFIS with 9 rules; (b) corresponding fuzzy subspaces .*

Figure 4.3 shows a 2-input, type-3 ANFIS with 9 rules. Three membership functions are associated with each input, so the input space is partitioned into 9 fuzzy subspaces, each of which is governed by a fuzzy if-then rules. The premise part of a rule defines a fuzzy subspace, while the consequent part specifies the output within this fuzzy subspace.

Next we will demonstrate how to apply the hybrid learning algorithms developed in the previous chapter to identify the parameters in the ANFIS architectures mentioned above.

4.3 Hybrid Learning Algorithm

From the proposed type-3 ANFIS architecture (Figure 4.1), it is observed that given the values of premise parameters, the overall output can be expressed as a linear combinations of the consequent parameters. More precisely, the output f in Figure 4.1 can

be rewritten as

$$\begin{aligned} f &= \frac{w_1}{w_1+w_2} f_1 + \frac{w_2}{w_1+w_2} f_2 \\ &= \overline{w}_1 f_1 + \overline{w}_2 f_2 \\ &= (\overline{w}_1 x)p_1 + (\overline{w}_1 y)q_1 + (\overline{w}_1)r_1 + (\overline{w}_2 x)p_2 + (\overline{w}_2 y)q_2 + (\overline{w}_2)r_2, \end{aligned} \tag{4.8}$$

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 equation (3.10); $H(\cdot)$ and $F(\cdot,\cdot)$ are the identity function and the function of the fuzzy inference system, respectively. Therefore the hybrid learning algorithm developed in the previous chapter 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.

-	forward pass	backward pass
premise parameters	fixed	gradient descent
consequent parameters	least squares estimate	fixed
signals	node outputs	error rates

Table 4.1: *Two passes in the hybrid learning procedure 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.

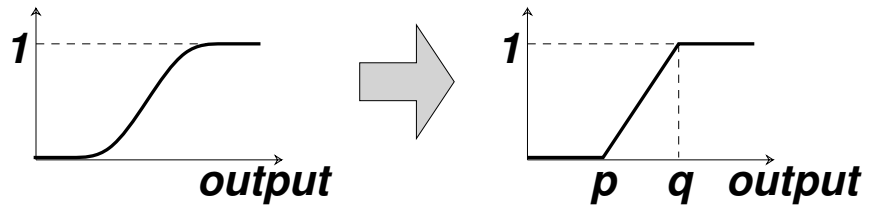


Figure 4.4: *Piecewise linear approximation of membership functions on the consequent part of type-1 ANFIS.*

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 equation (3.10). 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 4.4). 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:

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 the very 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 w.r.t. the premise parameters and the extended Kalman filter algorithm is employed to update all parameters. This has been proposed in the neural network literature [79, 77, 76].

The choice of above methods should be based on the trade-off between computation complexity and resulting performance. Our simulations presented in the next section are performed by the third method. Note that the consequent parameters can also be updated by the Widrow-Hoff LMS algorithm [103], as reported in [80]. 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 of the author's paper [30], the learning mechanisms should not be applied to the determination of membership functions since they convey linguistic and subjective description of ill-defined concepts. We think this is a case-by-case situation and the decision should be left to the users. In principle, if the 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 functions 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 the system under consideration. In this situation, the 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 [39, 64] where the "enhanced representation" of the input variables are achieved by the member-

ship functions. This "enhanced representation" which takes advantage of human knowledge are apparently more insight-revealing than the functional expansion and the tensor (outerproduct) models [64]. 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 (see Table 4.1), further speedup of learning is possible by using other versions of the gradient method on the premise parameters, such as conjugate gradient descent, second-order back-propagation [66], quick-propagation [17], nonlinear optimization [98] and many others.

4.4 Functional Equivalence to RBFN's

From equation (2.24), (2.25) and equation (4.7), it is obvious that the functional equivalence between an RBFN and a fuzzy inference system can be established if

1. The number of receptive field units is equal to the number of fuzzy if-then rules.
2. The output of each fuzzy if-then rule is composed of a constant. (Namely, p_1, q_1, p_2 and q_2 are zeros in Figure 4.1 (a).)
3. The membership functions within each rule are chosen as Gaussian functions with the same variance.
4. The T-norm operator used to compute each rule's firing strength is multiplication.
5. Both the RBFN and the fuzzy inference system under consideration use the same method (i.e., either weighted average or weighted sum) to derive their overall outputs.

Under these conditions, the membership functions of linguistic labels A_1 and B_1 in Figure 4.1 (a) can be expressed as

$$\mu_{A_1}(x_1) = exp[-\frac{(x_1 - c_{A_1})^2}{\sigma_1^2}], \quad \mu_{B_1}(x_2) = exp[-\frac{(x_2 - c_{B_1})^2}{\sigma_1^2}]. \tag{4.9}$$

Hence the firing strength (or weight) of rule 1 (the output of the first node in layer 2) is

$$w_1(x_1, x_2) = \mu_{A_1}(x_1)\mu_{B_1}(x_2) = exp[-\frac{\|\vec{x} - \vec{c_1}\|^2}{\sigma_1^2}] = R_i(\vec{x}), \tag{4.10}$$

where $\vec{c_1} = (c_{A_1}, c_{B_1})$, the center of the corresponding receptive field. The same argument applies to w_2. Therefore under the above constraints, the output of Figure 4.1(a) or (b) is exactly the same as a RBFN (with two receptive field units) where the receptive field units and output units are functionally equivalent to the cascades of layer 1, 2 and layer 3, 4, 5, respectively, in Figure 4.1. Without the above constraints, RBFN's are only a special case of fuzzy inference systems.

This functional equivalence provides us with a shortcut for better understanding of ANFIS and RBFN and advances in either literatures apply to both directly. For instance, the hybrid learning rule of ANFIS can be apply to RBFN directly and, vice versa, the approaches used to identify RBFN parameters, such as clustering preprocess [57, 58], orthogonal least squares learning [9], generalization properties [6], sequential adaptation [32], among others [31, 59], are all applicable techniques for ANFIS.

4.5 ANFIS as a Universal Approximator

4.5.1 Simplified Fuzzy If-Then Rules and the Stone-Weierstrass Theorem

Though the reasoning mechanisms (Figure 2.7) introduced in Chapter 2 are commonly used in the literature, each of them has inherent drawbacks. For type-1 reasoning (Figure 2.7 or 4.2), the membership functions on the consequence part are restricted to monotonically non-decreasing functions which are not compatible with linguistic terms such as "medium" whose membership function should be bell-shaped. For type-2 reasoning (Figure 2.7), the defuzzification process is time-consuming and systematic fine-tuning of the parameters are not easy. For type-3 reasoning (Figure 2.7 or 4.1), it is just hard to assign any appropriate linguistic terms to the consequence part which is a nonfuzzy 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 d is a *crisp* value. Due to the fact that the output 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 specifically, 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. The three reasoning mechanisms are unified under this simplified fuzzy if-then rules.

Most of 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. We will proceed this in a descriptive way by applying the Stone-Weierstrass theorem [34, 72] stated below.

Theorem 4.5.1 *Let domain D be a compact space of N dimensions, and let $\mathcal{F}$ be a set of continuous real-valued functions on D, satisfying the following criteria:*

1. **Identity function**: *The constant $f(x) = 1$ is in $\mathcal{F}$.*

2. **Separability**: *For any two points $x_1 \neq x_2$ in D, there is an f in $\mathcal{F}$ such that $f(x_1) \neq f(x_2)$.*

3. **Algebraic closure**: *If f and g are any two functions in $\mathcal{F}$, then fg and $af + bg$ are in F for any two real numbers a and b.*

Then $\mathcal{F}$ is dense in $C(D)$, the set of continuous real-valued functions on D. In other words, for any $\epsilon > 0$, and any function g in $C(D)$, there is a function f in $\mathcal{F}$ such that $|g(x) - f(x)| < \epsilon$ for all $x \in D$.

4.5.2 Application of the Stone-Weierstrass Theorem

In application of fuzzy inference systems, the domain in which we operate is almost always compact. It is a standard result in real analysis that every closed and bounded set in R^N is compact. Now we shall apply the Stone-Weierstrass theorem to show the representational power of fuzzy inference systems with simplified fuzzy if-then rules.

Identity Function

The first hypothesis of the Stone-Weierstrass theorem requires that our fuzzy inference system be able to compute the identity function $f(x) = 1$. An obvious way to compute the function is to set the consequence part of each rule equal to 1. In fact, a fuzzy inference system with only one rule suffices to satisfy this requirement.

Separability

The second hypothesis of the Stone-Weierstrass theorem requires that our fuzzy inference system be able to compute functions that have different values for different points. Without this requirement, the trivial set of functions $f : f(x) = c,\ c \in R$ would satisfy the Stone-Weierstrass theorem. Separability is satisfied whenever a fuzzy inference system can compute strictly monotonic functions of each input variable. This can easily be achieved by adjusting the membership functions of the premise part.

Algebraic Closure-Additive

The third hypothesis of the Stone-Weierstrass theorem requires that our fuzzy inference system be able to approximate sums and products of functions. Suppose we have two fuzzy inference systems S and $\overline{S}$, each of which has two rules. The ANFIS representations of S and $\overline{S}$ are shown in Figure 4.5 and Figure 4.6, respectively. The output of each system can be expressed as

$$S : \ z = \frac{w_1 f_1 + w_2 f_2}{w_1 + w_2} \tag{4.11}$$

$$\overline{S} : \ \overline{z} = \frac{\overline{w}_1 \overline{f}_1 + \overline{w}_2 \overline{f}_2}{\overline{w}_1 + \overline{w}_2} \tag{4.12}$$

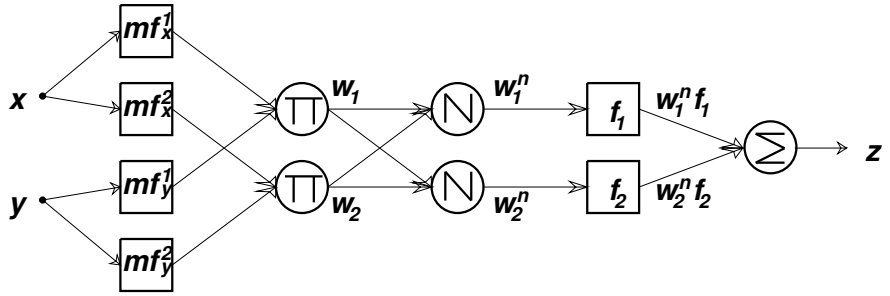


Figure 4.5: *ANFIS representation of fuzzy inference system* S.

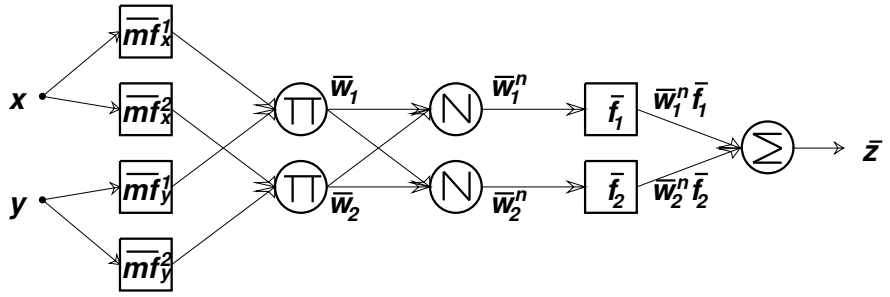


Figure 4.6: *ANFIS representation of fuzzy inference system* $\overline{S}$.

Then the sum of z and $\overline{z}$ is equal to

$$\begin{aligned} az + b\overline{z} &= a\frac{w_1 f_1 + w_2 f_2}{w_1 + w_2} + b\frac{\overline{w}_1 \overline{f}_1 + \overline{w}_2 \overline{f}_2}{\overline{w}_1 + \overline{w}_2} \\ &= \frac{w_1\overline{w}_1(af_1 + b\overline{f}_1) + w_1\overline{w}_2(af_1 + b\overline{f}_2) + w_2\overline{w}_1(af_2 + b\overline{f}_1) + w_2\overline{w}_2(af_2 + b\overline{f}_2)}{w_1\overline{w}_1 + w_1\overline{w}_2 + w_2\overline{w}_1 + w_2\overline{w}_2} \end{aligned}$$

Therefore we can construct a fuzzy inference system that computes $az + b\overline{z}$ as shown in Figure 4.7.

Algebraic Closure-Multiplicative

Modeling the product of $z\overline{z}$ of two fuzzy inference systems is the last capability we must demonstrate before we can conclude that the Stone-Weierstrass theorem can be applied to the proposed reasoning mechanism. The product $z\overline{z}$ can be expressed as

$$z\overline{z} = \frac{w_1\overline{w}_1 f_1\overline{f}_1 + w_1\overline{w}_2 f_1\overline{f}_2 + w_2\overline{w}_1 f_2\overline{f}_1 + w_2\overline{w}_2 f_2\overline{f}_2}{w_1\overline{w}_1 + w_1\overline{w}_2 + w_2\overline{w}_1 + w_2\overline{w}_2} \tag{4.13}$$

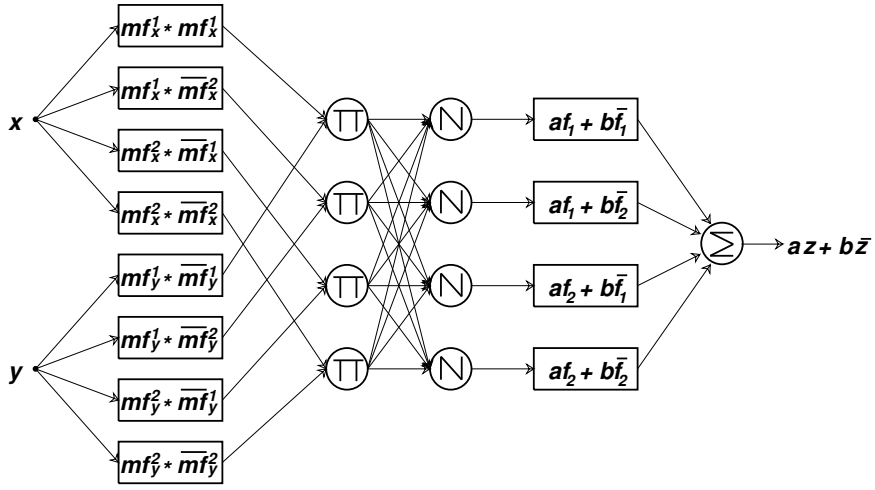


Figure 4.7: *ANFIS representation of a fuzzy inference system that computes* $az + b\overline{z}$.

Therefore we can construct a fuzzy inference system that computes $z\overline{z}$ as shown in Figure 4.8.

Apparently the ANFIS architectures that compute $az + b\overline{z}$ and $z\overline{z}$ are of the same class of S and $\overline{S}$ 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 criteria, as pointed out by Wang [94, 97], is the scaled Gaussian membership function:

$$\mu_{A_i}(x) = a_i exp[-(\frac{x - c_i}{a_i})^2], \tag{4.14}$$

Therefore by choosing an appropriate class of membership functions, we can conclude that the ANFIS with simplified fuzzy if-then rules satisfy the four criteria of the Stone-Weierstrass theorem. Consequently, for any given $\epsilon > 0$, and any real-valued function g, there is a fuzzy inference system S such that $|g(\vec{x}) - S(\vec{x})| < \epsilon$ for all $\vec{x}$ in the

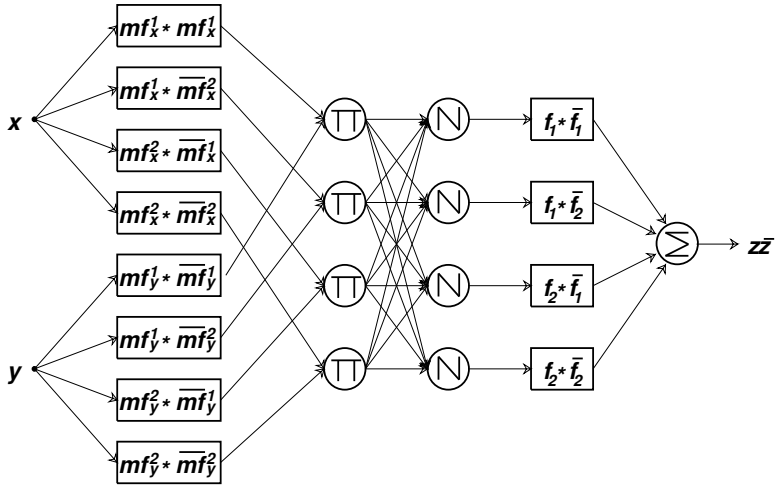


Figure 4.8: *ANFIS representation of a fuzzy inference system that computes* $z\overline{z}$.

underlying compact set. Moreover, since the simplified ANFIS is a proper subset of all three types of ANFIS in Figure 2.7, 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.

4.6 Application Examples

This section presents the simulation results of the proposed type-3 ANFIS with both batch (off-line) and pattern (on-line) learning. In the first two examples, ANFIS is used to model highly nonlinear functions and the results are compared with neural network approach and earlier work. In the third example, ANFIS is used as an identifier to identify a nonlinear component on-linely in a discrete control system. Lastly, we use ANFIS to predict a chaotic time series and compare the results with various statistical and connectionist approaches.

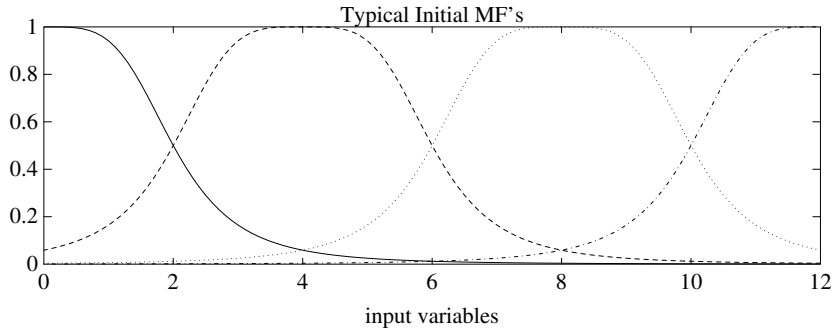


Figure 4.9: *A typical initial membership function setting in our simulation. (The operating range is assumed to be* $[0, 12]$*.)*

4.6.1 Practical Considerations

In a conventional fuzzy inference system, the number of rules is decided by an expert who is familiar with the system to be modeled. In our simulation, however, no expert is available and the number of membership functions (MF's) assigned to each input variable is chosen empirically, i.e., by examining the desired input-output data and/or by trial and error. This situation is much the same as that of neural networks; there are no simple ways to determine in advance the minimal number of hidden nodes necessary to achieve a desired performance level.

After the number of MF's associated with each inputs are fixed, the initial values of premise parameters are set in such a way that the MF's are equally spaced along the operating range of each input variable. Moreover, they satisfy ϵ-*completeness* [46, 47] with $\epsilon = 0.5$, which means that given a value x of one of the inputs in the operating range, we can always find a linguistic label A such that $\mu_A(x) \geq \epsilon$. In this manner, the fuzzy inference system can provide smooth transition and sufficient overlapping from one linguistic label to another. Though we did not attempt to keep the *epsilon*-completeness during the learning

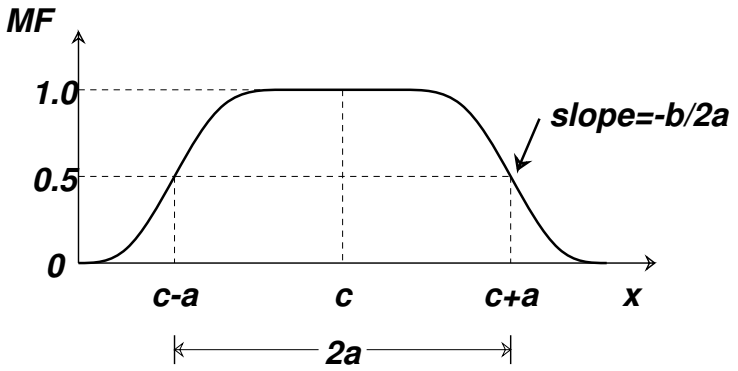


Figure 4.10: *Physical meanings of the parameters in the bell membership function* $\mu_A(x) = \frac{1}{1+[(\frac{x-c}{a})^2]^b}$. .

in our simulation, it can be easily achieved by using the constrained gradient method [105]. Figure 4.9 shows a typical initial MF setting when the number of MF is 4 and the operating range is $[0, 12]$. Note that throughout the simulation examples presented below, all the membership functions used are the **bell function** defined in equation (6.4):

$$\mu_A(x) = \frac{1}{1 + [(\frac{x-c}{a})^2]^b}, \tag{4.15}$$

which contains three fitting parameters a, b and c. Each of these parameters has a physical meaning: c determines the center of the corresponding membership function; a is the half width; and b (together with a) controls the slopes at the crossover points (where MF value is 0.5). Figure 4.10 shows these concepts.

We mentioned that the step size k in equation (3.8) may influence the speed of convergence. It is observed that if k is small, the gradient method will closely approximate the gradient path, but convergence will be slow since the gradient must be calculated many times. On the other hand, if k is large, convergence will initially be very fast, but the algorithm will oscillate about the optimum. Based on these observations, we update k

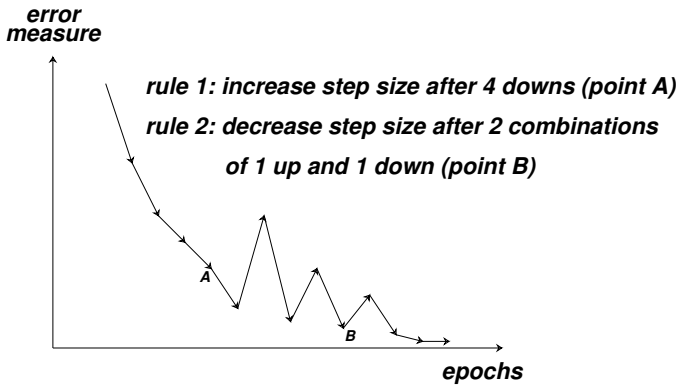


Figure 4.11: *Two heuristic rules for updating step size k.*

according to the following two heuristic rules (see Figure 4.11):

1. If the error measure undergoes 4 consecutive reductions, increase k by 10%.

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

Though the numbers 10%, 4 and 2 are chosen more or less arbitrarily, the results shown in our simulation appear to be satisfactory. Furthermore, due to this dynamical update strategy, the initial value of k is usually not critical as long as it is not too big.

4.6.2 Example 1: Modeling a Two-Input Nonlinear Function

In this example, we consider using ANFIS to model a nonlinear *sinc* equation

$$z = sinc(x, y) = \frac{sin(x)}{x} \times \frac{sin(y)}{y}. \tag{4.16}$$

From the grid points of the range $[-10, 10] \times [-10, 10]$ within the input space of the above equation, 121 training data pairs were obtained first. The ANFIS used here contains 16 rules, with four membership functions being assigned to each input variable and the total

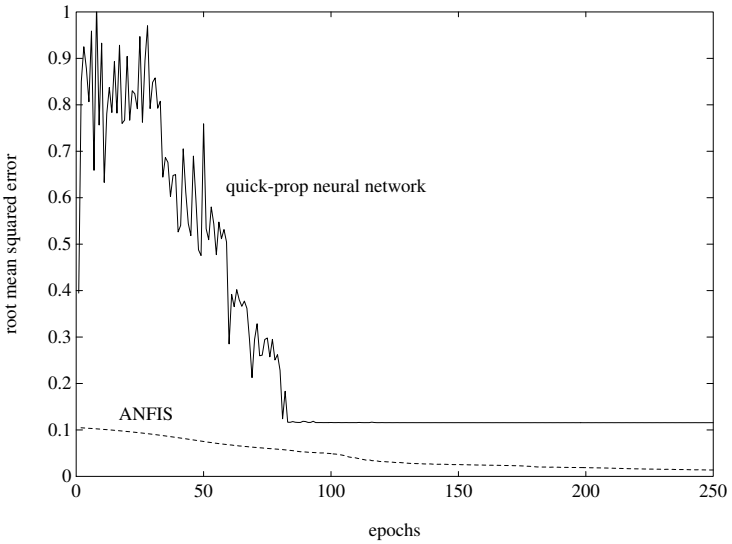


Figure 4.12: *RMSE curves for the quick-propagation neural networks and the ANFIS.*

number of fitting parameters is 72 which are composed of 24 premise parameters and 48 consequent parameters. (We also tried ANFIS with 4 rules and 9 rules, but obviously they are too simple to describe the highly nonlinear sinc function.)

Figure 4.12 shows the RMSE (root mean squared error) curves for both the 2-18-1 neural network and the ANFIS. Each curve is the average of ten runs: for the neural network, this ten runs were started from 10 different set of initial random weights; for the ANFIS, 10 different initial step size ($= 0.01, 0.02, \ldots, 0.10$) were used. The neural network, containing 73 fitting parameters (connection weights and thresholds), was trained with quick propagation [17] which is considered one of the best learning algorithms for connectionist models. Figure 4.12 demonstrate how ANFIS can effectively model a highly nonlinear surface as compared to neural networks. However, this comparison cannot taken to be universal since we did not attempt an exhaustive search to find the optimal settings for the quick-propagation learning rule of the neural networks.

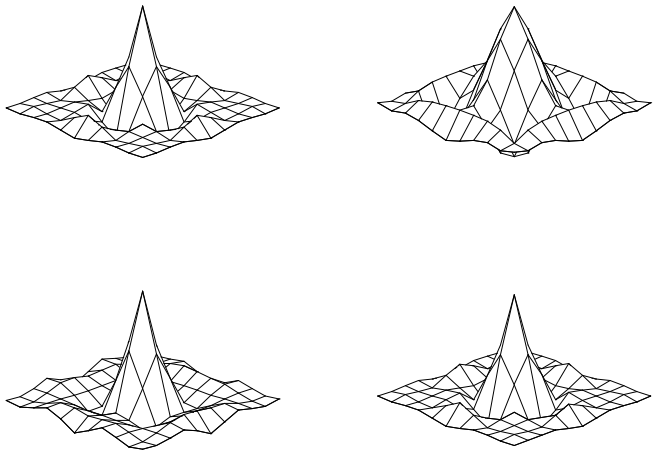


Figure 4.13: *Training data (upper left) and reconstructed surfaces at 0.5 (upper right), 99.5 (lower left) and 249.5 (lower right) epochs. (Example 1).*

The training data and other reconstructed surfaces at different epoch numbers are shown in Figure 4.13. (Since the error measure is always computed after the forward pass is over, the epoch numbers shown in Figure 4.13 always end with ".5".) Note that the reconstructed surface after 0.5 epoch is due to the identification of consequent parameters only and it already looks similar to the training data surface.

Figure 4.14 lists the initial and final membership functions. It is interesting to observe that the sharp changes of the training data surface around the origin is accounted for by the moving of the membership functions toward the origin. Theoretically, the final MF's on both x and y should be symmetric with respect to the origin. However, they are not symmetric due to the computer truncation errors and the approximate initial conditions for bootstrapping the calculation of the sequential least squares estimate 3.14.

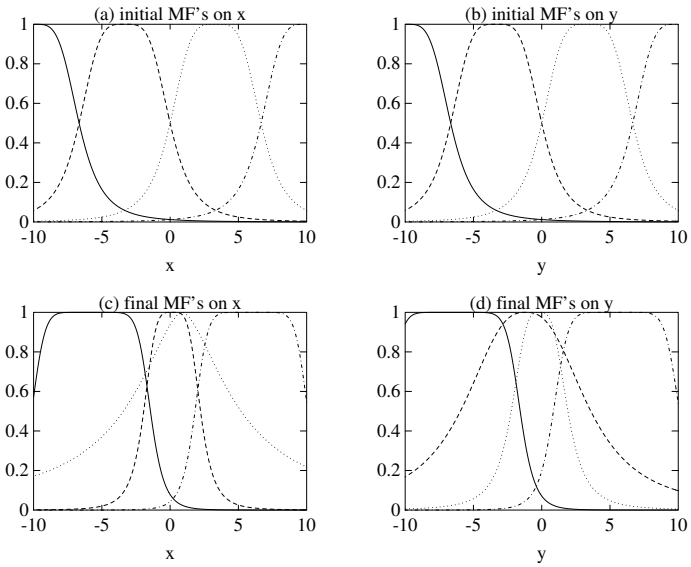


Figure 4.14: *Initial and final membership functions of example 1.*

4.6.3 Example 2: Modeling a Three-Input Nonlinear Function

The training data in this example are obtained from

$$output = (1 + x^{0.5} + y^{-1} + z^{-1.5})^2, \tag{4.17}$$

which was also used by Takagi et al. [89], Sugeno et al. [83] and Kondo [40] to verify their approaches. The ANFIS (see Figure 4.15) used here contains 8 rules, with 2 membership functions being assigned to each input variable. 216 training data and 125 checking data were sampled uniformly from the input ranges $[1,6]\times[1,6]\times[1,6]$ and $[1.5,5.5]\times[1.5,5.5]\times[1.5,5.5]$, respectively. The training data was used for the training of ANFIS, while the checking data was used for verifying the identified ANFIS only. To allow comparison, we use the same performance index adopted in [83, 40]:

$$APE \;=\; average\,percentage\,error \;=\; \frac{1}{P}\sum_{i=1}^{P}\frac{\mid T(i)-O(i)\mid}{\mid T(i)\mid} * 100\%. \tag{4.18}$$

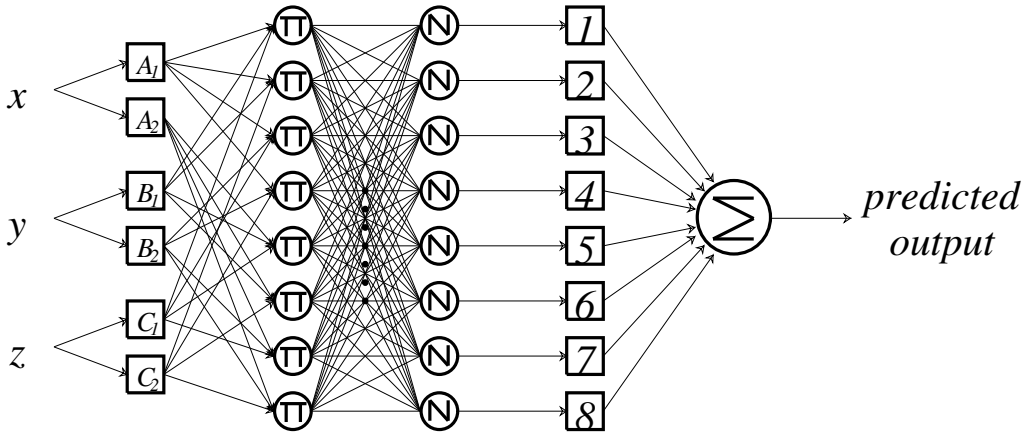


Figure 4.15: *The ANFIS architecture for example 2. (The connections from inputs to layer 4 are not shown.)*

where P is the number of data pairs; $T(i)$ and $O(i)$ are i-th desired output and calculated output, respectively.

Figure 4.16 illustrates the membership functions before and after training. The training error curves with different initial step sizes (from 0.01 to 0.09) are shown in Figure 4.17(a), which demonstrates that the initial step size is not too critical on the final performance as long as it is not too big. Figure 4.17(b) is the training and checking error curves with initial step size equal to 0.1. After 199.5 epochs, the final results are $APE_{trn} = 0.043\%$ and $APE_{chk} = 1.066\%$, which is listed in Table 4.2 along with other earlier work [83, 40]. Since each simulation cited here was performed under different assumptions and with different training and checking data sets, we cannot make conclusive comments here.

4.6.4 Example 3: On-line Identification in Control Systems

Here we repeat the simulation example 1 of [60] where a 1-20-10-1 neural network is employed to identify a nonlinear component in a control system, except that we use

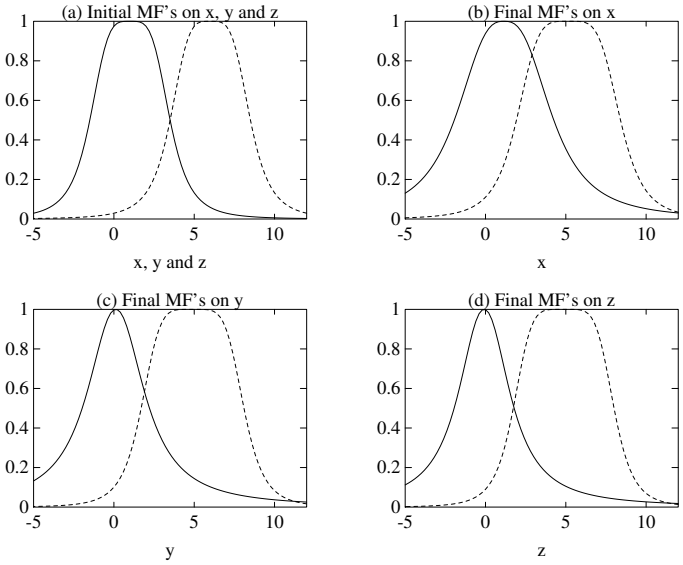


Figure 4.16: *Example 2, (a) membership functions before learning; (b)(c) (d) membership functions after learning.*

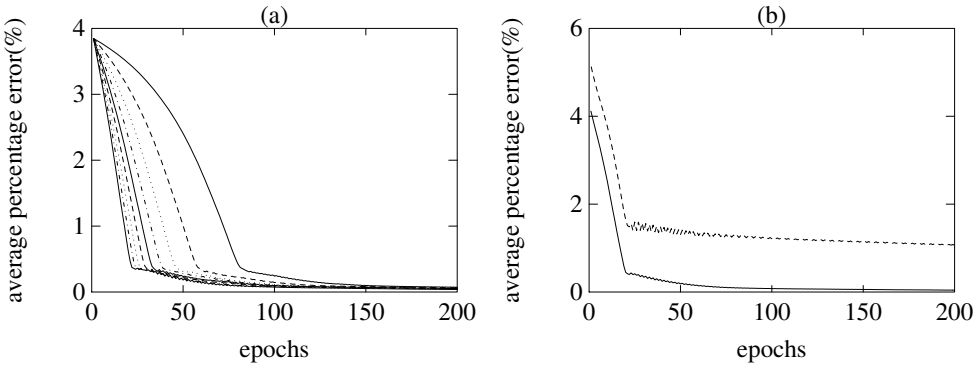


Figure 4.17: *Error curves of example 2: (a) 9 training error curves for 9 initial step size from 0.01 (solid line) to 0.09; (b) training (solid line) and checking (dashed line) error curves with initial step size equal to 0.1.*

Model	APE_{trn}	APE_{chk}	Para. no.	Training Size	Checking Size
ANFIS	0.043%	1.066%	50	216	125
GMDH model [40]	4.7%	5.7%	-	20	20
Fuzzy model 1 [83]	1.5%	2.1%	22	20	20
Fuzzy model 2 [83]	0.59%	3.4%	32	20	20

Table 4.2: *Example 2: comparisons with earlier work. (The last three rows are from [83].)*

ANFIS to replace the neural network. The plant under consideration is governed by the following difference equation:

$$y(k+1) = 0.3y(k) + 0.6y(k-1) + f(u(k)), \tag{4.19}$$

where y(k) and u(k) are the output and input, respectively, at time index k, and the unknown function $f(\cdot)$ has the form

$$f(u) = 0.6sin(\pi u) + 0.3sin(3\pi u) + 0.1sin(5\pi u). \tag{4.20}$$

In order to identify the plant, a series-parallel model governed by the difference equation

$$\hat{y}(k+1) = 0.3\hat{y}(k) + 0.6\hat{y}(k-1) + F(u(k)) \tag{4.21}$$

was used where $F(\cdot)$ is the function implemented by ANFIS and its parameters are updated at each time index. Here the ANFIS has 7 membership functions on its input (thus 7 rules, and 35 fitting parameters) and the pattern (on-line) learning paradigm was adopted with a learning rate $\eta = 0.1$ and a forgetting factor $\lambda = 0.99$. The input to the plant and the model was a sinusoid $u(k) = sin(2\pi k/250)$ and the adaptation started at $k = 1$ and stopped at $k = 250$. As shown in Figure 4.18, the output of the model follows the output of the plant almost immediately even after the adaptation stopped at $k = 250$ and the $u(k)$ is changed to $0.5sin(2\pi k/250) + 0.5sin(2\pi k/25)$ after $k = 500$. As a comparison, the

neural network in [60] fails to follow the plant when the adaptation stopped at $k = 500$ and the identification procedure had to continue for $50,000$ time steps using a random input. Table 4.3 summarizes the comparison.

Method	Parameter Number	Time Steps of Adaptation
NN	261	50000
ANFIS	35	250

Table 4.3: *Example 3: comparison with NN identifier [60].)*

In the above, the MF number is determined by trial and errors. If the MF number is below 7 then the model output will not follow the plant output satisfactorily after 250 adaptations. But can we decrease the parameter numbers by using batch learning which is supposed to be more effective? Figure 4.19, 4.20 and 4.21 show the results after 49.5 epochs of batch learning when the MF numbers are 5, 4 and 3, respectively. As can be seen, the ANFIS is a good model even when the MF is as small as 3. However, as the MF number is getting smaller, the correlation between $F(u)$ and each rule's output is getting less obvious in the sense that it is harder to sketch $F(u)$ from each rule's consequent part. In other words, when the parameter number is reduced mildly, usually the ANFIS can still do the job but at the cost of sacrificing its semantics in terms of the local-description nature of fuzzy if-then rules; it is less of a structured knowledge representation and more of a black-box model (like neural networks).

4.6.5 Example 4: Predicting Chaotic Dynamics

Example 1, 2 and 3 show that the ANFIS can be used to model highly nonlinear functions effectively. In this example, we will demonstrate how the proposed ANFIS can be

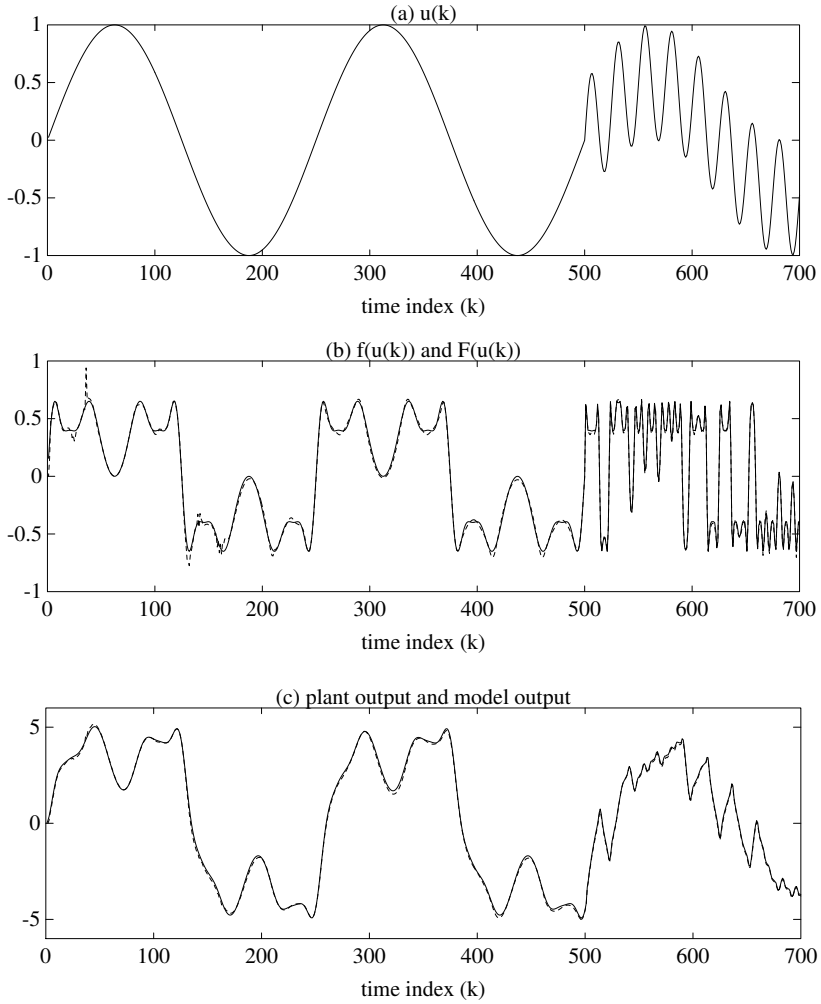


Figure 4.18: *Example 3: (a) $u(k)$; (a) $f(u(k))$ and $F(u(k))$; (b) plant output and model output.*

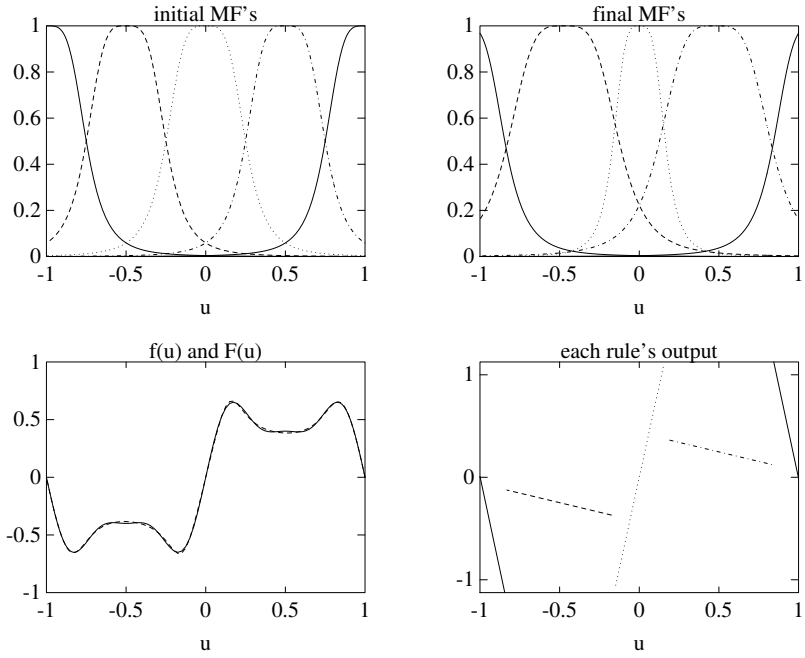


Figure 4.19: *Example 3: batch learning with 5 MF's.*

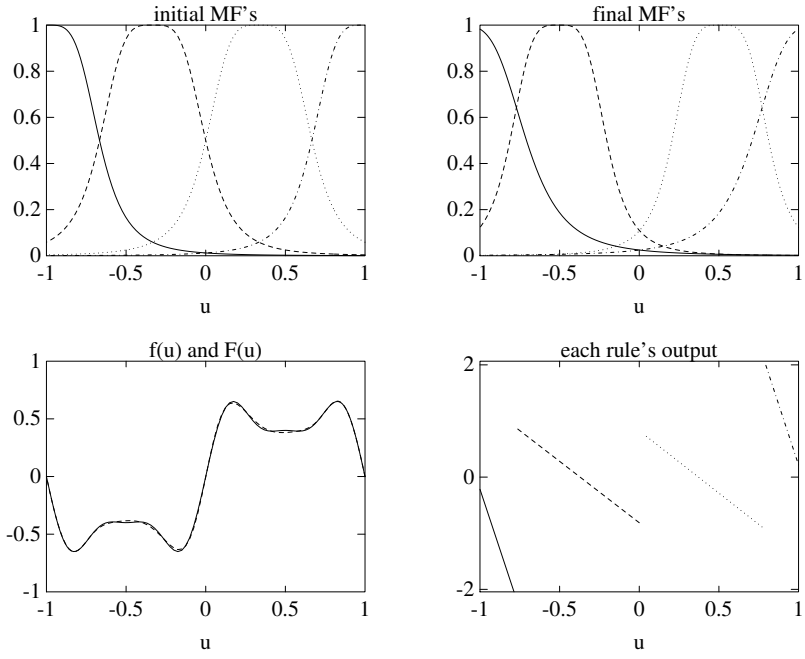


Figure 4.20: *Example 3: batch learning with 4 MF's.*

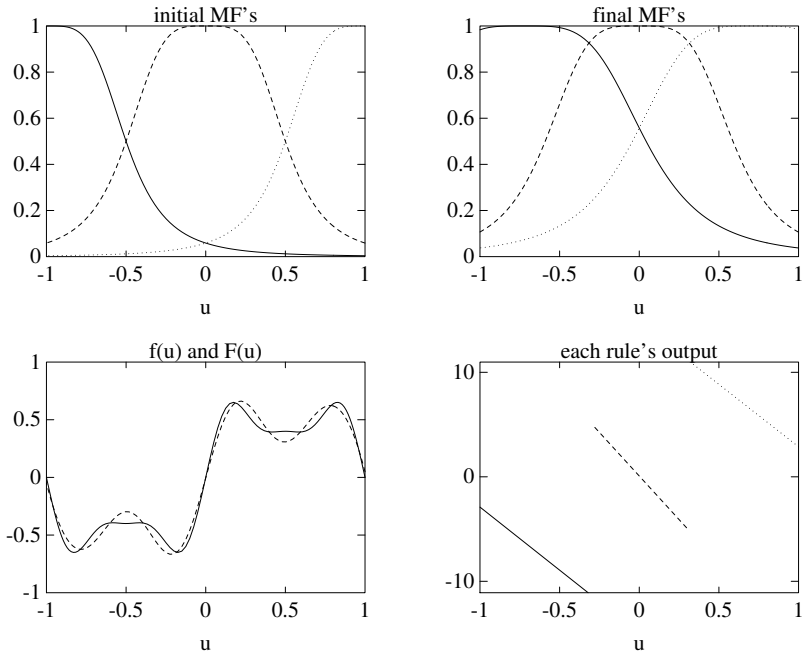


Figure 4.21: *Example 3: batch learning with 3 MF's.*

employed to predict future values of a chaotic time series. The performance obtained in this example will be compared with the results of a cascade-correlation neural network approach reported in [69] and a simple conventional statistical approach, the auto-regressive (AR) model.

The time series used in our simulation is generated by the chaotic Mackey-Glass differential delay equation [54] defined below:

$$\dot{x}(t) = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1x(t). \tag{4.22}$$

The prediction of future values of this time series is a benchmark problem which has been considered by a number of connectionist researchers (Lapedes and Farber [45], Moody [58, 56], Jones et al. [31], Crower [69] and Sanger [74]).

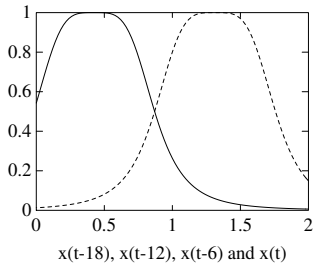
The goal of the task is to use known values of the time series up to the point $x = t$ to predict the value at some point in the future $x = t + P$. The standard method for this type of prediction is to create a mapping from D points of the time series spaced $\triangle$ apart, that is, $(x(t-(D-1)\triangle), ..., x(t-\triangle), x(t))$, to a predicted future value $x(t+P)$. To allow comparison with earlier work (Lapedes and Farber [45], Moody [58, 56], Crower [69]), the values $D = 4$ and $\triangle = P = 6$ were used. All other simulation settings in this example were purposedly arranged to be as close as possible to those reported in [69].

To obtain the time series value at each integer point, we applied the fourth-order Runge-Kutta method to find the numerical solution to equation (4.22). The time step used in the method is 0.1, initial condition $x(0) = 1.2$, $\tau = 17$, and $x(t)$ is thus derived for $0 \leq t \leq 2000$. (We assume $x(t) = 0$ for $t < 0$ in the integration.) From the Mackey-Glass time series $x(t)$, we extracted 1000 input-output data pairs of the following format:

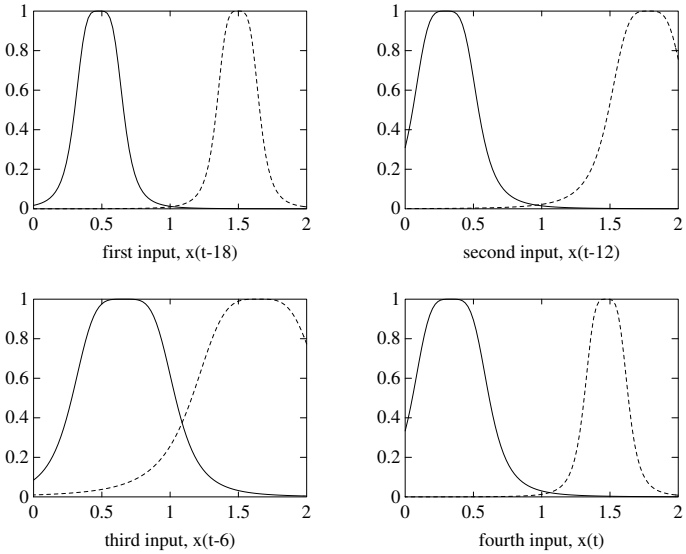
$$[x(t-18), x(t-12), x(t-6), x(t); x(t+6)], \tag{4.23}$$

where $t = 118$ to 1117. The first 500 pairs (training data set) was used for training the ANFIS while the remaining 500 pairs (checking data set) were used for validating the identified model. The number of membership functions assigned to each input of the ANFIS was arbitrarily set to 2, so the rule number is 16. Figure 4.22 (a) is the initial membership functions for each input variable. The ANFIS used here contains a total of 104 fitting parameters, of which 24 are premise parameters and 80 are consequent parameters

After 499.5 epochs, we had $RMSE_{trn} = 0.0016$ and $RMSE_{chk} = 0.0015$, which are much better when compared with other approaches explained below. The resulting 16 fuzzy if-then rules are listed in the Appendix. The desired and predicted values for



(a) Initial MF's for all four input variables.



(b) Final MF's for each input variable.

Figure 4.22: *Membership functions of example 4, (a) before learning; (b) after learning.*

both training data and checking data are essentially the same in Figure 4.23(a); their differences (Figure 4.23(b)) can only be seen on a finer scale. Figure 4.22 (b) is the final membership functions; Figure 4.24 shows the RMSE curves which indicate most of the learning was done in the first 100 epochs. It is quite unusual to observe the phenomenon that $RMSE_{trn} < RMSE_{chk}$ during the training process. Considering both the RMSE's are vary small, we conclude that: (1) the ANFIS has captured the essential components of the underlying dynamics; (2) the training data contains the effects of the initial conditions

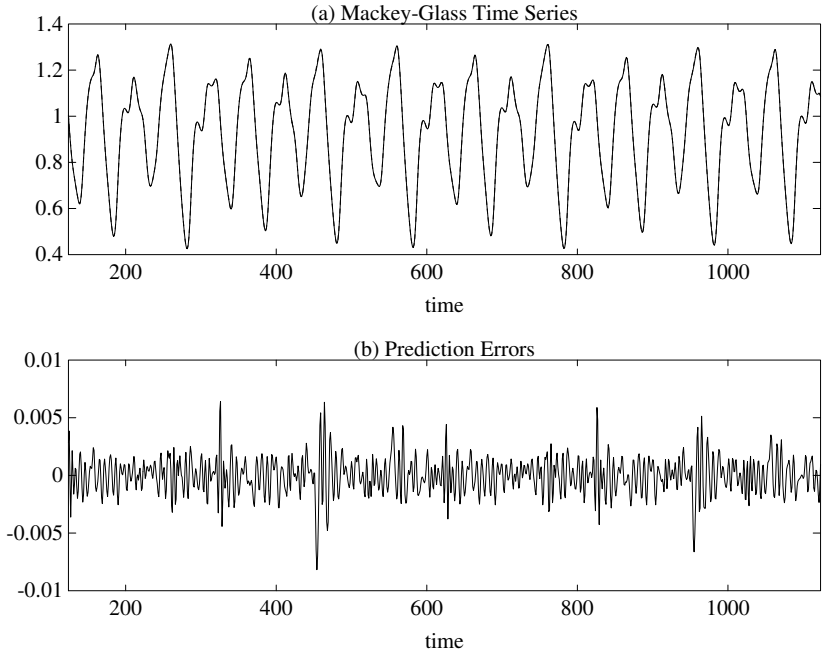


Figure 4.23: Example 3, (a) Mackey-Glass time series from $t = 124$ to 1123 and six-step ahead prediction (which is indistinguishable from the time series here); (b) prediction error.

(remember that we set $x(t) = 0$ for $t \leq 0$ in the integration) which might not be easily accounted for by the essential components identified by the ANFIS.

As a comparison, we performed the same prediction by using the auto-regressive (AR) model with the same number of parameters:

$$x(t+6) = a_0 + a_1 x(t) + a_2 x(t-6) + ... + a_{103} x(t - 102 * 6), \tag{4.24}$$

where there are 104 fitting parameters a_k, $k = 0$ to 103. From $t = 712$ to 1711, we extracted 1000 data pairs, of which the first 500 were used to identify a_k and the remaining were used for checking. The results obtained through the standard least squares estimate are $RMSE_{trn} = 0.005$ and $RMSE_{chk} = 0.078$ which is much worse than those of ANFIS. Figure 4.25 shows the predicted values and the prediction errors. Obviously, the over-

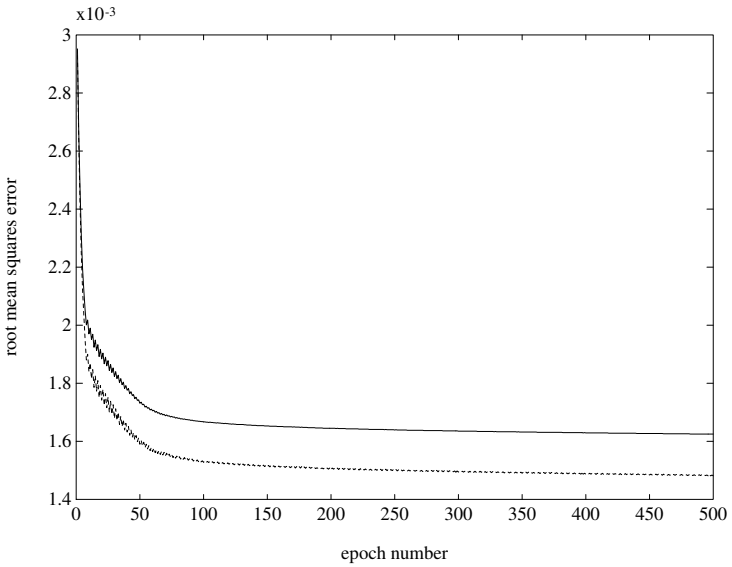


Figure 4.24: *Training and checking RMSE curves for ANFIS modeling.*

parameterization of the AR model causes over-fitting in the training data and large errors in the checking data. To search for the best AR model in terms of generalization capability, we tried out different AR models with parameter number being varied from 2 to 104; Figure 4.26 shows the results where the AR model with the best generalization capability is obtained when the parameter number is 45. Based on this best AR model, we repeat the generalization test and Figure 4.27 shows the results where there is no over-fitting at the price of larger training errors.

It goes with saying that the nonlinear ANFIS outperforms the linear AR model. However, it should be noted that the identification of the AR model took only a few seconds, while the ANFIS simulation took about 2.5 hours on a HP Apollo 700 Series workstation. (We did not pay special attention on the optimization of the codes, though.)

Table 4.4 lists other methods' generalization capabilities which are measured by using each method to predict 500 points immediately following the training set. Here the

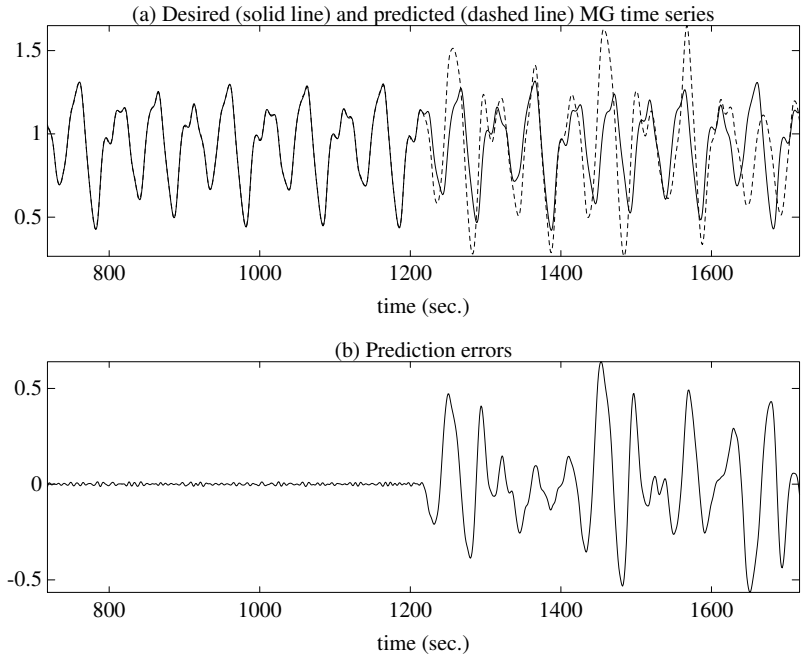


Figure 4.25: *(a) Mackey-Glass time series (solid line) from* $t = 718$ *to* 1717 *and six-step ahead prediction (dashed line) by AR model with parameter* $= 104$*; (b) prediction errors.*

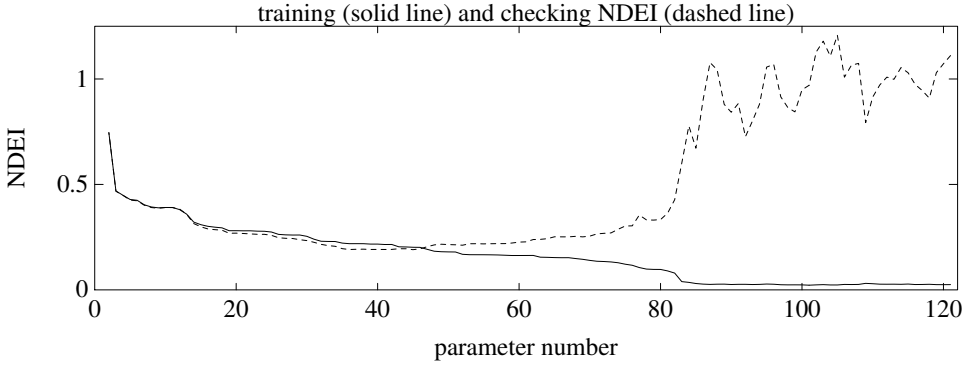


Figure 4.26: *Training (solid line) and checking (dashed line) errors of AR models with different parameter numbers.*

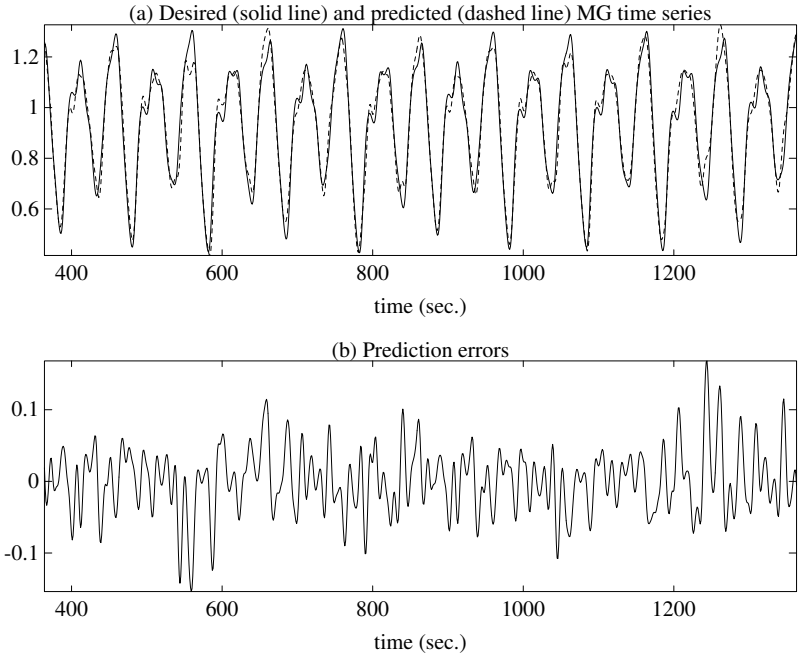


Figure 4.27: *Example 3, (a) Mackey-Glass time series (solid line) from* $t = 364$ *to* 1363 *and six-step ahead prediction (dashed line) by the best AR model (parameter number* $= 45$*); (b) prediction errors.*

non-dimensional error index (NDEI) [45, 69] is defined as the root mean square error divided by the standard deviation of the target series. (Note that the *average relative variance* used in [99, 100] is equal to the square of NDEI.) The remarkable generalization capability of the ANFIS, we believe, comes from the following facts:

1. The ANFIS can achieve a highly nonlinear mapping as shown in Example 1, 2 and 3, therefore it is superior to common linear methods in reproducing nonlinear time series.

2. The ANFIS used here has 104 adjustable parameters, much less than those of the cascade-correlation NN (693, the median size) and back-prop NN (about 540) listed in Table 4.4.

3. Though without *a priori* knowledge, the initial parameter settings of ANFIS are intuitively reasonable and it leads to fast learning that captures the underlying dynamics.

Method	Training Cases	Non-Dimensional Error Index
ANFIS	500	0.007
AR Model	500	0.19
Cascaded-Correlation NN	500	0.06
Back-Prop NN	500	0.02
6th-order Polynomial	500	0.04
Linear Predictive Method	2000	0.55

Table 4.4: *Generalization result comparisons for* $P = 6$. *(The last four rows are from [69].)*

Table 4.5 lists the results of the more challenging generalization test when $P = 84$ (the first six rows) and $P = 85$ (the last four rows). The results of the first six rows were obtained by iterating the prediction of $P = 6$ till $P = 84$. ANFIS still outperforms these

Method	Training Cases	Non-Dimensional Error Index
ANFIS	500	0.036
AR Model	500	0.39
Cascaded-Correlation NN	500	0.32
Back-Prop NN	500	0.05
6th-order Polynomial	500	0.85
Linear Predictive Method	2000	0.60
LRF	500	0.10-0.25
LRF	10000	0.025-0.05
MRH	500	0.05
MRH	10000	0.02

Table 4.5: *Generalization result comparisons for* $P = 84$ *(the first six rows) and* 85 *(the last four rows). Results for the first six methods are generated by iterating the solution at* $P = 6$. *Results for localized receptive fields (LRF) are multi-resolution hierarchies (MRH) are for networks trained for* $P = 85$. *(The last eight rows are from [69].)*

statistical and connectionist approaches unless a substantially large amount of training data (i.e., the last row of Table 4.5) were used instead. Figure 4.28 illustrates the generalization test for the ANFIS where the first 500 points were used for the desired outputs while the last 500 are the predicted outputs for $P = 84$.

4.7 Concluding Remarks

4.7.1 Summary and Extensions of Current work

We have described the architecture of adaptive-network-based fuzzy inference systems (ANFIS) with type-1 and type-3 reasoning mechanisms. By employing a hybrid learning procedure, the proposed architecture can refine fuzzy if-then rules obtained from human experts to describe the input-output behavior of a complex system. However, if human expertise is not available, we can still set up intuitively reasonable initial membership functions

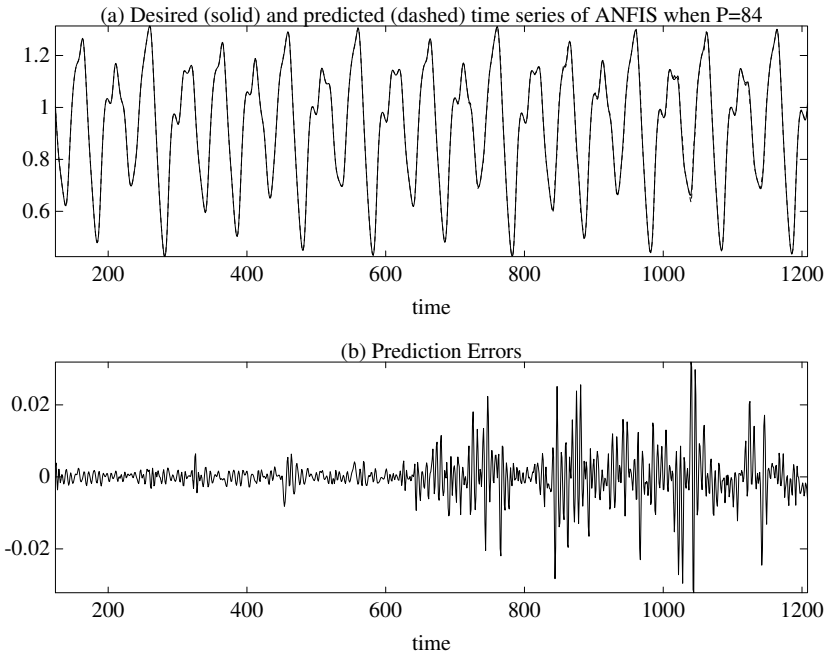


Figure 4.28: *Generalization test of ANFIS for* $P = 84$.

and start the learning process to generate a set of fuzzy if-then rules to approximate a desired data set, as shown in the simulation examples of nonlinear function modeling and chaotic time series prediction.

Due to the high flexibility of adaptive networks, the ANFIS can have a number of variants from what we have proposed here. For instance, the membership functions can be changed to *L-R representation* [15] which could be asymmetric, Furthermore, we can replace Π nodes in layer 2 with the parameterized *T-norm* [15] and let the learning rule to decide the best T-norm operator for a specific application. By employing the adaptive network as a common framework, we have also proposed other adaptive fuzzy models tailored for data classification [85, 86] and feature extraction [87] purposes.

Another important issue in the training of ANFIS is how to preserve the human-

plausible features such as bell-shaped membership functions, ϵ-completeness [46, 47] or sufficient overlapping between adjacent membership functions, minimal uncertainty, etc. Though we did not pursue along this direction in this paper, mostly it can be achieved by maintaining certain constraints and/or modifying the original error measure as explained below.

- To keep bell-shaped membership functions, we need the membership functions to be bell-shaped regardless of the parameter values. In particular, equation (6.4) and equation (6.5) become up-side-down bell-shaped if $b_i < 0$; one easy way to correct this is to replace b_i with b_i^2 in both equations.

- The ϵ-completeness can be maintained by the constrained gradient descent [105]. For instance, suppose that $\epsilon = 0.5$ and the adjacent membership functions are of the form of equation (6.4) with parameter sets $\{a_i, b_i, c_i\}$ and $\{a_{i+1}, b_{i+1}, c_{i+1}\}$. Then the ϵ-completeness is satisfied if $c_i + a_i = c_{i+1} - a_{i+1}$ and this can be ensured throughout the training if the constrained gradient descent is employed.

- Minimal uncertainty refers to the situation that within most region of the input space, there should be a dominant fuzzy if-then rule to account for the final output, instead of multiple rules with similar firing strengths. This minimizes the uncertainty and make the rule set more informative. One way to do this is to use a modified error measure

$$E' = E + \beta \sum_{i=1}^{P} [-\overline{w}_i \times ln(\overline{w}_i)], \tag{4.25}$$

where E is the original squared error; β is a weighting constant; P is the size of training data set; $\overline{w}_i$ is the normalized firing strength of the i-th rule (see equation (4.5)) and

$\sum_{i=1}^{P}[-\overline{w}_i \times ln(\overline{w}_i)]$ is the *information entropy*. Since this modified error measure is not based on data fitting along, the ANFIS thus trained can also have a potentially better generalization capability. (However, due to this new error measure, the training should be based on the gradient descent alone.) The improvement of generalization by using an error measure based both data fitting and weight elimination has been reported in the neural network literature [99, 100].

In this paper, we assume the structure of the ANFIS is fixed and the *parameter identification* is solved through the hybrid learning rule. However, to make the whole approach more complete, the *structure identification* [83, 84] (which concerns with the selection of an appropriate input-space partition style and the number of membership functions on each input, etc.) is equally important to the successful applications of ANFIS. Effective partition of the input space can decrease the rule number and thus increase the speed in both learning and application phases. Advances on neural networks' structure identification [18, 50] can shed some lights on this aspect.

4.7.2 Applications to Automatic Control and Signal Processing

Fuzzy control is by far the most successful applications of the fuzzy set theory and fuzzy inference systems. Due to the adaptive capability of ANFIS, its applications to adaptive control and learning control are immediate. Most of all, it can replace almost any neural networks in control systems to serve the same purposes. For instance, Narendra's pioneering work of using neural networks in adaptive control [60] can be all achieved similarly by ANFIS. Moreover, four of the generic designs (i.e., *supervised control*, *direct*

inverse control, *neural adaptive control* and *back-propagation of utility*) of neural networks in control, as proposed by Werbos [102, 25], are also directly applicable schemes for ANFIS. Particularly we have employed a similar method of the *back-propagation through time* [62] or *unfolding in time* to achieve a self-learning fuzzy controller with four rules that can balance an inverted pendulum in an near-optimal manner [29]; this will be introduced in the next chapter. It is expected that the advances of neural network techniques in control can promote those of ANFIS as well, and vice versa.

The active role of neural networks in signal processing [104, 42] also suggests similar applications of ANFIS. The nonlinearity and structured knowledge representation of ANFIS are the primary advantages over classical linear approaches in adaptive filtering [22] and adaptive signal processing [103], such as identification, inverse modeling, predictive coding, adaptive channel equalization, adaptive interference (noise or echo) canceling, etc.

Chapter 5

Self-Learning Intelligent Control

5.1 Introduction

Fuzzy controllers (FC's) have recently found various applications in industry as well as in household appliances. For complex and/or ill-defined systems that are not easily controlled by conventional control schemes, FC's provide a feasible alternative since they can easily capture the approximate, qualitative aspects of human knowledge and reasoning. However, the performance of FC's relies on two important factors: the soundness of knowledge acquisition techniques and the availability of domain (human) experts. These two factors substantially restrict the application domains of FC's.

The ANFIS (Adaptive-Network-based Fuzzy Inference System) architecture described in Chapter 3 is designed to solve the first problem concerning the automatic elicitation of knowledge in the form of fuzzy if-then rules. The proposed architecture can identify the near-optimal membership functions and other parameters of a rule base for achieving a desired input-output mapping. The basics of the ANFIS architecture are introduced in the

next section.

This chapter addresses the second problem: how to control a system through a self-learning FC. In other words, without resorting to human experts, we want to construct an FC that can perform a prescribed control task. The learning aspects of FC's have always been an interesting topic, and recent developments are mostly based on reinforcement learning [48, 49, 5]. Our learning method is based on a special form of gradient descent (called back propagation), which is used for training artificial neural networks [73, 101]. To control the plant's trajectory, we apply the back-propagation-type gradient descent method to propagate the error signals through different time stages. This is called TBP (*temporal back propagation*) and it is explained in Section 3.

The proposed control strategy is quite general and can be used to control plants with diverse characteristics. Moreover, the *a priori* knowledge that we have about the plant can be applied in an auxiliary manner to speed up the learning process. In our simulation described in Section 4, we successfully employ the TBP to construct a fuzzy controller with only 4 fuzzy if-then rules for balancing an inverted pendulum system.

5.2 Self-Learning Fuzzy Controllers through Temporal Back Propagation

In this section, we propose a generalized control scheme which can construct a fuzzy controller through *temporal back propagation*, such that the state variables can follow a given desired trajectory as close as possible. The basic idea is to implement both the controller and the plant at each time stage as a *stage adaptive network*, and cascade these

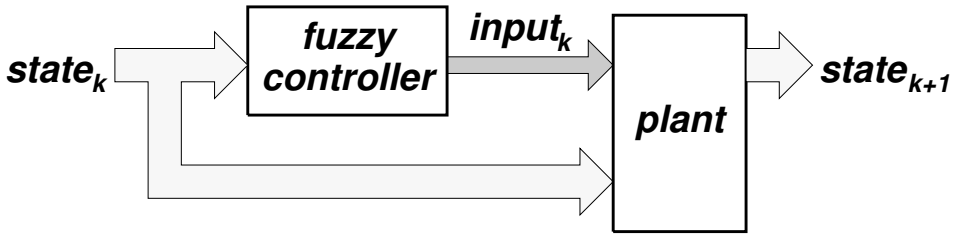


Figure 5.1: *Block diagram of a fuzzy controller and a plant. Also a stage adaptive network at time stage k.*

stage adaptive networks into a *trajectory adaptive network* to facilitate the temporal back propagation learning process.

5.2.1 Stage Adaptive Network

Figure 5.1 illustrates the block diagram of a feedback control system consisting of a fuzzy controller and a plant. We assume the delay through the controller is small and the state variables are accessible with accuracy. Moreover, the plant block is viewed as a static system since the dependency of the next state on the present state is shown explicitly. Before finding a controller to control the plant state, we have to find mathematical expressions for both the controller block and the plant block. This step is referred to as the *implementation*. In our case, we are going to implement both blocks as adaptive networks.

An obvious candidate for implementing the FC block in Figure 5.1 is the ANFIS architecture, since it has exactly the same function as a fuzzy controller, as shown in Figure 4.1. If we have p inputs to the plant, then the FC block can be implemented either as p ANFIS's, or as an ANFIS that has rules with multiple consequents.

Suppose that we have a human expert who knows how to control the plant. Then the domain knowledge can be transformed into fuzzy if-then rules and the corresponding

parameters (which characterize membership functions) can be used as the initial parameters of the FC block in the learning process. As a result, the domain knowledge can guide the TBP learning process to get started from a point in the parameter space that is not far from the optimal one, and the TBP can fine-tune the domain knowledge for achieving a better performance. This cooperative relation between the domain knowledge and the TBP learning process is not always present in other types of controllers.

On the other hand, if we do not have *a priori* knowledge about controlling the plant, then the number of fuzzy if-then rules has to be decided more or less by trial and error. Fortunately, due to ANFIS's remarkable representational power [27, 30], usually we do not need many rules to construct the desired mapping from state variables to control action.

As for the implementation of the plant block, we can choose whatever function approximators that can best represent the input-output behavior of the plant. This model-insensitive attribute is mostly due to the flexibility of adaptive networks, which allows us to choose either conventional models (difference or differential equations, transfer functions, etc) or unconventional ones (ANFIS, neural networks [73], radial basis function networks [58], GMDH structure [26], etc.) to implement the plant block.

If the plant can be modeled as a set of n (= number of state variables) first-order difference equations, then the plant block can be replaced with n nodes, each of which uses one difference equation to obtain the state variable at the next time step. Furthermore, if the state equations of the plant are a set of first-order differential equations:

$$\dot{\vec{x}}(t) = \vec{f}(\vec{x}(t),\ \vec{in}(t),\ t), \tag{5.1}$$

where $\vec{x}(t)$ is a vector consisting of state variables at time t and $\vec{in}(t)$ is the input vector to the plant, then we can just employ a linear approximation to get the difference equations as below

$$\vec{x}(h*k+h) = h*\vec{f}(\vec{x}(h*k),\ \vec{in}(h*k),\ h*k) + \vec{x}(h*k), \tag{5.2}$$

where k is an integer and h is the sampling time. Therefore the plant block still has n nodes, each of which performs a component function of equation (5.2).

When the sampling time h is too big or the plant has fast dynamics, the linear approximation may not be a reasonable estimate of the next state. In this case, we can utilize a large body of numerical analysis techniques to obtain a more precise estimate, for instance, the second-order Runge-Kutta method:

$$\begin{cases} \vec{a} = h*\vec{f}(\vec{x}(h*k),\ \vec{in}(h*k),\ h*k), \\ \vec{b} = h*\vec{f}(\vec{x}(h*k)+\vec{a},\ \vec{in}(h*k),\ h*k+h), \\ \vec{x}(h*k+h) = \vec{x}(h*k) + 0.5*(\vec{a}+\vec{b}). \end{cases} \tag{5.3}$$

However, to implement the above equations as an adaptive network (without modifiable parameters) is more complex and some intermediate nodes would be present in the resulting network for the intermediate variable vector $\vec{a}$ and $\vec{b}$. Higher order Runge-Kutta formulas may be used to implement the plant block as well, but the increased complexity of the adaptive network could slow down the learning process even more.

Consequently, the block diagram of Figure 5.1 can also be viewed as an adaptive network containing two sub-networks, the FC block (ANFIS) and the plant block. Subsequently, we refer to the adaptive network of Figure 5.1 as SAN_k, representing the *stage adaptive network* at time stage k.

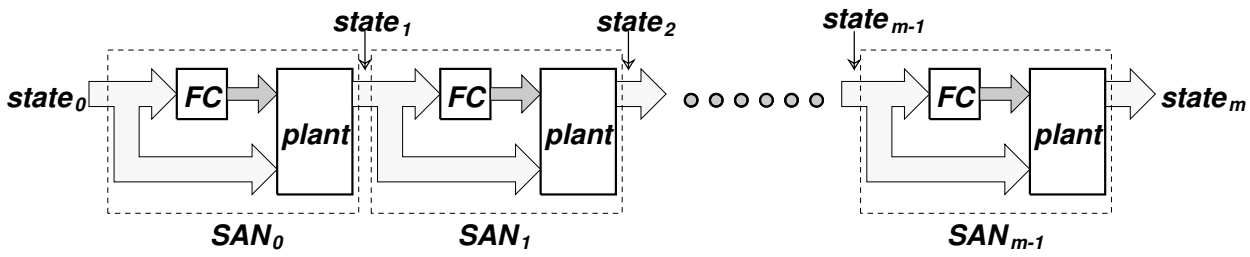


Figure 5.2: *State transition diagram.*

5.2.2 Trajectory Adaptive Network

Given the state of the plant at time $t = k * h$, the FC will generate an input to the plant and the plant will evolve to the next state at time $(k + 1) * h$. By repeating this process starting from $t = 0$, we obtain a plant state trajectory determined by the initial state and the parameters of the FC. The state transition from $t = 0$ to $t = m * h$ is shown conceptually in Figure 5.2 which is, again, an adaptive network consisting of m SAN_k, $k = 0$ to $m - 1$. Accordingly we can still apply the back propagation gradient descent to minimize the differences between adaptive network outputs and desired outputs. In order to make the inputs and outputs more explicit, we redraw Figure 5.2 to get the *trajectory adaptive network* shown in Figure 5.3, where the inputs to the network are the initial state of the plant at time = 0; the outputs of the network are the state trajectory from $t = h$ to $m * h$; and the adjustable parameters are all pertaining to the FC block implemented as an ANFIS. Hence each entry of the training data is of the form

$$(initial\ state;\ desired\ trajectory), \tag{5.4}$$

and the corresponding error measure to be minimized is

$$E = \sum_{k=1}^{m} \|\vec{x}(h * k) - \vec{x}_d(h * k)\|^2 \tag{5.5}$$

where $\vec{x}_d(h*k)$ is the desired trajectory at $t = h*k$. With some minor modifications of Figure 5.3, the above error measure can be revised as

$$E = \sum_{k=1}^{m} \|\vec{x}(h*k) - \vec{x}_d(h*k)\|^2 + \lambda * \sum_{k=0}^{m-1} \|\vec{in}(h*k)\|^2 \tag{5.6}$$

where $\vec{in}(h*k)$ is the controller's output at time $h*k$. By a proper selection of λ, a compromise between trajectory error and control effort can be obtained.

Since the error signals of back propagation can propagate through different time stages, this control methodology is called *temporal back propagation*, or simply *TBP*. As a matter of fact, the basic idea of TBP is similar to Nguyen and Widrow's approach [62] to constructing a self-learning neural controller, which Werbos has called back propagation through time [25]. We generalize the idea to a much more flexible building block, the adaptive network, which has two advantages over neural networks:

1. Accommodation of *a priori* knowledge from a human operator, in the form of fuzzy if-then rules.

2. No need to remodel the plant in neural networks if we already have other existing models for it, such as difference equations.

In the trajectory adaptive network shown in Figure 5.3, though there are m FC blocks, all of them refer to the same fuzzy controller at different time stages. Namely, there is only one parameter set which belongs to all m FC blocks at different time stages. For clarity, this parameter set is shown explicitly in Figure 5.3 and it is updated according to the output of the error measure block.

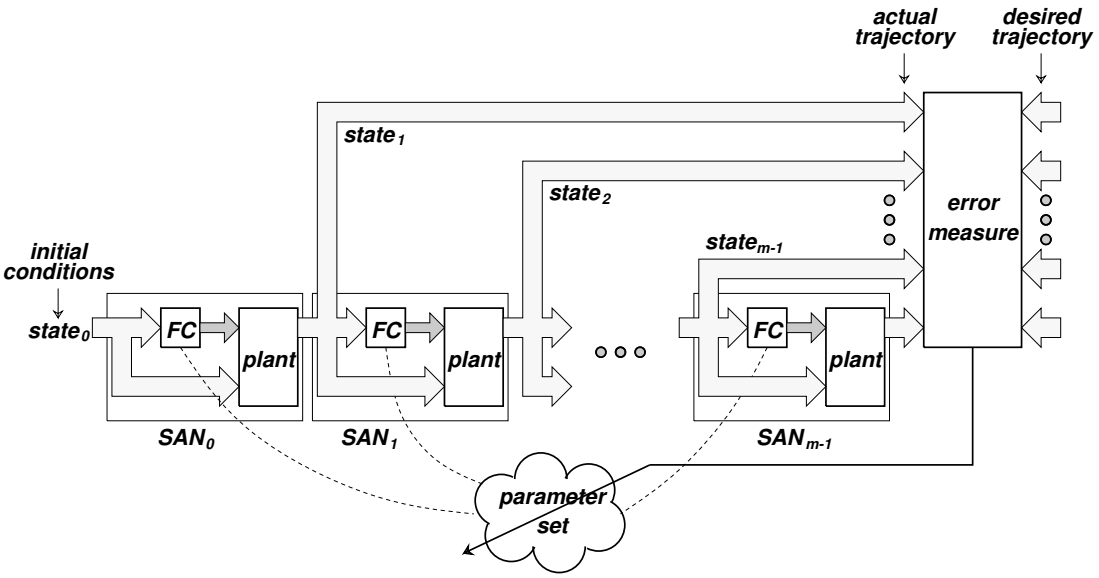


Figure 5.3: *A trajectory adaptive network for control application.*

5.3 Application to the Inverted Pendulum System

The proposed control scheme is quite general and it can be applied to a variety of control problems. In this section, we demonstrate the effectiveness of the TBP by applying it to a benchmark problem in intelligent control — the inverted pendulum system.

The Inverted Pendulum System

The inverted pendulum system (Figure 5.4) is composed of a rigid pole and a cart on which the pole is hinged. The cart moves on the rail tracks to its right or left, depending on the force exerted on the cart. The pole is hinged to the cart through a frictionless free joint such that it has only one degree of freedom. The control goal is to balance the pole starting from nonzero conditions by supplying appropriate force to the cart.

The dynamics of the inverted pendulum system are characterized by four state variables: θ (angle of the pole with respect to the vertical axis), $\dot{\theta}$ (angular velocity of the

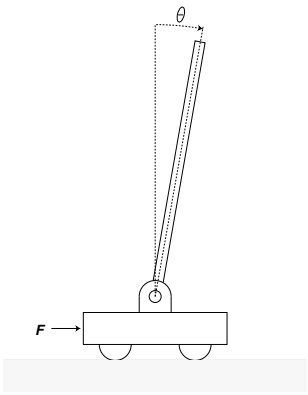


Figure 5.4: *The inverted pendulum system.*

pole), z (position of the cart on the track) and $\dot{z}$ (velocity of the cart). The behavior of these four state variables is governed by the following two second-order differential equations [7, 4]:

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

$$\ddot{z} = \frac{F + m * l * (\dot{\theta}^2 * sin\theta - \ddot{\theta} * cos\theta)}{m_c + m}, \tag{5.8}$$

where g (acceleration due to gravity) is 9.8 $meter/sec^2$, m_c (mass of cart) is 1.0 kg, m (mass of pole) is 0.1 kg, l (half length of pole) is 0.5 m, and F is the applied force in newtons. Our control goal here is to balance the pole without regard to the cart's position and velocity, hence only equation (5.7) is relevant in our simulation.

5.3.1 Simulation Settings

Figure 5.5 shows the stage adaptive network used in our simulation. Both the controller and the plant block, together with the learning rule, are explained below.

Plant Block

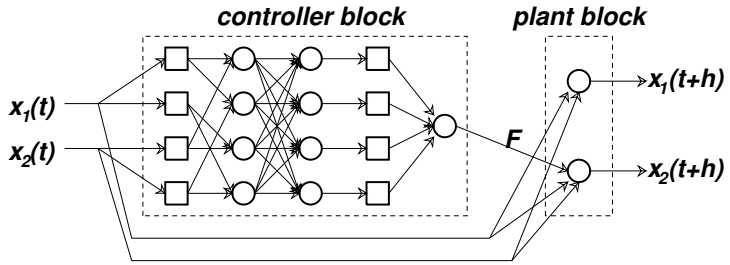


Figure 5.5: *The implementation of a stage adaptive network.*

As mentioned earlier, there are several ways to implement the plant block depending on how well we know the plant. In this case, the plant is a deterministic nonlinear dynamic system with precisely defined differential equations, so we can just use 2 nodes to calculate the state variables at the next time step by linear approximation

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

where $x_1(\cdot) = \theta(\cdot)$, $x_2(\cdot) = \dot{\theta}(\cdot)$. These two equations are the node functions of the plant block in Figure 5.5.

Controller Block

We assume that no domain knowledge (from a human operator's point of view) about the inverted pendulum system is available. The controller block in Figure 5.1 is implemented as an ANFIS with two inputs, each of which is assigned two membership functions, so it is a fuzzy controller with 4 fuzzy if-then rules of Takagi and Sugeno's type [90]. See the controller block in Figure 5.5. (Though the number of fuzzy rules can be more than four, the simulation indicates 4 rules are enough for balancing the pole.)

Without any domain knowledge, we have to set the initial parameters subjectively. The consequent parameters of the FC are all set at zero, which means the control action is

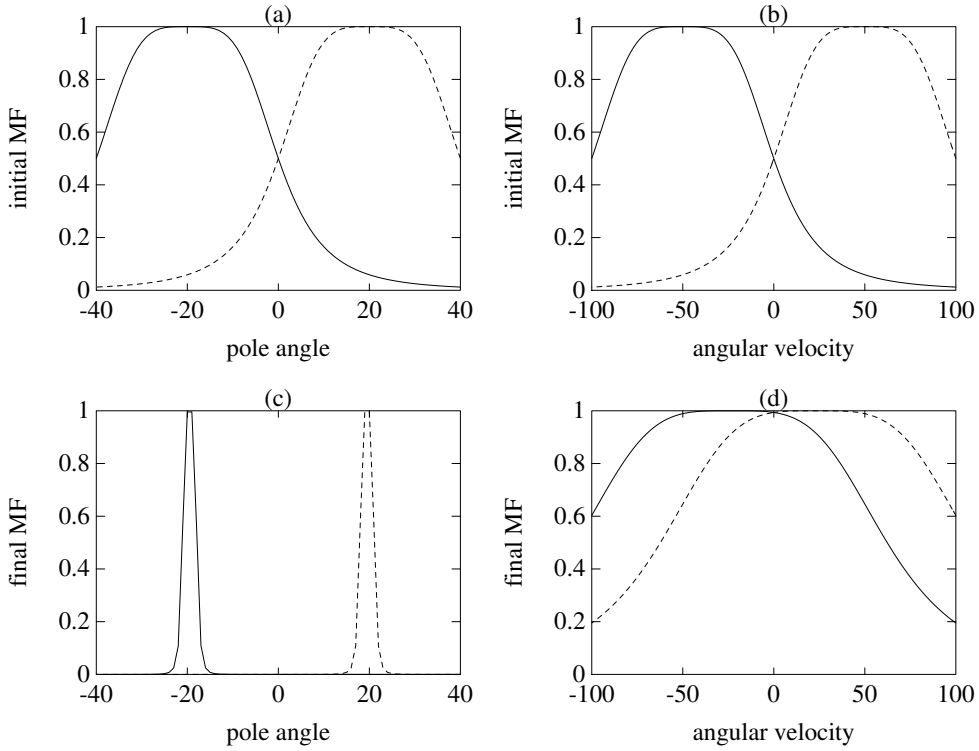


Figure 5.6: *(a)(b) Initial membership functions; (c)(d) final membership functions.*

zero initially as shown in Figure 5.7. As a conventional way of setting membership functions in a fuzzy controller, the premise parameters are set in such a way that the membership functions can cover the domain interval (or universe of discourse) completely with sufficient overlapping of each other. Figures 5.6(a) and (b) illustrate the initial membership functions in the form of equation (6.4); the domain interval for θ (degrees) and $\dot{\theta}$ (degrees/sec) are assumed to be [-20, 20] and [-50, 50], respectively.

Temporal Back Propagation

We employ 100 stage adaptive networks to construct the trajectory adaptive network, and each stage adaptive network corresponds to the time transition of 10 ms. That is, the time step (h) used is 10 ms, and the trajectory adaptive network corresponds to a

time interval from $t = 0$ to $t = 1$ *sec*. If h is too small, a large network has to be built to cover the same time span, which increases the signal propagation time and thus delays the whole learning process. On the other hand, if h is too big, then the linear approximation of the plant behavior may not be precise enough and a higher order approximation has to be used instead.

The training data set contains desired input-output pairs of the format

$$(\textit{initial condition};\ \textit{desired trajectory}), \tag{5.10}$$

where the initial condition is a two-element vector which specifies the initial condition of the pole; the desired trajectory is a 100-element vector which contains the desired pole angle at each time step. In our simulation, only 2 entries of training data are used: the initial conditions are $(10, 0)$ and $(-10, 0)$, respectively, and the desired trajectory is always a zero vector. In short, we expect that the trajectory adaptive network can not only learn to balance the pole from an initial pole angle of $+10$ or -10 degrees, but also achieve the control goal in an near-optimal manner which minimizes the error measure

$$E = \sum_{k=1}^{100} \theta^2(0.01 * k) + \lambda * \sum_{k=0}^{99} f^2(0.01 * k), \tag{5.11}$$

where $f(0.01 * k)$ is the controller's output force and $\lambda\ (= 10)$ accounts for the relative unit cost of control effort.

To speed up the convergence, we follow a strict gradient descent in the sense that each transition of the parameters will lead to a smaller error measure. If the error measure increases after parameter update, we back up to the original point in the parameter space and decrease the current step size by half. This process is repeated until the weight update

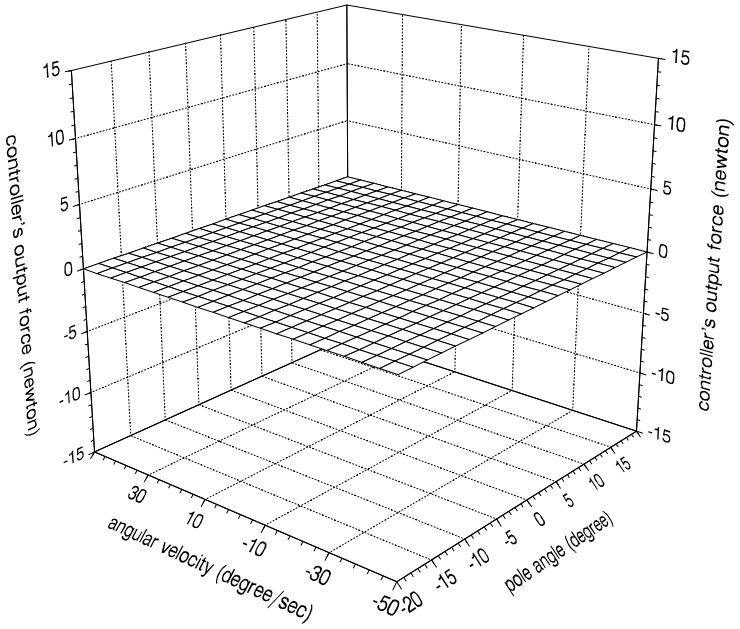


Figure 5.7: *Initial control action surface*.

leads to a smaller error measure. However, this step size update rule tends to use a small step size if the error measure surface encountered in the first few updates is not smooth. Therefore we multiply the step size by 4 after observing 3 consecutive transitions without any back-up actions. The initial step size in the simulation is 20 and the learning process stops whenever the number of transitions in parameter space (which is equal to the number of reductions in error measure) reaches 10.

5.3.2 Simulation Results

All the simulation settings mentioned above are referred to as the *reference setting*; other simulations are based on this setting with minor changes. In the learning task with

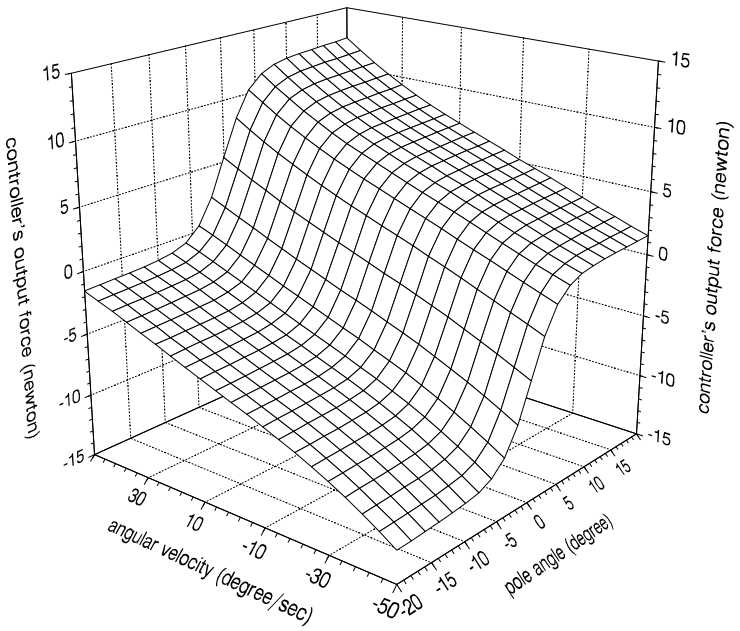


Figure 5.8: *Final control action surface.*

the reference setting, it is amazing to observe that the FC is able to balance the pole right after the first parameter transition, and it keeps on refining the controller (minimizing the error measure) till the 10th parameter transition is done. Figures 5.6 (a) and (b) show the initial membership functions on pole angle and angular velocity; (c) and (d) show the final membership functions. Each membership function is characterized by 3 parameters as described in equation (6.4). If θ is in $degree$, $\dot{\theta}$ in $degree/sec$, the initial fuzzy if-then rules are

$$\begin{cases} If\ \theta\ is\ A_1\ and\ \dot{\theta}\ is\ B_1,\ then\ force = 0 \\ If\ \theta\ is\ A_1\ and\ \dot{\theta}\ is\ B_2,\ then\ force = 0 \\ If\ \theta\ is\ A_2\ and\ \dot{\theta}\ is\ B_1,\ then\ force = 0 \\ If\ \theta\ is\ A_2\ and\ \dot{\theta}\ is\ B_2,\ then\ force = 0 \end{cases} \tag{5.12}$$

where A_1, A_2, B_1 and B_2 are the linguistic labels characterized by (20, 2, -20), (20, 2, 20), (50, 2, -50) and (50, 2, 50), respectively. Figure 5.7 is the initial control action surface.

The final fuzzy if-then rules derived from the reference settings are

$$\begin{cases} If\ \theta\ is\ A_1\ and\ \dot{\theta}\ is\ B_1,\ then\ force = 0.0502 * \theta + 0.1646 * \dot{\theta} - 10.09 \\ If\ \theta\ is\ A_1\ and\ \dot{\theta}\ is\ B_2,\ then\ force = 0.0083 * \theta + 0.0119 * \dot{\theta} - 1.09 \\ If\ \theta\ is\ A_2\ and\ \dot{\theta}\ is\ B_1,\ then\ force = 0.0083 * \theta + 0.0119 * \dot{\theta} + 1.09 \\ If\ \theta\ is\ A_2\ and\ \dot{\theta}\ is\ B_2,\ then\ force = 0.0502 * \theta + 0.1646 * \dot{\theta} + 10.09 \end{cases} \tag{5.13}$$

where A_1, A_2, B_1 and B_2 are the linguistic labels characterized by (-1.59, 2.34, -19.49), (-1.59, 2.34, 19.49), (85.51, 1.94, -23.21) and (85.51, 1.94, 23.21), respectively. Figure 5.8 is the final control action surface.

Figure 5.6 indicates that the final membership functions for θ are quite different from the initial membership functions. Visually there are no membership functions covering

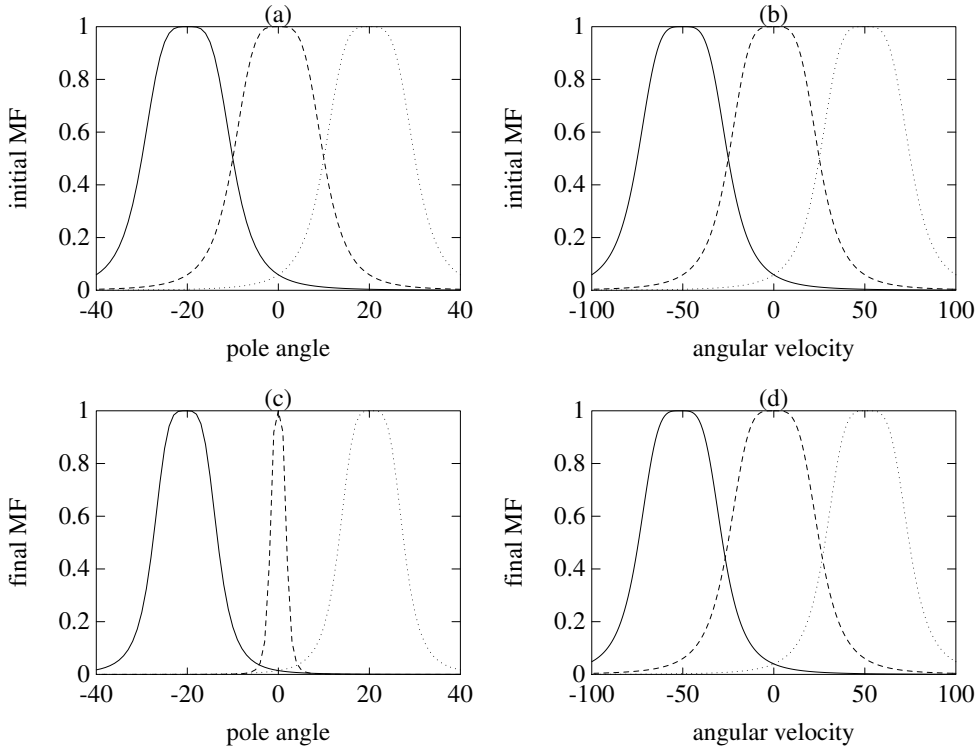


Figure 5.9: *(a)(b) Initial membership functions and (c)(d) final membership functions of a 9-rule fuzzy controller.*

the interval [-10, 10] of θ, making the linguistic interpretation of the fuzzy rules difficult. However, since we are utilizing the FC as a functional approximator that can generate the required nonlinear mapping, linguistically desirable features (such as enough overlapping between membership functions and total coverage of the domain interval) do not have to be one of the FC's attributes in this case. If we want to keep those desirable features, we can either impose some constraints on the premise parameters, or simply increase the number of parameters of the fuzzy controller to endow it with more degree of freedom. Figure 5.9 shows the membership functions of a 9-rule fuzzy controller which has about the same performance as the 4-rule fuzzy controller. Due to its higher degrees of freedom, the premise parameters of the 9-rule fuzzy controller do not have to change a lot to minimize the

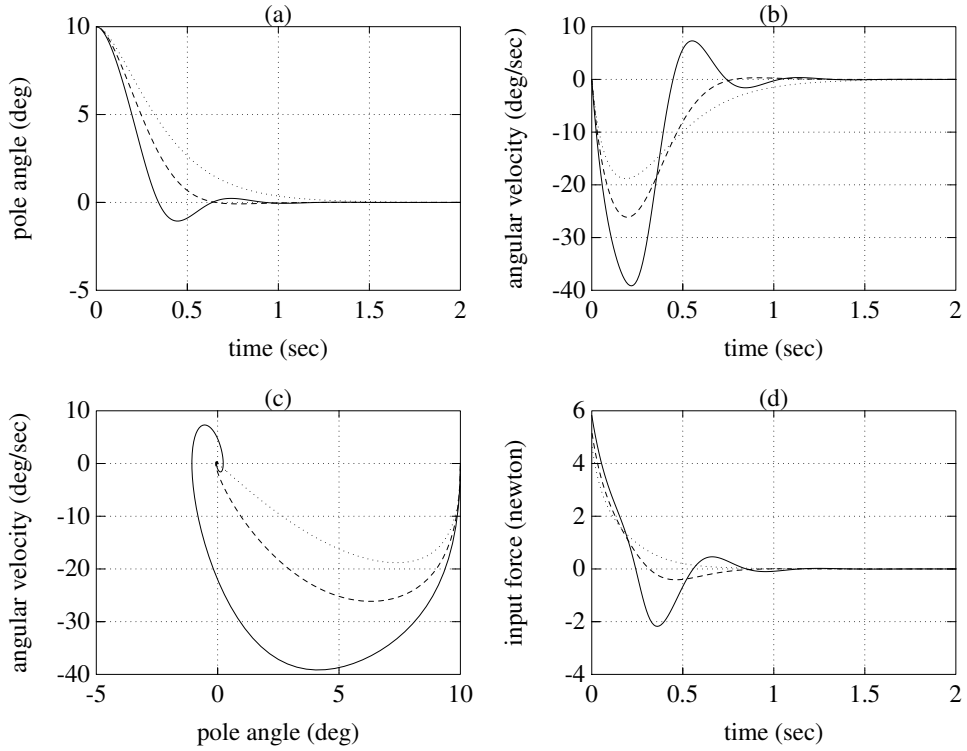


Figure 5.10: *(a) Pole angle; (b) pole angular velocity; (c) state space; (d) input force. (Solid, dashed and dotted curves correspond to λ = 10, 40 and 100, respectively.)*

error measure; therefore the final membership functions can cover all the domain intervals with desired overlapping.

Solid curves in Figure 5.10 demonstrate the state variable trajectories under the reference setting: (a), (b) and (d) show the pole angle (degree), angular velocity (degree/sec) and control actions (newtons) from $t = 0$ to $t = 2$ *sec*; (c) is the state space plot which reveals how the trajectory approaches the origin from the initial point $(10, 0)$. Dashed and dotted curves in Figure 5.10 correspond to λ equal to 40 and 100, respectively. From (a), it is observed that a smaller λ (solid curve) achieves the control goal faster since the controller can apply a larger force to balance the pole. For a large λ (dotted curve), the controller's output has to be kept small, thus slowing down the approach to the goal.

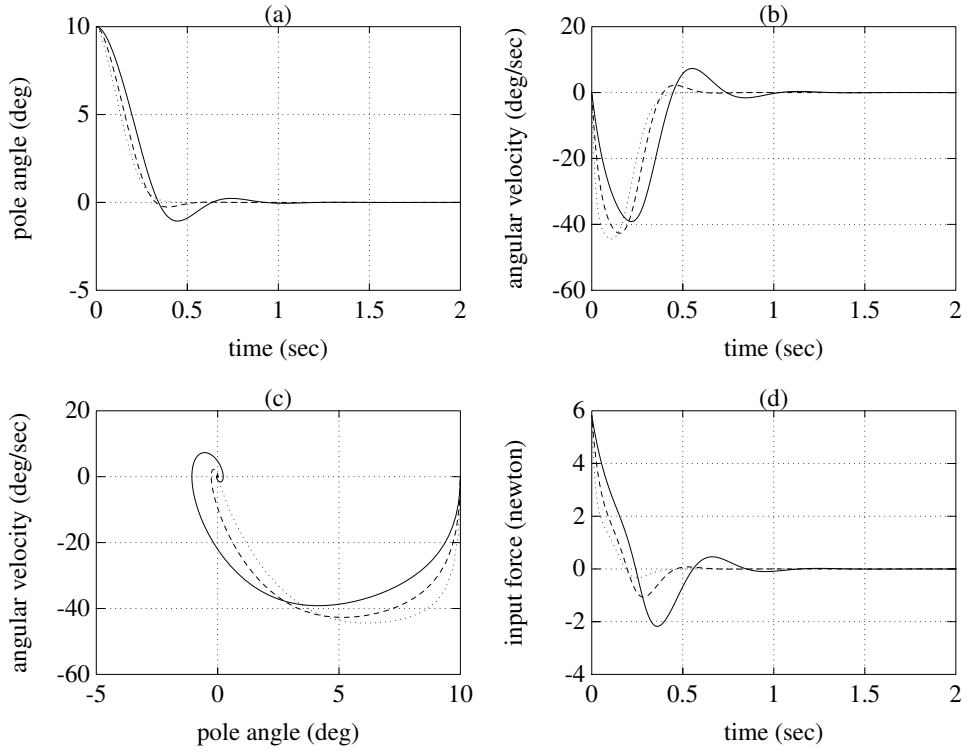


Figure 5.11: *(a) Pole angle; (b) pole angular velocity; (c) state space; (d) input force; (Solid, dashed and dotted curves correspond to half pole lengths of 0.5, 0.25 and 0.125 m, respectively.)*

To demonstrate how the fuzzy controller can survive substantial changes of plant parameters, we use poles of different lengths to test the controller obtained from the reference setting. The results are shown in Figure 5.11, where solid, dashed and dotted curves correspond to half-lengths of the pole equal to 0.5 (reference setting), 0.25 and 0.125 m respectively. It is remarkable to note how the controller can handle the shorter pole easily and gracefully.

In the learning phase, we supply only two training data corresponding to initial conditions (10, 0) and (-10, 0) of the pole. Now it would be interesting to know how the FC (obtained from the reference setting) deals with other initial conditions. In this part,

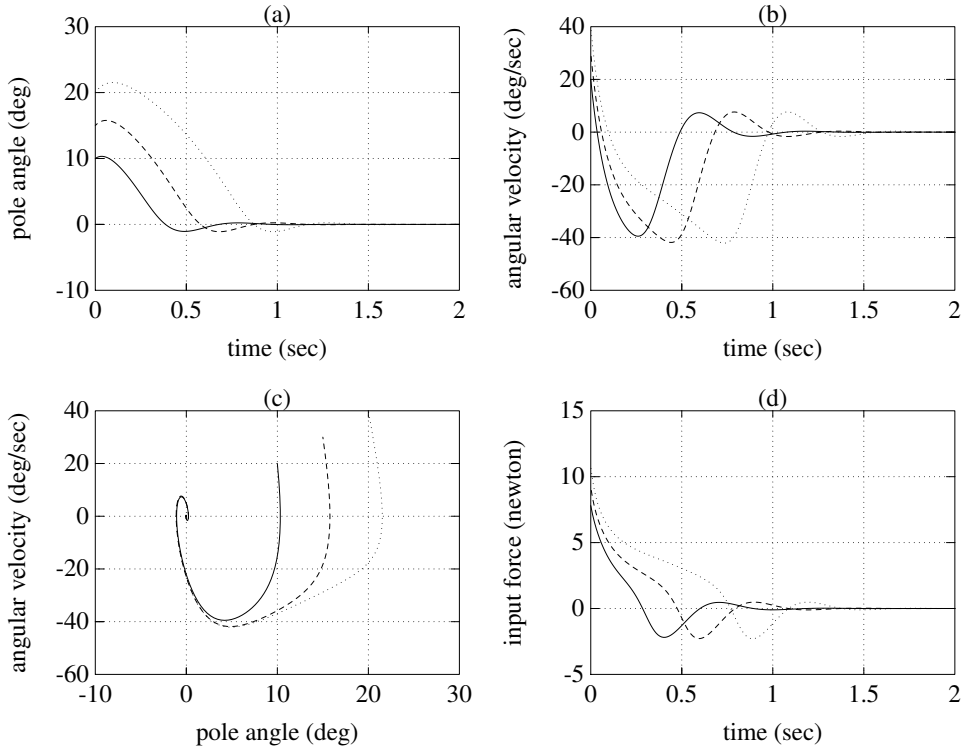


Figure 5.12: *(a) Pole angle; (b) pole angular velocity; (c) state space; (d) input force. (Solid, dashed and dotted curves correspond to initial conditions (10, 20), (15, 30) and (20, 40), respectively.)*

we monitor the pole behavior starting from other initial conditions which make the control goal even harder. Figure 5.12 shows the results, where the curve solid, dashed and dotted curves correspond to the initial conditions (10, 20), (15, 30) and (20, 40), respectively. Again, the same fuzzy controller can perform the control task starting from the unseen initial conditions. Figure 5.12 together with Figure 5.11 reveals the robustness and fault tolerance of the fuzzy controller obtained from the TBP.

Chapter 6

Neuro-Fuzzy Classifiers

6.1 Introduction

In the previous two chapters, we introduced the ANFIS architecture and its applications to nonlinear function modeling, time series prediction and self-learning fuzzy controllers. In this chapter, we will propose another neuro-fuzzy architecture which is devoted to the applications of pattern classification. The proposed architecture, called the *neuro-fuzzy classifier* (NFC), is a class of adaptive networks which can update parameters through the hybrid learning rule introduced in chapter 3. NFC is better than neural network classifiers in the sense that prior knowledge about the training data set can be encoded into the NFC's parameters. This encoded knowledge, usually acquired from human experts or data visualization techniques, can almost always allow the learning process to begin from a good initial point not far away from the optimal one in the parameter space, thus speeding up the convergence to the optimal or a near-optimal point. Moreover, the parameters obtained after the learning process can be easily transformed into structure knowledge in

the form of fuzzy if-then rules. The basic architecture of the NFC is covered in the next section.

Apparently the problem of (supervised) pattern classification is quite similar to the problem of neural network or fuzzy logic modeling: the parameters are identified through a learning process which makes use of a data set composed of desired input-output pairs, and the input-output mapping of the resulting architecture should be able to make reasonable generalization and interpolation for unseen input data. However, classification is different from continuous modeling problems in that its output is discrete. In other words, if we want to divide a data set into two classes, then the output of a classifier should be either class 1 or class 2, and nothing in between (unless we adopt the fuzzy classification paradigm). In this perspective, the original definition of least squared error measure is no longer appropriate since it cannot reflect the number of misclassifications. Therefore in section 6.3, we devise another error measure, called *maximum-type error measure*, which is a continuous function that can appropriately reflect the number of misclassifications.

To demonstrate the strengths of the proposed NFC architecture, we perform two simulation experiments in section 6.3. The first problem we attempted is the two-spiral problem, a benchmark problem in neural network literature which is often used to verify the feasibility of new learning algorithms or new network structures for neural network classifiers. In the second experiment, we try the NFC architecture on the IRIS data classification which deals with the classification of 150 iris flowers into 3 classes based on 4 features. These simulations show both the advantages and the potential application domains of the proposed NFC architecture.

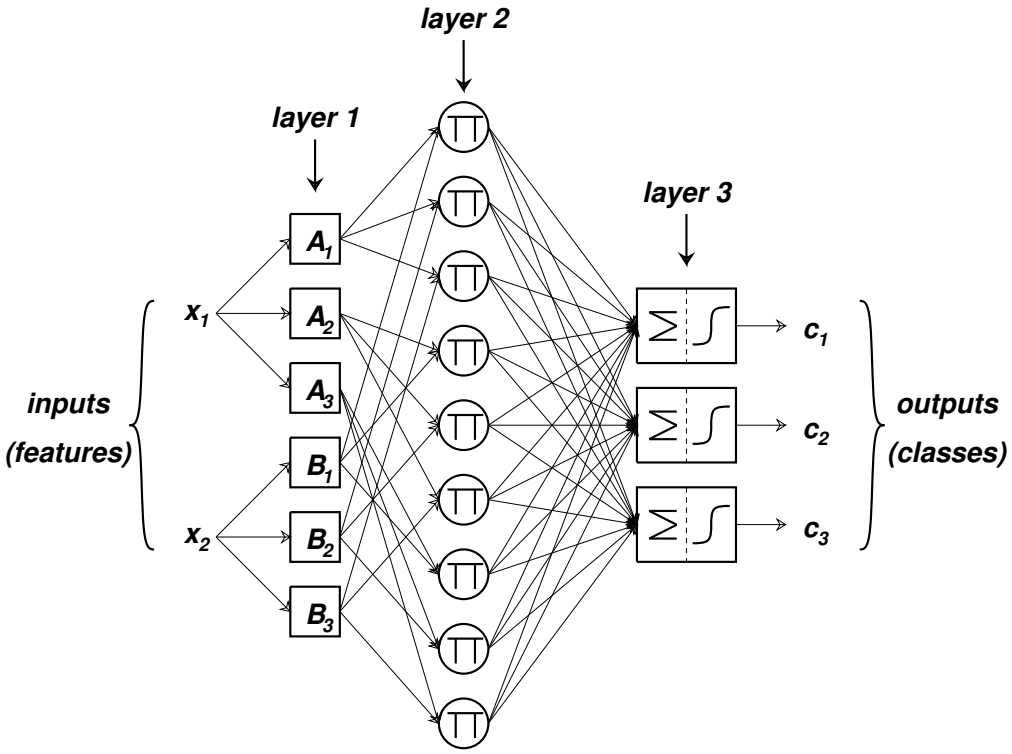


Figure 6.1: *NFC architecture*.

6.2 NFC Architecture

The format of fuzzy if-then rules used in a classifier can be expressed as

$$\text{If } X_1 \text{ is } A_1 \text{ and } X_2 \text{ is } A_2, \text{ then } Z \text{ is } C. \tag{6.1}$$

where X_1 and X_2 are *input variables* or *observed features*; A_1 and A_2 are linguistic labels characterized by appropriate membership functions; Z is the object with X_1 and X_2 as its observed features; and C is the name of a class or category. Usually we do not distinguish between the object Z and its feature vector (X_1, X_2); therefore in the following text, $\vec{X} = (X_1, X_2)$ is used loosely to represent both the feature vector and the object Z. Some examples of this type of fuzzy if-then rule are:

If $Time$ is $around\ eight$ and $Traffic$ is $good$, then Day is $weekend$.

If $Voice$ is $high$ and $Hair$ is $long$, then $Person$ is $female$.

Similar rules can be found repeatedly in the reasoning activities of our daily life. Note that the classes in the above two examples are inherently nonfuzzy since a day should be either weekday or weekend and a person should be either male or female. However, some other applications may favor *fuzzy classification* and thus C in equation (6.1) may assume a fuzzy term; some such examples are:

If $Temperature$ is $high$ and Sky is $bright$, then $Time$ is $daytime$.

If $Pulse$ is $weak$ and $Face$ is $pale$, then $Patient's\ Condition$ is $critical$.

Based on adaptive networks, we propose a neuro-fuzzy classifier (NFC) architecture [85, 86] that can incorporate the above form of fuzzy if-then rules into its structure. For simplicity, we assume the NFC architecture under consideration has two inputs (observed features) x_1 and x_2 and three outputs c_1, c_2 and c_3, corresponding to three different classes or categories. As mentioned above, the consequent part of the fuzzy if-then rule in equation (6.1) may have a crisp or fuzzy term C. For crisp-output NFC, we adopt the convention that c_i is 1 if and only if the presented feature vector $\vec{x} = (x_1, x_2)$ belongs to class i; otherwise c_i is 0. For fuzzy-output NFC, c_i denotes the degree to which the presented feature vector belongs to class i and its summation over i is equal to 1:

$$\sum_i c_i = 1. \tag{6.2}$$

Figure 6.1 shows the NFC architecture where each of the two inputs is assigned three membership functions. Node function in each layer of Figure 6.1 is explained below:

Layer 1 Every node i in this layer is a square node with a node function

$$O_i^1 = \mu_{A_i}(x), \tag{6.3}$$

where x is the input to node i, and A_i is the linguistic label (*small* , *large*, etc.) associated with this node function. This layer is exactly the same as the first layer in the ANFIS structure of Figure 4.1 or Figure 4.3. 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

$$\mu_{A_i}(x) = \frac{1}{1 + [(\frac{x - c_i}{a_i})^2]^{b_i}}, \tag{6.4}$$

or

$$\mu_{A_i}(x) = exp\{-[(\frac{x - c_i}{a_i})^2]^{b_i}\}, \tag{6.5}$$

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. In fact, any continuous and piecewise differentiable functions, such as commonly used trapezoidal or triangular-shaped membership functions, are also qualified candidates for node functions in this layer. Parameters in this layer are referred to as *premise parameters*.

Layer 2 Every node in this layer is a circle node labeled Π which multiplies the incoming

signals and sends the product out. For instance, the output of node 4 in this layer is

$$w_4 = \mu_{A_2}(x)\mu_{B_1}(y), \tag{6.6}$$

which specifies the degree of match on the premise part of the three fuzzy if-then rules with A_2 and B_1 on the premise part. From another viewpoint, it represents the degree to which the presented feature vector belongs to the fuzzy region formed jointly by the linguistic label A_2 and B_1.

Layer 3 If the data is to be classified into C categories, there are C nodes in this layer and the output of each node indicates to what degree the presented pattern belongs to a corresponding category. Node function in this layer is the cascade of two functions: a weighted sum and a squashing function. The weighted sum function first multiplies each output (w_i) from the previous layer with a parameter (p_{ij}) and then sums them up. The squashing function, usually a sigmoidal function, forces the output of this layer to be within $[0, 1]$. For instance, the output of node j is

$$O_i^3 = sig(\sum_{i=1}^{9} w_i p_{ij}), \tag{6.7}$$

where $sig(\cdot)$ denotes the sigmoidal function. For convenience, parameters in this layer are called *consequent parameters*.

Thus we have constructed the NFC architecture, a three-layer adaptive network which can perform fuzzy if-then rules for pattern classification. For crisp-output NFC, we usually place a maximum selector following layer 3 to single out the class corresponding to the maximum of c_i's. Alternatively, for fuzzy-output NFC, we often add another layer which calculates the normalized version of the outputs in Figure 6.1; the function of this

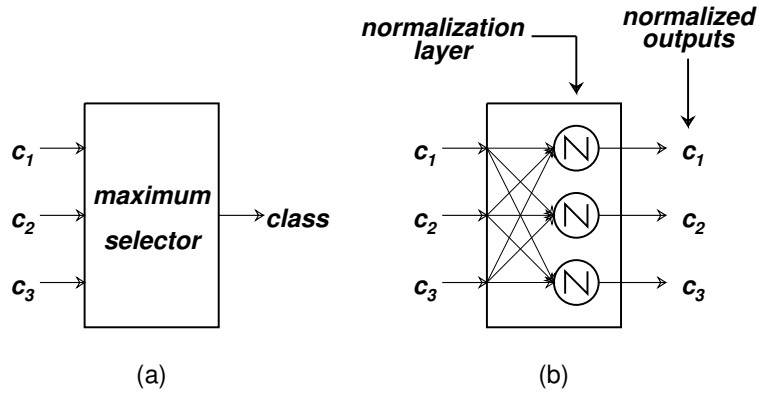


Figure 6.2: *Post-processor for NFC with (a) crisp outputs; (b) fuzzy outputs.*

normalization layer is exactly the same as that of layer 3 in the ANFIS of Figure 4.1. Both the maximum selector and the normalization layer can be viewed as post-processors for different types of outputs and they are depicted in Figure 6.2. In the following discussion, we will confine our discussion to crisp-output NFC, though many of the arguments can also be applied to fuzzy-output NFC.

Note that since the weighted sum mentioned above does not involve an offset term, the node function in layer 3 is not the same as that of a back-propagation neural net. Moreover, the parameter p_{ij} serves as an indicator to show how much the fuzzy region i belongs to class j. In other words, a positive p_{ij} strengthens the statement that region i belongs to class j while a negative p_{ij} strengthens the opposite statement that region i does not belong to class j. Therefore, either the parameter set a_i, b_i, c_i in layer 1 or the parameter set p_{ij} in layer 3 has specific physical meanings in the fuzzy if-then rules. This meaningful representation of NFC is a primary advantage over the conventional neural network classifiers.

Next we will describe how to apply the hybrid learning algorithms developed in Chapter 3 to identify the parameters in the NFC architecture mentioned above. Besides,

we will consider other error measures employed in the literature and their relationships with the hybrid learning rule.

6.3 Learning Rule and Error Measure

From the description of the NFC architecture in the previous section, it is obvious that if we take the final outputs as the outputs of the weighted sum functions instead of the squashing functions, then we have output expressions which are linear in the consequent parameters. In terms of the expression of equation (3.11), $H(\cdot)$ is the inverse sigmoidal function:

$$H(x) = sig^{-1}(x) = ln\frac{x}{1-x} \tag{6.8}$$

And the decomposition of the parameter set S into S_1 and S_2 (see equation (3.10)) is

S_1 = set of premise parameters (layer 1),

S_2 = set of consequent parameters (layer 3).

As a result, the hybrid learning algorithm proposed in Chapter 3 can be applied directly if the error measure is chosen as the squared errors of the transformed outputs.

Without loss of generality, we assume the class number is 3; the desired and calculated outputs of NFC are denoted as $\vec{do} = (do_1, do_2, do_3)$ and $\vec{co} = (co_1, co_2, co_3)$. For fuzzy-output NFC, each element of $\vec{do}$ is a real number between 0 and 1 and the outputs of NFC are supposed to match the desired outputs $\vec{do}$ as closely as possible. Therefore we can just employ the hybrid learning algorithm as usual: use the gradient descent to update the premise parameter in layer 1 and apply the Kalman filter algorithm for identifying the consequent parameters in layer 3.

However, when we are dealing with crisp-output classification, only one element of $\vec{do}$ is non-zero (usually 1); denoting the class which a corresponding feature vector belongs to. Hence a maximum selector (Figure 6.2 (a)) is often utilized to decide the class according to the maximum of NFC's outputs. In this case, the error measure defined as squared errors in equation (3.2) is no longer appropriate since the calculated outputs are not expected to match the desired outputs as closely as possible. For instance, suppose that the desired output vector $\vec{do}$ is $(1,0,0)$ and we have two calculated output vectors $\vec{co1} = (0.9,0,0)$ and $\vec{co2} = (0.9,0.8,0.8)$. If we employ the squared error as the error measure, then $\vec{co2}$ will have a much larger error than that of $\vec{co1}$. This is not quite right since after the maximum selector, both $\vec{co1}$ and $\vec{co2}$ will be correctly assigned as class 1. In other words, we need a new error measure that can more accurately reflect the number of misclassified cases. Furthermore, by choosing such an error measure, we relax the requirement that the calculated output should be as close as possible to the desired output, thus introducing more degrees of freedom into the NFC architecture for reaching the desired outputs.

Assuming the desired class is 1, then we are looking for an error measure E with the following properties:

if $o_1 > o_2$ and $o_1 > o_3$, then E is small,

if $o_1 > o_2$ and $o_1 < o_3$, then E is medium,

if $o_1 < o_2$ and $o_1 > o_3$, then E is medium,

if $o_1 < o_2$ and $o_1 < o_3$, then E is large,

where $\vec{co} = (co_1, co_2, co_3)$ is the calculated output vector. This kind of error measure can be implemented as

$$E = sgn(co_2 - co_1) + sgn(co_3 - co_1), \tag{6.9}$$

where $sgn(\cdot)$ is the *signum function* defined below:

$$sgn(x) = \begin{cases} 0 \; if \; x < 0, \\ 1 \; if \; x \geq 0. \end{cases} \tag{6.10}$$

However, in order to preserve differentiability, we choose the sigmoidal functions to replace the signum functions in equation (6.10):

$$E = sig[m(co_2 - co_1)] + sig[m(co_3 - co_1)] \tag{6.11}$$

where m is a positive large number that makes the sigmoidal function approximate the signum function. For different desired classes, we have similar expressions. Thus for a given desired output $\vec{do} = (do_1, do_2, do_3)$ and the corresponding calculated output $\vec{co} = (co_1, co_2, co_3)$, the desired error measure E can be formulated as:

$$\begin{aligned} E \quad = \quad & do_1(1 - do_2)(1 - do_3)\{(sig[m(co_2 - co_1)] + sig[m(co_3 - co_1)]\} \\ & + (1 - do_1)do_2(1 - do_3)\{(sig[m(co_1 - co_2)] + sig[m(co_3 - co_2)]\} \\ & + (1 - do_1)(1 - do_2)do_3\{(sig[m(co_1 - co_3)] + sig[m(co_2 - co_3)]\} \end{aligned} \tag{6.12}$$

where the second and the third terms account for the conditions when the desired classes are class 2 and 3, respectively. Since this error measure is reasonable when the maximum selector is used, it is therefore referred to as *maximum-type error measure*.

The maximum-type error measure introduced above can increase the degrees of freedom and therefore is suitable for crisp-output NFC. Note that by adopting this error

measure, we cannot apply the hybrid learning rules any more and the learning proceeds by the gradient descent alone. Meanwhile, the slow convergence of the gradient descent is compensated for by the proper choice of the maximum-type error measure, so the learning process will not suffer from the drawbacks of the gradient descent. The next section presents two application examples which employ NFC with maximum-type error measures to do crisp pattern classification.

6.4 Application Examples

In this section, we apply the proposed NFC architecture to two classification problems: the two-spiral problem and the IRIS data classification. Computer simulation results demonstrate how NFC can easily solve these two problems with the use of its adaptive capability and the prior knowledge about the training data.

6.4.1 Two-Spiral Problem

The two-spiral problem was proposed by Alexis P. Wieland on the connectionist mailing list as an interesting benchmark task for neural networks. The task requires a neural network classifier with two inputs and one output to learning a mapping that distinguishes between points of two intertwined spirals. The two sets of spiral data consist of 194 points, with 97 points for each spiral. One spiral is generated as a mirror image of the other, making the problem highly nonlinear-separable. The formulas used to generate the spirals are given below, and the two spirals are shown in Figure 6.3:

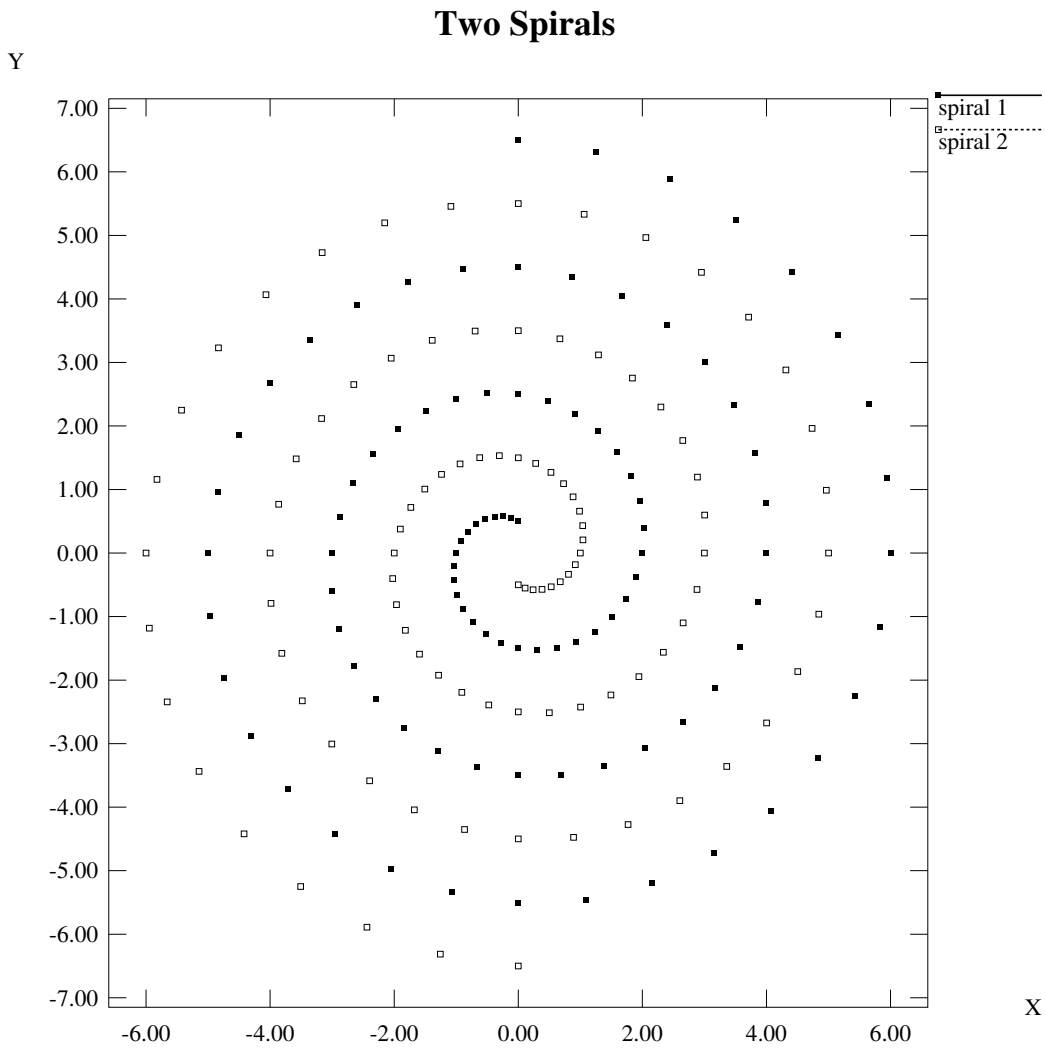


Figure 6.3: *Training data for the two-spiral problem.*

$$\begin{cases} \gamma = \frac{1}{\pi}(\theta + \frac{\pi}{2}) \\ \theta = \frac{k\pi}{16}, \ \ k = 0,\ 1,\ 2,\ ...,\ 96. \end{cases} \tag{6.13}$$

$$\text{spiral 1:} \begin{cases} x = \gamma cos(\theta) \\ y = \gamma sin(\theta) \end{cases} \tag{6.14}$$

$$\text{spiral 2:} \begin{cases} x = -\gamma cos(\theta) \\ y = -\gamma sin(\theta) \end{cases} \tag{6.15}$$

As pointed out by Wieland, this task has several features that makes it an interesting test for neural network's learning algorithms. First of all, it requires the neural networks to learn the highly nonlinear separation of the input space, which is difficult for most current learning algorithms. Secondly, its 2-dimensional input space makes it easy to plot the overall input-output relations as a 3-dimensional surface or 2-dimensional image for visual inspection and analysis.

Lang and Witbrock [43] attempted to learn the task with standard back-propagation nets with a couple of hidden layers, but they failed. In Cios and Liu's experiments [13], the number of hidden layers for the back-propagation nets varied from 2 to 4, and the number of nodes per layer varied from 10 to 100, but none of the trials showed signs of convergence even after 10,000 epochs. These unsuccessful trials show that this problem is non-trivial for standard feed-forward back-propagation nets, and therefore new learning rules as well as new architectures in connectionist models are invoked to tackle this problem. Lang and Witbrock [43] employ back-propagation nets with short-cut connections as pathways for providing information to all parts of the network, thus eliminating the problem of atten-

uated error signals when back-propagating through layers [70]. Cios and Liu [13] devise a continuous version of the ID3 [68] algorithm to solve this problem and then map the result back to a neural network structure. Fahlman and Lebiere [18] develop a new connectionist structure called cascade-correlation nets and use quick-propagation [17] as the learning rule to successfully separate the two spirals.

Although this problem has been solved by the approaches mentioned above, it remains difficult to figure out the network configurations and to choose a suitable learning algorithm. Moreover, it is hard, if not impossible, to extract information from a neural network after learning the mapping perfectly. In contrast, the proposed NFC architecture can solve this problem with meaningful representation both before and after learning, and the network structure, or primarily the rule number, can be estimated roughly through a glance at the training data distribution.

To proceed with the simulation, first the rule number has to be decided. Since the input partition of NFC is checkerboard-like (similar to the case of ANFIS, see the right sub-figure of Figure 4.3), we expect that the partition number (or equivalently, the number of membership functions) on either input x and y should be equal to the maximum number of alternations between classes along one dimension when the other is fixed. In the two-spiral problem, the maximum number of alternations on y is approximately 14, which occurs on the straight line $x = 0$; the maximum number of alternations on x is approximately 13, which occurs on $y = 0$. Our simulation consists of two groups which utilize the squared error measure and maximum-type error measure, respectively. Within each group, we perform four runs of simulation; the number of membership functions on both inputs is varied from

10 to 13 sequentially for runs in both group. That is, we have 8 runs in our simulation: the four runs in the first group employ the hybrid learning rule and they are stopped at 199.5 epochs, while those in the second group use the gradient descent method and they are stopped at 200 epochs. It is found that for both groups, 13 is the minimum number for NFC to classify the two spirals correctly. This agrees with our observation of the maximum number of alternations along each dimension.

Figure 6.4 shows the initial and final membership functions when the 2-spirals problem can be solved correctly, namely, when the number of membership functions on both inputs is 13. Figure 6.4 (a) and (b) are initial membership functions which are set uniformly to cover the whole universe of discourse; (c)(d) and (e)(f) are the final membership functions for run 4 (in group 1) and run 8 (in group 2), respectively. Note that the NFC with maximum-type error measure has more degrees of freedom, therefore the final membership functions (Figure 6.4 (e) and (f)) do not have to go through as many variations as those in group 1 (Figure 6.4 (c) and (d)) to reach the same goal.

Figure 6.5 and Figure 6.6 display the error measures and misclassification numbers as functions of epoch numbers for group 1 (run 1 through 4) and group 2 (run 5 through 8). It is observed that the initial misclassification numbers of the runs in group 1 are always smaller than those of the corresponding runs in group 2 due to the powerful Kalman filter algorithm employed in the hybrid learning rule. However, because the squared error is not an appropriate measure for misclassification numbers, the reduction in misclassification numbers in group 1 is not as significant as that of group 2 (except run 2 vs. run 6, which is an anomaly).

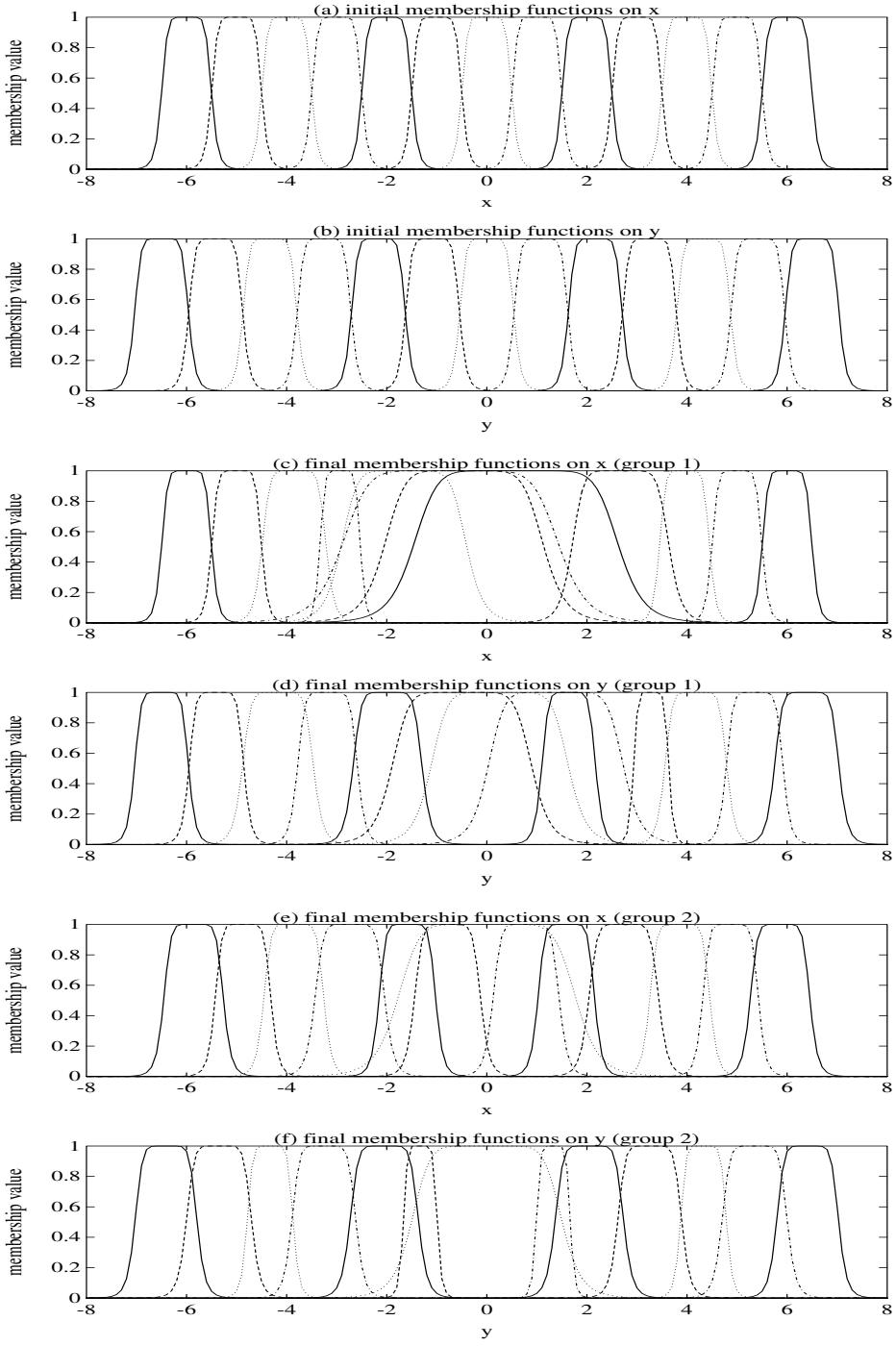


Figure 6.4: *Membership functions on input x and y for solving the two-spiral problems: (a)(b) initial membership functions for group 1 and 2; (c)(d) final membership functions for group 1; (e)(f) final membership functions for group 2.*

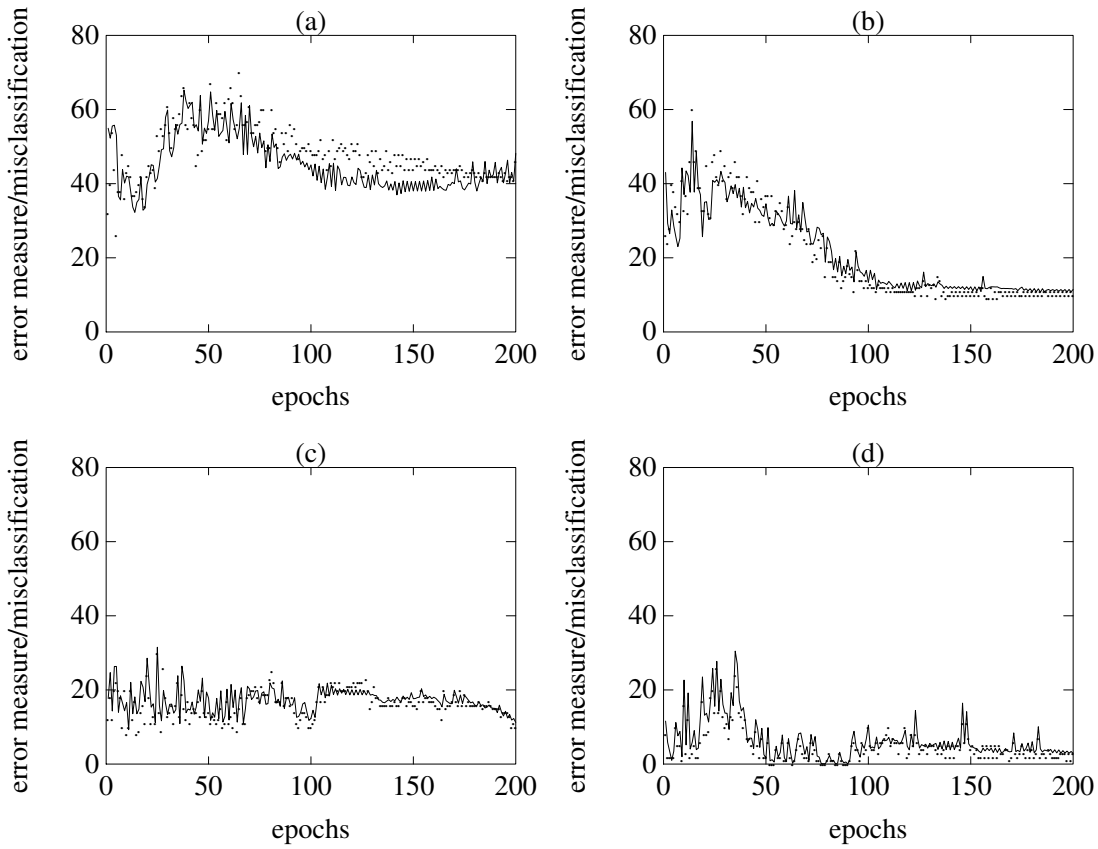


Figure 6.5: *(Group 1) Error measure and misclassification numbers w.r.t. epoch number: (a) run 1; (b) run 2; (c) run 3; (d) run 4. (Solid line: error measure, dot: misclassification number).*

As mentioned earlier, this problem is suitable for visual inspection or analysis on the classifier's input-output behavior through data visualization techniques such as 3-D surface or 2-D image. Figure 6.7 and Figure 6.8 depict the NFC's input-output behavior obtained from the two groups; each of the eight images is composed of 22,500 (150×150) pixels which are 8 bit deep. Since the NFC's outputs c_1 and c_2 are always between 0 and 1, in order to preserve the gray shades, they are converted to pixel values by the following

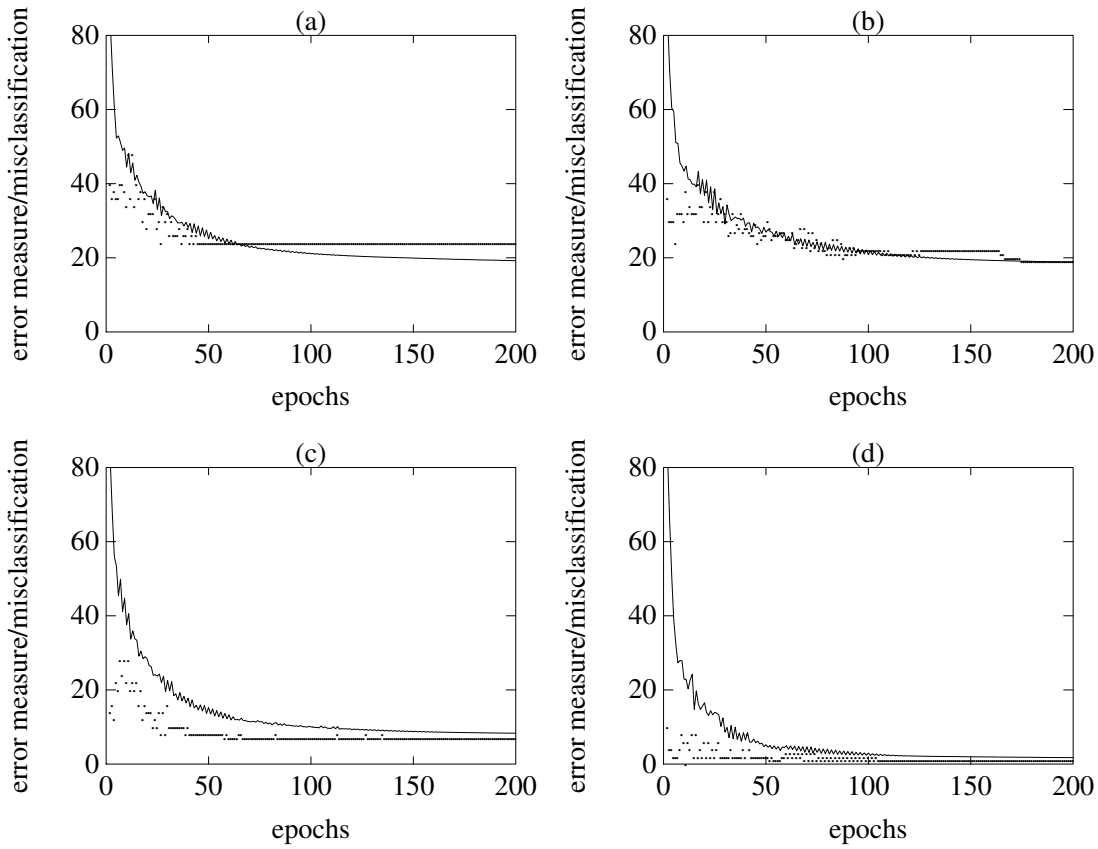
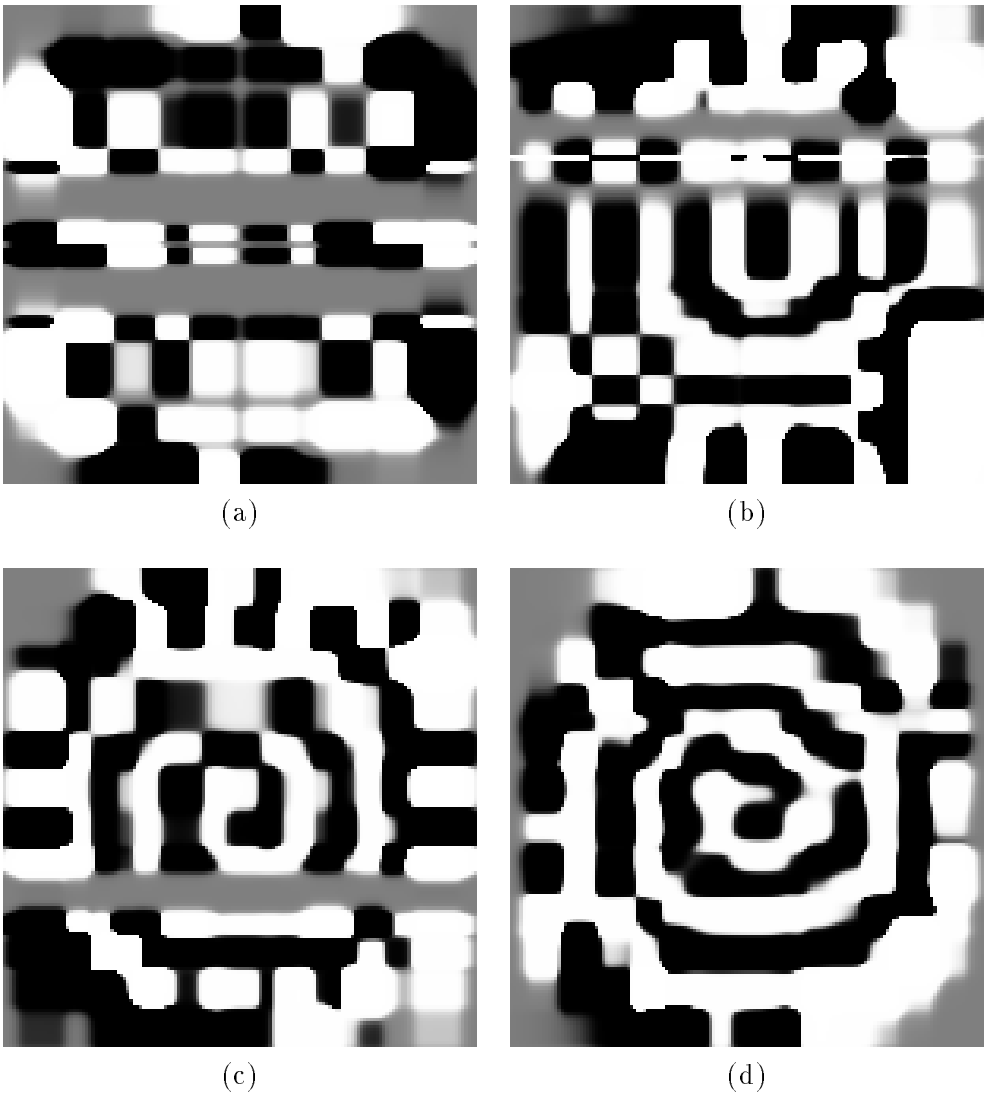


Figure 6.6: *(Group 2) Error measure and misclassification numbers w.r.t. epoch number: (a) run 5; (b) run 6; (c) run 7; (d) run 8. (Solid line: error measure, dot: misclassification number).*

equation (with round-off to the nearest integer):

$$image(x, y) = \frac{c_1(x, y) - c_2(x, y) + 1}{2} \times 255, \tag{6.16}$$

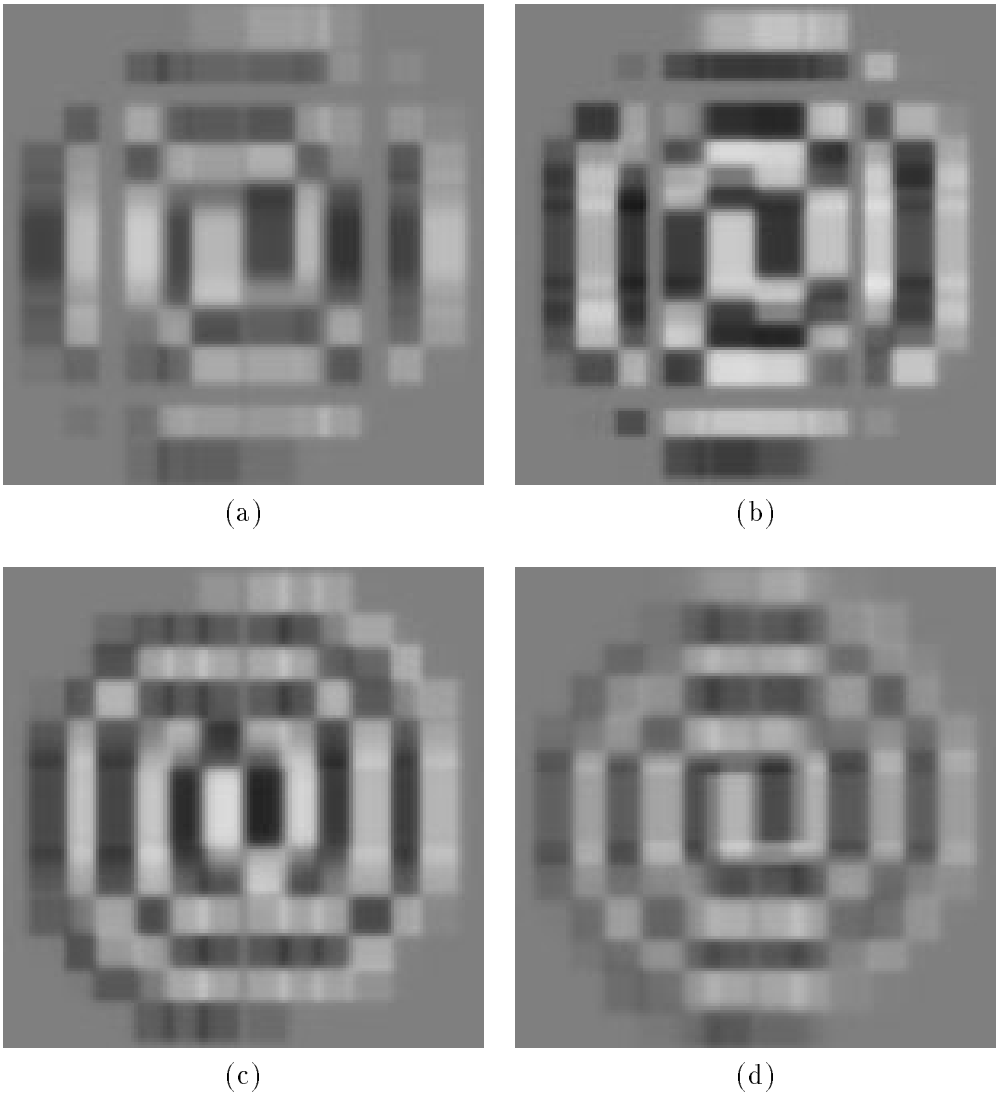
where $image(x, y)$ is the pixel value at position (x, y). It is clearly shown in Figure 6.7 (a), (b) and (c) that when the number of membership functions is not enough, there will be some "grey bands" in the images indicating that NFC cannot decide which class these regions belong to. This is caused by the fact that the universe of discourse (domain interval) is not covered properly. This situation does not appear in Figure 6.8 because the NFC



(a) (b)

(c) (d)

Figure 6.7: *Group 1, image representation of NFC's input-output behavior, with number of membership functions on x and y equal to (1) 10 (run 1); (b) 11 (run 2); (c) 12 (run 3) and (d) 13 (run 4).*



(a) (b)

(c) (d)

Figure 6.8: *Group 2, image representation of NFC's input-output behavior, with number of membership functions on x and y equal to (1) 10 (run 5); (b) 11 (run 6); (c) 12 (run 7) and (d) 13 (run 8).*

with maximum-type error measure has higher degrees of freedom. The images shown in Figure 6.7 are more distinct since the squared error measure causes the calculated outputs to be as close as possible to the desired outputs which are nonfuzzy. The images shown in Figure 6.8 are less distinct; however, if we pass them through a threshold filter shown below:

$$image_{enhanced}(x,y) = \begin{cases} 255, \text{ if } image(x,y) > 127, \\ 127, \text{ if } image(x,y) = 127, \\ 0, \text{ if } image(x,y) < 127, \end{cases} \tag{6.17}$$

then we get a more distinct version of Figure 6.8 as shown in Figure 6.9. This is equivalent to using a maximum selector, expressed as follows, to convert the NFC's output $c_1(x,y)$ and $c_2(x,y)$ into pixels with three values only:

$$image_{enhanced}(x,y) = \begin{cases} 255, \text{ if } c_1 > c_2, \\ 127, \text{ if } c_1 = c_2, \\ 0, \text{ if } c_1 < c_2. \end{cases} \tag{6.18}$$

Needless to say, Figure 6.9 appears to be more regular and symmetric than Figure 6.7, thus fitting the training data more closely and having better generalization capability.

Although we can always employ a large number of membership functions in NFC to achieve a perfect classification, this kind of over-parameterized structure is not recommended since it not only slows down the learning but also degrades the generalization power for unseen data sets; this is just like the case in over-parameterized neural networks. Therefore, the ability to determine the number of membership functions from visual inspection is a very practical and useful technique that enables us to find roughly a minimum structure

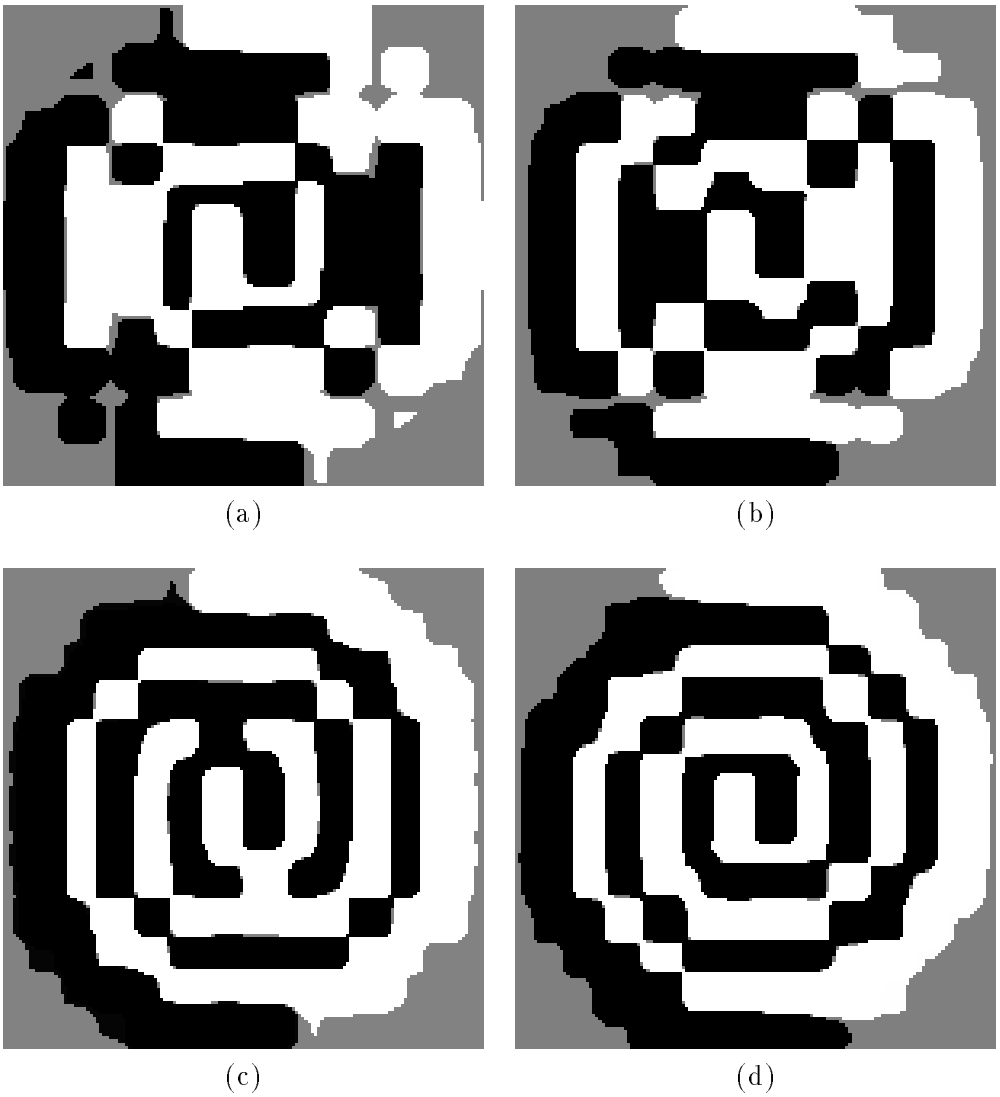


Figure 6.9: *Enhanced images of experiments in group 2.*

for NFC to do the job. For neural networks, we do not have similar quick and easy techniques to determine the minimum structure (node numbers and layer numbers) simply due to the uniformity in the node function.

6.4.2 IRIS Classification

The IRIS data set [1] is the measurements of the sepal length (SL) and width (SW) and petal length (PL) and width (PW) in millimeters of fifty plants for each of three types of iris: iris setosa (class 1), iris versicolor (class 2) and iris virginica (class 3). The data set contains 150 feature vectors and the corresponding classes; each of the feature vectors has 4 elements (SL, SW, PL and PW) which are used for classifying the data set into three classes.

The training data format is

$$(SL, SW, PL, PW;\ d_1, d_2, d_3),$$

where $\vec{fv} = (SL, SW, PL, PW)$ is the presented feature vector, $\vec{do} = (d_1, d_2, d_3)$ is the desired output specified by

$$d_i = \begin{cases} 1 \text{ if } \vec{fv} \text{ belongs to class } i, \\ 0 \text{ otherwise.} \end{cases} \tag{6.19}$$

Since the feature vector $\vec{fv}$ is the input to our NFC classifier, it will be referred to as the input vector with input variables $x1$ (SL), $x2$ (SW), $x3$ (PL) and $x4$ (PW).

Figure 6.10 contains four plots showing the discriminating power of each input variables. Clearly $x1$ and $x2$ have less discriminating power since we cannot find a single value of $x1$ or $x2$ that can be used as a threshold to identify any one of the three classes,

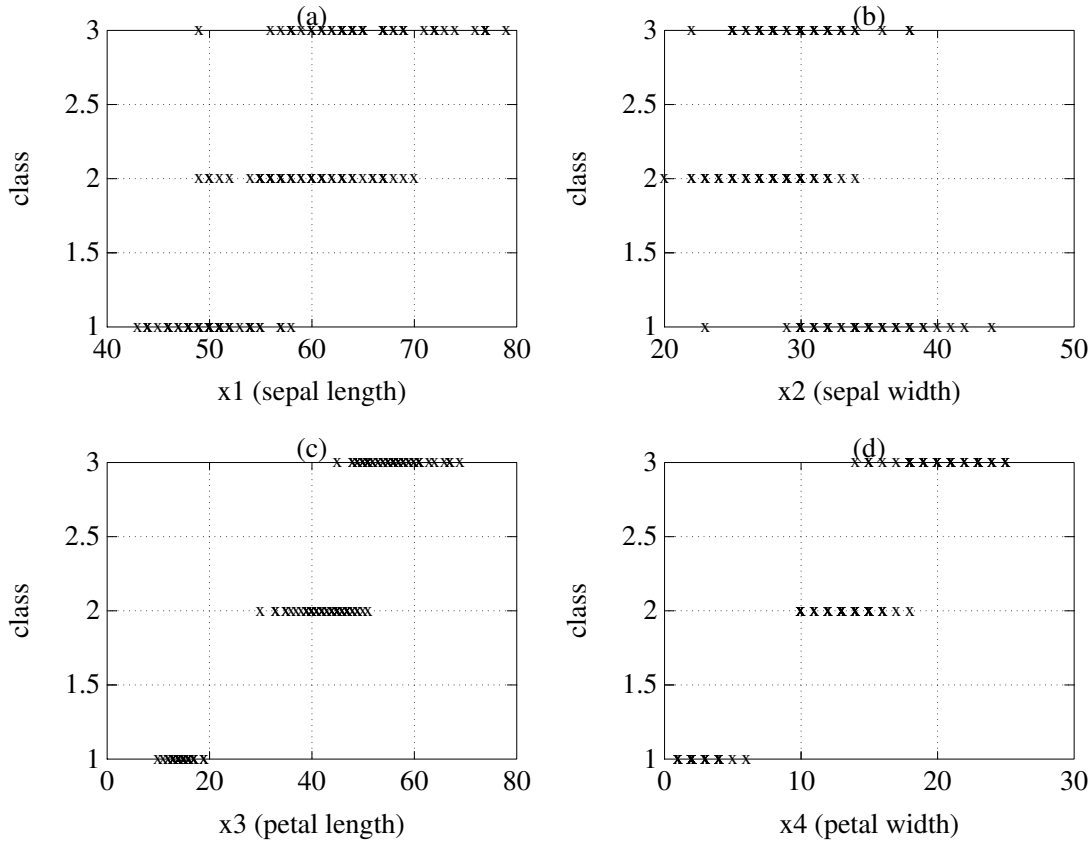


Figure 6.10: *Plots of classes w.r.t. single input variable (feature).*

as shown in Figure 6.10 (a) and (b). However, from Figure 6.10 (c) and (d), we notice that $x3$ and $x4$ have more discriminating power since, for instance, $x3 = 25$ or $x4 = 8$ can be used as a threshold to distinguish class 1 from others. These statements can be confirmed through the plots of Figure 6.11 which shows the plots of classes with respect to two of the input variables. Again it is obvious that class 1 can be identified by either $x3$ or $x4$. But for class 2 or 3, the discriminating power combining any two variables does not seem good enough and we have to rely on the combined discriminating power of three or four variables.

In our simulation, we assume the knowledge obtained from the class-variable plot (Figure 6.10 and 6.11) is not available, so the number of membership functions on each variable is set arbitrarily. We perform two experiments on the IRIS data set: experiment 1 has two membership functions on each input variable while experiment 2 has three.

As observed in the previous section, the adoption of the squared error measure causes a smaller initial misclassification number and less reduction in the misclassification number in the learning process. On the other hand, when the maximum-type error measure is chosen, it causes a larger initial misclassification number which decreases rapidly afterwards. As a result, we can take advantage of both sides and employ a combined approach. Namely, in order to get a smaller initial misclassification number, we choose the squared error measure first and identify the consequent parameters by the Kalman filter algorithm in the first half epoch. Then, to achieve a large reduction in the misclassification number, we switch to the maximum-type error measure after the first epoch and the adaptation proceeds by the gradient descent only. This approach is used for both experiment 1 and 2.

As for the initial parameters for the membership functions, they are set in such a

way that the domain intervals (obtained as the range between the minimum and maximum of each variable in the training data set) are totally covered and adjacent membership functions have overlappings satisfying ϵ-completeness with ϵ equal to 0.5. Figure 6.12 (a) and (c) show the initial membership functions for experiment 1 and 2; (b) and (d) are the membership functions after 200 epochs of adaptation. Since we adopt the maximum-type error measure after the first epoch, the final membership functions for both experiments do not differ too much from the initial ones.

We apply the same heuristics in Section 3.5 to dynamically adjust the step size according to the error measures obtained so far. The maximum-type error measure and the number of misclassified cases for both experiments are demonstrated as functions of epoch numbers in Figure 6.13. Because of the use of the Kalman filter algorithm to set up the initial consequent parameters, the initial misclassification numbers for both experiments are fairly small (both are below 10) compared to the number of cases to be classified (150). After 200 epochs of learning, experiment 1 still has one case of misclassification while experiment 2 can classify all cases correctly. This conforms to our observation on the scatter plot (Figure 6.11) that three membership functions for each input variable seem to be the minimum structure to perform the correct classification.

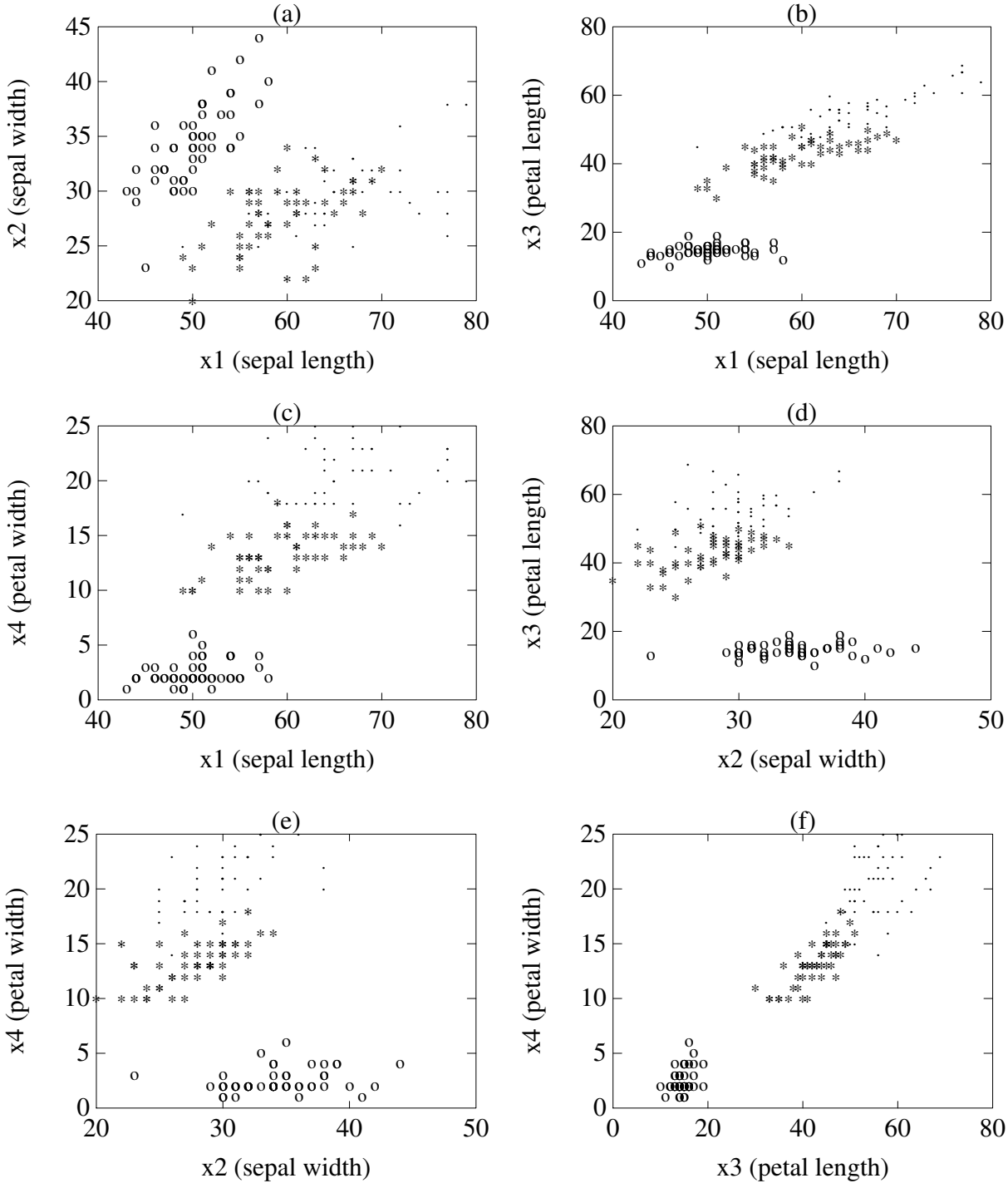


Figure 6.11: *Plots of classes w.r.t. two input variables. (o: class 1, *: class 2, .: class 3)*

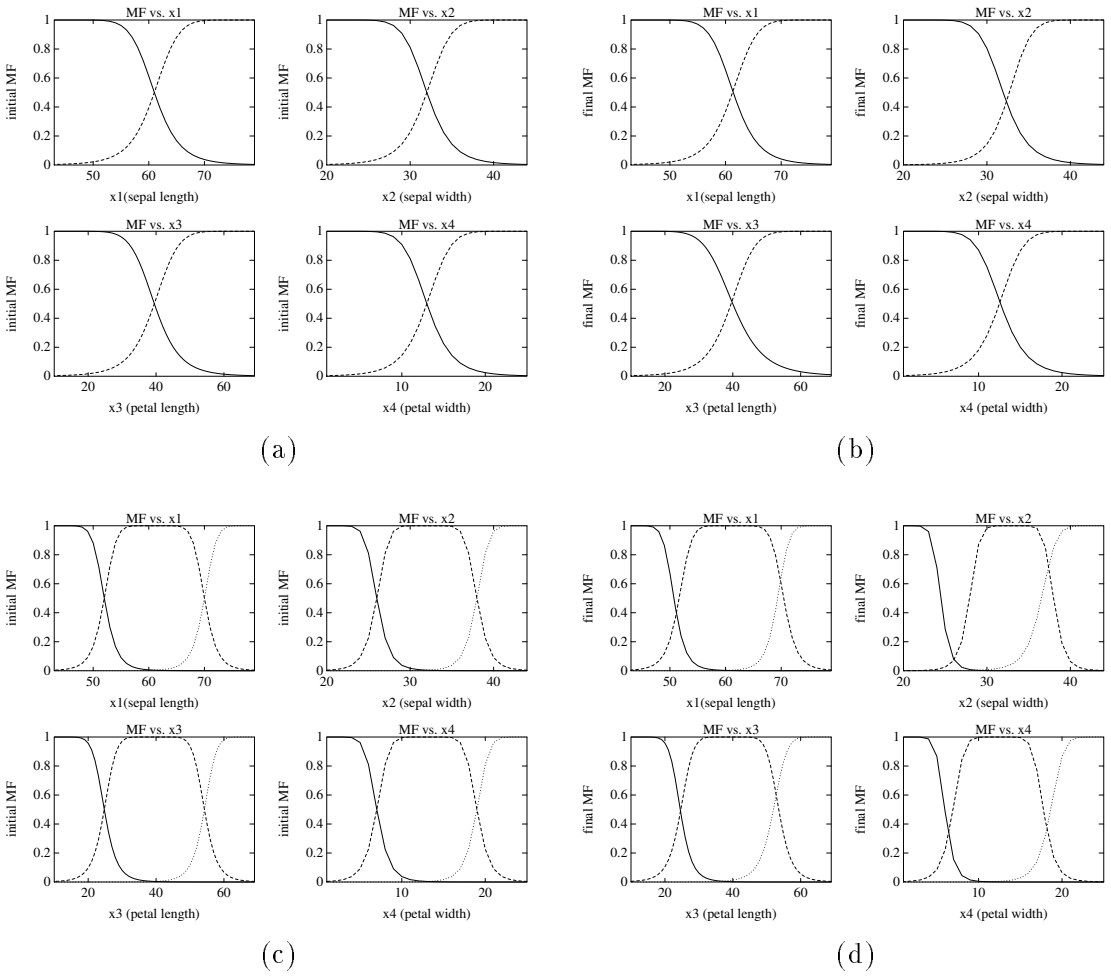


Figure 6.12: *Initial and final membership functions: (a)(b) experiment 1; (c)(d) experiment 2.*

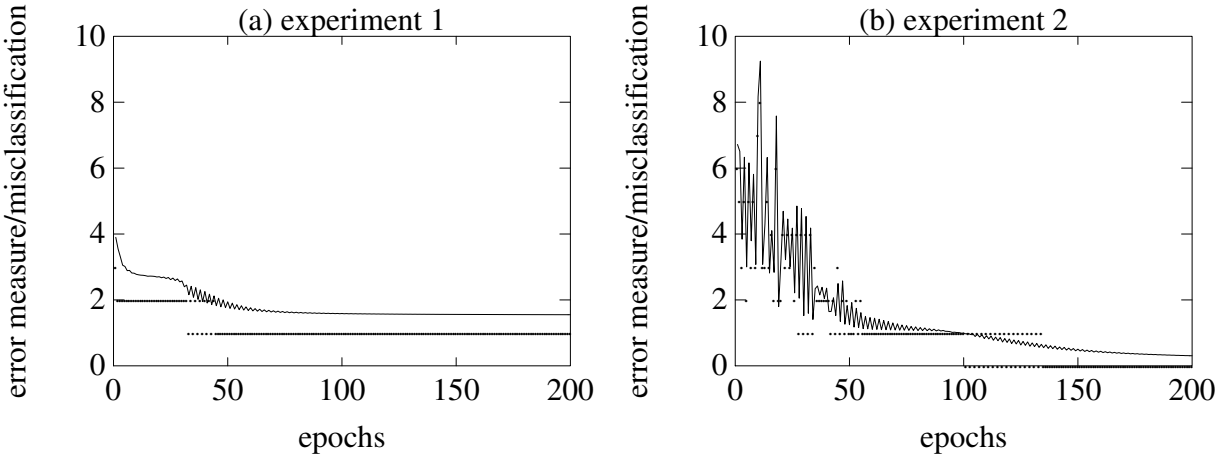


Figure 6.13: *Error curves and misclassified case numbers of (a) experiment 1 and (b) experiment 2. (Note that solid lines are maximum-type error measures; the dotted lines are misclassification numbers.)*

Chapter 7

Conclusions and Future Directions

7.1 Concluding Remarks

Before the introduction of learning algorithms into the fuzzy inference system, the relationships between the neural network (NN) and the fuzzy inference system (FIS) can be viewed as two extreme endpoints on a spectrum of modeling approaches. At one end, FIS has meaningful representations (fuzzy if-then rules and fuzzy reasoning) derived from human expertise, but it has no adaptive capability (learning from examples) to take advantage of a desired input-output data set. At the other end, NN represents a totally different paradigm with learning capability that adapts its parameters based on desired input-output pairs, but neither can it accommodate *a priori* knowledge from human experts, nor can we transform network configurations and connection weights into a meaningful representation to account for structured knowledge. Conceptually speaking, we may say that FIS is a top-down approach which employs high-level knowledge (rules) to describe a system, while NN is a bottom-up approach which uses low-level knowledge (input-output pairs) to tackle the same

problem.

However, ever since the introduction of learning rules or adaptive capability into FIS, FIS has gained an obvious advantage over NN, that is, the knowledge-representation feature that can both speed up learning (by encoding prior knowledge, in the form of fuzzy if-then rules, into parameters) and interpret the parameters after learning (by transforming the parameters back to fuzzy if-then rules). This breakthrough is significant because of the following:

1. For most applications, prior knowledge and/or human experts do exist and the "rules of thumb" can be expressed in various forms of fuzzy if-then rules. For a simple example, NN needs a couple of learning epochs to solve the XOR problem which is not linearly separable. However, for FIS, the specification of fuzzy if-then rules and corresponding membership functions to solve the same problem is trivially straightforward.

2. The knowledge base (rule base and database) derived from human experts may be inconsistent or biased due to personal differences, or it could be incomplete in some regions (*ignorant regions*) of input space due to the absence of knowledge. Based on desired input-output pairs, we can employ the learning rules to refine the knowledge base and thus improve the performance of FIS. (Note that the ignorant regions could cover the whole input space, resulting in the case of rule extraction from scratch.)

3. The structured knowledge representation in FIS is important for both rule-level manipulations (such as rule consistency checking, rule merging, incremental learning, etc.) and system-level operations (such as system compositions, modular design, etc.). Therefore the ability to keep meaningful representation after learning is also

crucial for other post-processes. For NN, it is hard to give physical meanings of each connection weight (especially when the number of layers is large, e.g., more than four) even when the NN can discover a perfect input-output mapping. For instance, suppose that we successfully employ a 2-1000-1 NN to solve the two-spiral problem, then how can we benefit from the obtained parameters and structure to solve a similar problem but with another added dimension, i.e., a 3-D two-spiral problem (with 2-D projection the same as the original problem)?

4. The basic learning rule of FIS is of gradient descent type which is the same as that of NN's. As a result, almost all kinds of new developments of NN's learning algorithms [79, 61, 66, 3, 98, 24, 17] which do not depend on NN's peculiar configuration can also be applied to FIS's learning algorithm, and vice versa. This fact allows these two modeling approaches to benefit from research findings and results in both literatures.

	Neural Networks	Conventional FIS	Adaptive FIS
Meaningful representation?	No	Yes	Yes
Reasoning mechanism?	No	Yes	Yes
Adaptive capability?	Yes	No	Yes
Hardware implementation?	Easier	Harder	Harder

Table 7.1: *Comparisons of NN, FIS and adaptive FIS.*

Thus, the demonstrated superiority of FIS to NN is mostly due to the fact that NN's uniform structure is hard to analyze, making it hard to represent structured knowledge. Quite a few researchers [20, 19, 67, 8, 13] are working on this problem in order to give appropriate interpretation of NN's configuration and connection weights, though with lim-

ited results for the time being. However, on the other hand, the uniform structure of NN's do offer some advantages: they simplify the computation in both learning and application phases and thus favor VLSI circuit implementations.

Table 7.1 summarizes the comparisons between NN, conventional FIS and adaptive FIS.

7.2 Future Directions

This work provides a foundation for future expansion of integrated modeling approaches based on both fuzzy if-then rules (prior knowledge) and input-output pairs (data set). We shall conclude this dissertation with several crucial and promising research directions along these lines.

- *Constrained membership functions*: As shown in the simulations in Section 3.6, 4.3 and 5.4, the initial membership functions are always set in such a way that the universe of discourse is covered with minimal membership value equal to 0.5. This setting is in spirit close to human concepts of reasonable distribution of membership functions. However, the membership functions after learning do not always conform to this guideline used for setting initial membership functions. This fact indicates that the adaptation, though almost always leading to a good function approximator, does not always lead to a parameter set that allows for intuitive and reasonable interpretation for humans. To remedy this, a constrained gradient method [105] has to be invoked for the adaptation process.

- *Structure-level adaptation*: Up to now, the adaptation (learning) process is only concerned with parameter-level adaptation within fixed structures. This is satisfactory only on the condition that we have the ability to set up an appropriate structure (premise structure, consequent structure, rule number, etc.) before the learning takes place. For large-scale problems, this is not always possible, and structure-level adaptation has to be introduced to reduce the cost of trial-and-error. The counterpart of this in NN, such as cascaded-correlation neural nets [18] or FUNNET (FUNction NETwork) [50], has been investigated for some time. Usually the structure-level adaptation for NN's can be classified into either constructive learning (start learning with a small NN and add neurons and/or connections if necessary) or destructive learning (start learning with a large NN and cut out connections and/or neurons if possible). We believe the structure-level adaptation for FIS can be pursued similarly.

- *On-line self-learning neuro-fuzzy controllers*: The self-learning fuzzy controller proposed in Chapter 5 needs a plant model before the learning can be started; therefore it falls into the category of *off-line learning*. For a real-world problem, the plant model is not always available and the learning has to be performed in an on-line manner. Thus it would be desirable if the TBP proposed in Chapter 5 has an on-line version. One way to achieve this is to establish the identification of the plant as part of the learning process such that the plant model and the controller are identified simultaneously. The other way is to incorporate the *reinforcement learning* [4, 25, 102, 49, 11], a well-known on-line learning paradigm derived from animal behavior, as a part of our original learning scheme.

- *Stability analysis for neuro-fuzzy controllers*: The lack of stability analysis for both neural networks and fuzzy controllers is the major reason why conventional control literature is resistant to these controllers. Due to the success of fuzzy control, the investigation of stability in fuzzy control is already underway [12, 10, 44, 37]. We think that the establishment of stability analysis can greatly expand the application domains of neuro-fuzzy controllers.

Appendix A

Appendix

Here we list the final 16 fuzzy if-then rules in example 4 of Chapter 4 which predicts the Mackey-Glass chaotic time series. Suppose that the i-th input variable is assigned two linguistic values $SMALL_i$ and $LARGE_i$, then the fuzzy if-then rules after training can be expressed as:

$$
\begin{cases}
If\ x(t-18)\ is\ SMALL_1\ and\ x(t-12)\ is\ SMALL_2\ and\ x(t-6)\ is\ SMALL_3\ and\ x(t)\ is\ SMALL_4,\ then\ x(t+6) = \vec{c}_1 \cdot \vec{X} \\
If\ x(t-18)\ is\ SMALL_1\ and\ x(t-12)\ is\ SMALL_2\ and\ x(t-6)\ is\ SMALL_3\ and\ x(t)\ is\ LARGE_4,\ then\ x(t+6) = \vec{c}_2 \cdot \vec{X} \\
If\ x(t-18)\ is\ SMALL_1\ and\ x(t-12)\ is\ SMALL_2\ and\ x(t-6)\ is\ LARGE_3\ and\ x(t)\ is\ SMALL_4,\ then\ x(t+6) = \vec{c}_3 \cdot \vec{X} \\
If\ x(t-18)\ is\ SMALL_1\ and\ x(t-12)\ is\ SMALL_2\ and\ x(t-6)\ is\ LARGE_3\ and\ x(t)\ is\ LARGE_4,\ then\ x(t+6) = \vec{c}_4 \cdot \vec{X} \\
If\ x(t-18)\ is\ SMALL_1\ and\ x(t-12)\ is\ LARGE_2\ and\ x(t-6)\ is\ SMALL_3\ and\ x(t)\ is\ SMALL_4,\ then\ x(t+6) = \vec{c}_5 \cdot \vec{X} \\
If\ x(t-18)\ is\ SMALL_1\ and\ x(t-12)\ is\ LARGE_2\ and\ x(t-6)\ is\ SMALL_3\ and\ x(t)\ is\ LARGE_4,\ then\ x(t+6) = \vec{c}_6 \cdot \vec{X} \\
If\ x(t-18)\ is\ SMALL_1\ and\ x(t-12)\ is\ LARGE_2\ and\ x(t-6)\ is\ LARGE_3\ and\ x(t)\ is\ SMALL_4,\ then\ x(t+6) = \vec{c}_7 \cdot \vec{X} \\
If\ x(t-18)\ is\ SMALL_1\ and\ x(t-12)\ is\ LARGE_2\ and\ x(t-6)\ is\ LARGE_3\ and\ x(t)\ is\ LARGE_4,\ then\ x(t+6) = \vec{c}_8 \cdot \vec{X} \\
If\ x(t-18)\ is\ LARGE_1\ and\ x(t-12)\ is\ SMALL_2\ and\ x(t-6)\ is\ SMALL_3\ and\ x(t)\ is\ SMALL_4,\ then\ x(t+6) = \vec{c}_9 \cdot \vec{X} \\
If\ x(t-18)\ is\ LARGE_1\ and\ x(t-12)\ is\ SMALL_2\ and\ x(t-6)\ is\ SMALL_3\ and\ x(t)\ is\ LARGE_4,\ then\ x(t+6) = \vec{c}_{10} \cdot \vec{X} \\
If\ x(t-18)\ is\ LARGE_1\ and\ x(t-12)\ is\ SMALL_2\ and\ x(t-6)\ is\ LARGE_3\ and\ x(t)\ is\ SMALL_4,\ then\ x(t+6) = \vec{c}_{11} \cdot \vec{X} \\
If\ x(t-18)\ is\ LARGE_1\ and\ x(t-12)\ is\ SMALL_2\ and\ x(t-6)\ is\ LARGE_3\ and\ x(t)\ is\ LARGE_4,\ then\ x(t+6) = \vec{c}_{12} \cdot \vec{X} \\
If\ x(t-18)\ is\ LARGE_1\ and\ x(t-12)\ is\ LARGE_2\ and\ x(t-6)\ is\ SMALL_3\ and\ x(t)\ is\ SMALL_4,\ then\ x(t+6) = \vec{c}_{13} \cdot \vec{X} \\
If\ x(t-18)\ is\ LARGE_1\ and\ x(t-12)\ is\ LARGE_2\ and\ x(t-6)\ is\ SMALL_3\ and\ x(t)\ is\ LARGE_4,\ then\ x(t+6) = \vec{c}_{14} \cdot \vec{X} \\
If\ x(t-18)\ is\ LARGE_1\ and\ x(t-12)\ is\ LARGE_2\ and\ x(t-6)\ is\ LARGE_3\ and\ x(t)\ is\ SMALL_4,\ then\ x(t+6) = \vec{c}_{15} \cdot \vec{X} \\
If\ x(t-18)\ is\ LARGE_1\ and\ x(t-12)\ is\ LARGE_2\ and\ x(t-6)\ is\ LARGE_3\ and\ x(t)\ is\ LARGE_4,\ then\ x(t+6) = \vec{c}_{16} \cdot \vec{X}
\end{cases}
\tag{A.1}
$$

where $\vec{X} = [x(t-18), x(t-12), x(t-6), x(t), 1]$ and $\vec{c}_i$ is the i-th row of the following consequent parameter matrix C:

$$C = \begin{bmatrix} 0.2167 & 0.7233 & -0.0365 & 0.5433 & 0.0276 \\ 0.2141 & 0.5704 & -0.4826 & 1.2452 & -0.3778 \\ -0.0683 & 0.0022 & 0.6495 & 2.7320 & -2.2916 \\ -0.2616 & 0.9190 & -2.9931 & 1.9467 & 1.6555 \\ -0.3293 & -0.8943 & 1.4290 & -1.6550 & 2.3735 \\ 2.5820 & -2.3109 & 3.7925 & -5.8068 & 4.0478 \\ 0.8797 & -0.9407 & 2.2487 & 0.7759 & -2.0714 \\ -0.8417 & -1.5394 & -1.5329 & 2.2834 & 2.4140 \\ -0.6422 & -0.4384 & 0.9792 & -0.3993 & 1.5593 \\ 1.5534 & -0.0542 & -4.7256 & 0.7244 & 2.7350 \\ -0.6864 & -2.2435 & 0.1585 & 0.5304 & 3.5411 \\ -0.3190 & -1.3160 & 0.9689 & 1.4887 & 0.7079 \\ -0.3200 & -0.4654 & 0.4880 & -0.0559 & 0.9622 \\ 4.0220 & -3.8886 & 1.0547 & -0.7427 & -0.4464 \\ 0.3338 & -0.3306 & -0.5961 & 1.1220 & 0.3529 \\ -0.5572 & 0.9190 & -0.8745 & 2.1899 & -0.9497 \end{bmatrix} \quad \text{(A.2)}$$

The linguistic labels $SMALL_i$ and $LARGE_i$ (i=1 to 4) are defined by the bell membership function (with different parameters a, b and c):

$$\mu_A(x) = \frac{1}{1 + [(\frac{x-c}{a})^2]^b}. \quad \text{(A.3)}$$

These membership functions are shown in Figure 4.22. The following table lists the linguistic labels and the corresponding consequent parameters in equation (A.3):

A	a	b	c
$SMALL_1$	0.1790	2.0456	0.4798
$LARGE_1$	0.1584	2.0103	1.4975
$SMALL_2$	0.2410	1.9533	0.2960
$LARGE_2$	0.2923	1.9178	1.7824
$SMALL_3$	0.3798	2.1490	0.6599
$LARGE_3$	0.4884	1.8967	1.6465
$SMALL_4$	0.2815	2.0170	0.3341
$LARGE_4$	0.1616	2.0165	1.4727

Table A.1: *Table of premise parameters in example 4.*

Bibliography

[1] D. F. Andrews and A. M. Herzberg. *Data: a collection of problems from many fields for the student and research worker*. Springer-Verlag, 1985.

[2] K. J. Aström and B. Wittenmark. *Computer Controller Systems: Theory and Design*. Prentice-Hall, 1984.

[3] N. Baba. A new approach for finding the global minimum of error function of neural networks. *Neural Networks*, 2:367–373, 1989.

[4] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. on Systems, Man, and Cybernetics*, 13(5):834–846, 1983.

[5] H. R. Berenji. Refinement of approximate reasoning-based controllers by reinforcement learning. In *Proc. of the Eighth International Workshop of Machine Learning*, Evanston, Illinois, June 1991.

[6] S. M. Botros and C. G. Atkeson. Generalization properties of radial basis functions. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems III*, pages 707–713. Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[7] R. H. Cannon. *Dynamics of Physical Systems.* McGraw-Hill, New York, 1967.

[8] L.-W. Chan. Analysis of the internal representations in neural networks for machine intelligence. In *Proc. of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pages 578–583, July 1991.

[9] S. Chen, C. F. N. Cowan, and P. M. Grant. Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Trans. on Neural Networks*, 2(2):302–309, March 1991.

[10] Y.-Y. Chen. *The global analysis of fuzzy dynamical systems.* PhD thesis, University of California at Berkeley, 1989.

[11] Y.-Y. Chen. A self-learning fuzzy controller. In *Proc. of IEEE international conference on fuzzy systems*, March 1992.

[12] Y.-Y. Chen and T.-C. Tsao. A description of the dynamic behavior of fuzzy systems. *IEEE Trans. on Systems, Man, and Cybernetics*, 19(4):745–755, July 1989.

[13] K. J. Cios and N. Liu. A machine learning method for generation of a neural network architecture: a continuous ID3 algorithm. *IEEE Trans. on Neural Networks*, 3(2):280–291, March 1992.

[14] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2:303–314, 1989.

[15] D. Dubois and H. Prade. *Fuzzy Sets and Systems: Theory and Applications.* Academic press, New York, 1980.

[16] D. Dubois and H. Prade. New results about properties and semantics of fuzzy set theoretic operators. In P. P. Wang and S. K. Chang, editors, *Fuzzy Sets: Theory and Applications to Policy Analysis and Information Systems*, pages 59–75. Plenum, New York, 1980.

[17] S. E. Fahlman. Faster-learning variations on back-propagation: an empirical study. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proc. of the 1988 Connectionist Models Summer School*, pages 38–51, Carnegic Mellon University, 1988.

[18] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D. S. Touretzky, G. Hinton, and T. Sejnowski, editors, *Advances in Neural Information Processing Systems II*. Morgan Kaufmann, 1990.

[19] Limin Fu. Rule learning by searching on adapted nets. In *Proc. of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pages 590–595, July 1991.

[20] S. Gallant. Connectionist expert systems. *Communication of Association for Computing Maching*, 31(2):152–169, February 1988.

[21] G. C. Goodwin and K. S. Sin. *Adaptive filtering prediction and control*. Prentice-Hall, Englewood Cliffs, N.J., 1984.

[22] S. S. Haykin. *Adaptive filter theory*. Prentice Hall, Englewood Cliffs, NJ, 2nd edition, 1991.

[23] S. Horiawa, T. Furuhashi, S. Ouma, and Y. Uchikawa. A fuzzy controller using a neural network and its capability to learn expert's control rules. In *Proc. of the*

International Conference on Fuzzy Logic and Neural Networks, pages 103–106, Japan, 1990.

[24] S.-C. Huang and Y.-F. Huang. Learning algorithms for perceptrons using back-propagation with selective updates. *IEEE Control Systems Magazine*, pages 56–61, April 1990.

[25] W. T. Miller III, R. S. Sutton, and P. J. Werbos, editors. *Neural Networks for Control*. Massachusetts Institute of Technology, 1990.

[26] A. G. Ivakhnenko. Polynomial theory of complex systems. *IEEE Trans. on Systems, Man, and Cybernetics*, 1(4):364–378, October 1971.

[27] J.-S. Roger Jang. Fuzzy modeling using generalized neural networks and Kalman filter algorithm. In *Proc. of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pages 762–767, July 1991.

[28] J.-S. Roger Jang. Rule extraction using generalized neural networks. In *Proc. of the 4th IFSA World Congress*, pages 82–86 (in the Volume for Artificial Intelligence), July 1991.

[29] J.-S. Roger Jang. Self-learning fuzzy controller based on temporal back-propagation. *IEEE Trans. on Neural Networks*, 3(5):714–723, September 1992.

[30] J.-S. Roger Jang. ANFIS: Adaptive-network-based fuzzy inference systems. *IEEE Trans. on Systems, Man, and Cybernetics*, 23(03):665–685, May 1993.

[31] R. D. Jones, Y. C. Lee, C. W. Barnes, G. W. Flake, K. Lee, and P. S. Lewis. Function approximation and time series prediction with neural networks. In *Proc. of IEEE International Joint Conference on Neural Networks*, pages I–649–665, 1990.

[32] V. Kadirkamanathan, M. Niranjan, and F. Fallside. Sequential adaptation of radial basis function neural networks. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems III*, pages 721–727. Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[33] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, pages 35–45, March 1960.

[34] L. V. Kantorovich and G. P. Akilov. *Functional analysis.* Pergamon, Oxford, 2nd edition, 1982.

[35] A. Kaufmann. *Introduction to Theory of Fuzzy Subsets.* Academic, New York, 1975.

[36] A. Kaufmann and M. M. Gupta. *Introduction to Fuzzy Arithmetic.* Van Nostrand Reinhold Company, 1985.

[37] S. Kawamoto, K. Tada, A. Ishigame, and T. Taniguchi. An approach to stability analysis of second order fuzzy systems. In *Proc. of IEEE international conference on fuzzy systems*, pages 1427–1434, March 1992.

[38] J. Keyes. AI on a chip. *AI Expert Magazine*, pages 33–38, April 1991.

[39] M. S. Klassen and Y.-H. Pao. Characteristics of the functional-link net: A higher order delta rule net. In *IEEE Proc. of the International Conference on Neural Networks*, San Diego, June 1988.

[40] T. Kondo. Revised GMDH algorithm estimating degree of the complete polynomial. *Tran. of the Society of Instrument and Control Engineers*, 22(9):928–934, 1986. (Japanese).

[41] B. Kosko. *Neural networks and fuzzy systems: a dynamical systems approach*. Prentice Hall, Englewood Ciffs, NJ, 1991.

[42] B. Kosko. *Neural networks for signal processing*. Prentice Hall, Englewood Ciffs, NJ, 1991.

[43] K. J. Lang and M. J. Witbrock. Learning to tell two spirals apart. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proc. of the 1988 Connectionist Models Summer School*, pages 52–59, Carnegic Mellon University, 1988.

[44] G. Langari. *A framework for analysis and synthesis of fuzzy linguistic control systems*. PhD thesis, University of California at Berkeley, 1990.

[45] A. S. Lapedes and R. Farber. Nonlinear signal processing using neural networks: prediction and system modeling. Technical Report LA-UR-87-2662, Los Alamos National Laboratory, Los Alamos, New Mexico 87545, 1987.

[46] C.-C. Lee. Fuzzy logic in control systems: fuzzy logic controller-part 1. *IEEE Trans. on Systems, Man, and Cybernetics*, 20(2):404–418, 1990.

[47] C.-C. Lee. Fuzzy logic in control systems: fuzzy logic controller-part 2. *IEEE Trans. on Systems, Man, and Cybernetics*, 20(2):419–435, 1990.

[48] C.-C. Lee. Intelligent control based on fuzzy logic and neural network theory. In *Proc. of the Int. Conf. on Fuzzy Logic and Neural Networks, Iizuka*, pages 759–764, 1990.

[49] C.-C. Lee. A self-learning rule-based controller employing approximate reasoning and neural net concepts. *International Journal of Intelligent Systems*, 5(3):71–93, 1991.

[50] T.-C. Lee. *Structure level adaptation for artificial neural networks.* Kluwer Academic Publishers, 1991.

[51] C.-T. Lin and C. S. G. Lee. Neural-network-based fuzzy logic control and decision system. *IEEE Trans. on Computers*, 40(12):1320–1336, December 1991.

[52] R. P. Lippmann. An introduction to computing with neural networks. *IEEE Acoustics, Speech, and Signal Processing Magazine*, 4(2):4–22, 1987.

[53] L. Ljung. *System identification: theory for the user.* Prentice-Hall, Englewood Cliffs, N.J., 1987.

[54] M. C. Mackey and L. Glass. Oscillation and chaos in physiological control systems. *Science*, 197:287–289, July 1977.

[55] M. Minsky and S. Papert. *Perceptrons.* MIT Press, MA, 1969.

[56] J. Moody. Fast learning in multi-resolution hierarchies. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems I*, chapter 1, pages 29–39. Morgan Kaufmann, San Mateo, CA, 1989.

[57] J. Moody and C. Darken. Learning with localized receptive fields. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proc. of the 1988 Connectionist Models Summer School*. Carnegie Mellon University, Morgan Kaufmann Publishers, 1988.

[58] J. Moody and C. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:281–294, 1989.

[59] M. T. Musavi, W. Ahmed, K. H. Chan, K. B. Faris, and D. M. Hummels. On the training of radial basis function classifiers. *Neural Networks*, 5(4):595–603, 1992.

[60] K. S. Narendra and K. Parthsarathy. Identification and control of dynamical systems using neural networks. *IEEE Trans. on Neural Networks*, 1(1):4–27, 1990.

[61] D. Nguyen and B. Widrow. Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. In *Proc. of IEEE International Joint Conference on Neural Networks*, pages III–21–26, 1990.

[62] D. H. Nguyen and B. Widrow. Neural networks for self-learning control systems. *IEEE Control Systems Magazine*, pages 18–23, April 1990.

[63] N.J. Nilsson. *Learning machines: foundations of trainable pattern classifying systems.* McGraw-Hill, New York, 1965.

[64] Y.-H. Pao. *Adaptive Pattern Recognition and Neural Networks*, chapter 8, pages 197–222. Addison-Wesley Publishing Company, Inc., 1989.

[65] D. B. Parker. Learning logic. Invention Report S81-64, File 1, Office of Technology Licensing, Standford University, October 1982.

[66] D. B. Parker. Optimal algorithms for adaptive networks: Second order back propagation, second order direct propagation, and second order Hebbian learning. In *Proc. of IEEE International Conference on Neural Networks*, pages 593–600, 1987.

[67] L. Y. Pratt, J. Mostow, and C. A. Kamm. Direct transfer of learned information among neural networks. In *Proc. of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pages 584–589, July 1991.

[68] J. R. Quinlan. Learning efficient classification procedures and hteir application to chess and games. In R. S. Michalski, J. G. Carbonell, and T. M Mitchell, editors, *Maching Learning*, chapter 15, pages 463–483. Morgan Kaufmann, Los Altos, 1983.

[69] III R. S. Crowder ˙ Predicting the Mackey-Glass timeseries with cascade-correlation learning. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proc. of the 1990 Connectionist Models Summer School*, pages 117–123, Carnegic Mellon University, 1990.

[70] A. K. Rigler, J. M. Irvine, and T. P. Vogl. Rescaling of variables in back propagation learning. *Neural Networks*, 4(2):225–229, 1991.

[71] F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the theory of brain mechanisms*. Spartan, New York, 1962.

[72] H. L. Royden. *Real analysis*. Macmillan, New York, 2nd edition, 1968.

[73] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and James L. McClelland, editors, *Parallel*

Distributed Processing: Explorations in the Microstructure of Cognition, volum 1, chapter 8, pages 318–362. The MIT Press, 1986.

[74] T. D. Sanger. A tree-structured adaptive network for function approximate in high-dimensional spaces. *IEEE Trans. on Neural Networks*, 2(2):285–293, March 1991.

[75] B. Schweizer and A. Sklar. Associative functions and abstract semi-groups. *Publ. Math. Debrecen*, 10:69–81, 1963.

[76] S. Shah, F. Palmieri, and M. Datum. Optimal filtering algorithms for fast learning in feedforward neural networks. *Neural Networks*, 5(5):779–787, 1992.

[77] S. Shar and F. Palmieri. MEKA-a fast, local algorithm for training feedforward neural networks. In *Proc. of International Joint Conference on Neural Networks*, pages III 41–46, 1990.

[78] J. M. Sibigtroth. Implementing fuzzy expert rules in hardware. *AI Expert Magazine*, pages 25–31, April 1992.

[79] S. Singhal and L. Wu. Training multilayer perceptrons with the extended kalman algorithm. In David S. Touretzky, editor, *Advances in neural information processing systems I*, pages 133–140. Morgan Kaufmann Publishers, 1989.

[80] S. M. Smith and D. J. Comer. Automated calibration of a fuzzy logic controller using a cell state space algorithm. *IEEE Control Systems Magazine*, 11(5):18–28, August 1991.

[81] P. Strobach. *Linear prediction theory: a mathematical basis for adaptive systems.* Springer-Verlag, 1990.

[82] M. Sugeno. Fuzzy measures and fuzzy integrals: a survey. In M. M. Gupta, G. N. Saridis, and B. R. Gaines, editors, *Fuzzy Automata and Decision Processes*, pages 89–102. North-Holland, New York, 1977.

[83] M. Sugeno and G. T. Kang. Structure identification of fuzzy model. *Fuzzy Sets and Systems*, 28:15–33, 1988.

[84] C.-T. Sun. Rulebase structure identification in an adaptive network based fuzzy inference system. *IEEE Trans. on Fuzzy Systems*, 2(1), 1994. (Forthcoming).

[85] C.-T Sun and J.-S. Roger Jang. Adaptive network based fuzzy classification. In *Proc. of the Japan-U.S.A. Symposium on Flexible Automation*, July 1992.

[86] C.-T. Sun and J.-S. Roger Jang. A neuro-fuzzy classifier and its applications. In *Proc. of IEEE international conference on fuzzy systems*, San Francisco, March 1993.

[87] C.-T. Sun, J.-S. Roger Jang, and C.-Y. Fu. Neural network analysis of plasma spectra. In *Proc. of the International Conference on Artificial Neural Networks*, Amsterdam, September 1993.

[88] H. Takagi and I. Hayashi. Artificial-neural-network-driven fuzzy reasoning. In *Proc. of International Workshop on Fuzzy System Applications*, pages 217–218, August 1988.

[89] H. Takagi and I. Hayashi. NN-driven fuzzy reasoning. *International Journal of Approximate Reasoning*, 5(3):191–212, 1991.

[90] T. Takagi and M. Sugeno. Derivation of fuzzy control rules from human operator's control actions. *Proc. of the IFAC Symp. on Fuzzy Information, Knowledge Representation and Decision Analysis*, pages 55–60, July 1983.

[91] T. Takagi and M. Sugeno. Fuzzy identification of systems and its applications to modeling and control. *IEEE Trans. on Systems, Man, and Cybernetics*, 15:116–132, 1985.

[92] M. Togai and H. Watanabe. Expert system on a chip: an engine for real-time approximate reasoning. *IEEE Expert*, pages 55–62, 1986. Fall issue.

[93] Y. Tsukamoto. An approach to fuzzy reasoning method. In Madan M. Gupta, Rammohan K. Ragade, and Ronald R. Yager, editors, *Advances in Fuzzy Set Theory and Applications*, pages 137–149. North-Holland, Amsterdam, 1979.

[94] L.-X. Wang. Fuzzy systems are universal approximators. In *Proc. of the IEEE International Conference on Fuzzy Systems*, San Diego, March 1992.

[95] L.-X. Wang and J. M. Mendel. Generating fuzzy rules from numerical data, with applications. Technical Report USC-SIPI-169, Signal and Image Processing Institute, Univ. of Southern California, Los Angeles, 1990.

[96] L.-X. Wang and J. M. Mendel. Back-propagation fuzzy systems as nonlinear dynamic system identifiers. In *Proc. of the IEEE International Conference on Fuzzy Systems*, San Diego, March 1992.

[97] L.-X. Wang and J. M. Mendel. Fuzzy basis function, universal approximation, and orthogonal least squares learning. *IEEE Trans. on Neural Networks*, 3(5):807–814, September 1992.

[98] R. L. Watrous. Learning algorithms for connectionist network: applied gradient methods of nonlinear optimization. In *Proc. of IEEE International Conference on Neural Networks*, pages 619–627, 1991.

[99] A. A. Weigend, D. E. Rumelhart, and B. A. Huberman. Back-propagation, weight-elimination and time series prediction. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proc. of the 1990 Connectionist Models Summer School*, pages 105–116, Carnegic Mellon University, 1990.

[100] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman. Generalization by weight-elimination with application to forecasting. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems III*, pages 875–882. Morgan Kaufmann, San Mateo, CA, 1991.

[101] P. Werbos. *Beyond regression: New tools for prediction and analysis in the behavioral sciences.* PhD thesis, Harvard University, 1974.

[102] P. J. Werbos. An overview of neural networks for control. *IEEE Control Systems Magazine*, 11(1):40–41, January 1991.

[103] B. Widrow and D. Stearns. *Adaptive Signal Processing.* Prentice-Hall, Englewood Cliffs, N.J., 1985.

[104] B. Widrow and R. Winter. Neural nets for adaptive filtering and adaptive pattern recognition. *IEEE Computer*, pages 25–39, March 1988.

[105] D. A. Wismer and R. Chattergy. *Introduction to nonlinear optimization: a problem solving approach*, chapter 6, pages 139–162. North-Holland Publishing Company, 1978.

[106] R. Yager. On a general class of fuzzy connectives. *Fuzzy Sets and Systems*, 4:235–242, 1980.

[107] T. Yamakawa. High-speed fuzzy controller hardware systems: the mega-FIPS machine. *Information Sciences*, 45:113–128, 1988.

[108] T. Yamakawa and H. Kabuo. A programmable fuzzifier integrated circuit-synthesis, design and fabrication. *Information Sciences*, 45:75–112, 1988.

[109] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[110] L. A. Zadeh. Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Trans. on Systems, Man, and Cybernetics*, 3(1):28–44, January 1973.

[111] L. A. Zadeh. Fuzzy logic. *Computer*, 1(4):83–93, 1988.

[112] H. J. Zimmermann, editor. *Fuzzy Set Theory and Its Applications*. Kluwer Academic, Boston, 1985.

[113] H. J. Zimmermann. *Fuzzy Sets, Decision Making, and Expert Systems*. Kluwer Academic Publishers, 1987.