1

Neuro-Fuzzy Modeling and Control

Jyh-Shing Roger Jang, Chuen-Tsai Sun

Abstract— **Fundamental and advanced developments in neuro-fuzzy synergisms for modeling and control are reviewed. The essential part of neuro-fuzzy synergisms comes from a common framework called adaptive networks, which unifies both neural networks and fuzzy models. The fuzzy models under the framework of adaptive networks is called ANFIS (Adaptive-Network-based Fuzzy Inference System), which possess certain advantages over neural networks. We introduce the design methods for ANFIS in both modeling and control applications. Current problems and future directions for neuro-fuzzy approaches are also addressed.**

Keywords— **Fuzzy logic, neural networks, fuzzy modeling, neuro-fuzzy modeling, neuro-fuzzy control, ANFIS.**

I. Introduction

In 1965, Zadeh published the first paper on a novel way of characterizing non-probabilistic uncertainties, which he called **fuzzy sets** [118]. This year marks the 30th anniversary of fuzzy logic and fuzzy set theory, which has now evolved into a fruitful area containing various disciplines, such as calculus of fuzzy if-then rules, fuzzy graphs, fuzzy interpolation, fuzzy topology, fuzzy reasoning, fuzzy inferences systems, and fuzzy modeling. The applications, which are multi-disciplinary in nature, includes automatic control, consumer electronics, signal processing, time-series prediction, information retrieval, database management, computer vision, data classification, decision-making, and so on.

Recently, the resurgence of interest in the field of artificial neural networks has injected a new driving force into the "fuzzy" literature. The back-propagation learning rule, which drew little attention till its applications to artificial neural networks was discovered, is actually an universal learning paradigm for any smooth parameterized models, including fuzzy inference systems (or fuzzy models). As a result, a fuzzy inference system can now not only take linguistic information (linguistic rules) from human experts, but also adapt itself using numerical data (input/output pairs) to achieve better performance. This gives fuzzy inference systems an edge over neural networks, which cannot take linguistic information directly.

In this paper, we formalize the adaptive networks as a universal representation for any parameterized models. Under this common framework, we reexamine back-propagation algorithm and propose speedup schemes utilizing the least-squared method. We explain why neural networks and fuzzy inference systems are all special instances of adaptive networks when proper node functions are assigned, and all learning schemes applicable to adaptive networks are also qualified methods for neural networks and fuzzy inference systems.

When represented as an adaptive network, a fuzzy inference system is called ANFIS (Adaptive-Networks-based Fuzzy Inference Systems). For three of the most commonly used fuzzy inference systems, the equivalent ANFIS can be derived directly. Moreover, the training of ANFIS follows the spirit of the **minimum disturbance principle** [111] and is thus more efficient than sigmoidal neural networks.

Once a fuzzy inference system is equipped with learning capability, all the design methodologies for neural network controllers become directly applicable to fuzzy controllers. We briefly review these design techniques and give related references for further studies.

The arrangement of this article is as follows. In Section 2, an in-depth introduction to the basic concepts of fuzzy sets, fuzzy reasoning, fuzzy if-then rules, and fuzzy inference systems are given. Section 3 is devoted to the formalization of adaptive networks and their learning rules, where the back-propagation neural network and radial basis function network are included as special cases. Section 4 explains the ANFIS architecture and demonstrates its superiority over back-propagation neural networks. A number of design techniques for fuzzy and neural controllers is described in Section 5. Section 6 concludes this paper by pointing out current problems and future directions.

II. Fuzzy Sets, Fuzzy Rules, Fuzzy Reasoning, and Fuzzy Models

This section provides a concise introduction to and a summary of the basic concepts central to the study of fuzzy sets. Detailed treatments of specific subjects can be found in the reference list.

A. Fuzzy Sets

A classical set is a set with a crisp boundary. For example, a classical set A can be expressed as

$$A = \{x \mid x > 6\}, \tag{1}$$

where there is a clear, unambiguous boundary point 6 such that if x is greater than this number, then x belongs to the set A, otherwise x does not belong to this set. In contrast to a classical set, a **fuzzy set**, as the name implies, is a set without a crisp boundary. That is, the transition from "belonging to a set" to "not belonging to a set" is gradual, and this smooth transition is characterized by **membership functions** that give fuzzy sets flexibility in modeling commonly used linguistic expressions, such as "the water is hot" or "the temperature is high." As Zadeh pointed out in 1965 in his seminal paper entitled "Fuzzy Sets" [118],

This paper is to appear in the Proceedings of the IEEE, March 1995

Jyh-Shing Roger Jang is with the Control and Simulation Group, The MathWorks, Inc., Natick, Massachusetts. Email: jang@mathworks.com.

Chuen-Tsai Sun is with the Department of Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan. Email: ctsun@cis.nctu.edu.tw.

such imprecisely defined sets or classes "play an important role in human thinking, particularly in the domains of pattern recognition, communication of information, and abstraction." Note that the fuzziness does not come from the randomness of the constituent members of the sets, but from the uncertain and imprecise nature of abstract thoughts and concepts.

Definition 1: Fuzzy sets and membership functions
If X is a collection of objects denoted generically by x, then a fuzzy set A in X is defined as a set of ordered pairs:

$$A = \{(x, \mu_A(x)) \mid x \in X\} \tag{2}$$

$\mu_A(x)$ is called the membership function (MF for short) of x in A. The MF maps each element of X to a continuous membership value (or membership grade) between 0 and 1.

□

Obviously the definition of a fuzzy set is a simple extension of the definition of a classical set in which the characteristic function is permitted to have continuous values between 0 and 1. If the value of the membership function $\mu_A(x)$ is restricted to either 0 or 1, then A is reduced to a classical set and $\mu_A(x)$ is the characteristic function of A.

Usually X is referred to as the **universe of discourse**, or simply the **universe**, and it may contain either discrete objects or continuous values. Two examples are given below.

Example 1: Fuzzy sets with discrete X
Let X = {1, 2, 3, 4, 5, 6, 7, 8} be the set of numbers of courses a student may take in a semester. Then the fuzzy set A = "appropriate number of courses taken" may be described as follows:

$$A = \{(1, 0.1), (2, 0.3), (3, 0.8), (4, 1), (5, 0.9), (6, 0.5), (7, 0.2), (8, 0.1)\}.$$

This fuzzy set is shown in Figure 1 (a).

□

Example 2: Fuzzy sets with continuous X
Let $X = R^+$ be the set of possibles ages for human beings. Then the fuzzy set B = "about 50 years old" may be expressed as

$$B = \{(x, \mu_B(x)) | x \in X\},$$

where

$$\mu_B(x) = \frac{1}{1 + \left(\frac{x-50}{5}\right)^4}.$$

This is illustrated in Figure 1 (b).

□

An alternative way of denoting a fuzzy set A is

$$A = \begin{cases} \sum_{x_i \in X} \mu_A(x_i)/x_i, & \text{if } X \text{ is discrete.} \\ \int_X \mu_A(x)/x, & \text{if } X \text{ is continuous.} \end{cases} \tag{3}$$

The summation and integration signs in equation (3) stand for the union of $(x, \mu_A(x))$ pairs; they do not indicate summation or integration. Similarly, "/" is only a marker and does not imply division. Using this notation, we can rewrite the fuzzy sets in examples 1 and 2 as

$$A = 0.1/1 + 0.3/2 + 0.8/3 + 1.0/4 + 0.9/5 + 0.5/6 + 0.2/7 + 0.1/8,$$

and

$$B = \int_{R^+} \frac{1}{1 + (\frac{x-50}{5})^4} \Big/ x,$$

respectively.

From example 1 and 2, we see that the construction of a fuzzy set depends on two things: the identification of a suitable universe of discourse and the specification of an appropriate membership function. It should be noted that the specification of membership functions is quite *subjective*, which means the membership functions specified for the same concept (say, "cold") by different persons may vary considerably. This subjectivity comes from the indefinite nature of abstract concepts and has nothing to do with randomness. Therefore the *subjectivity* and *nonrandomness* of fuzzy sets is the primary difference between the study of fuzzy sets and probability theory, which deals with objective treatment of random phenomena.

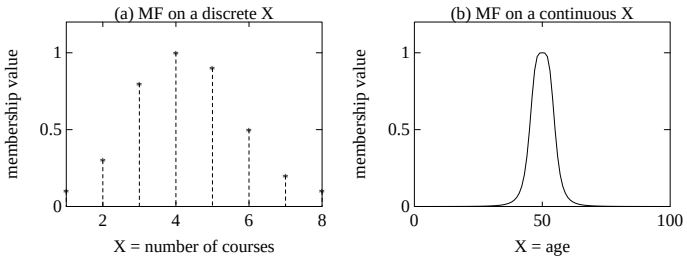


Fig. 1. *(a) A = "appropriate number of courses taken"; (b) B = "about 50 years old".*

Corresponding to the ordinary set operations of union, intersection, and complement, fuzzy sets have similar operations, which were initially defined in Zadeh's paper [118]. Before introducing these three fuzzy set operations, first we will define the notion of containment, which plays a central role in both ordinary and fuzzy sets. This definition of containment is, of course, a natural extension of the case for ordinary sets.

Definition 2: Containment or subset
Fuzzy set A is **contained** in fuzzy set B (or, equivalently, A is a **subset** of B, or A is smaller than or equal to B) if and only if $\mu_A(x) \leq \mu_B(x)$ for all x. In symbols,

$$A \subseteq B \Longleftrightarrow \mu_A(x) \leq \mu_B(x). \tag{4}$$

□

Definition 3: Union (disjunction)
The **union** of two fuzzy sets A and B is a fuzzy set C, written as $C = A \cup B$ or $C = A \ \ OR \ \ B$, whose MF is related to those of A and B by

$$\mu_C(x) = max(\mu_A(x), \mu_B(x)) = \mu_A(x) \vee \mu_B(x). \tag{5}$$

□

As pointed out by Zadeh [118], a more intuitive and appealing definition of union is the smallest fuzzy set containing both A and B. Alternatively, if D is any fuzzy set that contains both A and B, then it also contains $A \cup B$. The intersection of fuzzy sets can be defined analogously.

Definition 4: Intersection (conjunction)
The **intersection** of two fuzzy sets A and B is a fuzzy set C, written as $C = A \cap B$ or $C = A \ \ AND \ \ B$, whose MF is related to those of A and B by

$$\mu_C(x) = min(\mu_A(x), \mu_B(x)) = \mu_A(x) \wedge \mu_B(x). \quad (6)$$

□

As in the case of the union, it is obvious that the intersection of A and B is the largest fuzzy set which is contained in both A and B. This reduces to the ordinary intersection operation if both A and B are nonfuzzy.

Definition 5: Complement (negation)
The **complement** of fuzzy set A, denoted by $\overline{A}$ ($\neg A$, $NOT\ A$), is defined as

$$\mu_{\overline{A}}(x) = 1 - \mu_A(x). \quad (7)$$

□

Figure 2 demonstrates these three basic operations: (a) illustrates two fuzzy sets A and B, (b) is the complement of A, (c) is the union of A and B, and (d) is the intersection of A and B.

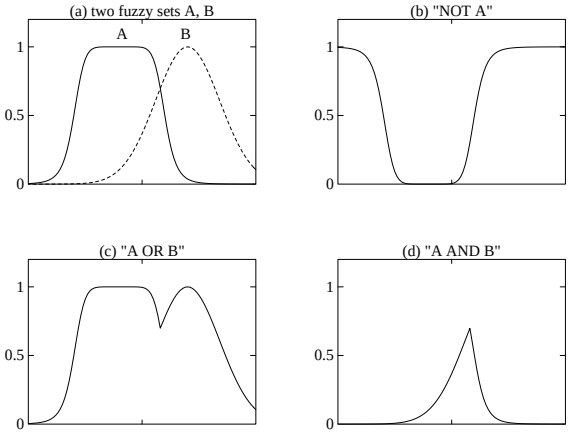


Fig. 2. *Operations on fuzzy sets: (a) two fuzzy sets A and B; (b) $\overline{A}$; (c) $A \cup B$; (d) $A \cap B$.*

Note that other consistent definitions for fuzzy AND and OR have been proposed in the literature under the names **T-norm** and **T-conorm** operators [16], respectively. Except for *min* and *max*, none of these operators satisfy the law of distributivity:

$$\begin{aligned} \mu_{A\cup(B\cap C)}(x) &= \mu_{(A\cup B)\cap(A\cup C)}(x), \\ \mu_{A\cap(B\cup C)}(x) &= \mu_{(A\cap B)\cup(A\cap C)}(x). \end{aligned}$$

However, *min* and *max* do incur some difficulties in analyzing fuzzy inference systems. A popular alternative is to use the probabilistic AND and OR:

$$\begin{aligned} \mu_{A\cap B}(x) &= \mu_A(x)\mu_B(x). \\ \mu_{A\cup B}(x) &= \mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x). \end{aligned}$$

In the following, we shall give several classes of parameterized functions commonly used to define MF's. These parameterized MF's play an important role in adaptive fuzzy inference systems.

Definition 6: Triangular MF's
A **triangular MF** is specified by three parameters $\{a, b, c\}$, which determine the x coordinates of three corners:

$$triangle(x; a, b, c) = max\left(min\left(\frac{x-a}{b-a}, \frac{c-x}{c-b}\right), 0\right). \quad (8)$$

Figure 3 (a) illustrates an example of the triangular MF defined by $triangle(x; 20, 60, 80)$.

□

Definition 7: Trapezoidal MF's
A **trapezoidal MF** is specified by four parameters $\{a, b, c, d\}$ as follows:

$$trapezoid(x; a, b, c, d) = max\left(min\left(\frac{x-a}{b-a}, 1, \frac{d-x}{d-c}\right), 0\right) \quad (9)$$

Figure 3 (b) illustrates an example of a trapezoidal MF defined by $trapezoid(x; 10, 20, 60, 95)$. Obviously, the triangular MF is a special case of the trapezoidal MF.

□

Due to their simple formulas and computational efficiency, both triangular MF's and trapezoidal MF's have been used extensively, especially in real-time implementations. However, since the MF's are composed of straight line segments, they are not smooth at the switching points specified by the parameters. In the following we introduce other types of MF's defined by smooth and nonlinear functions.

Definition 8: Gaussian MF's
A **Gaussian MF** is specified by two parameters $\{\sigma, c\}$:

$$gaussian(x; \sigma, c) = e^{-\left(\frac{x-c}{\sigma}\right)^2}, \quad (10)$$

where c represents the MF's center and σ determines the MF's width. Figure 3 (c) plots a Gaussian MF defined by $gaussian(x; 20, 50)$.

□

Definition 9: Generalized bell MF's
A **generalized bell MF** (or **bell MF**) is specified by three parameters $\{a, b, c\}$:

$$bell(x; a, b, c) = \frac{1}{1 + \left|\frac{x-c}{a}\right|^{2b}}, \quad (11)$$

where the parameter b is usually positive. Note that this MF is a direct generalization of the Cauchy distribution

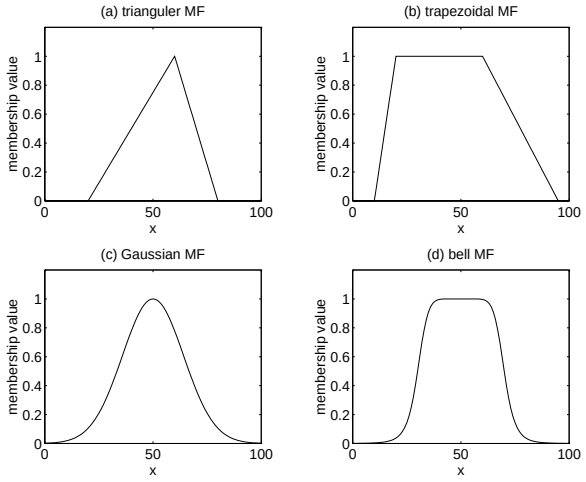


Fig. 3. *Examples of various classes of MF's: (a) triangle*$(x;20,60,80)$*; (b) trapezoid*$(x;10,20,60,95)$*; (c) gaussian*$(x;20,50)$*; (d) bell*$(x;20,4,50)$.

used in probability theory. Figure 3 illustrates a generalized bell MF defined by $bell(x;20,4,50)$.

□

A desired generalized bell MF can be obtained by a proper selection of the parameter set $\{a,b,c\}$. Specifically, we can adjust c and a to vary the center and width of the MF, and then use b to control the slopes at the crossover points. Figure 4 shows the physical meanings of each parameter in a bell MF.

Because of their smoothness and concise notation, Gaussian MF's and bell MF's are becoming increasingly popular methods for specifying fuzzy sets. Gaussian functions are well known in the fields of probability and statistics, and they possess useful properties such as invariance under multiplication and Fourier transform. The bell MF has one more parameter than the Gaussian MF, so it can approach a nonfuzzy set if $b \to \infty$.

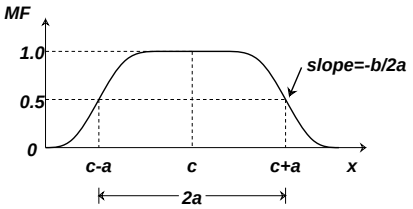


Fig. 4. *Physical meaning of parameters in a generalized bell function.*

Definition 10: Sigmoidal MF's
A **sigmoidal MF** is defined by

$$sigmoid(x;a,c) = \frac{1}{1+exp[-a(x-c)]}, \qquad (12)$$

where a controls the slope at the crossover point $x = c$.

□

Depending on the sign of the parameter a, a sigmoidal MF is inherently open right or left and thus is appropriate for representing concepts such as "very large" or "very negative." Sigmoidal functions of this kind are employed widely as the activation function of artificial neural networks. Therefore, for a neural network to simulate the behavior of a fuzzy inference system, the first problem we face is how to synthesize a close MF through a sigmoidal function. There are two simple ways to achieve this: one is to take the product of two sigmoidal MF's; the other is to take the absolute difference of two sigmoidal MF's.

It should be noted that the list of MF's introduced in this section is by no means exhaustive; other specialized MF's can be created for specific applications if necessary. In particular, any types of continuous probability distribution functions can be used as an MF here, provided that a set of parameters are given to specify the appropriate meanings of the MF.

B. Fuzzy If-Then Rules

A **fuzzy if-then rule** (**fuzzy rule**, **fuzzy implication** or **fuzzy conditional statement**) assumes the form

$$\textit{if } x \textit{ is } A \textit{ then } y \textit{ is } B, \qquad (13)$$

where A and B are linguistic values defined by fuzzy sets on universes of discourse X and Y, respectively. Often "x *is* A" is called the **antecedent** or **premise** while "y *is* B" is called the **consequence** or **conclusion**. Examples of fuzzy if-then rules are widespread in our daily linguistic expressions, such as the following:

- If pressure is high then volume is small.
- If the road is slippery then driving is dangerous.
- If a tomato is red then it is ripe.
- If the speed is high then apply the brake a little.

Before we can employ fuzzy if-then rules to model and analyze a system, we first have to formalize what is meant by the expression "*if* x *is* A *then* y *is* B", which is sometimes abbreviated as $A \to B$. In essence, the expression describes a relation between two variables x and y; this suggests that a fuzzy if-then rule be defined as a **binary fuzzy relation** R on the product space $X \times Y$. Note that a binary fuzzy relation R is an extension of the classical Cartesian product, where each element $(x,y) \in X \times Y$ is associated with a membership grade denoted by $\mu_R(x,y)$. Alternatively, a binary fuzzy relation R can be viewed as a fuzzy set with universe $X \times Y$, and this fuzzy set is characterized by a **two-dimensional MF** $\mu_R(x,y)$.

Generally speaking, there are two ways to interpret the fuzzy rule $A \to B$. If we interpret $A \to B$ as **A coupled with B**, then

$$R = A \to B = A \times B = \int_{X\times Y} \mu_A(x) \tilde{*} \mu_B(y)/(x,y),$$

where $\tilde{*}$ is a fuzzy AND (or more generally, T-norm) operator and $A \to B$ is used again to represent the fuzzy relation R. On the other hand, if $A \to B$ is interpreted as **A entails B**, then it can be written as four different formulas:

- Material implication: $R = A \to B = \neg A \cup B$.
- Propositional calculus: $R = A \to B = \neg A \cup (A \cap B)$.

- Extended propositional calculus: $R = A \rightarrow B = (\neg A \cap \neg B) \cup B$.
- Generalization of modus ponens: $\mu_R(x,y) = sup\{c \mid \mu_A(x) \tilde{*} c \leq \mu_B(y)\ and\ 0 \leq c \leq 1\}$, where $R = A \rightarrow B$ and $\tilde{*}$ is a T-norm operator.

Though these four formulas are different in appearance, they all reduce to the familiar identity $A \rightarrow B \equiv \neg A \cup B$ when A and B are propositions in the sense of two-valued logic. Figure 5 illustrates these two interpretations of a fuzzy rule $A \rightarrow B$. Here we shall adopt the first interpretation, where $A \rightarrow B$ implies **A coupled with B**. The treatment of the second interpretation can be found in [34], [49], [50].

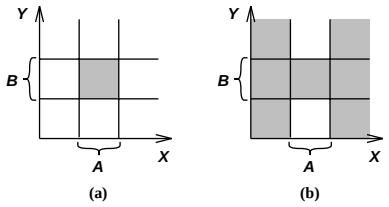


Fig. 5. *Two interpretations of fuzzy implication: (a) A coupled with B; (b) A entails B..*

C. Fuzzy Reasoning (Approximate Reasoning)

Fuzzy reasoning (also known as **approximate reasoning**) is an inference procedure used to derive conclusions from a set of fuzzy if-then rules and one or more conditions. Before introducing fuzzy reasoning, we shall discuss the **compositional rule of inference** [119], which is the essential rationale behind fuzzy reasoning.

The compositional rule of inference is a generalization of the following familiar notion. Suppose that we have a curve $y = f(x)$ that regulates the relation between x and y. When we are given $x = a$, then from $y = f(x)$ we can infer that $y = b = f(a)$; see Figure 6 (a). A generalization of the above process would allow a to be an interval and $f(x)$ to be an interval-valued function, as shown in Figure 6 (b). To find the resulting interval $y = b$ corresponding to the interval $x = a$, we first construct a cylindrical extension of a (that is, extend the domain of a from X to $X \times Y$) and then find its intersection I with the interval-valued curve. The projection of I onto the y-axis yields the interval $y = b$.

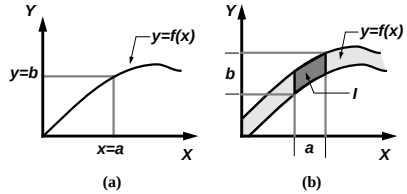


Fig. 6. *Derivation of $y = b$ from $x = a$ and $y = f(x)$: (a) a and b are points, $y = f(x)$ is a curve; (b) a and b are intervals, $y = f(x)$ is an interval-valued function.*

Going one step further in our generalization, we assume that A is a fuzzy set of X and F is a fuzzy relation on $X \times Y$, as shown in Figure 7 (a) and (b). To find the resulting fuzzy set B, again, we construct a cylindrical extension $c(A)$ with base A (that is, we expand the domain of A from X to $X \times Y$ to get $c(A)$). The intersection of $c(A)$ and F (Figure 7 (c)) forms the analog of the region of intersection I in Figure 6 (b). By projecting $c(A) \cap F$ onto the y-axis, we infer y as a fuzzy set B on the y-axis, as shown in Figure 7 (d).

Specifically, let μ_A, $\mu_{c(A)}$, μ_B, and μ_F be the MF's of A, $c(A)$, B, and F, respectively, where $\mu_{c(A)}$ is related to μ_A through

$$\mu_{c(A)}(x,y) = \mu_A(x).$$

Then

$$\begin{aligned} \mu_{c(A)\cap F}(x,y) &= \min[\mu_{c(A)}(x,y), \mu_F(x,y)] \\ &= \min[\mu_A(x), \mu_F(x,y)]. \end{aligned}$$

By projecting $c(A) \cap F$ onto the y-axis, we have

$$\begin{aligned} \mu_B(y) &= \max_x \min[\mu_A(x), \mu_F(x,y)] \\ &= \vee_x [\mu_A(x) \wedge \mu_F(x,y)]. \end{aligned}$$

This formula is referred to as **max-min composition** and B is represented as

$$B = A \circ F,$$

where $\circ$ denotes the composition operator. If we choose product for fuzzy AND and *max* for fuzzy OR, then we have **max-product composition** and $\mu_B(y)$ is equal $\vee_x [\mu_A(x)\mu_F(x,y)]$.

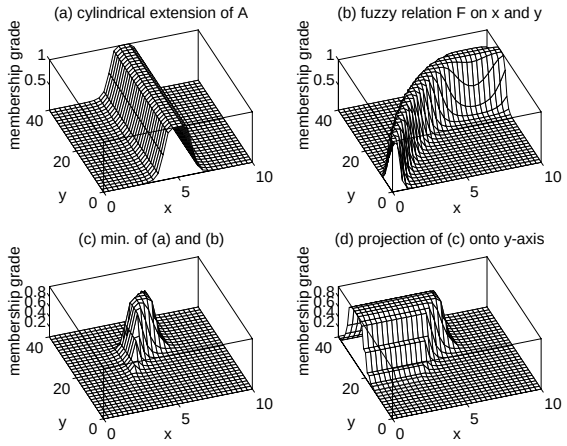


Fig. 7. *Compositional rule of inference.*

Using the compositional rule of inference, we can formalize an inference procedure, called fuzzy reasoning, upon a set of fuzzy if-then rules. The basic rule of inference in traditional two-valued logic is **modus ponens**, according to which we can infer the truth of a proposition B from the truth of A and the implication $A \rightarrow B$. For instance, if A is identified with "the tomato is red" and B with "the tomato is ripe," then if it is true that "the tomato is red," it is also true that "the tomato is ripe." This concept is illustrated below.

premise 1 (fact):	x is A,
premise 2 (rule):	if x is A then y is B,
consequence (conclusion):	y is B.

However, in much of human reasoning, modus ponens is employed in an approximate manner. For example, if we have the same implication rule "if the tomato is red then it is ripe" and we know that "the tomato is more or less red," then we may infer that "the tomato is more or less ripe." This is written as

premise 1 (fact):	x is A',
premise 2 (rule):	if x is A then y is B,
consequence (conclusion):	y is B',

where A' is close to A and B' is close to B. When A, B, A', and B' are fuzzy sets of appropriate universes, the above inference procedure is called fuzzy reasoning or approximate reasoning; it is also called **generalized modus ponens**, since it has modus ponens as a special case.

Using the composition rule of inference introduced earlier, we can formulate the inference procedure of fuzzy reasoning as the following definition.

Definition 11: Fuzzy Reasoning Based On Max-Min Composition.

Let A, A', and B be fuzzy sets of X, X, and Y, respectively. Assume that the fuzzy implication $A \rightarrow B$ is expressed as a fuzzy relation R on $X \times Y$. Then the fuzzy set B' induced by "*x is* A'" and the fuzzy rule "*if x is A then y is B*" is defined by

$$\begin{aligned} \mu_{B'}(y) &= \max_x \min[\mu_{A'}(x), \mu_R(x,y)] \\ &= \vee_x[\mu_{A'}(x) \wedge \mu_R(x,y)], \end{aligned} \tag{14}$$

or, equivalently,

$$B' = A' \circ R = A' \circ (A \rightarrow B). \tag{15}$$

□

Remember that equation (15) is a general expression for fuzzy reasoning, while equation (14) is an instance of fuzzy reasoning where *min* and *max* are the operators for fuzzy AND and OR, respectively.

Now we can use the inference procedure of the generalized modus ponens to derive conclusions, provided that the fuzzy implication $A \rightarrow B$ is defined as an appropriate binary fuzzy relation.

C.1 Single rule with single antecedent

For a single rule with a single antecedent, the formula is available in equation (14). A further simplification of the equation yields

$$\begin{aligned} \mu_{B'}(y) &= [\vee_x(\mu_{A'}(x) \wedge \mu_A(x)] \wedge \mu_B(y) \\ &= w \wedge \mu_B(y). \end{aligned}$$

In other words, first we find the degree of match w as the maximum of $\mu_{A'}(x) \wedge \mu_A(x)$ (the shaded area in the antecedent part of Figure 8); then the MF of the resulting B' is equal to the MF of B clipped by w, shown as the shaded area in the consequent part of Figure 8.

A fuzzy if-then rule with two antecedents is usually written as "*if x is A and y is B then z is C*." The corresponding problem for approximate reasoning is expressed as

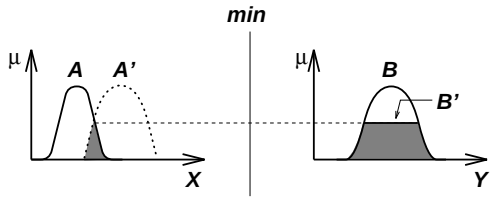


Fig. 8. *Fuzzy reasoning for a single rule with a single antecedent.*

premise 1 (fact):	x is A' and y is B',
premise 2 (rule):	if x is A and y is B then z is C,
consequence (conclusion):	z is C'.

The fuzzy rule in premise 2 above can be put into the simpler form "$A \times B \rightarrow C$." Intuitively, this fuzzy rule can be transformed into a ternary fuzzy relation R, which is specified by the following MF:

$$\mu_R(x,y,z) = \mu_{(A\times B)\times C}(x,y,z) = \mu_A(x) \wedge \mu_B(y) \wedge \mu_C(z).$$

And the resulting C' is expressed as

$$C' = (A' \times B') \circ (A \times B \rightarrow C).$$

Thus

$$\begin{aligned} \mu_{C'}(z) &= \vee_{x,y}[\mu_{A'}(x) \wedge \mu_{B'}(y)] \wedge [\mu_A(x) \wedge \mu_B(y) \wedge \mu_C(z)] \\ &= \vee_{x,y}\{[\mu_{A'}(x) \wedge \mu_{B'}(y) \wedge \mu_A(x) \wedge \mu_B(y)]\} \wedge \mu_C(z) \\ &= \{\underbrace{\vee_x[\mu_{A'}(x) \wedge \mu_A(x)]}_{w_1}\} \wedge \{\underbrace{\vee_y[\mu_{B'}(y) \wedge \mu_B(y)]}_{w_2}\} \wedge \mu_C(z) \\ &= \underbrace{w_1 \wedge w_2}_{\substack{\text{firing} \\ \text{strength}}} \wedge \mu_C(z), \end{aligned} \tag{16}$$

where w_1 is the degree of match between A and A'; w_2 is the degree of match between B and B'; and $w_1 \wedge w_2$ is called the **firing strength** or **degree of fulfillment** of this fuzzy rule. A graphic interpretation is shown in Figure 9, where the MF of the resulting C' is equal to the MF of C clipped by the firing strength w, $w = w_1 \wedge w_2$. The generalization to more than two antecedents is straightforward.

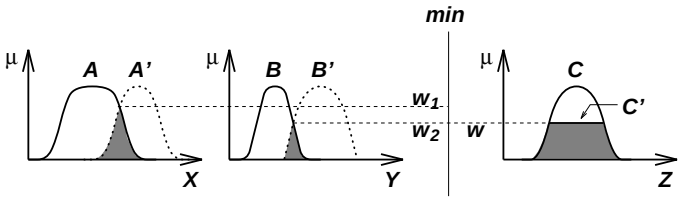


Fig. 9. *Approximate reasoning for multiple antecedents.*

C.2 Multiple rules with multiple antecedents

The interpretation of multiple rules is usually taken as the union of the fuzzy relations corresponding to the fuzzy rules. For instance, given the following fact and rules:

premise 1 (fact):	x is A' and y is B',
premise 2 (rule 1):	if x is A_1 and y is B_1 then z is C_1,
premise 3 (rule 2):	if x is A_2 and y is B_2 then z is C_2,
consequence (conclusion):	z is C',

we can employ the fuzzy reasoning shown in Figure 10 as an inference procedure to derive the resulting output fuzzy set C'.

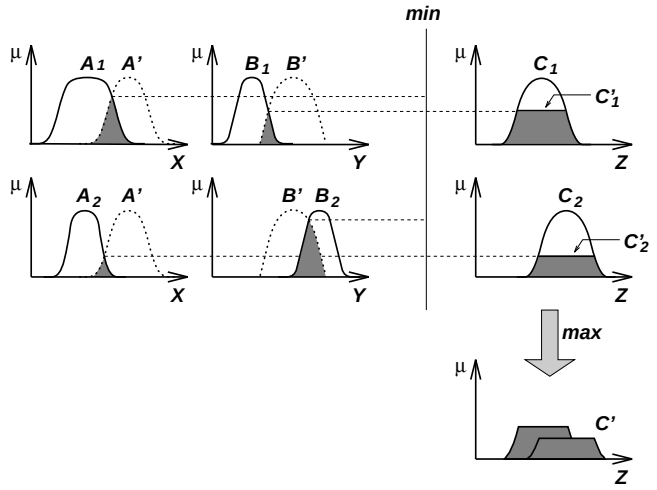


Fig. 10. *Fuzzy reasoning for multiple rules with multiple antecedents.*

To verify this inference procedure, let $R_1 = A_1 \times B_1 \rightarrow C_1$ and $R_2 = A_2 \times B_2 \rightarrow C_2$. Since the max-min composition operator $\circ$ is distributive over the $\cup$ operator, it follows that

$$\begin{aligned} C' &= (A' \times B') \circ (R_1 \cup R_2) \\ &= [(A' \times B') \circ R_1] \cup [(A' \times B') \circ R_2] \\ &= C'_1 \cup C'_2, \end{aligned} \tag{17}$$

where C'_1 and C'_2 are the inferred fuzzy sets for rule 1 and 2, respectively. Figure 10 shows graphically the operation of fuzzy reasoning for multiple rules with multiple antecedents.

When a given fuzzy rule assumes the form "if x is A or y is B then z is C," then firing strength is given as the maximum of degree of match on the antecedent part for a given condition. This fuzzy rule is equivalent to the union of the two fuzzy rules "if x is A then z is C" and "if y is B then z is C" if and only if the max-min composition is adopted.

D. Fuzzy Models (Fuzzy Inference Systems)

The **Fuzzy inference system** is a popular computing framework based on the concepts of fuzzy set theory, fuzzy if-then rules, and fuzzy reasoning. It has been successfully applied in fields such as automatic control, data classification, decision analysis, expert systems, and computer vision. Because of its multi-disciplinary nature, the fuzzy inference system is known by a number of names, such as **fuzzy-rule-based system**, **fuzzy expert system** [37], **fuzzy model** [98], [91], **fuzzy associative memory** [47], **fuzzy logic controller** [60], [49], [50], and simply (and ambiguously) **fuzzy system**.

The basic structure of a fuzzy inference system consists of three conceptual components: a **rule base**, which contains a selection of fuzzy rules, a **database** or **dictionary**, which defines the membership functions used in the fuzzy rules, and a **reasoning mechanism**, which performs the inference procedure (usually the fuzzy reasoning introduced earlier) upon the rules and a given condition to derive a reasonable output or conclusion.

Note that the basic fuzzy inference system can take either fuzzy inputs or crisp inputs (which can be viewed as fuzzy singletons that have zero membership grade everywhere except at certain points where the membership grades achieve unity), but the outputs it produces are almost always fuzzy sets. Often it is necessary to have a crisp output, especially in a situation where a fuzzy inference system is used as a controller. Therefore we need a **defuzzification strategy** to extract a crisp value that best summarize a fuzzy set. A fuzzy inference system with a crisp output is shown in Figure 11, where the dashed line indicates a basic fuzzy inference system with fuzzy output and the defuzzification block serves the purpose of transforming a fuzzy output into a crisp one. An example of a basic fuzzy inference system is the two-rule two-input system of Figure 10. The function of the defuzzification block will be explained at a later point.

With crisp inputs and outputs, a fuzzy inference system implements a nonlinear mapping from its input space to output space. This mapping is accomplished by a number of fuzzy if-then rules, each of which describes the local behavior of the mapping. In particular, the antecedent of each rule defines a fuzzy region of the input space, and the consequent specifies the corresponding outputs.

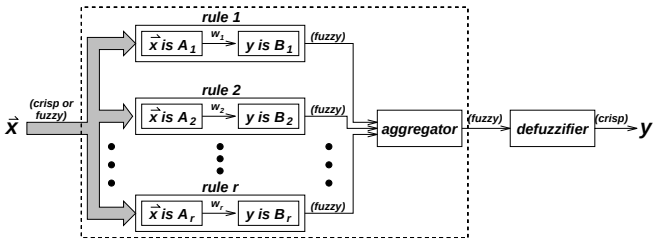


Fig. 11. *Block diagram for a fuzzy inference system.*

In what follows, we will first introduce three types of fuzzy inference systems that have been widely employed in various applications. The differences between these three fuzzy inference systems lie in the consequents of their fuzzy rules, and thus their aggregation and defuzzification procedures differ accordingly. Then we will introduce three ways of partitioning the input space for any type of fuzzy inference system. Last, we will address briefly the features and the problems of fuzzy modeling, which is concerned with the construction of a fuzzy inference system for modeling a specific target system.

D.1 Mamdani Fuzzy Model

The **Mamdani fuzzy model** [60] was proposed as the very first attempt to control a steam engine and boiler combination by a set of linguistic control rules obtained from experienced human operators. Figure 12 is an illustration of how a two-rule fuzzy inference system of the Mamdani type derives the overall output z when subjected to two crisp inputs x and y.

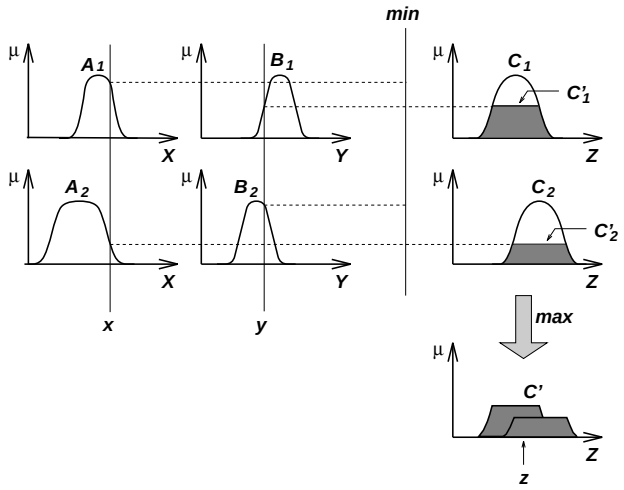


Fig. 12. *The Mamdani fuzzy inference system using min and max for fuzzy AND and OR operators, respectively.*

If we adopt product and *max* as our choice for the fuzzy AND and OR operators, respectively, and use max-product composition instead of the original max-min composition, then the resulting fuzzy reasoning is shown in Figure 13, where the inferred output of each rule is a fuzzy set scaled down by its firing strength via the algebraic product. Though this type of fuzzy reasoning was not employed in Mamdani's original paper, it has often been used in the literature. Other variations are possible if we have different choices of fuzzy AND (T-norm) and OR (T-conorm) operators.

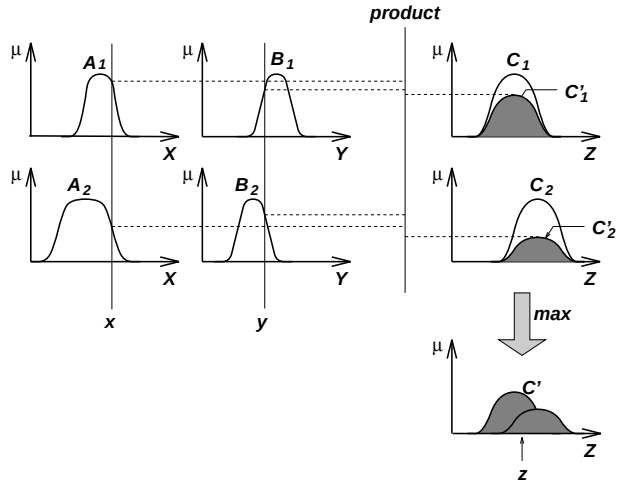


Fig. 13. *The Mamdani fuzzy inference system using product and max for fuzzy AND and OR operators, respectively.*

In Mamdani's application [60], two fuzzy inference systems were used as two controllers to generate the heat input to the boiler and throttle opening of the engine cylinder, respectively, in order to regulate the steam pressure in the boiler and the speed of the engine. Since the plant takes only crisp values as inputs, we have to use a defuzzifier to convert a fuzzy set to a crisp value. **Defuzzification** refers to the way a crisp value is extracted from a fuzzy set as a representative value. The most frequently used defuzzification strategy is the centroid of area, which is defined as

$$z_{COA} = \frac{\int_Z \mu_{C'}(z) z \, dz}{\int_Z \mu_{C'}(z) \, dz}, \tag{18}$$

where $\mu_{C'}(z)$ is the aggregated output MF. This formula is reminiscent of the calculation of expected values in probability distributions. Other defuzzification strategies arise for specific applications, which includes bisector of area, mean of maximum, largest of maximum, and smallest of maximum, and so on. Figure 14 demonstrate these defuzzification strategies. Generally speaking, these defuzzification methods are computation intensive and there is no rigorous way to analyze them except through experiment-based studies. Other more flexible defuzzification methods can be found in [73], [115], [80].

Both Figure 12 and 13 conform to the fuzzy reasoning defined previously. In practice, however, a fuzzy inference system may have certain reasoning mechanisms that do not follow the strict definition of the compositional rule of inference. For instance, one might use either *min* or product for computing firing strengths and/or qualified rule outputs. Another variation is to use pointwise summation (*sum*) instead of *max* in the standard fuzzy reasoning, though *sum* is not really a fuzzy OR operators. An advantage of this **sum-product composition** [47] is that the final crisp output via centroid defuzzification is equal to the weighted average of each rule's crisp output, where the weighting factor for a rule is equal to its firing strength multiplied by the area of the rule's output MF, and the crisp output of a rule is equal to the the centroid defuzzified value of its output MF. This reduces the computation burden if we can obtain the area and the centroid of each output MF in advance.

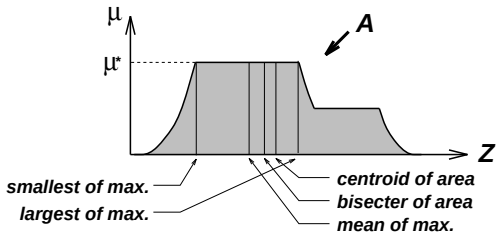


Fig. 14. *Various defuzzification schemes for obtaining a crisp output.*

D.2 Sugeno Fuzzy Model

The **Sugeno fuzzy model** (also known as the **TSK fuzzy model**) was proposed by Takagi, Sugeno, and

Kang [98], [91] in an effort to develop a systematic approach to generating fuzzy rules from a given input-output data set. A typical fuzzy rule in a Sugeno fuzzy model has the form

$$\textit{if } x \textit{ is } A \textit{ and } y \textit{ is } B \textit{ then } z = f(x,y),$$

where A and B are fuzzy sets in the antecedent, while $z = f(x,y)$ is a crisp function in the consequent. Usually $f(x,y)$ is a polynomial in the input variables x and y, but it can be any function as long as it can appropriately describe the output of the system within the fuzzy region specified by the antecedent of the rule. When $f(x,y)$ is a first-order polynomial, the resulting fuzzy inference system is called a **first-order Sugeno fuzzy model**, which was originally proposed in [98], [91]. When f is a constant, we then have a **zero-order Sugeno fuzzy model**, which can be viewed either as a special case of the Mamdani fuzzy inference system, in which each rule's consequent is specified by a fuzzy singleton (or a pre-defuzzified consequent), or a special case of the Tsukamoto fuzzy model (to be introduce later), in which each rule's consequent is specified by an MF of a step function crossing at the constant. Moreover, a zero-order Sugeno fuzzy model is functionally equivalent to a radial basis function network under certain minor constraints [32].

It should be pointed out that the output of a zero-order Sugeno model is a smooth function of its input variables as long as the neighboring MF's in the premise have enough overlap. In other words, the overlap of MF's in the consequent does not have a decisive effect on the smoothness of the interpolation; it is the overlap of the MF's in the premise that determines the smoothness of the resulting input-output behavior.

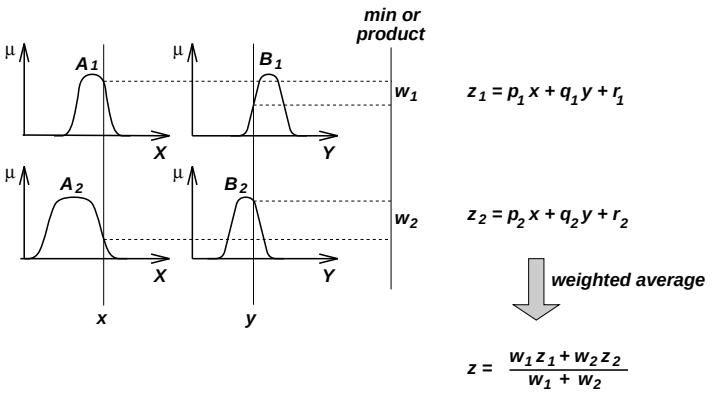


Fig. 15. *The Sugeno fuzzy model.*

Figure 15 shows the fuzzy reasoning procedure for a first-order Sugeno fuzzy model. Note that the aggregator and defuzzifier blocks in Figure 11 are replaced by the operation of **weighted average**, thus avoiding the time-consuming procedure of defuzzification. In practice, sometimes the weighted average operator is replaced with the **weighted sum** operator (that is, $z = w_1 z_1 + w_2 z_2$ in Figure 15) in order to further reduce computation load, especially in training a fuzzy inference system. However, this simplification could lead to the loss of MF linguistic meanings unless the sum of firing strengths (that is, $\sum_i w_i$) is close to unity.

D.3 Tsukamoto Fuzzy Model

In the **Tsukamoto fuzzy models** [101], the consequent of each fuzzy if-then rule is represented by a fuzzy set with a monotonical MF, as shown in Figure 16. As a result, the inferred output of each rule is defined as a crisp value induced by the rule's firing strength. The overall output is taken as the weighted average of each rule's output. Figure 16 illustrates the whole reasoning procedure for a two-input two-rule system.

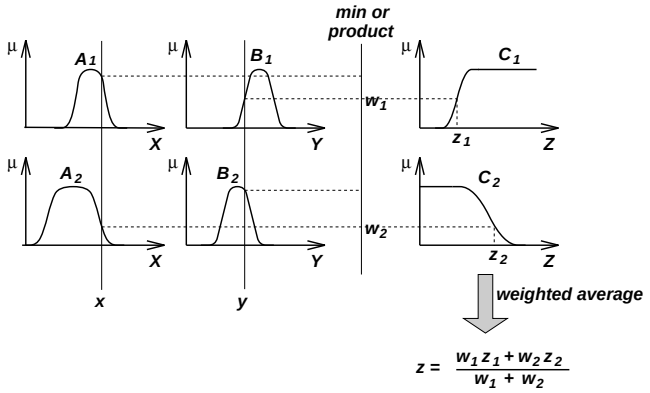


Fig. 16. *The Tsukamoto fuzzy model.*

Since each rule infers a crisp output, the Tsukamoto fuzzy model aggregates each rule's output by the method of weighted average and thus also avoids the time-consuming process of defuzzification.

D.4 Partition Styles for Fuzzy Models

By now it should be clear that the spirit of fuzzy inference systems resembles that of "divide and conquer" – the antecedents of fuzzy rules partition the input space into a number of local fuzzy regions, while the consequents describe the behavior within a given region via various constituents. The consequent constituent could be an output MF (Mamdani and Tsukamoto fuzzy models), a constant (zero-order Sugeno model), or a linear equation (first-order Sugeno model). Different consequent constituents result in different fuzzy inference systems, but their antecedents are always the same. Therefore the following discussion of methods of partitioning input spaces to form the antecedents of fuzzy rules is applicable to all three types of fuzzy inference systems.

- **Grid partition**: Figure 17 (a) illustrates a typical grid partition in a two-dimensional input space. This partition method is often chosen in designing a fuzzy controller, which usually involves only several state variables as the inputs to the controller. This partition strategy needs only a small number of MF's for each input. However, it encounters problems when we have a moderately large number of inputs. For instance, a fuzzy model with 10 inputs and two MF's on each input would result in $2^{10} = 1024$ fuzzy if-then rules, which is prohibitively large. This problem, usually

referred to as the **curse of dimensionality**, can be alleviated by the other partition strategies introduced below.

- **Tree partition**: Figure 17 (b) shows a typical tree partition, in which each region can be uniquely specified along a corresponding decision tree. The tree partition relieves the problem of an exponential increase in the number of rules. However, more MF's for each input are needed to define these fuzzy regions, and these MF's do not usually bear clear linguistic meanings such as "small," "big," and so on.
- **Scatter partition**: As shown in Figure 17 (c), by covering a subset of the whole input space that characterizes a region of possible occurrence of the input vectors, the scatter partition can also limit the number of rules to a reasonable amount.

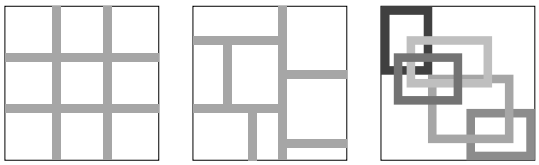


Fig. 17. *Various methods for partitioning the input space: (a) grid partition; (b) tree partition; (b) scatter partition.*

D.5 Neuro-Fuzzy Modeling

The process for constructing a fuzzy inference system is usually called **fuzzy modeling**, which has the following features:

- Due to the rule structure of a fuzzy inference system, it is easy to incorporate human expertise about the target system directly into the modeling process. Namely, fuzzy modeling takes advantage of **domain knowledge** that might not be easily or directly employed in other modeling approaches.
- When the input-output data of a system to be modeled is available, conventional system identification techniques can be used for fuzzy modeling. In other words, the use of **numerical data** also plays an important role in fuzzy modeling, just as in other mathematical modeling methods.

A common practice is to use domain knowledge for **structure determination** (that is, determine relevant inputs, number of MF's for each input, number of rules, types of fuzzy models, and so on) and numerical data for **parameter identification** (that is, identify the values of parameters that can generate best the performance). In particular, the term **neuro-fuzzy modeling** refers to the way of applying various learning techniques developed in the neural network literature to fuzzy inference systems. In the subsequent sections, we will apply the concept of the adaptive network, which is a generalization of the common back-propagation neural network, to tackle the parameter identification problem in a fuzzy inference system.

III. Adaptive Networks

This section describes the architectures and learning procedures of adaptive networks, which are a superset of all kinds of neural network paradigms with supervised learning capability. In particular, we shall address two of the most popular network paradigms adopted in the neural network literature: the back-propagation neural network (BPNN) and the radial basis function network (RBFN). Other network paradigms that can be interpreted as a set of fuzzy if-then rules are described in the next section.

A. Architecture

As the name implies, an **adaptive network** (Figure 18) is a network structure whose overall input-output behavior is determined by the values of a collection of modifiable parameters. More specifically, the configuration of an adaptive network is composed of a set of nodes connected through directed links, where each node is a process unit that performs a static **node function** on its incoming signals to generate a single **node output** and each link specifies the direction of signal flow from one node to another. Usually a node function is a parameterized function with modifiable parameters; by changing these parameters, we are actually changing the node function as well as the overall behavior of the adaptive network.

In the most general case, an adaptive network is heterogeneous and each node may have a different node function. Also remember that each link in an adaptive network are merely used to specify the propagation direction of a node's output; generally there are no weights or parameters associated with links. Figure 18 shows a typical adaptive network with two inputs and two outputs.

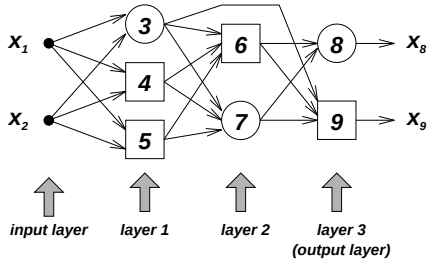


Fig. 18. *A feedforward adaptive network in layered representation.*

The parameters of an adaptive network are distributed into the network's nodes, so each node has a local parameter set. The union of these local parameter sets is the network's overall parameter set. If a node's parameter set is non-empty, then its node function depends on the parameter values; we use a square to represent this kind of **adaptive node**. On the other hand, if a node has an empty parameter set, then its function is fixed; we use a circle to denote this type of **fixed node**.

Adaptive networks are generally classified into two categories on the basis of the type of connections they have: **feedforward** and **recurrent** types. The adaptive network shown in Figure 18 is a feedforward network, since the output of each node propagates from the input side (left) to the

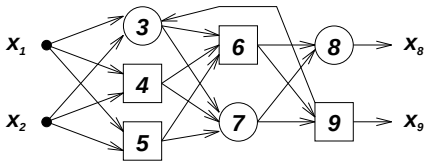


Fig. 19. *A recurrent adaptive network.*

output side (right) unanimously. If there is a feedback link that forms a circular path in a network, then the network is a recurrent network; Figure 19 is an example. (From the viewpoint of graph theory, a feedforward network is represented by an *acyclic* directed graph which contains no directed cycles, while a recurrent network always contains at least one directed cycle.)

In the **layered representation** of the feedforward adaptive network in Figure 18, there are no links between nodes in the same layer and outputs of nodes in a specific layer are always connected to nodes in succeeding layers. This representation is usually preferred because of its modularity, in that nodes in the same layer have the same functionality or generate the same level of abstraction about input vectors.

Another representation of feedforward networks is the **topological ordering representation**, which labels the nodes in an ordered sequence 1, 2, 3, ..., such that there are no links from node i to node j whenever $i \geq j$. Figure 20 is the topological ordering representation of the network in Figure 18. This representation is less modular than the layer representation, but it facilitates the formulation of the learning rule, as will be seen in the next section. (Note that the topological ordering representation is in fact a special case of the layered representation, with one node per layer.)

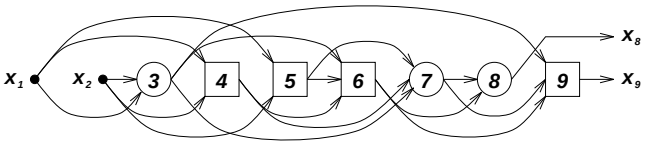


Fig. 20. *A feedforward adaptive network in topological ordering representation.*

Conceptually, a feedforward adaptive network is actually a static mapping between its input and output spaces; this mapping may be either a simple linear relationship or a highly nonlinear one, depending on the structure (node arrangement and connections, and so on) for the network and the function for each node. Here our aim is to construct a network for achieving a desired nonlinear mapping that is regulated by a data set consisting of a number of desired input-output pairs of a target system. This data set is usually called the **training data set** and the procedure we follow in adjusting the parameters to improve the performance of the network are often referred to as the **learning rule** or **learning algorithm** . Usually an adaptive network's performance is measured as the discrepancy between the desired output and the network's output under the same input conditions. This discrepancy is called the **error measure** and it can assume different forms for different applications. Generally speaking, a learning rule is derived by applying a specific optimization technique to a given error measure.

Before introducing a basic learning algorithm for adaptive networks, we shall present several examples of adaptive networks.

Example 3: An adaptive network with a single linear node.

Figure 21 is an adaptive network with a single node specified by

$$x_3 = f_3(x_1, x_2; a_1, a_2, a_3) = a_1 x_1 + a_2 x_2 + a_3,$$

where x_1 and x_2 are inputs and a_1, a_2, and a_3 are modifiable parameters. Obviously this function defines a plane in $x_1 - x_2 - x_3$ space, and by setting appropriate values for the parameters, we can place this plane arbitrarily. By adopting the squared error as the error measure for this network, we can identify the optimal parameters via the linear least-squares estimation method.

□

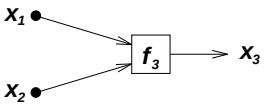


Fig. 21. *A linear single-node adaptive network.*

Example 4: A building block for the perceptron or the back-propagation neural network.

If we add another node to let the output of the adaptive network in Figure 21 have only two values 0 and 1, then the nonlinear network shown in Figure 22 is obtained. Specifically, the node outputs are expressed as

$$x_3 = f_3(x_1, x_2; a_1, a_2, a_3) = a_1 x_1 + a_2 x_2 + a_3,$$

and

$$x_4 = f_4(x_3) = \begin{cases} 1 & \text{if } x_3 \geq 0 \\ 0 & \text{if } x_3 < 0 \end{cases},$$

where f_3 is a linearly parameterized function and f_4 is a step function which maps x_3 to either 0 or 1. The overall function of this network can be viewed as a **linear classifier**: the first node forms a decision boundary as a straight line in $x_1 - x_2$ space, and the second node indicates which half plane the input vector (x_1, x_2) resides in. Obviously we can form an equivalent network with a single node whose function is the composition of f_3 and f_4; the resulting node is the building block of the classical **perceptron**.

Since the step function is discontinuous at one point and flat at all the other points, it is not suitable for learning procedures based on gradient descent. One way to get around this difficulty is to use the sigmoid function:

$$x_4 = f_4(x_3) = \frac{1}{1 + e^{-x_3}},$$

which is a continuous and differentiable approximation to the step function. The composition of f_3 and this differentiable f_4 is the building block for the back-propagation neural network in the following example.

□

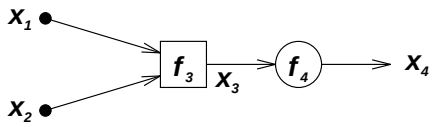


Fig. 22. *A nonlinear single-node adaptive network.*

Example 5: A back-propagation neural network.
Figure 23 is a typical architecture for a back-propagation neural network with three inputs, two outputs, and three hidden nodes that do not connect directly to either inputs or outputs. (The term back-propagation refers to the way the learning procedure is performed, that is, by propagating gradient information from the network's outputs to its inputs; details on this are to be introduced next.) Each node in a network of this kind has the same node function, which is the composition of a linear f_3 and a sigmoidal f_4 in example 4. For instance, the node function of node 7 in Figure 23 is

$$x_7 = \frac{1}{1 + exp[-(w_{4,7}x_4 + w_{5,7}x_5 + w_{6,7}x_6 + t_7)]},$$

where x_4, x_5, and x_6 are outputs from nodes 4, 5, and 6, respectively, and $\{w_{4,7}, w_{5,7}, w_{6,7}, t_7\}$ is the parameter set. Usually we view $w_{i,j}$ as the **weight** associated with the link connecting node i and j and t_j as the **threshold** associated with node j. However, it should be noted that this weight-link association is only valid in this type of network. In general, a link only indicates the signal flow direction and the causal relationship between connected nodes, as will be shown in other types of adaptive networks in the subsequent development. A more detailed discussion about the structure and learning rules of the artificial neural network will be presented later.

□

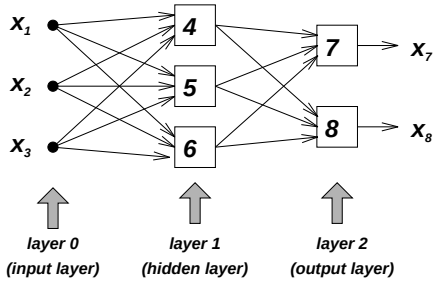


Fig. 23. *A 3-3-2 neural network.*

B. Back-Propagation Learning Rule for Feedforward Networks

The central part of a learning rule for an adaptive network concerns how to recursively obtain a gradient vector in which each element is defined as the derivative of an error measure with respect to a parameter. This is done by means of the chain rule, and the method is generally referred to as the **back-propagation learning rule** because the gradient vector is calculated in the direction opposite to the flow of the output of each node. Details follow below.

Suppose that a given feedforward adaptive network in the layered representation has L layers and layer l ($l = 0, 1, \ldots, L$; $l = 0$ represents the input layer) has $N(l)$ nodes. Then the output and function of node i ($i = 1, \ldots, N(l)$) of layer l can be represented as $x_{l,i}$ and $f_{l,i}$, respectively, as shown in Figure 24 (a). Without loss of generality, we assume there are no jumping links, that is, links connecting non-consecutive layers. Since the output of a node depends on the incoming signals and the parameter set of the node, we have the following general expression for the node function $f_{l,i}$:

$$x_{l,i} = f_{l,i}(x_{l-1,1}, \ldots x_{l-1,N(l-1)}, \alpha, \beta, \gamma, \ldots), \qquad (19)$$

where α, β, γ, etc. are the parameters pertaining to this node.

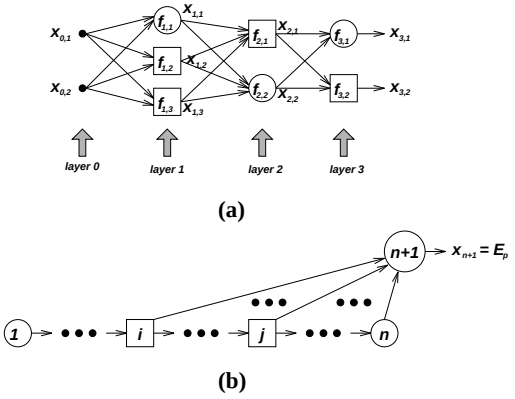


Fig. 24. *Our notational conventions: (a) layered representation; (b) topological ordering representation.*

Assuming the given training data set has P entries, we can define an **error measure** for the p-th ($1 \le p \le P$) entry of the training data as the sum of squared errors:

$$E_p = \sum_{k=1}^{N(L)} (d_k - x_{L,k})^2, \qquad (20)$$

where d_k is the k-th component of the p-th desired output vector and $x_{L,k}$ is the k-th component of the actual output vector produced by presenting the p-th input vector to the network. (For notational simplicity, we omit the subscript p for both d_k and $x_{L,k}$.) Obviously, when E_p is equal to zero, the network is able to reproduce exactly the desired output vector in the p-th training data pair. Thus our task here is to minimize an overall error measure, which is defined as $E = \sum_{p=1}^{P} E_p$.

Remember that the definition of E_p in equation (20) is not universal; other definitions of E_p are possible for specific situations or applications. Therefore we shall avoid using an explicit expression for the error measure E_p in order to emphasize the generality. In addition, we assume that E_p depends on the output nodes only; more general situations will be discussed below.

To use the gradient method to minimize the error measure, first we have to obtain the gradient vector. Before calculating the gradient vector, we should observe that

$$\boxed{\begin{array}{c}\text{change in}\\ \text{parameter } \alpha\end{array}} \Rightarrow \boxed{\begin{array}{c}\text{change in the output}\\ \text{of node containing } \alpha\end{array}} \Rightarrow \boxed{\begin{array}{c}\text{change in the output}\\ \text{of the final layer}\end{array}} \Rightarrow \boxed{\begin{array}{c}\text{change in the}\\ \text{error measure}\end{array}}$$

where the arrows $\Rightarrow$ indicate causal relationships. In other words, a small change in a parameter α will affect the output of the node containing α; this in turn will affect the output of the final layer and thus the error measure. Therefore the basic concept in calculating the gradient vector of the parameters is to pass a form of derivative information starting from the output layer and going backward layer by layer until the input layer is reached.

To facilitate the discussion, we define the **error signal** $\epsilon_{l,i}$ as the derivative of the error measure E_p with respect to the output of node i in layer l, taking both direct and indirect paths into consideration. In symbols,

$$\epsilon_{l,i} = \frac{\partial^+ E_p}{\partial x_{l,i}}. \tag{21}$$

This expression was called the **ordered derivative** by Werbos [109]. The difference between the ordered derivative and the ordinary partial derivative lies in the way we view the function to be differentiated. For an internal node output $x_{l,i}$ (where $l \neq L$), the partial derivative $\frac{\partial E_p}{\partial x_{l,i}}$ is equal to zero, since E_p does not depend on $x_{l,i}$ directly. However, it is obvious that E_p does depend on $x_{l,i}$ indirectly, since a change in $x_{l,i}$ will propagate through indirect paths to the output layer and thus produce a corresponding change in the value of E_p. Therefore $\epsilon_{l,i}$ can be viewed as the ratio of these two changes when they are made infinitesimal. The following example demonstrates the difference between the ordered derivative and the ordinary partial derivative.

Example 6: Ordered derivatives and ordinary partial derivatives

Consider the simple adaptive network shown in Figure 25, where z is a function of x and y, and y is in turn a function of x:

$$\begin{cases} y &= f(x), \\ z &= g(x,y). \end{cases}$$

For the ordinary partial derivative $\frac{\partial z}{\partial x}$, we assume that all the other input variables (in this case, y) are constant:

$$\frac{\partial z}{\partial x} = \frac{\partial g(x,y)}{\partial x}.$$

In other words, we assume the direct inputs x and y are independent, without paying attention to the fact that y is actually a function of x. For the ordered derivative, we take this indirect causal relationship into consideration:

$$\begin{aligned} \frac{\partial^+ z}{\partial x} &= \frac{\partial g(x, f(x))}{\partial x} \\ &= \left.\frac{\partial g(x,y)}{\partial x}\right|_{y=f(x)} + \left.\frac{\partial g(x,y)}{\partial y}\right|_{y=f(x)} \cdot \frac{\partial f(x)}{\partial x}. \end{aligned}$$

Therefore the ordered derivative takes into consideration both the direct and indirect paths that lead to the causal relationship.

□

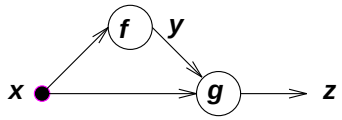


Fig. 25. *Ordered derivatives and ordinary partial derivatives (see text for details).*

The error signal for the i-th output node (at layer L) can be calculated directly:

$$\epsilon_{L,i} = \frac{\partial^+ E_p}{\partial x_{L,i}} = \frac{\partial E_p}{\partial x_{L,i}}. \tag{22}$$

This is equal to $\epsilon_{L,i} = -2(d_i - x_{L,i})$ if E_p is defined as in equation (20). For the internal (non-output) node at the i-th position of layer l, the error signal can be derived by the chain rule:

$$\begin{aligned} \epsilon_{l,i} &= \underbrace{\frac{\partial^+ E_p}{\partial x_{l,i}}}_{\substack{\text{error signal}\\ \text{at layer } l}} = \sum_{m=1}^{N(l+1)} \underbrace{\frac{\partial^+ E_p}{\partial x_{l+1,m}}}_{\substack{\text{error signal}\\ \text{at layer } l+1}} \times \frac{\partial f_{l+1,m}}{\partial x_{l,i}} \\ &= \sum_{m=1}^{N(l+1)} \epsilon_{l+1,m} \frac{\partial f_{l+1,m}}{\partial x_{l,i}}, \end{aligned} \tag{23}$$

where $0 \leq l \leq L-1$. That is, the error signal of an internal node at layer l can be expressed as a linear combination of the error signal of the nodes at layer $l+1$. Therefore for any l and i ($0 \leq l \leq L$ and $1 \leq i \leq N(l)$), we can find $\epsilon_{l,i} = \frac{\partial^+ E_p}{\partial x_{l,i}}$ by first applying equation (22) once to get error signals at the output layer, and then applying equation (23) iteratively until we reach the desired layer l. Since the error signals are obtained sequentially from the output layer back to the input layer, this learning paradigm is called the **back-propagation** learning rule by Rumelhart, Hinton and Williams [79].

The gradient vector is defined as the derivative of the error measure with respect to each parameter, so we have to apply the chain rule again to find the gradient vector. If α is a parameter of the i-th node at layer l, we have

$$\frac{\partial^+ E_p}{\partial \alpha} = \frac{\partial^+ E_p}{\partial x_{l,i}} \frac{\partial f_{l,i}}{\partial \alpha} = \epsilon_{l,i} \frac{\partial f_{l,i}}{\partial \alpha}. \tag{24}$$

Note that if we allow the parameter α to be shared between different nodes, then equation (24) should be changed to a more general form:

$$\frac{\partial^+ E_p}{\partial \alpha} = \sum_{x^* \in S} \frac{\partial^+ E_p}{\partial x^*} \frac{\partial f^*}{\partial \alpha}, \tag{25}$$

where S is the set of nodes containing α as a parameter and f^* is the node function for calculating x^*.

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

$$\frac{\partial^+ E}{\partial \alpha} = \sum_{p=1}^{P} \frac{\partial^+ E_p}{\partial \alpha}. \tag{26}$$

Accordingly, the update formula for the generic parameter α is

$$\triangle \alpha = -\eta \frac{\partial^+ E}{\partial \alpha}, \tag{27}$$

in which η is the **learning rate**, which can be further expressed as

$$\eta = \frac{\kappa}{\sqrt{\sum_\alpha (\frac{\partial E}{\partial \alpha})^2}}, \tag{28}$$

where κ is the **step size**, the length of each transition along the gradient direction in the parameter space. Usually we can change the step size to vary the speed of convergence; two heuristic rules for updating the value of κ are described in [29].

When an n-node feedforward network is represented in its topological order, we can envision the error measure E_p as the output of an additional node with index $n+1$, whose node function f_{n+1} can be defined on the outputs of any nodes with smaller indices; see Figure 24 (b). (Therefore E_p may depend directly on any internal nodes.) Applying the chain rule again, we have the following concise formula for calculating the error signal $\epsilon_i = \partial E_p / \partial x_i$:

$$\frac{\partial^+ E_p}{\partial x_i} = \frac{\partial f_{n+1}}{\partial x_i} + \sum_{i<j\leq n} \frac{\partial^+ E_p}{\partial x_j} \frac{\partial f_j}{\partial x_i}, \tag{29}$$

or

$$\epsilon_i = \frac{\partial f_{n+1}}{\partial x_i} + \sum_{i<j\leq n} \epsilon_j \frac{\partial f_j}{\partial x_i}, \tag{30}$$

where the first term shows the direct effect of x_i on E_p via the direct path from node i to node $n+1$ and each product term in the summation indicates the indirect effect of x_i on E_p. Once we find the error signal for each node, then the gradient vector for the parameters is derived as before.

Another simple and systematic way to calculate the error signals is through the representation of the **error-propagation network** (or **sensitivity model**), which is obtained from the original adaptive network by reversing the links and supplying the error signals at the output layer as inputs. The following example illustrates this idea.

Example 7: Adaptive network and its error-propagation model

Figure 26 (a) is an adaptive network, where each node is indexed by a unique number. Again, we use f_i and x_i to denote the function and output of node i. In order to calculate the error signals at internal nodes, an error-propagation network is constructed in Figure 26 (b), where the output of node i is the error signal of this node in the original adaptive network. In symbols, if we choose the squared error measure for E_p, then we have the following:

$$\epsilon_9 = \frac{\partial^+ E_p}{\partial x_9} = \frac{\partial E_p}{\partial x_9} = -2\,(d_9 - x_9),$$

$$\epsilon_8 = \frac{\partial^+ E_p}{\partial x_8} = \frac{\partial E_p}{\partial x_8} = -2\,(d_8 - x_8),$$

(Thus nodes 9 and 8 in the error-propagation network are only buffer nodes.)

$$\epsilon_7 = \frac{\partial^+ E_p}{\partial x_7} = \frac{\partial^+ E_p}{\partial x_8}\frac{\partial f_8}{\partial x_7} + \frac{\partial^+ E_p}{\partial x_9}\frac{\partial f_9}{\partial x_7} = \epsilon_8 \frac{\partial f_8}{\partial x_7} + \epsilon_9 \frac{\partial f_9}{\partial x_7},$$

$$\epsilon_6 = \frac{\partial^+ E_p}{\partial x_6} = \frac{\partial^+ E_p}{\partial x_8}\frac{\partial f_8}{\partial x_6} + \frac{\partial^+ E_p}{\partial x_9}\frac{\partial f_9}{\partial x_6} = \epsilon_8 \frac{\partial f_8}{\partial x_6} + \epsilon_9 \frac{\partial f_9}{\partial x_6}.$$

Similar expressions can be written for the error signals of node 1, 2, 3, 4 and 5. It is interesting to observe that in the error-propagation net, if we associate each link connecting nodes i and j ($i < j$) with a weight $w_{ij} = \frac{\partial f_j}{\partial x_i}$, then each node performs a linear function and the error-propagation net is actually a linear network. The error-propagation network is helpful in correctly formulating the expressions for error signals. The same concept applies to recurrent networks with either synchronous or continuous operations [34].

□

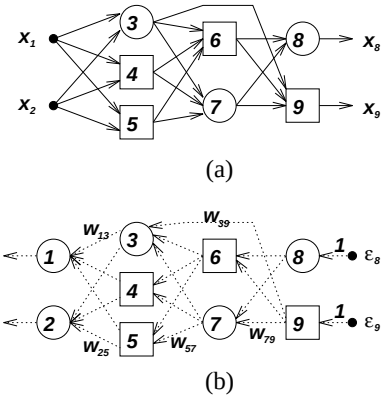


Fig. 26. *(a) An adaptive network and (b) its error-propagation model.*

Depending on the applications we are interested in, two types of learning paradigms for adaptive networks are available to suit our needs. In **off-line learning** (or **batch learning**), the update formula for parameter α is based on equation (26) and the update action takes place only after the whole training data set has been presented, that is, only after each **epoch** or **sweep**. On the other hand, in **on-line learning** (or **pattern learning**), the parameters are updated immediately after each input-output pair has been presented, and the update formula is based on equation (24). In practice, it is possible to combine these two learning modes and update the parameter after k training data entries have been presented, where k is between 1 and P and it is sometimes referred to as the **epoch size**.

C. Back-Propagation Learning Rule for Recurrent Networks

For recurrent adaptive networks, the back-propagation learning rule is still applicable if we can transform the network configurations to be of the feedforward type. To simplify our notation, we shall use the network in Figure 27 for

our discussion, where x_1 and x_2 are inputs and x_5 and x_6 are outputs. Because it has directional loops 3-4-5, 3-4-6-5, and 6 (a self loop), this is a typical recurrent network with node functions denoted as follows:

$$\begin{cases} x_3 &= f_3(x_1, x_5) \\ x_4 &= f_4(x_2, x_3) \\ x_5 &= f_5(x_4, x_6) \\ x_6 &= f_6(x_4, x_6) \end{cases} \tag{31}$$

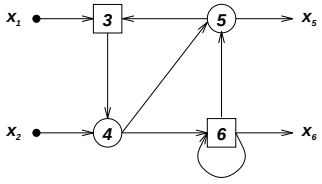


Fig. 27. *A simple recurrent network.*

In order to correctly derive the back-propagation learning rule for the recurrent net in Figure 27, we have to distinguish two operating modes through which the network may satisfy equation (31). These two modes are **synchronous operation** and **continuous operation**.

For continuously operated networks, all nodes continuously change their outputs until equation (31) is satisfied. This operating mode is of particular interest for analog circuit implementations, where a certain kind of dynamical evolution rule is imposed on the network. For instance, the dynamical formula for node 3 can be written as

$$\tau_3 \frac{dx_3}{dt} + x_3 = f_3(x_1, x_5). \tag{32}$$

Similar formulas can be devised for other nodes. It is obvious that when $x_3(t)$ stops changing (i.e., $\frac{dx_3}{dt} = 0$), equation (32) leads to the correct fixed points satisfying equation (31). However, this kind of recurrent networks do pose some problems in software simulation, as the stable fixed point satisfying equation (31) may be hard to find. Here we shall not go into details about continuously operated networks. A detailed treatment of continuously operated networks which use the Mason gain formula [62] as a learning rule can be found in [34].

On the other hand, if a network is operated synchronously, all nodes change their outputs simultaneously according to a global clock signal and there is a time delay associated with each link. This synchronization is reflected by adding the time t as an argument to the output of each node in equation (31) (assuming there is a unit time delay associated with each link):

$$\begin{cases} x_3(t+1) &= f_3(x_1(t), x_5(t)) \\ x_4(t+1) &= f_4(x_2(t), x_3(t)) \\ x_5(t+1) &= f_5(x_4(t), x_6(t)) \\ x_6(t+1) &= f_6(x_4(t), x_6(t)) \end{cases} \tag{33}$$

C.1 Back-Propagation Through Time (BPTT)

When using synchronously operated networks, we usually are interested in identifying a set of parameters that will make the output of a node (or several nodes) follow a given trajectory (or trajectories) in a discrete time domain. This problem of **tracking** or **trajectory following** is usually solved by using a method called **unfolding of time** to transform a recurrent network into a feedforward one, as long as the time t does not exceed a reasonable maximum T. This idea was originally introduced by Minsky and Papert [64] and combined with back-propagation by Rumelhart, Hinton, and Williams [79]. Consider the recurrent net in Figure 27, which is redrawn in Figure 28 (a) with the same configuration except that the input variables x_1 and x_2 are omitted for simplicity. The same network in a feedforward architecture is shown in Figure 28 (b) with the time index t running from 1 to 4. In other words, for a recurrent net that synchronously evaluates each of its node functions from $t = 1, 2, ..., T$, we can simply duplicate all units T times and arrange the resulting network in a layered feedforward manner.

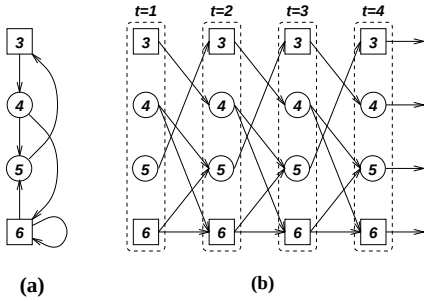


Fig. 28. *(a) A synchronously operated recurrent network and (b) its feedforward equivalent obtained via unfolding of time.*

It is obvious that the two networks in Figure 28 (a) and (b) will behave identically for $t = 1$ to T, provided that all copies of each parameter at different time steps remain identical. For instance, the parameter in node 3 of Figure 28 (a) must be the same at all time instants. This is the problem of **parameter sharing**; a quick solution is to move the parameters from node 3 and 6 into the so-called **parameter nodes**, which are independent of the time step, as shown in Figure 29. (Without loss of generality, we assume nodes 3 and 6 both have only one parameter, denoted by a and b, respectively.) After setting up the parameter nodes in this way, we can apply the back-propagation learning rule as usual to the network in Figure 29 (which is still feedforward in nature) without the slightest concern about the parameter sharing constraint. Note that the error signals of parameter nodes come from the error signals of nodes located at layers of different time instants; thus the BP for this kind of unfolded network is often called **back-propagation through time** (**BPTT**).

C.2 Real Time Recurrent Learning (RTRL)

BPTT generally works well for most problems; the only complication is that it requires extensive computing resources when the sequence length T is large, because the duplication of nodes makes both memory requirements and simulation time proportional to T. Therefore for long sequences or sequences of unknown length, **real time recur-**

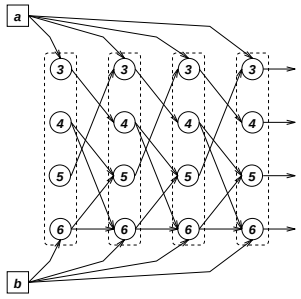


Fig. 29. *An alternative representation of Figure 28 (b) that automatically satisfies the parameter-sharing requirement.*

rent learning (RTRL) [114] is employed instead to perform on-line learning, that is, to update parameters while the network is running rather than at the end of the presented sequences.

To explain the rationale behind the RTRL algorithm, we take as an example the simple recurrent network in Figure 30 (a), where there is only one node with one parameter a. After moving the parameter out of the unfolded architecture, we obtain the feedforward network shown in Figure 30 (b). Figure 30 (c) is the corresponding error-propagation network. Here we assume $E = \sum_i^T E_i = \sum_i^T (d_i - x_i)^2$, where i is the index for time and d_i and x_i are the desired and the actual node output, respectively, at time instant i.

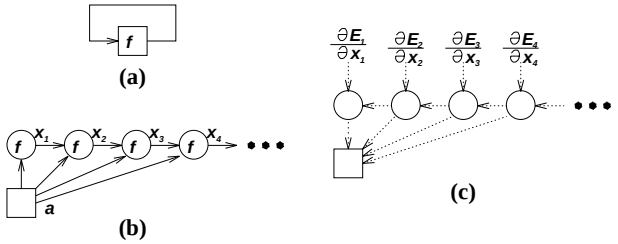


Fig. 30. *A simple recurrent adaptive network to illustrate RTRL: (a) a recurrent net with single node and single parameter; (b) unfolding-of-time architecture; (c) error-propagation network.*

To save computation and memory requirements, a sensible choice is to minimize E_i at each time step instead of trying to minimize E at the end of a sequences. To achieve this, we need to calculate $\partial^+ E/\partial a$ recursively at each time step i. For $i = 1$, the error-propagation network is as shown in Figure 31 (a) and we have

$$\frac{\partial^+ x_1}{\partial a} = \frac{\partial x_1}{\partial a} \text{ and } \frac{\partial^+ E_1}{\partial a} = \frac{\partial E_1}{\partial x_1}\frac{\partial^+ x_1}{\partial a}. \tag{34}$$

For $i = 2$, the error-propagation network is as shown in Figure 31 (b) and we have

$$\frac{\partial^+ x_2}{\partial a} = \frac{\partial x_2}{\partial a} + \frac{\partial x_2}{\partial x_1}\frac{\partial^+ x_1}{\partial a} \text{ and } \frac{\partial^+ E_2}{\partial a} = \frac{\partial E_2}{\partial x_2}\frac{\partial^+ x_2}{\partial a}. \tag{35}$$

For $i = 3$, the error-propagation network is as shown in Figure 31 (c) and we have

$$\frac{\partial^+ x_3}{\partial a} = \frac{\partial x_3}{\partial a} + \frac{\partial x_3}{\partial x_2}\frac{\partial^+ x_2}{\partial a} \text{ and } \frac{\partial^+ E_3}{\partial a} = \frac{\partial E_3}{\partial x_3}\frac{\partial^+ x_3}{\partial a}. \tag{36}$$

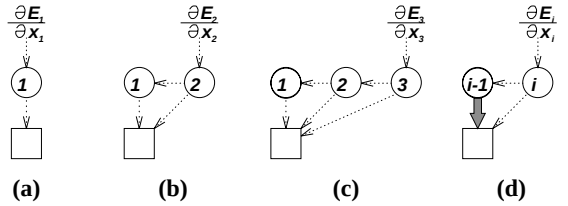


Fig. 31. *Error-propagation networks at different time steps: (a) $i = 1$; (b) $i = 2$; (c) $i = 3$; (d) a general situation, where the thick arrow represents $\frac{\partial^+ x_{i-1}}{\partial a}$.*

In general, for the error-propagation at time instant i, we have

$$\frac{\partial^+ x_i}{\partial a} = \frac{\partial x_i}{\partial a} + \frac{\partial x_i}{\partial x_{i-1}}\frac{\partial^+ x_{i-1}}{\partial a} \text{ and } \frac{\partial^+ E_i}{\partial a} = \frac{\partial E_i}{\partial x_i}\frac{\partial^+ x_3}{\partial a}, \tag{37}$$

where $\frac{\partial^+ x_{i-1}}{\partial a}$ is already available from the calculation at the previous time instant. Figure 31 shows this general situation, where the thick arrow represents $\frac{\partial^+ x_{i-1}}{\partial a}$, which is already available at the time instant $i-1$.

Therefore, by trying to minimize each individual E_i, we can recursively find the gradient $\frac{\partial^+ E_i}{\partial a}$ at each time instant; there is no need to wait until the end of the presented sequence. Since this is an approximation of the original BPTT, the learning rate η in the update formula

$$\Delta a = -\eta \frac{\partial^+ E_i}{\partial a}$$

should be kept small and, as a result, the learning process usually takes longer.

D. Hybrid Learning Rule: Combining BP and LSE

It is observed that if an adaptive network's output (assuming only one) or its transformation is linear in some of the network's parameters, then we can identify these linear parameters by the well-known linear least-squares method. This observation leads to a hybrid learning rule [24], [29] which combines the gradient method and the least-squares estimator (LSE) for fast identification of parameters.

D.1 Off-Line Learning (Batch Learning)

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

$$output = F(\vec{I}, S), \tag{38}$$

where $\vec{I}$ is the vector 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

$$S = S_1 \oplus S_2, \tag{39}$$

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

(38), we have

$$H(output) = H \circ F(\vec{I}, S), \tag{40}$$

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 (40) and obtain a matrix equation:

$$A\boldsymbol{\theta} = B \tag{41}$$

where $\boldsymbol{\theta}$ is an unknown vector whose elements are parameters in S_2. This equation represents the standard linear least-squares problem and the best solution for $\boldsymbol{\theta}$, which minimizes $\|A\boldsymbol{\theta} - B\|^2$, is the least-squares estimator (LSE) $\boldsymbol{\theta}^*$:

$$\boldsymbol{\theta}^* = (A^T A)^{-1} A^T B, \tag{42}$$

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. Of course, we can also employ the recursive LSE formula [23], [1], [58]. Specifically, let the i-th row vector of matrix A defined in equation (41) be a_i^T and the i-th element of B be b_i^T; then $\boldsymbol{\theta}$ can be calculated iteratively as follows:

$$\left.\begin{array}{rcl} \boldsymbol{\theta}_{i+1} &=& \boldsymbol{\theta}_i + S_{i+1} a_{i+1} (b_{i+1}^T - a_{i+1}^T \boldsymbol{\theta}_i) \\ S_{i+1} &=& S_i - \frac{S_i a_{i+1} a_{i+1}^T S_i}{1 + a_{i+1}^T S_i a_{i+1}}, \quad i = 0, 1, \cdots, P-1 \end{array}\right\}, \tag{43}$$

where the least-squares estimator $\boldsymbol{\theta}^*$ is equal to $\boldsymbol{\theta}_P$. The initial conditions needed to bootstrap equation (43) are $\boldsymbol{\theta}_0 = 0$ and $S_0 = \gamma I$, where γ is a positive large number and I is the identity matrix of dimension $M \times M$. When we are dealing with multi-output adaptive networks (*output* in equation (38) is a column vector), equation (43) still applies except that b_i^T is the i-th row of matrix B.

Now we can combine the gradient method and the least-squares estimator to update the parameters in an adaptive network. For hybrid learning to be applied in a batch mode, each epoch is composed of a forward pass and a backward pass. In the forward pass, after an input vector is presented, we calculate the node outputs in the network layer by layer until a corresponding row in the matrices A and B in equation (41) are obtained. This process is repreated for all the training data entries to form the complete A and B; then parameters in S_2 are identified by either the pseudo-inverse formula in equation (42) or the recursive least-squares formulas in equation (43). After the parameters in S_2 are identified, we can compute the error measure for each training data entry. In the backward pass, the error signals (the derivative of the error measure w.r.t. each node output, see equations (22) and (23)) propagate from the output end toward the input end; the gradient vector is accumulated for each training data entry. At the end of the backward pass for all training data, the parameters in S_1 are updated by the gradient method in equation (27).

For given fixed values of the 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 because of 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 substantially reduce the time needed to reach convergence.

It should be kept 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. In practice, this usually will not cause a problem as long as $H(\cdot)$ is monotonically increasing and the training data are not too noisy. A more detailed treatment of this transformation method can be found in [34].

D.2 On-Line Learning (Pattern Learning)

If the parameters are updated after each data presentation, we have a **on-line learning** or **pattern learning** scheme. This learning strategy is vital to on-line parameter identification for systems with changing characteristics. To modify the batch learning rule to obtain an on-line version, it is obvious that the gradient descent should be based on E_p (see equation (24)) instead of E. Strictly speaking, this is not a truly gradient search procedure for minimizing E, yet it will approximate one if the learning rate is small.

For the recursive least-squares formula to account for the time-varying characteristics of the incoming data, the effects of old data pairs must decay 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 [20]. 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 recursive formula:

$$\left.\begin{array}{rcl} \boldsymbol{\theta}_{i+1} &=& \boldsymbol{\theta}_i + S_{i+1} a_{i+1} (b_{i+1}^T - a_{i+1}^T \boldsymbol{\theta}_i) \\ S_{i+1} &=& \frac{1}{\lambda}[S_i - \frac{S_i a_{i+1} a_{i+1}^T S_i}{\lambda + a_{i+1}^T S_i a_{i+1}}] \end{array}\right\}, \tag{44}$$

where the typical value of λ in practice is between 0.9 and 1. The smaller λ is, the faster the effects of old data decay. A small λ sometimes causes numerical instability, however, and thus should be avoided. For a complete discussion and derivation of equation (44), the reader is referred to [34], [58], [20].

D.3 Different Ways of Combining GD and LSE

The computational complexity of the least-squares estimator (LSE) is usually higher than that of the gradient descent (GD) method for one-step adaptation. However, for achieving a prescribed performance level, the LSE is usually much faster. Consequently, depending on the available computing resources and required level of performance, we can choose from among at least five types of hybrid learning rules combining GD and LSE in different degrees, as follows.

1. One pass of LSE only: Nonlinear parameters are fixed while linear parameters are identified by one-time application of LSE.
2. GD only: All parameters are updated by GD iteratively.

3. One pass of LSE followed by GD: LSE is employed only once at the very beginning to obtain the initial values of linear parameters and then GD takes over to update all parameters iteratively.
4. GD and LSE: This is the proposed hybrid learning rule, where each iteration (epoch) of GD used to update the nonlinear parameters is followed by LSE to identify the linear parameters.
5. Sequential (approximate) LSE only: The outputs of an adaptive network are linearized with respect to its parameters, and then the extended Kalman filter algorithm [21] is employed to update all parameters. This method has been proposed in the neural network literature [85], [84], [83].

The choice of one of the above methods should be based on a trade-off between computational complexity and performance. Moreover, the whole concept of fitting data to parameterized models is called **regression** in statistics literature, and there are a number of other techniques for either linear or nonlinear regression, such as the Guass-Newton method (linearization method) and the Marquardt procedure [61]. These methods can be found in advanced textbooks on regression and they are also viable techniques for finding optimal parameters in adaptive networks.

E. Neural Networks as Special Cases of Adaptive Networks

Some special cases of adaptive networks have been explored extensively in the neural network literature. In particular, we will introduce two types of neural networks: the back-propagation neural network (BPNN) and the radial basis function network (RBFN). Other types of adaptive networks that can be interpreted as a set of fuzzy if-then rules are investigated in the next section.

E.1 Back Propagation Neural Networks (BPNN's)

A back-propagation neural network (BPNN), as already mentioned in examples 4 and 5, is an adaptive network whose nodes (called **neurons**) perform the same function on incoming signals; this node function is usually a composite function of the weighted sum and a nonlinear function called the **activation function** or **transfer function**. Usually the activation functions are of either a sigmoidal or a hyper-tangent type which approximates the **step function** (or **hard limiter**) and yet provides differentiability with respect to input signals. Figure 32 depicts the four different types of activation functions $f(x)$ defined below.

Step function: $f(x) = \begin{cases} 1 \text{ if x} \geq 0. \\ 0 \text{ if x} < 0. \end{cases}$

Sigmoid function: $f(x) = \dfrac{1}{1+e^{-x}}.$

Hyper-tangent function: $f(x) = \tanh(x/2) = \dfrac{1-e^{-x}}{1+e^{-x}}.$

Identity function: $f(x) = x.$

When the step function (hard-limiter) is used as the activation function for a layered network, the network is often called a **perceptron** [78], [70], as explained in example 4. For a neural network to approximate a continuous-valued function not necessarily limited to the interval $[0, 1]$ or $[1, -1]$, we usually let the node function for the output layer be a weighted sum with no limiting-type activation functions. This is equivalent to the situation where the activation function is an identity function, and output nodes of this type are often called **linear nodes**.

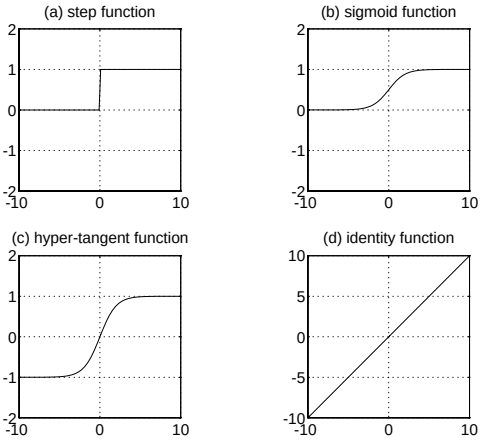


Fig. 32. *Activation functions for BPNN's: (a) step function; (b) sigmoid function; (c) hyper-tangent function; (d) identity function.*

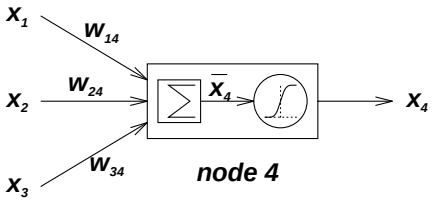


Fig. 33. *A BPNN node.*

For simplicity, we assume the BPNN in question uses the sigmoidal function as its activation function. The *net input* $\bar{x}$ 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 33 (where $j = 4$) are

$$\begin{aligned} \bar{x}_j &= \textstyle\sum_i w_{ij} x_i + t_j, \\ x_j &= f(\bar{x}_j) = \dfrac{1}{1+e^{-\bar{x}_j}}, \end{aligned} \tag{45}$$

where x_i is the output of node i located in the previous layer, w_{ij} is the weight associated with the link connecting nodes i and j, and t_j is the threshold of node j. Since the weights w_{ij} 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 23 shows a two-layer BPNN with 3 inputs in the input layer, 3 neurons in the hidden layer, and 2 output neurons in the output layer. For simplicity, this BPNN will be referred to as a 3-3-2 structure, corresponding to the number of nodes in each layer. (Note that the input layer is composed of three buffer nodes for distributing the input signals; therefore this layer is conventionally not counted as a physical layer of the BPNN.)

BPNN's are by far the most commonly used NN struc-

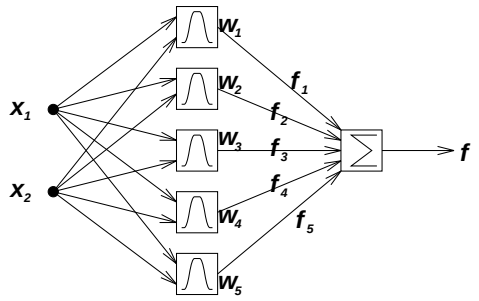


Fig. 34. *A radial basis function network (RBFN)*.

ture for applications in a wide range of areas, such as speech recognition, optical character recognition (OCR), signal processing, data compression, and automatic control.

E.2 Radial Basis Function Networks (RBFN's)

The locally-tuned and overlapping receptive field is a well-known structure that has been studied in the regions of the cerebral cortex, the visual cortex, and so forth. Drawing on the knowledge of biological receptive fields, Moody and Darken [66], [67] proposed a network structure that employs local receptive fields to perform function mappings. Similar schemes have been proposed by Powell [74], Broomhead and Lowe [7], and many others in the areas of interpolation and approximation theory; these schemes are collectively called radial basis function approximations. Here we shall call this network structure the **radial basis function network** or **RBFN**. Figure 34 shows a schematic diagram of an RBFN with five receptive field units; the activation level of the i-th receptive field unit (or hidden unit) is

$$w_i = R_i(\vec{x}) = R_i(\|\vec{x} - \vec{c_i}\|/\sigma_i),\ i = 1, 2, ..., H \qquad (46)$$

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

$$R_i(\vec{x}) = exp[-\frac{\|\vec{x} - \vec{c_i}\|^2}{\sigma_i^2}] \qquad (47)$$

or as a logistic function

$$R_i(\vec{x}) = \frac{1}{1 + exp[\|\vec{x} - \vec{c_i}\|^2/\sigma_i^2]} \qquad (48)$$

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

The output of a radial basis function network can be computed in two ways. In the simpler method, as shown in Figure 34, the final output is the weighted sum of the output value associated with each receptive field:

$$f(\vec{x}) = \sum_{i=1}^{H} f_i w_i = \sum_{i=1}^{H} f_i R_i(\vec{x}), \qquad (49)$$

where f_i is the output value associated with the i-th receptive field. A more complicated method for calculating the overall output is to take the weighted average of the output associated with each receptive field:

$$f(\vec{x}) = \frac{\sum_{i=1}^{H} f_i w_i}{\sum_{i=1}^{H} w_i} = \frac{\sum_{i=1}^{H} f_i R_i(\vec{x})}{\sum_{i=1}^{H} R_i(\vec{x})}. \qquad (50)$$

This mode of calculation, though has a higher degree of computational complexity, possesses the advantage that points in the overlapping area of two receptive fields will have a well interpolated output value between the output values of the two receptive fields. For representation purposes, if we change the radial basis function $R_i(\vec{x})$ in each node of layer 2 in Figure 34 by its **normalized** counterpart $R_i(\vec{x})/\sum_i R_i(\vec{x})$, then the overall output is specified by equation (50).

Several learning algorithms have been proposed to identify the parameters ($\vec{c_i}$, σ_i and f_i) of an RBFN. Note that the RBFN is an ideal example of the hybrid learning described in the previous section, where the linear parameters are f_i and the nonlinear parameters are c_i and σ_i. In practice, the $\vec{c_i}$ are usually found by means of vector quantization or clustering techniques (which assume similar input vectors produce similar outputs) and the σ_i are obtained heuristically (such as by taking the average distance to the first several nearest neighbors of $\vec{c_i}$'s). Once these nonlinear parameters are fixed, the linear parameters can be found by either the least-squares method or the gradient method. Chen et al. [8] used an alternative method that employs the orthogonal least-squares algorithm to determine the c_i's and f_i's while keeping the σ_i's at a predetermined constant.

An extension of Moody-Darken's RBFN is to assign a linear function as the output function of each receptive field; that is, f_i is a linear function of the input variables instead of a constant:

$$f_i = \vec{a}_i \cdot \vec{x} + b_i, \qquad (51)$$

where $\vec{a}_i$ is a parameter vector and b_i is a scalar parameter. Stokbro et al. [89] used this structure to model the Mackey-Glass chaotic time series [59] and found that this extended version performed better than the original RBFN with the same number of fitting parameters.

It was pointed out by the authors that under certain constraints, the RBFN is functionally equivalent to the the zero-order Sugeno fuzzy model. See [32] or [34] for details.

IV. ANFIS: Adaptive Neuro-Fuzzy Inference Systems

A class of adaptive networks that act as a fundamental framework for adaptive fuzzy inference systems is introduced in this section. This type of networks is referred to as **ANFIS** [25], [24], [29], which stands for **Adaptive-Network-based Fuzzy Inference System**, or semantically equivalently, **Adaptive Neuro-Fuzzy Inference System**. We will describe primarily the ANFIS architecture and its learning algorithm for the Sugeno fuzzy model,

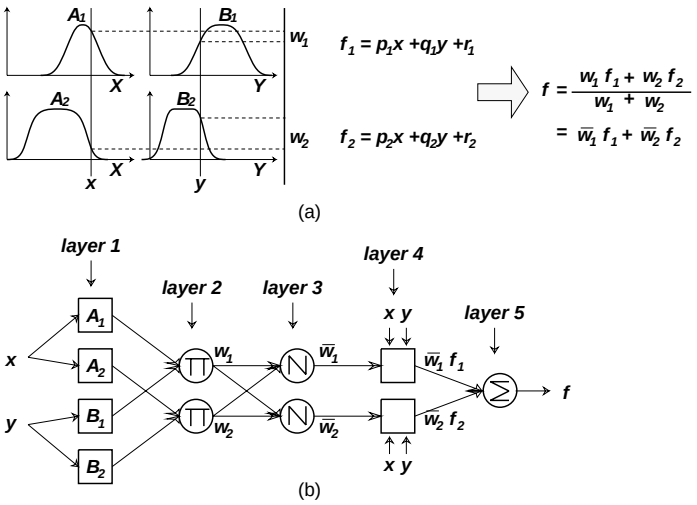


Fig. 35. *(a) A two-input first-order Sugeno fuzzy model with two rules; (b) equivalent ANFIS architecture.*

with an application example of chaotic time series prediction.

Note that similar network structures were also proposed independently by Lin and Lee [55] and Wang and Mendel [106]).

A. ANFIS Architecture

For simplicity, we assume the fuzzy inference system under consideration has two inputs x and y and one output z. For a first-order Sugeno fuzzy model [98], [91], a typical rule set with two fuzzy if-then rules can be expressed as

Rule 1: If x is A_1 and y is B_1, then $f_1 = p_1x + q_1y + r_1$,
Rule 2: If x is A_2 and y is B_2, then $f_2 = p_2x + q_2y + r_2$.

Figure 35 (a) illustrates the reasoning mechanism for this Sugeno model. The corresponding equivalent ANFIS architecture is as shown in Figure 35(b), where nodes of the same layer have similar functions, as described below. (Here we denote the output node i in layer l as $O_{l,i}$.)

Layer 1: Every node i in this layer is an adaptive node with a node output defined by

$$\begin{aligned} O_{1,i} &= \mu_{A_i}(x), \quad \text{for } i = 1, 2, \text{ or} \\ O_{1,i} &= \mu_{B_{i-2}}(y), \quad \text{for } i = 3, 4, \end{aligned} \tag{52}$$

where x (or y) is the input to the node and A_i (or B_{i-2}) is a fuzzy set associated with this node. In other words, outputs of this layer are the membership values of the premise part. Here the membership functions for A_i and B_i can be any appropriate parameterized membership functions introduced in Section II. For example, A_i can be characterized by the generalized bell function:

$$\mu_A(x) = \frac{1}{1 + [(\frac{x - c_i}{a_i})^2]^{b_i}}, \tag{53}$$

where $\{a_i, b_i, c_i\}$ is the parameter set. Parameters in this layer are referred to as *premise parameters*.

Layer 2: Every node in this layer is a fixed node labeled $\mathbf{\Pi}$, which multiplies the incoming signals and outputs the product. For instance,

$$O_{2,i} = w_i = \mu_{A_i}(x) \times \mu_{B_i}(y), \; i = 1, 2. \tag{54}$$

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

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

$$O_{3,i} = \overline{w}_i = \frac{w_i}{w_1 + w_2}, \; i = 1, 2. \tag{55}$$

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

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

$$O_{4,i} = \overline{w}_i f_i = \overline{w}_i (p_i x + q_i y + r_i), \tag{56}$$

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

Layer 5: The single node in this layer is a fixed node labeled Σ, which computes the overall output as the summation of all incoming signals:

$$O_{5,1} = \textit{overall output} = \sum_i \overline{w}_i f_i = \frac{\sum_i w_i f_i}{\sum_i w_i} \tag{57}$$

Thus we have constructed an adaptive network that has exactly the same function as a Sugeno fuzzy model. Note that the structure of this adaptive network is not unique; we can easily combine layers 3 and 4 to obtain an equivalent network with only four layers. Similarly, we can perform weight normalization at the last layer; Figure 36 illustrates an ANFIS of this type.

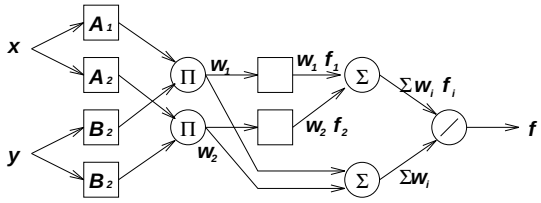


Fig. 36. *Another ANFIS architecture for the two-input two-rule Sugeno fuzzy model.*

Figure 37 (a) is an ANFIS architecture that is equivalent to a two-input first-order Sugeno fuzzy model with nine rules, where each input is assumed to have three associated MF's. Figure 37 (b) illustrates how the 2-D input space is partitioned into nine overlapping fuzzy regions, each of which is governed by fuzzy if-then rules. In other words, the premise part of a rule defines a fuzzy region, while the consequent part specifies the output within this region.

For ANFIS architectures for the Mamdani and Tsukamoto fuzzy models, the reader is referred to [29] and [34] for more details.

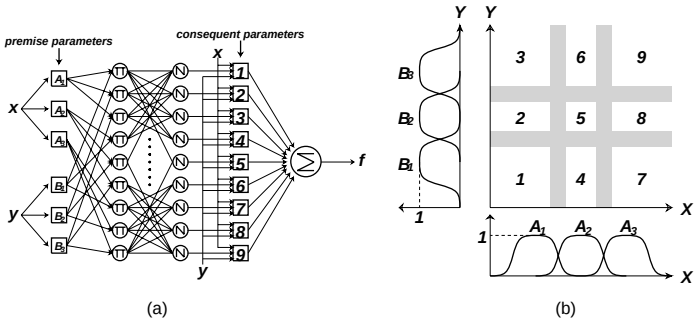


Fig. 37. *(a) ANFIS architecture for a two-input first-order Sugeno fuzzy model with nine rules; (b) partition of the input space into nine fuzzy regions.*

B. Hybrid Learning Algorithm

From the ANFIS architecture shown in Figure 35 (b), we observe that when the values of the premise parameters are fixed, the overall output can be expressed as a linear combination of the consequent parameters. In symbols, the output f in Figure 35 (b) can be rewritten as

$$\begin{aligned} f &= \frac{w_1}{w_1+w_2}f_1 + \frac{w_2}{w_1+w_2}f_2 \\ &= \overline{w}_1 f_1 + \overline{w}_2 f_2 \\ &= (\overline{w}_1 x)p_1 + (\overline{w}_1 y)q_1 + (\overline{w}_1)r_1 + (\overline{w}_2 x)p_2 + (\overline{w}_2 y)q_2 + (\overline{w}_2)r_2, \end{aligned} \qquad (58)$$

which is linear in the consequent parameters p_1, q_1, r_1, p_2, q_2, and r_2. Therefore the hybrid learning algorithm developed in the previous section can be applied directly. More specifically, in the forward pass of the hybrid learning algorithm, node outputs go forward until layer 4 and the consequent parameters are identified by the least-squares method. In the backward pass, the error signals propagate backward and the premise parameters are updated by gradient descent. Table I summarizes the activities in each pass.

TABLE I
TWO PASSES IN THE HYBRID LEARNING PROCEDURE FOR ANFIS.

	Forward Pass	Backward Pass
Premise Parameters	Fixed	Gradient Descent
Consequent Parameters	Least-Squares Estimate	Fixed
Signals	Node Outputs	Error Signals

As mentioned earlier, the consequent parameters thus identified are optimal under the condition that the premise parameters are fixed. Accordingly, the hybrid approach converges much faster since it reduces the dimension of the search space of the original back-propagation method.

If we fix the membership functions and adapt only the consequent part, then ANFIS can be viewed as a functional-link network [46], [71] where the "enhanced representations" of the input variables are obtained via the membership functions. These "enhanced representations", which take advantage of human knowledge, apparently express more insight than the functional expansion and the tensor (outer product) models [71]. By fine-tuning the membership functions, we actually make this "enhanced representation" also adaptive.

From equations (49), (50), and equation (57), it is not too hard to see the resemblance between the radial basis function network (RBFN) and the ANFIS for the Sugeno model. Actually these two computing framework are functionally equivalent under certain minor conditions [32]; this cross-fertilize both disciplines in many respects.

C. Application to Chaotic Time Series Prediction

ANFIS can be applied to a wide range of areas, such as nonlinear function modeling [24], [29], time series prediction [33], [29], on-line parameter identification for control systems [29], and fuzzy controller design [26], [28]. In particular, GE has been using ANFIS for modeling correction factors in steel rolling mills [6]. Here we will briefly report the application of ANFIS to chaotic time series prediction [33], [29].

The time series used in our simulation is generated by the Mackey-Glass differential delay equation [59]:

$$\dot{x}(t) = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1x(t). \qquad (59)$$

The prediction of future values of this time series is a benchmark problem that has been used and reported by a number of connectionist researchers, such as Lapedes and Farber [48], Moody [67], [65], Jones et al. [35], Crower [77], and Sanger [81]. The simulation results presented here were reported in [33], [29]; more details can be found therein.

The goal of the task is to use past 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 [48], Moody [67], [65], Crower [77]), the values $D = 4$ and $\triangle = P = 6$ were used. All other simulation settings were arranged to be as similar as possible to those reported in [77].

From the Mackey-Glass time series $x(t)$, we extracted 1000 input-output data pairs of the following format:

$$[x(t-18), x(t-12), x(t-6), x(t); x(t+6)], \qquad (60)$$

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

Figure 38 shows the results after about 500 epochs of learning. The desired and predicted values for both training data and checking data are essentially the same in Fig-

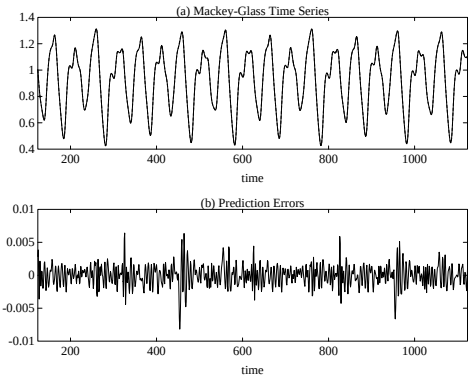


Fig. 38. (a) Mackey-Glass time series from $t = 124$ to 1123 and six-step ahead prediction (which is indistinguishable from the time series here); (b) prediction error. (Note that the first 500 data points are training data, while the remaining are for validation.)

ure 38 (a); the differences between them can only be seen on a much finer scale, such as that in Figure 38 (b).

TABLE II
GENERALIZATION RESULT COMPARISONS FOR $P = 6$.

Methods	Training Data	NDEI
ANFIS	500	0.007
AR Model	500	0.19
Cascade-Correlation NN	500	0.06
Back-Prop NN	500	0.02
6th-order Polynomial	500	0.04
Linear Predictive Method	2000	0.55

Table II lists the generalization capabilities of other methods, which were measured by using each method to predict 500 points immediately following the training set. The last four row of Table II are from [77] directly. The *non-dimensional error index* (NDEI) [48], [77] is defined as the root mean square error divided by the standard deviation of the target series. The remarkable generalization capability of ANFIS is attributed to the following facts:

- ANFIS can achieve a highly nonlinear mapping, therefore it is well-suited for predicting nonlinear time series.
- The ANFIS used here has 104 adjustable parameters, far fewer than those used in the cascade-correlation NN (693, the median) and back-prop NN (about 540) listed in Table II.
- Though not based on *a priori* knowledge, the initial parameter settings of ANFIS are intuitively reasonable and results in fast convergence to good parameter values that captures the underlying dynamics.
- ANFIS consists of fuzzy rules which are actually local mappings (which are called local experts in [36]) instead of global ones. These local mappings facilitate the **minimal disturbance principle** [111], which states that the adaptation should not only reduce the output error for the current training pattern but also minimize disturbance to response already learned. This is particularly important in on-line learning. We also found the use of least-squares method to determine the output of each local mapping is of particular importance. Without using LSE, the learning time would be ten times longer.

Other generalization tests and comparisons with neural network approaches can be found in [29].

The original ANFIS C codes and several examples (including this one) can be retrieved via anonymous ftp in `user/ai/areas/fuzzy/systems/anfis` at `ftp.cs.cmu.edu` (CMU Artificial Intelligence Repository).

V. Neuro-Fuzzy Control

Once a fuzzy controller is transformed into an adaptive network, the resulting ANFIS can take advantage of all the NN controller design techniques proposed in the literature. In this section we shall introduce common design techniques for ANFIS controllers. Most of these methodologies are derived directly from their counterparts for NN controllers. However, certain design techniques apply exclusively to ANFIS, which will be pointed out explicitly.

As shown in Figure 39, the block diagram of a typical feedback control system consists of a plant block and a controller block. The plant block is usually represented by a set of differential equations that describe the physical system to be controlled. These equations govern the behavior of the plant state $\mathbf{x}(t)$, which is assumed to be accessible in our discussion. In contrast, the controller block is usually a static function denoted by $\mathbf{g}$; it maps the the plant state $\mathbf{x}(t)$ into a control action $\mathbf{u}(t)$ that can hopefully achieve a given control objective. Thus for a general time-invariant control system, we have the following equations:

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) \quad \text{(plant dynamics)}, \\ \mathbf{u}(t) &= \mathbf{g}(\mathbf{x}(t)) \quad \text{(controller)}. \end{aligned}$$

The control objective here is to design a controller function $\mathbf{g}(\cdot)$ such that the plant state $\mathbf{x}(t)$ can follow a desired trajectory $\mathbf{x}_d(t)$ as closely as possible.

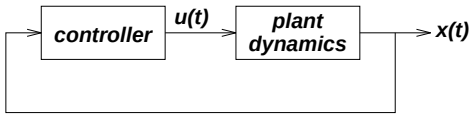


Fig. 39. *Block diagram for a continuous time feedback control system.*

A simple example of a feedback control system is the inverted pendulum system (Figure 40) where a rigid pole is hinged to a cart through a free joint with only one degree of freedom, and the cart moves on the rail tracks to its right or left depending on the force exerted on it. The control goal is to find the applied force u as a function of the state variable $\mathbf{x} = [\theta, \dot{\theta}, z, \dot{z}]$ (where θ is the pole angle and z is the cart position) such that the pole can be balanced from a given non-zero initial condition.

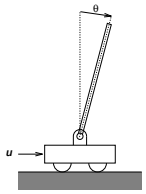


Fig. 40. *The inverted pendulum system.*

For a feedback control system in a discrete time domain, a general block diagram representation is as shown in Figure 41. Note that the inputs to the plant block include the control action $\mathbf{u}(k)$ and the previous plant output $\mathbf{x}(k)$, so the plant block now represents a static mapping. In symbols, we have

$$\begin{array}{lcll} \mathbf{x}(k+1) & = & \mathbf{f}(\mathbf{x}(k),\mathbf{u}(k)) & \text{(plant)}, \\ \mathbf{u}(k) & = & \mathbf{g}(\mathbf{x}(k)) & \text{(controller)}. \end{array}$$

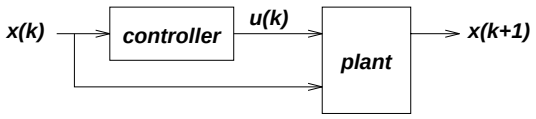


Fig. 41. *Block diagram for a discrete-time feedback control system.*

A central problem in control engineering is that of finding the control action $\mathbf{u}$ as a function of the plant output $\mathbf{x}$ in order to achieve a given control goal. Each design method for neuro-fuzzy controllers corresponds to a way of obtaining the control action; these methods are discussed next.

A. Mimicking Another Working Controller

Most of the time, the controller being mimicked is an experienced human operator who can control the plant satisfactorily. In fact, the whole concept of mimicking a human expert is the original intention of fuzzy controllers whose ultimate goal is to replace human operators who can control complex systems such as chemical reaction processes, subway trains, and traffic systems. An experienced human operator usually can summarize his or her control actions as a set of fuzzy if-then rules with roughly correct membership functions; this corresponds to the linguistic information. Prior to the emergence of neuro-fuzzy approaches, refining membership function is usually obtained via a lengthy trial-and-error process. Now with learning algorithms, we can further take advantage of the numerical information (input/output data pairs) and refine the membership functions in a systematic way. Note that the capability to utilize linguistic information is specific to fuzzy inference systems; it is not always available in neural networks. Successful applications of fuzzy controller based on linguistic information plus trial-and-error tuning includes steam engine and boiler control [60], Sendai subway systems [117], container ship crane control [116], elevator control [54], nuclear reaction control [5], automobile transmission control [40], aircraft control [14], and many others [90]. With the availability of learning algorithms, a wider range of applications is expected.

Note that this approach is not only for control applications. If the target system to be emulated is a human physician or a credit analyst, then the resulting fuzzy inference systems become a fuzzy expert system for diagnosis and credit analysis, respectively.

B. Inverse Control

Another scheme for obtaining desired control action is the inverse control method shown in Figure 42. For simplicity, we assume that the plant has only one state $x(k)$ and one input $u(k)$. In the learning phase, a training set is obtained by generating inputs $u(k)$ at random, and observing the corresponding outputs $x(k)$ produced by the plant. The ANFIS in Figure 42 (a) is then used to learn the inverse model of the plant by fitting the data pairs $(x(k), x(k+1); u(k))$. In the application phase, the ANFIS identifier is copied to the ANFIS controller in Figure 42 for generating the desired output. The input to the ANFIS controller is $(x(k), x_d(k))$; if the inverse model (ANFIS identifier) that maps $(x(k), x(k+1))$ to $u(k)$ is accurate, then the generated $u(k)$ should result in $x(k+1)$ that is close to $x_d(k)$. That is, the whole system in Figure 42 will behave like a pure unit-delay system.

This method seems straightforward and only one learning task is needed to find the inverse model of the plant. However, it assumes existence of the inverse of a plant, which is not valid in general. Moreover, minimization of the network error $||e_u(k)||^2$ does not guarantee minimization of the overall system error $||x_d(k) - x(k)||^2$.

Using ANFIS for adaptive inverse control can be found in [42].

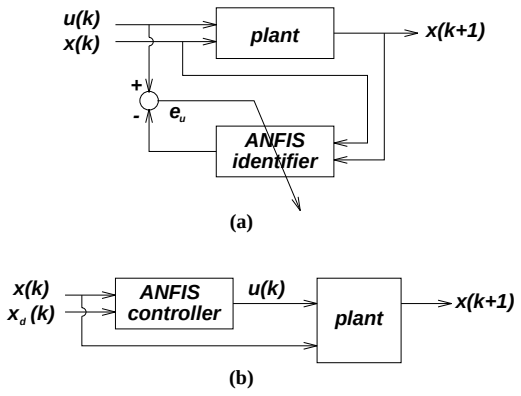


Fig. 42. *Block diagram for inverse control method: (a) learning phase; (b) application phase.*

C. Specialized Learning

The major problem with the inverse control scheme is that we are minimizing the network error instead of the overall system error. An alternative is to minimize the system error directly; this is called **specialized learning** [76]. In order to back-propagate error signals through the plant block in Figure 43, we need to find a model representing

the behavior of the plant. In fact, in order to apply back-propagation learning, all we need to know is the **Jacobian matrix** of the plant, where the element at row i and column j is equal to the derivative of the plant's i-th output with respect to its j-th input.

If the Jacobian matrix is not easy to find, an alternative is to estimate it on-line from the changes of the plant's inputs and outputs during two consecutive time instants. Other similar methods that aim at using an approximate Jacobian matrix to achieve the same learning effects can be found in [41], [11], [103]. Applying specialized learning to find an ANFIS controller for the inverted pendulum was reported in [27].

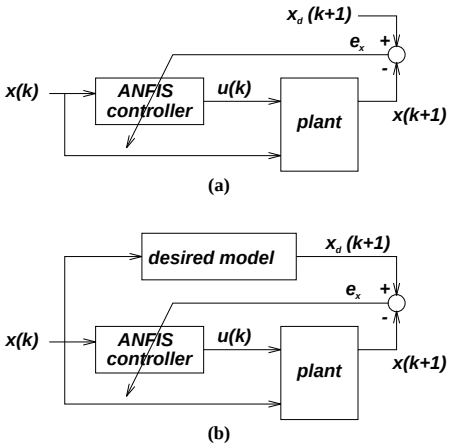


Fig. 43. *Block diagram for (a) specialized learning; (b) specialized learning with model reference.*

It is not always convenient to specify the desired plant output $x_d(k)$ at every time instant k. As a standard approach in model reference adaptive control, the desired behavior of the overall system can be implicitly specified by a (usually linear) model that is able to achieve the control goal satisfactorily. This alternative approach is shown in Figure 43 (b), where the desired output $x_d(k+1)$ is generated through a desired model.

D. Back-Propagation Through Time and Real Time Recurrent Learning

If we replace the controller and the plant block in Figure 39 with two adaptive networks, the feedback control system becomes a recurrent adaptive network discussed in Section III. Assuming the synchronous operation is adopted here (which virtually convert the system into the discrete time domain), we can apply the same scheme of unfolding of time to obtain a feedforward network, and then use the same back-propagation learning algorithm to identify the optimal parameters.

In terms of the inverted pendulum system (pole only), Figure 41 becomes Figure 44 if the controller block is replaced with a four-rule ANFIS and the plant block is replaced with a two-node adaptive network. To obtain the state trajectory, we cascade the network in Figure 44 to obtain the **trajectory network** shown in Figure 45. In particular, the inputs to the trajectory network are initial

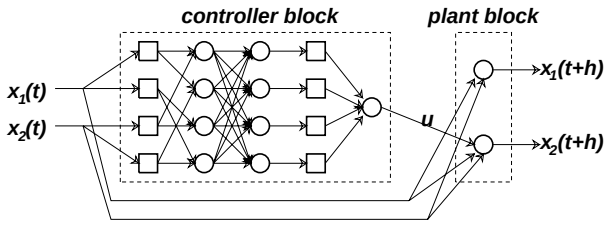


Fig. 44. *Network implementation of Figure 41 .*

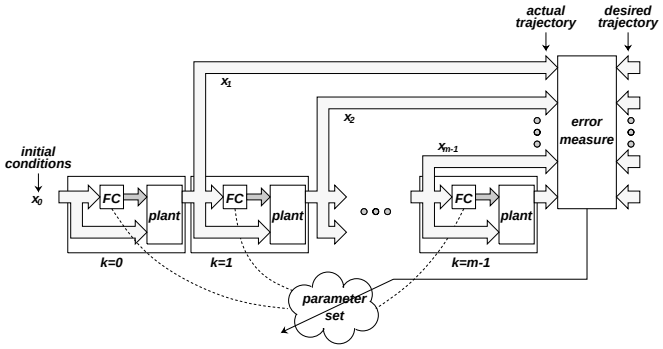


Fig. 45. *A trajectory network for control application (FC stands for "fuzzy controller").*

conditions of the plant; the outputs are the state trajectory from $k = 1$ to $k = m$. The adjustable parameters are all pertaining to the FC (fuzzy controller) block implemented as an four-rule ANFIS. Though there are m FC blocks, all of them refer to the same parameter set. For clarity, this parameter set is shown explicitly in Figure 45 and it is updated according to the output of the error measure block.

Each entry of the training data is of the following format:

$$\text{(initial conditions; desired trajectory)},$$

and the corresponding error measure to be minimized is

$$E = \sum_{k=1}^{m} \|\mathbf{x}(k) - \mathbf{x_d}(k)\|^2,$$

where $\mathbf{x}_d(k)$ is a desired state vector at $t = k * T$ (T is the sampling period). If we take control efforts into consideration, a revised error measure would be

$$E = \sum_{k=1}^{m} \|\mathbf{x}(k) - \mathbf{x_d}(k)\|^2 + \lambda \sum_{k=0}^{m-1} \|\mathbf{u}(k)\|^2,$$

where $\mathbf{u}(k)$ is the control action at time step k. By a proper selection of λ, a compromise between trajectory error and control efforts can be obtained.

Use of back-propagation through time to train a neural network for backing up a tractor-trailer system is reported in [69]. The same technique was used to design an ANFIS controller for balancing an inverted pendulum [28]. Note that back-propagation through time is usually an off-line learning algorithms in the sense that the parameters will not be updated till the sequence ($k = 1$ to m) is over. If the sequence is too long or if we want to update the parameters in the middle of the sequence, we can always apply RTRL (real time recurrent learning) introduced earlier.

E. Feedback Linearization and Sliding Control

The equations of motion of a class of dynamic systems in continuous time domain can be expressed in the canonical form:

$$x^{(n)}(t) = f(x(t), \dot{x}(t), \cdots x^{(n-1)}(t)) + bu(t), \qquad (61)$$

where f is an unknown continuous function, b is the control gain, and $u \in R$ and $y \in R$ are the input and output of the system, respectively. The control objective is to force the state vector $\boldsymbol{x} = [x, \dot{x}, \ldots, x^{(n-1)}]^T$ to follow a specified desired trajectory $\boldsymbol{x}_d = [x_d, \dot{x}_d, \ldots, x_d^{(n-1)}]^T$. If we define the tracking error vector as $\mathbf{e} = \mathbf{x} - \mathbf{x}_d$, then the control objective is to design a control law $u(t)$ which ensures $\boldsymbol{e} \rightarrow 0$ as $t \rightarrow \infty$. (For simplicity, we assume $b = 1$ in the following discussion.)

Equation (61) is a typical **feedback linearizable** system since it can be reduced to a linear system if f is known exactly. Specifically, the following control law

$$u(t) = -f(\boldsymbol{x}(t)) + x_d^{(n)} + \boldsymbol{k}^T \boldsymbol{e} \qquad (62)$$

would transform the original nonlinear dynamics into a linear one:

$$e^{(n)}(t) + k_1 e^{(n-1)} + \cdots + k_n e = 0, \qquad (63)$$

where $\boldsymbol{k}=[k_n, \ldots, k_1]^T$ is an appropriately chosen vector that ensures satisfactory behavior of the close-loop linear system in equation (63).

Since f is unknown, an intuitive candidate of u would be

$$u = -F(\boldsymbol{x}, \boldsymbol{p}) + x_d^{(n)} + \boldsymbol{k}^T \boldsymbol{e} + v, \qquad (64)$$

where v is an additional control input to be determined later, F is an parameterized function (such as ANFIS, neural networks, or any other types of adaptive networks) that is rich enough to approximate f. Using this control law, the close-loop system becomes

$$e^{(n)} + k_1 e^{(n-1)} + \cdots + k_n e = (f - F) + v. \qquad (65)$$

Now the problem is divided into two tasks:

- How to update the parameter vector $\boldsymbol{p}$ incrementally so that $F(\boldsymbol{x}, \boldsymbol{p}) \approx f(\boldsymbol{x})$ for all $\boldsymbol{x}$.
- How to apply v to guarantee global stability while F is approximating f during the whole process.

The first task is not too difficult as long as F, which could be a neural network or a fuzzy inference system, is equipped with enough parameters to approximate f. For the second task, we need to apply the concept of a branch of nonlinear control theory called **sliding control** [102], [86]. The standard approach is to define an error metrics as

$$s(t) = (\frac{d}{dt} + \lambda)^{n-1} e(t), \text{ with } \lambda > 0. \qquad (66)$$

The equation $s(t) = 0$ defines a time varying hyperplane in R^n on which the tracking error vector $\boldsymbol{e}(t) = [e(t), \dot{e}(t), \ldots, e^{n-1}(t)]^T$ decays exponentially to zero, so that perfect tracking can be obtained asymptotically. Moreover, if we can maintain the following condition:

$$\frac{d|s(t)|}{dt} \leq -\eta, \qquad (67)$$

then $|s(t)|$ will approach the hyperplane $|s(t)| = 0$ in a finite time less than or equal to $|s(0)|/\eta$. In other words, by maintain the condition in equation (67), $s(t)$ will approaches the sliding surface $s(t) = 0$ in a finite time, and then the error vector $\boldsymbol{e}(t)$ will converge to the origin exponentially with a time constant $(n-1)/\lambda$.

From equation (66), s can be rearranged as follows:

$$s = (\lambda + \frac{d}{dt})^{n-1} e = [\lambda^{n-1}, (n-1)\lambda^{n-2}, \ldots, 1]\mathbf{e}. \qquad (68)$$

Differentiate the above equation and plug in $e^{(n)}$ from equation (65), we obtain

$$\begin{aligned} \frac{ds}{dt} &= e^{(n)} + [0, \lambda^{n-1}, (n-1)\lambda^{n-2}, \cdots, \lambda]\mathbf{e} \\ &= f - F + v - [k_n, k_{n-1}, \cdots, k_1]\mathbf{e} \\ &\quad + [0, \lambda^{n-1}, (n-1)\lambda^{n-2}, \cdots, \lambda]\mathbf{e} \end{aligned} \qquad (69)$$

By setting $[k_n, k_{n-1}, \cdots, k_1] = [0, \lambda^{n-1}, (n-1)\lambda^{n-2}, \cdots, \lambda]$, we have

$$\frac{ds}{dt} = f - F + v,$$

and

$$\begin{aligned} \frac{d|s|}{dt} &= \frac{ds}{dt} sgn(s) \\ &= (f - F + v) sgn(s). \end{aligned}$$

That is, equation (67) is satisfied if and only if

$$(f - F + v) sgn(s) \leq -\eta.$$

If we assume the approximation error $|f - F|$ is bounded by a positive number A, then the above equation is always satisfied if

$$v = -(A + \eta) sgn(s).$$

To sum up, if we choose the control law as

$$u(t) = -F(\mathbf{x}, \mathbf{p}) + x_d^{(n)} + [0, \lambda^{n-1}, (n-1)\lambda^{n-2} \ldots \lambda]\boldsymbol{e} - (A+\eta) sgn(s),$$

where $F(\mathbf{x}, \mathbf{p})$ is an adaptive network that approximates $f(\mathrm{x})$ and A is the error bound, then the close-loop system can achieve perfect tracking asymptotically with global stability.

This approach uses a number of nonlinear control design techniques and possesses rigorous proofs for global stability. However, its applicability is restricted to feedback linearizable systems. The reader is referred to [86] for a more detailed treatment of this subject. Applications of this technique to neural network and fuzzy control can be found in [82] and [104], respectively.

F. Gain Scheduling

Under certain arrangements, the first-order Sugeno fuzzy model becomes a gain scheduler that switches between several sets of feedback gains. For instance, a first-order Sugeno fuzzy controller for an hypothetical inverted pendulum system with varying pole length may have the following fuzzy if-then rules:

$$\begin{cases} \text{If pole is short, then } f_1 = k_{11}\theta + k_{12}\dot{\theta} + k_{13}z + k_{14}\dot{z}, \\ \text{If pole is medium, then } f_2 = k_{21}\theta + k_{22}\dot{\theta} + k_{23}z + k_{24}\dot{z}, \\ \text{If pole is long, then } f_3 = k_{31}\theta + k_{32}\dot{\theta} + k_{33}z + k_{34}\dot{z}. \end{cases} \tag{70}$$

This is in fact a gain scheduling controller, where the scheduling variable is the pole length and the control action is switching smoothly between three sets of feedback gains depending on the value of the scheduling variable. In general, the scheduling variables only appear in the premise part while the state variables only appear in the consequent part. The design method here is standard in gain scheduling: find several nominal points in the space formed by scheduling variables and employ any of the linear control design techniques to find appropriate feedback gains. If the number of nominal points is small, we can construct the fuzzy rules directly. On the other hand, if the number of nominal points is large, we can always use ANFIS to fit desired control actions to a fuzzy controller.

Examples of applying this method to both one-pole and two-pole inverted pendulum systems with varying pole lengths can be found in the demo programs in [31].

G. Others

Other design techniques that do not use the learning algorithm in neuro-fuzzy modeling are summarized here.

For complex control problems with perfect plant models, we can always use gradient-free optimization schemes, such as genetic algorithms [22], [19], simulated annealing [44], [45], downhill Simplex method [68], and random method [63], [88]. In particular, use of genetic algorithms for neural network controllers can be found in [113]; for fuzzy logic controllers, see [39], [52], [38].

If the plant model is not available, we can apply reinforcement learning [2] to find a working controller directly. The close relationship between reinforcement learning and dynamic programming was addressed in [3], [110]. Other variants of reinforcement learning includes temporal difference methods (TD(λ) algorithms) and Q-learning [108]. Representative applications of reinforcement learning to fuzzy control can be found in [4], [51], [12], [56].

Some other design and analysis approaches for fuzzy controllers include cell-to-cell mapping techniques [13], [87], model-based design method [99], self-organizing controllers [75], [100], and so on. As more and more people are working in this field, new design methods are coming out sooner than before.

VI. Concluding Remarks

A. A. Current Problems and Possible Solutions

A typical modeling problem includes **structure determination** and **parameter identification**. We address the parameter identification problem for ANFIS in this paper, which is solved via the back-propagation gradient descent and the least-squares method. The structure determination problem, which deals with the partition style, the number of MF's for each input, and the number of fuzzy if-then rules, and so on, is now an active research topic in the field. Work along this direction includes Jang's fuzzy CART approach [30], Lin's reinforcement learning method [57], Sun's fuzzy k-d trees [93], Sugeno's iterative method [92] and various clustering algorithms proposed by Chiu [15], Khedkar [43] and Wang [105]. Moreover, advances on the constructive and destructive learning of neural networks [18], [53] can also shed some lights on this problem.

Though we can speed up the parameter identification problem by introducing the least-squares estimator into the learning cycle, gradient descent still slows down the training process and the training time could be prohibitively long for a complicated task. Therefore the need to search for better learning algorithms hold equally true for both neural networks and fuzzy models. Variants of gradient descent proposed in the neural network literature; including second-order back-propagation [72], quick-propagation [17], and so on, can be used to speed up training. A number of techniques used in nonlinear regression can also contribute in this regard, such as the Guass-Newton method (linearization method) and the Marquardt procedure [61]. Another important resource is the rich literature of optimization, which offers many better gradient-based optimization routines, such as quadratic programming and conjugate gradient descent.

B. Future Directions

Due to the extreme flexibility of adaptive networks, ANFIS can have a number of variants that are different from what we have proposed here. For instance, we can replace the Π nodes in layer 2 of ANFIS with the parameterized T-norm operator [16] and let the learning algorithm decide the best T-norm function for a specific application. By employing the adaptive network as a common framework, we have also proposed other adaptive fuzzy models tailored for different purposes, such as the neuro-fuzzy classifier [94], [95] for data classification and the fuzzy filter scheme [96], [97] for feature extraction. There are a number of possible extensions and applications and they are currently under investigation.

During the past years, we have witnessed the rapid growth of the application of fuzzy logic and fuzzy set theory to consumer electronic products, automotive industry and process control. With the advent of fuzzy hardware with possibly on-chip learning capability, the applications to adaptive signal processing and control are expected. Potential applications within adaptive signal processing in-

cludes adaptive filtering [21], channel equalization [9], [10], [107], noise or echo cancelling [112], predictive coding [53], and so on.

Acknowledgments

The authors wish to thank Steve Chiu for providing numerous helpful comments. Most of this paper was finished while the first author was a research associate at UC Berkeley, so the authors would like to acknowledge the guidance and help of Professor Lotfi A. Zadeh and other members of the "fuzzy group" at UC Berkeley. Research supported in part by the BISC Program, NASA Grant NCC 2-275, EPRI Agreement RP 8010-34 and MICRO State Program No. 92-180.

References

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

[2] 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.

[3] A. G. Barto, R. S. Sutton, and C.J.C.H Watkins. Learning and sequential decision making. In M. Gabriel and J. W. Moore, editors, *Learning and Computational Neuroscience*. MIT Press, Cambridge, 1991.

[4] H. R. Berenji and P. Khedkar. Learning and tuning fuzzy logic controllers through reinforcements. *IEEE Trans. on Neural Networks*, 3(5):724–740, 1992.

[5] J. A. Bernard. Use of rule-based system for process control. *IEEE Control Systems Magazine*, 8(5):3–13, 1988.

[6] P. Bonissone, V. Badami, K. Chiang, P. Khedkar, K. Marcelle, and M. Schutten. Industrial applications of fuzzy logic at general electric. *The Proceedings of the IEEE*, March 1995.

[7] D. S. Broomhead and D. Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988.

[8] 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.

[9] S. Chen, G. J. Gibson, C. F. N. Cowan, and P. M. Grant. Adaptive equalization of finite nonlinear channels using multilayer perceptrons. *Signal Processing*, 20:107–119, 1990.

[10] S. Chen, G. J. Gibson, C. F. N. Cowan, and P. M. Grant. Reconstruction of binary signals using an adaptive radial-basis-function equalizer. *Signal Processing*, 22:77–93, 1991.

[11] V. C. Chen and Y. H. Pao. Learning control with neural networks. In *Proc. of International Conference on Robotics and Automation*, pages 1448–1453, 1989.

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

[13] 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.

[14] S. Chiu, S. Chand, D. Moore, and A. Chaudhary. Fuzzy logic for control of roll and moment for a flexible wing aircraft. *IEEE Control Systems Magazine*, 11(4):42–48, 1991. 1991.

[15] S. L. Chiu. Fuzzy model identification based on cluster estimation. *Journal of Intelligent and Fuzzy Systems*, 2(3), 1994.

[16] D. Dubois and H. Prade. *Fuzzy Sets and Systems: Theory and Applications*. Academic press, 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] D. E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, Reading, Massachusetts, 1989.

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

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

[22] J. H. Holland. *Adaptation in natural and artificial systems*. The University of Michigan Press, 1975.

[23] T. C. Hsia. *System Identification: Least-Squares Methods*. D.C. Heath and Company, 1977.

[24] 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.

[25] 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.

[26] J.-S Roger Jang. A self-learning fuzzy controller with application to automobile tracking problem. In *Proc. of IEEE Roundtable Discussion on Fuzzy and Neural Systems, and Vehicle Application*, page paper no. 10, Tokyo, Japan, November 1991. Institute of Industrial Science, Univ. of Tokyo.

[27] J.-S. Roger Jang. Fuzzy controller design without domain experts. In *Proc. of IEEE international conference on fuzzy systems*, March 1992.

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

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

[30] J.-S. Roger Jang. Structure determination in fuzzy modeling: a fuzzy CART approach. In *Proc. of IEEE international conference on fuzzy systems*, Orlando, Florida, June 1994.

[31] J.-S. Roger Jang and N. Gulley. The Fuzzy Logic Toolbox for use with MATLAB. The MathWorks, Inc., Natick, Massachusetts, 1995.

[32] J.-S. Roger Jang and C.-T. Sun. Functional equivalence between radial basis function networks and fuzzy inference systems. *IEEE Trans. on Neural Networks*, 4(1):156–159, January 1993.

[33] J.-S. Roger Jang and C.-T. Sun. Predicting chaotic time series with fuzzy if-then rules. In *Proc. of IEEE international conference on fuzzy systems*, San Francisco, March 1993.

[34] J.-S. Roger Jang and C.-T. Sun. Neuro-fuzzy modeling: an computational approach to intelligence, 1995. Submitted for publication.

[35] 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.

[36] M. I. Jordan and R. A. Jacobs. Hierarchical mextures of experts and the EM algorithm. Technical report, M.I.T., 1993.

[37] A. Kandel, editor. *Fuzzy expert systems*. CRC Press, Boca Raton, FL, 1992.

[38] C. L. Karr. GAs for fuzzy controllers. *AI Expert*, 6(2):26–33, February 1991.

[39] C. L. Karr and E. J. Gentry. Fuzzy control of pH using genetic algorithms. *IEEE Trans. on Fuzzy Systems*, 1(1):46–53, February 1993.

[40] Y. Kasai and Y. Morimoto. Electronically controlled continuously variable transmission. In *Proc. of International Congress Transportation Electronics*, Dearborn, Michigan, 1988.

[41] M. Kawato, K. Furukawa, and R. Suzuki. A hierarchical neural network model for control and learning of voluntary movement. *Biological Cybernetics*, 57:169–185, 1987.

[42] D. J. Kelly, P. D. Burton, and M.A. Rahman. The application of a neural fuzzy controller to process control. In *Proc. of the International Joint Conference of the North American Fuzzy Information Processing Society Biannual Conference, the Industrial Fuzzy Control and Intelligent Systems Conference, and the NASA Joint Technology Workshop on Neural Networks and Fuzzy Logic*, San Antonio, Texas, December 1994.

[43] P. S. Khedkar. *Learning as Adaptive Interpolation in Neural Fuzzy Systems*. PhD thesis, Computer Science Division, Department of EECS, University of California at Berkeley, 1993.

[44] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Research Report 9335 IBM T. J. Watson Center*, 1983.

[45] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[46] 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.

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

[48] 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.

[49] 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.

[50] 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.

[51] 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.

[52] M. A. Lee and H. Takagi. Integrating design stages of fuzzy systems using genetic algorithms. In *Proc. of the second IEEE International Conference on Fuzzy Systems*, pages 612–617, San Francisco, 1993.

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

[54] Fujitec Company Limited. FLEX-8800 series elevator group control system, 1988. Osaka, Japan.

[55] 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.

[56] C.-T. Lin and C.-S. G. Lee. Reinforcement structure/parameter learning for neural-network-based fuzzy logic control systems. In *Proc. of IEEE International Conference on Fuzzy Systems*, pages 88–93, San Francisco, March 1993.

[57] C.-T. Lin and C.-S. G. Lee. Reinforcement structure/parameter learning for neural-network-based fuzzy logic control systems. *IEEE Trans. on Fuzzy Systems*, 2(1):46–63, 1994.

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

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

[60] E. H. Mamdani and S. Assilian. An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Man-Machine Studies*, 7(1):1–13, 1975.

[61] D. W. Marquardt. An algorithm for least squares estimation of nonlinear parameters. *Journal of the Society of Industrial and Applied Mathematics*, 2:431–441, 1963.

[62] S. J. Mason. Feedback theory – further properties of signal flow graphs. *Proc. IRE*, 44(7):920–926, July 1956.

[63] J. Matyas. Random optimization. *Automation and Remote Control*, 26:246–253, 1965.

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

[65] 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.

[66] 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.

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

[68] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1964.

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

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

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

[72] 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.

[73] N. Pfluger, J. Yen, and R. Langari. A defuzzification strategy for a fuzzy logic controller employing prohibitive information in command formulation. In *Proc. of IEEE international conference on fuzzy systems*, pages 717–723, San Diego, March 1992.

[74] M.J.D. Powell. Radial basis functions for multivariable interpolation: a review. In J. C. Mason and M. G. Cox, editors, *Algorithms for Approximation*, pages 143–167. Oxford University Press, 1987.

[75] T. J. Procyk and E. H. Mamdani. A linguistic self-organizing process controller. *Automatica*, 15:15–30, 1978.

[76] D. Psaltis, A. Sideris, and A. Yamamura. A multilayered neural network controller. *IEEE Control Systems Magazine*, 8(4):17–21, April 1988.

[77] 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.

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

[79] 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.

[80] T. A. Runkler and M. Glesner. Defuzzification and ranking in the context of membership value semantics, rule modality, and measurement theory. In *European Congress on Fuzzy and Intelligent Technologies*, Aachen, September 1994.

[81] 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.

[82] R. M. Sanner and J. J. E. Slotine. Gaussian networks for direct adaptive control. *IEEE Trans. on Neural Networks*, 3:837–862, 1992.

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

[84] 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.

[85] 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.

[86] J.-J. E. Slotine and W. Li. *Applied nonlinear control.* Prentice Hall, 1991.

[87] 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.

[88] F. J. Solis and J. B. Wets. Minimization by random search techniques. *Mathematics of Operations Research*, 6(1):19–30, 1981.

[89] K. Stokbro, D. K. Umberger, and J. A. Hertz. Exploiting neurons with localized receptive fields to learn chaos. *Complex Systems*, 4:603–622, 1990.

[90] M. Sugeno, editor. *Industrial applications of fuzzy control.* Elsevier Science Pub. Co., 1985.

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

[92] M. Sugeno and T. Yasukawa. A fuzzy-logic-based approach to qualitative modeling. *IEEE Trans. on Fuzzy Systems*, 1(1):7–31, February 1993.

[93] C.-T. Sun. Rulebase structure identification in an adaptive network based fuzzy inference system. *IEEE Trans. on Fuzzy Systems*, 2(1):64–73, 1994.

[94] 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.

[95] 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.

[96] 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.

[97] C.-T. Sun, T.-Y. Shuai, and G.-L. Dai. Using fuzzy filters as feature detectors. In *Proc. of IEEE international conference on fuzzy systems*, pages 406–410 (Vol I), Orlando, Florida, June 1994.

[98] 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.

[99] K. Tanaka and M. Sugeno. Stability analysis and design of fuzzy control systems. *Fuzzy Sets and Systems*, 45:135–156, 1992.

[100] R. Tanscheit and E. M. Scharf. Experiments with the use of a rule-based self-organizing controller for robotics applications. *Fuzzy Sets and Systems*, 26:195–214, 1988.

[101] 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.

[102] V. I. Utkin. Variable structure systems with sliding mode: a survey. *IEEE Trans. on Automatic Control*, 22:212, 1977.

[103] K. P. Venugopal, R. Sudhakar, and A. S. Pandya. An improved scheme for direct adaptive control of dynamical systems using backpropagation neural networks. *Journal of Circuits, Systems and Signal Processing*, 1994. (Forthcoming).

[104] L.-X. Wang. Stable adaptive fuzzy control of nonlinear systems. *IEEE Trans. on Fuzzy Systems*, 1(1):146–155, 1993.

[105] L.-X. Wang. Traning fuzzy logic systems using nearest neighborhood clustering. In *Proc. of the IEEE International Conference on Fuzzy Systems*, San Francisco, March 1993.

[106] 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.

[107] L.-X. Wang and J. M. Mendel. Fuzzy adaptive filters, with application to nonlinear channel equalization. *IEEE Trans. on Fuzzy Systems*, 1(3):161–170, 1993.

[108] C.J.C.H Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

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

[110] Paul J. Werbos. A menu of designs for reinforcement learning over time. In III W. Thomas Miller, Richard S. Sutton, and Paul J. Werbos, editors, *Neural Networks for Control*, chapter 3. The MIT Press, Bradford, 1990.

[111] B. Widrow and M. A. Lehr. 30 years of adaptive neural networks: Perceptron, madline, and backpropagation. *Proceedings of the IEEE*, 78(9):1415–1442, 1990.

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

[113] Alexis P. Wieland. Evolving controls for unstable systems. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proc. of the 1990 Connectionist Models Summer School*, pages 91–102, Carnegie Mellon University, 1990.

[114] R. J. William and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270–280, 1989.

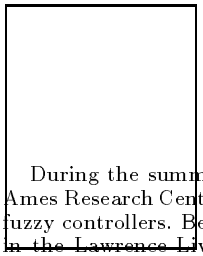
[115] R. R. Yager and D. P. Filev. SLIDE: A Simple Adaptive Defuzzification Method. *IEEE Transactions on Fuzzy Systems*, 1(1):69–78, February 1993.

[116] S. Yasunobu and G. Hasegawa. Evaluation of an automatic container crane operation system based on predictive fuzzy control. *Control Theory and Advanced Technology*, 2(2):419–432, 1986. 1986.

[117] S. Yasunobu and S. Miyamoto. Automatic train operation by predictive fuzzy control. In M. Sugeno, editor, *Industrial Applications of Fuzzy Control*, pages 1–18. North-Holland, Amsterdam, 1985.

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

[119] 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.

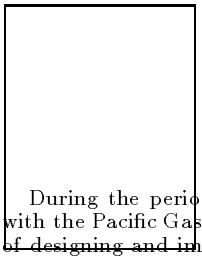


Jyh-Shing Roger Jang was born in Taipei, Taiwan in 1962. He received the B.S. degree in electrical engineering from National Taiwan University in 1984, and the Ph.D. degree in the Department of Electrical Engineering and Computer Sciences at the University of California, Berkeley, in 1992.

During the summer of 1989, he was a summer student in NASA Ames Research Center, working on the design and implementation of fuzzy controllers. Between 1991 and 1992 he was a research scientist in the Lawrence Livermore National Laboratory, working on spectrum modeling and analysis using neural networks and fuzzy logic. After obtaining his Ph.D. degree, he was a research associate in in the same department, working on machine learning techniques using fuzzy logic. Since September 1993, he has been with The MathWorks, Inc., working on the Fuzzy Logic Toolbox used with MATLAB.

His interests lie in the area of neuro-fuzzy modeling, system identification, machine learning, nonlinear regression, optimization, and computer aided control system design.

Dr. Jang is a member of IEEE.



Chuen-Tsai Sun received his B.S. degree in Electrical Engineering in 1979 and his M.A. degree in History in 1984, both from National Taiwan University, Taiwan. He received his Ph.D degree in Computer Science from the University of California at Berkeley in 1992. His Ph.D. research advisor was Professor Lotfi A. Zadeh, the initiator of fuzzy set theory.

During the period of 1989 and 1990 he worked as a consultant with the Pacific Gas and Electric Company, San Francisco, in charge of designing and implementing an expert system for protective device coordination in electric distribution circuits. Between 1991 and 1992 he was a research scientist in the Lawrence Livermore National Laboratory, working on plasma analysis using neural networks and fuzzy logic. Since August 1992, he has been on the faculty of the Department of Computer and Information Science at National Chiao Tung University. His current research interests include computational intelligence, system modeling, and computer assisted learning.

Dr. Sun is a member of IEEE. He was the Arthur Gould Tasheira Scholarship winner in 1986. He was also honored with the Phi Hua Scholar Award in 1985 for his publications in history.