# An On-Line Self-Constructing Neural Fuzzy Inference Network and Its Applications

Chia-Feng Juang and Chin-Teng Lin

***Abstract*—A self-constructing neural fuzzy inference network (SONFIN) with on-line learning ability is proposed in this paper. The SONFIN is inherently a modified Takagi–Sugeno–Kang (TSK)-type fuzzy rule-based model possessing neural network's learning ability. There are no rules initially in the SONFIN. They are created and adapted as on-line learning proceeds via simultaneous structure and parameter identification. In the structure identification of the precondition part, the input space is partitioned in a flexible way according to a aligned clustering-based algorithm. As to the structure identification of the consequent part, only a singleton value selected by a clustering method is assigned to each rule initially. Afterwards, some additional significant terms (input variables) selected via a projection-based correlation measure for each rule will be added to the consequent part (forming a linear equation of input variables) incrementally as learning proceeds. The combined precondition and consequent structure identification scheme can set up an economic and dynamically growing network, a main feature of the SONFIN. In the parameter identification, the consequent parameters are tuned optimally by either least mean squares (LMS) or recursive least squares (RLS) algorithms and the precondition parameters are tuned by backpropagation algorithm. Both the structure and parameter identification are done simultaneously to form a fast learning scheme, which is another feature of the SONFIN. Furthermore, to enhance the knowledge representation ability of the SONFIN, a linear transformation for each input variable can be incorporated into the network so that much fewer rules are needed or higher accuracy can be achieved. Proper linear transformations are also learned dynamically in the parameter identification phase of the SONFIN. To demonstrate the capability of the proposed SONFIN, simulations in different areas including control, communication, and signal processing are done. Effectiveness of the SONFIN is verified from these simulations.**

***Index Terms*—Equalizer, noisy speech recognition, projection-based correlation measure, similarity measure, TSK fuzzy rule.**

## I. Introduction

THE problem of system modeling is encountered in many areas such as control, communications, and pattern recognition, etc. Recently, the neural fuzzy approach to system modeling has become a popular research focus [1]–[4]. The key advantage of neural fuzzy approach over traditional ones lies on that the former doesn't require a mathematical description of the system while modeling. Moreover, in contrast to pure neural or fuzzy methods, the neural fuzzy method possesses both of their advantages; it brings the low-level learning and computational power of neural networks into fuzzy systems and provides the high-level human-like thinking and reasoning of fuzzy systems into neural networks [5]–[7].

A fuzzy system consists of a bunch of fuzzy if-then rules. Conventionally, the selection of fuzzy if-then rules often relies on a substantial amount of heuristic observation to express proper strategy's knowledge. Obviously, it is difficult for human experts to examine all the input–output data from a complex system to find a number of proper rules for the fuzzy system. To cope with this difficulty, several approaches to generating fuzzy if-then rules from numerical data, an active research topic in the neural fuzzy area, have been proposed [6]–[15]. Generally, these approaches consist of two learning phases, the structure learning phase and the parameter learning phase. Traditionally, these two phases are done sequentially; the structure learning phase is employed to decide the structure of fuzzy rules first and then the parameter learning phase is used to tune the coefficients of each rule (like the shapes and positions of the membership functions). One disadvantage of this sequential learning scheme is that it is suitable only for off-line instead of on-line operation. Moreover, to adopt this scheme a large amount of representative data should be collected in advance. Also, the independent realization of the structure and parameter learning usually each spends a lot of time. Owning to these problems, the structure as well as the parameter learning phases are done simultaneously in the proposed self-constructing neural fuzzy inference network (SONFIN). This ability makes the SONFIN suitable for fast on-line learning.

One important task in the structure identification of a neural fuzzy network is the partition of the input space, which influences the number of fuzzy rules generated. The most direct way is to partition the input space into grid types with each grid representing a fuzzy if-then rule [see Fig. 1(a)]. The major problem of such kind of partition is that the number of fuzzy rules increases exponentially as the dimension of the input space increases. Another frequently used method for input space partitioning is to cluster the input training vectors in the input space [24], [25]. Such a method provides a more flexible partition, as shown in Fig. 1(b). The resulting fuzzy rule is of the form, Rule $i$: IF $\mathbf{x}$ is $C_i$, THEN $y$ is $\cdots$, where $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ denotes the input vector with dimension $n$ and $C_i$ the $i$th cluster formed in the input space. One problem of this partition is that what $C_i$ means and what are the corresponding fuzzy terms in each input variable are always opaque to the user, especially in the case of high-input dimensions. This violates the spirit of fuzzy systems

Manuscript received July 10, 1996; revised January 7, 1997. This work was supported by the National Science Council, Republic of China, under Grant NSC 85-2212-E-009-044.

The authors are with the Department of Control Engineering, National Chiao-Tung University, Hsinchu, Taiwan, R.O.C.

Publisher Item Identifier S 1063-6706(98)00794-2.

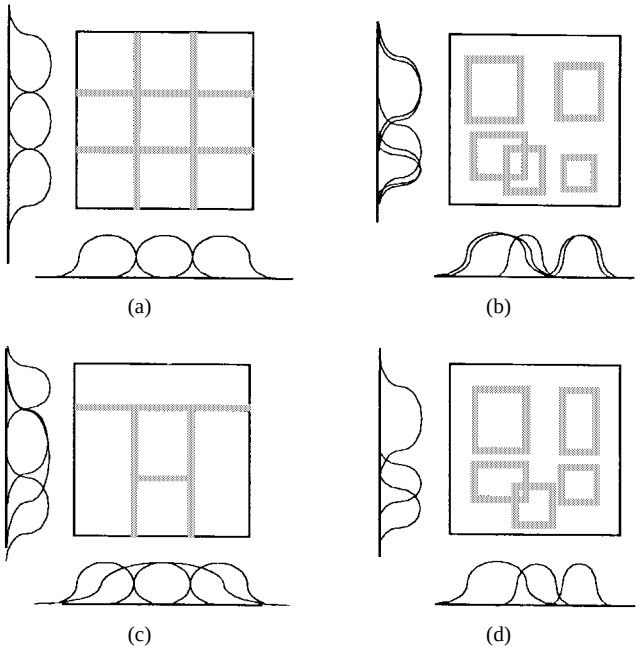



Fig. 1. Fuzzy partitions of two-dimensional input space. (a) Grid-type partitioning. (b) Clustering-based partitioning. (c) GA-based partitioning. (d) Proposed aligned clustering-based partitioning.

that what a fuzzy rule means and how it works should be easy to understand. We may solve this problem by projecting the generated cluster onto each dimension of the input space to form a projected one-dimensional (1-D) membership function for each input variable and represent a cluster by the product of the projected membership functions, as illustrated in Fig. 1(b). Compared with the grid-type partition, the clustering-based partition does reduce the number of generated rules, but not the number of membership functions of each input variable. To verify this, suppose there are $n$ input variables and each input variable is partitioned into $m$ parts ($m$ fuzzy terms). Then the total number of membership functions used is $nm$ for the grid-type partition. As to the clustering-based partition, if there are $k$ clusters formed, then the number of membership functions generated is $nk$. In general, $k$ is larger than $m$, meaning that the clustering-based partition creates more membership functions than the grid-type one dose. In fact, by observing the projected membership functions in Fig. 1(b), we find that some membership functions projected from different clusters have high similarity degrees. These highly similar membership functions should be eliminated. This phenomenon occurs not only in the clustering-based partitioning methods, but also in other approaches like those based on the orthogonal least square (OLS) method [16], [17].

Another flexible input space partitioning method is based on the genetic algorithm (GA) [28], which has the partition result as shown in Fig. 1(c). The major disadvantage of this method is that it is very time consuming; the computation cost to evaluate a partition result encoded in each individual is very high and many generations are needed to find the final partition. Hence, this scheme is obviously not suitable for on-line operation. Moreover, the GA-based partitioning methods might not find meaningful fuzzy terms for each input variable, as illustrated in Fig. 1(c). In this paper, we develop a novel on-line input space partitioning method, which is an aligned clustering-based approach. This method can produce a partition result like the one shown in Fig. 1(d). Basically, it aligns the clusters formed in the input space, so it reduces not only the number of rules but also the number of membership functions under a prespecified accuracy requirement. The proposed method creates only the significant membership functions on the universe of discourse of each input variable by using a fuzzy measure algorithm. It can thus generate necessary fuzzy rules from numerical data dynamically. In [17], the most significant rules are selected based upon OLS method. To use this method, the learning data should be collected in advance and the parameters of the fuzzy basis functions are fixed. The generated fuzzy rules by this method are significant only for the fixed input–output training pairs collected in advance, so it is not suitable for on-line learning. Since our objective is on-line learning, and the input membership functions are all tunable, a rule is considered to be necessary and is generated when it has a low overlapping degree with others.

Another objective of this paper is to provide an optimal way for determining the consequent part of fuzzy if-then rules during the structure learning phase. Different types of consequent parts (e.g., singletons, bell-shaped membership functions, or a linear combination of input variables) have been used in fuzzy systems [22]. It was pointed out by Sugeno and Tanaka [20] that a large number of rules are necessary when representing the behavior of a sophisticated system by the ordinary fuzzy model based on Mamdani's approach. Furthermore, they reported that the Takagi–Sugeno–Kang (TSK) model can represent a complex system in terms of a few rules. However, even though fewer rules are required for the TSK model, the terms used in the consequent part are quite considerable for multi-input/multi-output systems or for the systems with high-dimensional input or output spaces. Hence, we encounter a dilemma between the number of fuzzy rules and the number of consequent terms. A method is proposed in this paper to solve this dilemma, which is, in fact, a combinational optimization problem. A fuzzy rule of the following form is adopted in our system initially

$$\begin{aligned} \text{Rule } i\text{: } & \text{IF } x_1 \text{ is } A_{i1} \text{ and } \cdots \text{ and } x_n \text{ is } A_{in} \\ & \text{THEN } y_i \text{ is } m_i \end{aligned} \tag{1}$$

where $x_i$ and $y_i$ are the input and output variables, respectively, $A_{in}$ is a fuzzy set, and $m_i$ is the position of a symmetric membership function of the output variable with its width neglected during the defuzzification process. This type of fuzzy rule is used as the main body of the SONFIN. We call a SONFIN consisting of such kind of rules a *basic SONFIN*. By monitoring the change of the network output errors, additional terms (the linear terms used in the consequent part of the TSK model) will be added when necessary to further reduce the output error. If it is decided that some auxiliary terms should be added to the consequent part during the on-line learning process, a projection-based correlation measure using Gram–Schmidt orthogonalization algorithm will be performed on each rule to select the most significant terms to be incorporated into the rule. This consequent identification process is

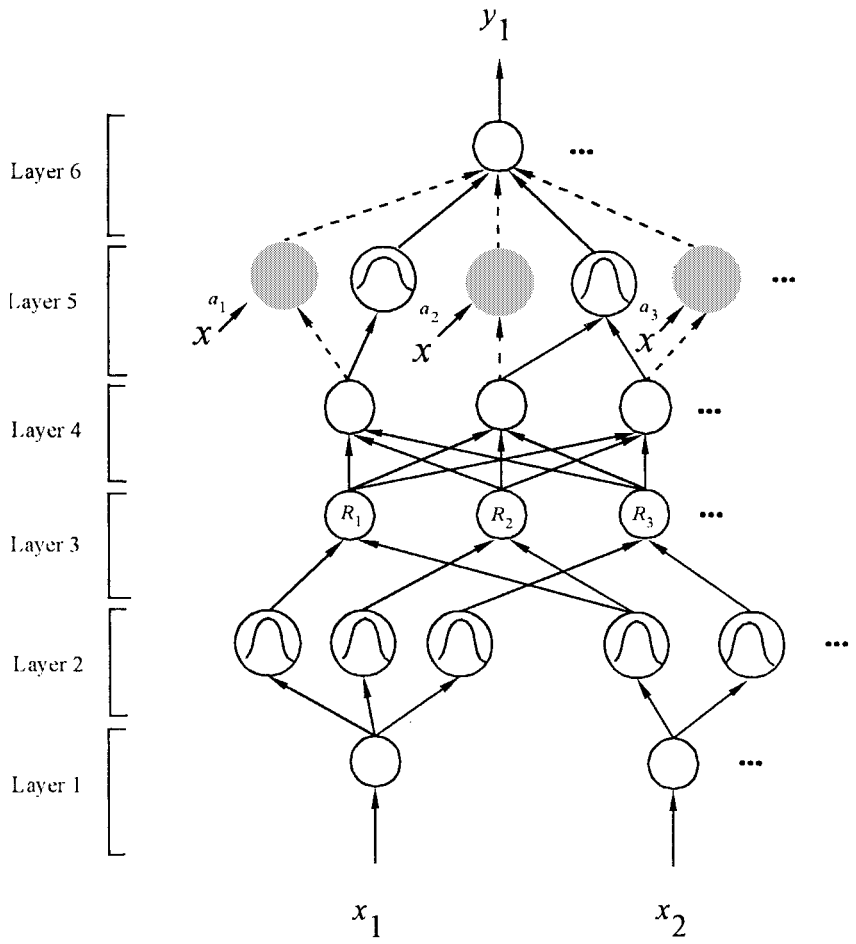



Fig. 2. Structure of the proposed SONFIN.

employed in conjunction with the precondition identification process to reduce both the number of rules and the number of consequent terms.

Associated with the structure identification scheme is the parameter identification scheme used in the SONFIN. In the parameter identification scheme, the consequent parameters (coefficients of the linear equations) are tuned by either least mean squares (LMS) or recursive least squares (RLS) algorithms and the precondition parameters (membership functions of input variables) are tuned by the backpropagation algorithm to meet the required output accuracy. Furthermore, to enhance the knowledge representation capability of the SONFIN, a linear transformation of the input variables can be incorporated into the network to further reduce the rule number or to achieve higher output accuracy. Proper linear transformation is also tuned automatically during the parameter learning phase. Both the structure and parameter learning are done simultaneously for each incoming training pattern to form a fast on-line learning scheme.

This paper is organized as follows. Section II describes the basic structure and functions of the SONFIN. The on-line structure/parameter learning algorithms of the SONFIN is presented in Section III. In Section IV, the SONFIN is applied to solve several problems covering the areas of control, communication, and signal processing. Finally, conclusions are summarized in the last section.

## II. Structure of the SONFIN

In this section, the structure of the SONFIN (as shown in Fig. 2) is introduced. This six-layered network realizes a fuzzy model of the following form:

$$\text{Rule } i\text{: IF } x_1 \text{is } A_{i1} \text{ and } \cdots \text{ and } x_n \text{ is } A_{in}$$
$$\text{THEN } y \text{ is } m_{0i} + a_{ji}x_j + \cdots$$

where $A_{ij}$ is a fuzzy set, $m_{0i}$ is the center of a symmetric membership function on $y$, and $a_{ji}$ is a consequent parameter. It is noted that unlike the traditional TSK model where all the input variables are used in the output linear equation, only the significant ones are used in the SONFIN, i.e., some $a_{ij}$'s in the above fuzzy rules are zero. With this six-layered network structure of the SONFIN, we shall define the function of each node in Section II-A and then introduce an enhanced structure of the SONFIN in Section II-B.

### A. Structure of the SONFIN

The SONFIN consists of nodes, each of which has some finite "fan-in" of connections represented by weight values from other nodes and "fan-out" of connections to other nodes. Associated with the fan-in of a node is an integration function $f$, which serves to combine information, activation, or evidence from other nodes. This function provides the net input

for this node

$$\text{net-input} = f[u_1^{(k)}, u_2^{(k)}, \cdots, u_p^{(k)}; w_1^{(k)}, w_2^{(k)}, \cdots, w_p^{(k)}]$$

where $u_1^{(k)}, u_2^{(k)}, \cdots, u_p^{(k)}$ are inputs to this node and $w_1^{(k)}, w_2^{(k)}, \cdots, w_p^{(k)}$ are the associated link weights. The superscript $(k)$ in the above equation indicates the layer number. This notation will also be used in the following equations. A second action of each node is to output an activation value as a function of its net-input

$$\text{output} = o_i^{(k)} = a(\text{net-input}) = a(f)$$

where $a(\cdot)$ denotes the activation function. We shall next describe the functions of the nodes in each of the six layers of the SONFIN.

*Layer 1:* No computation is done in this layer. Each node in this layer, which corresponds to one input variable, only transmits input values to the next layer directly. That is

$$f = u_i^{(1)}$$

and

$$a^{(1)} = f. \tag{2}$$

From the above equation, the link weight in layer one $[w_i^{(1)}]$ is unity.

*Layer 2:* Each node in this layer corresponds to one linguistic label (small, large, etc.) of one of the input variables in Layer 1. In other words, the membership value which specifies the degree to which an input value belongs a fuzzy set is calculated in Layer 2. There are many choices for the types of membership functions for use, such as triangular, trapezoidal, or Gaussian ones. In this paper, a Gaussian membership function is employed for two reasons. First, a fuzzy system with Gaussian membership function has been shown to be an universal approximator of any nonlinear functions on a compact set [16]. Second, a multidimensional Gaussian membership function generated during the learning process can be decomposed into the product of 1-D Gaussian membership functions easily. With the choice of Gaussian membership function, the operation performed in this layer is

$$f[u_{ij}^{(2)}] = -\frac{[u_i^{(2)} - m_{ij}]^2}{\sigma_{ij}^2}$$

and

$$a^{(2)}(f) = e^f \tag{3}$$

where $m_{ij}$ and $\sigma_{ij}$ are, respectively, the center (or mean) and the width (or variance) of the Gaussian membership function of the $j$th term of the $i$th input variable $x_i$. Hence, the link weight in this layer can be interpreted as $m_{ij}$. Unlike other clustering-based partitioning methods, where each input variable has the same number of fuzzy sets, the number of fuzzy sets of each input variable is not necessarily identical in the SONFIN.

*Layer 3:* A node in this layer represents one fuzzy logic rule and performs precondition matching of a rule. Here, we use the following AND operation for each Layer-3 node

$$f[u_i^{(3)}] = \prod_i u_i^{(3)} = e^{-[D_i(\mathbf{x}-\mathbf{m}_i)]^T[D_i(\mathbf{x}-\mathbf{m}_i)]}$$

and

$$a^{(3)}(f) = f \tag{4}$$

where $n$ is the number of Layer-2 nodes participating in the IF part of the rule $D_i = \text{diag}(1/\sigma_{i1}, 1/\sigma_{i2}, \cdots, 1/\sigma_{in})$ and $\mathbf{m}_i = (m_{i1}, m_{i2}, \cdots, m_{in})^T$. The link weight in Layer 3 $[w_i^{(3)}]$ is then unity. The output $f$ of a Layer-3 node represents the firing strength of the corresponding fuzzy rule.

*Layer 4:* The number of nodes in this layer is equal to that in Layer 3 and the firing strength calculated in Layer 3 is normalized in this layer by

$$f[u_i^{(4)}] = \sum_i u_i^{(4)}$$

and

$$a^{(4)}(f) = \frac{u_i^{(4)}}{f}. \tag{5}$$

Like Layer 3, the link weight $[w_i^{(4)}]$ in this layer is unity, too.

*Layer 5:* This layer is called the consequent layer. Two types of nodes are used in this layer and they are denoted as blank and shaded circles in Fig. 2, respectively. The node denoted by a blank circle (blank node) is the essential node representing a fuzzy set (described by a Gaussian membership function) of the output variable. Only the center of each Gaussian membership function is delivered to the next layer for the local mean of maximum (LMOM) defuzzification operation [23] and the width is used for output clustering only. Different nodes in Layer 4 may be connected to a same blank node in Layer 5, meaning that the same consequent fuzzy set is specified for different rules. The function of the blank node is

$$f = \sum_i u_i^{(5)}$$

and

$$a^{(5)}(f) = f \cdot a_{0i} \tag{6}$$

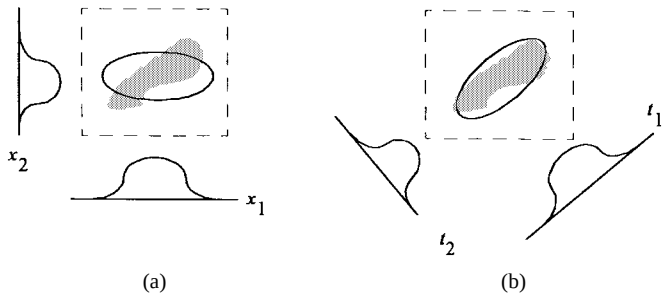
where $a_{0i} = m_{0i}$, the center of a Gaussian membership function. As to the shaded node, it is generated only when necessary. Each node in Layer 4 has its own corresponding shaded node in Layer 5. One of the inputs to a shaded node is the output delivered from Layer 4 and the other possible inputs (terms) are the input variables from Layer 1. The shaded node function is

$$f = \sum_j a_{ji} x_j$$

and

$$a^{(5)}(f) = f \cdot u_i^{(5)} \tag{7}$$

where the summation is over the significant terms connected to the shaded node only, and $a_{ji}$ is the corresponding parameter.




(a) (b)

Fig. 3. (a) The region covered by the original input membership functions. (b) The covered region after space transformation.

Combining these two types of nodes in Layer 5, we obtain the whole function performed by this layer as

$$a^{(5)}(f) = \left(\sum_j a_{ji}x_j + a_{0i}\right)u_i^{(5)}. \tag{8}$$

*Layer 6:* Each node in this layer corresponds to one output variable. The node integrates all the actions recommended by Layer 5 and acts as a defuzzifier with

$$f[u_i^{(6)}] = \sum_i u_i^{(6)}$$

and

$$a^{(6)}(f) = f. \tag{9}$$

### B. Enhanced Structure of the SONFIN

For the structure of the SONFIN introduced in the last subsection, the region that the input membership functions cover is restricted to be of an elliptic shape with the axes of the ellipse parallel to the corresponding input coordinate axes as shown in Fig. 3(a). Such an elliptic region usually cannot cover the distribution of a cluster of input data well. For example, if the distribution of the input data is like the shaded region shown in Fig. 3(a) indicating that the input variables are highly correlated each other, then we usually need more than one rule to map such kind of input region to its corresponding output region. To use as fewer rules as possible, linear transformation is performed on the input variables in the SONFIN. The transformation can be regarded as a change of input coordinates, while the parameters of each membership function are kept unchanged, i.e., the center and width of each membership function on the new coordinate axes are the same as the old ones. In mathematical form, we have

$$\text{Rule } i\text{: } \mathbf{t}_i = R_i(x - \mathbf{m}_i) + \mathbf{m}_i \tag{10}$$

where $\mathbf{t}_i \in \Re^n$ are the newly generated input variables and $R_i \in \Re^{n\times n}$ is the transformation matrix for rule $i$. After transformation, the region that the input membership functions cover is shown in Fig. 3(b). It is observed that the rotated ellipse covers the input data distribution well and, thus, a single fuzzy rule can associate this region with its proper output region (consequent).

With the transformation of input coordinates, the firing strength calculated in Layer 3 (i.e., the function of a Layer-3 node) is changed to

$$\begin{aligned} f[u_i^{(3)}] &= \prod_i u_i^{(3)} \\ &= e^{-[D_i(\mathbf{t}_i - \mathbf{m}_i)]^T[D_i(\mathbf{t}_i - \mathbf{m}_i)]} \\ &= e^{-[D_iR_i(\mathbf{x} - \mathbf{m}_i)]^T[D_iR_i(\mathbf{x} - \mathbf{m}_i)]} \end{aligned}$$

and

$$a^{(3)}(f) = f. \tag{11}$$

It is noted that basically all the parameters of the transformation matrix in (10) are free parameters. However, if we set the additional constraint that $R_i^TR_i = I$, then in geometric view, the operation $R_i$ is equivalent to a rotation of the region covered by the original membership functions. In this situation, $\mathbf{m}_i$ is responsible for the location of the membership function $D_i$ for the spread and $R_i$ for the orientation of each input coordinate axis. After transformation, the rule in (1) becomes

$$\begin{aligned} \text{Rule } i\text{: IF } & t_{i1} = \sum_{k=1}^{n} r_{1k}^i(x_k - m_{ik}) + m_{i1} \text{ is } A_{i1} \text{ and } \cdots \\ & \text{and } t_{in} = \sum_{k=1}^{n} r_{nk}^i(x_k - m_{ik}) + m_{in} \text{ is } A_{in} \\ & \text{THEN } y_i \text{ is } m_i \end{aligned}$$

where $r_{nk}^i$ the $(n, k)$th element of $R^i$. The linguistic implication $A_{in}$ of the original variable $x_n$ is now implicated by the new variable $t_{ij}$ (see Fig. 3 for clarity), which is a linear combination of the original variables. Note that when $R = I$, then the transformed rules are the same as the original ones.

Generally speaking, the flexibility provided by $R_i$ can reduce the number of rules needed or can increase the modeling accuracy of the SONFIN. This transformation is extremely useful for low-input dimension problems. For high-input dimension problems, these advantages may be traded off by the additional memories required for storing $R_i$.

## III. Learning Algorithms for the SONFIN

Two types of learning—structure and parameter learning—are used concurrently for constructing the SONFIN. The structure learning includes both the precondition and consequent structure identification of a fuzzy if-then rule. Here the precondition structure identification corresponds to the input-space partitioning and can be formulated as a combinational optimization problem with the following two objectives: to minimize the number of rules generated and to minimize the number of fuzzy sets on the universe of discourse of each input variable. As to the consequent structure identification, the main task is to decide when to generate a new membership function for the output variable and which significant terms (input variables) should be added to the consequent part (a linear equation) when necessary. For the parameter learning based upon supervised learning algorithms, the parameters of the linear equations in the consequent parts are adjusted by either LMS or RLS algorithms and the parameters in the precondition part are adjusted by the backpropagation algorithm to minimize a given cost function. The SONFIN can be used for normal operation at any time

during the learning process without repeated training on the input–output patterns when on-line operation is required. There are no rules (i.e., no nodes in the network except the input–output nodes) in the SONFIN initially. They are created dynamically as learning proceeds upon receiving on-line incoming training data by performing the following learning processes simultaneously:

1) input/output space partitioning;
2) construction of fuzzy rules;
3) optimal consequent structure identification;
4) parameter identification.

In the above, learning process 1), 2), and 3) belong to the structure learning phase and 4) belongs to the parameter learning phase. The details of these learning processes are described in the rest of this section.

### A. Input–Output Space Partitioning

The way the input space is partitioned determines the number of rules extracted from training data as well as the number of fuzzy sets on the universal of discourse of each input variable. Geometrically, a rule corresponds to a cluster in the input space, with $\mathbf{m}_i$ and $D_i$ representing the center and variance of that cluster. For each incoming pattern $\mathbf{x}$ the strength a rule is fired can be interpreted as the degree the incoming pattern belongs to the corresponding cluster. For computational efficiency, we can use the firing strength derived in (4) directly as this degree measure

$$F^i(\mathbf{x}) = \prod_i u_i^{(3)} = e^{-[D_i(\mathbf{x}-\mathbf{m}_i)]^T[D_i(\mathbf{x}-\mathbf{m}_i)]} \tag{12}$$

where $F^i \in [0, 1]$. In the above equation, the term $[D_i(\mathbf{x}-\mathbf{m}_i)]^T[D_i(\mathbf{x}-\mathbf{m}_i)]$ is, in fact, the distance between $\mathbf{x}$ and the center of cluster $i$. Using this measure, we can obtain the following criterion for the generation of a new fuzzy rule. Let $\mathbf{x}(t)$ be the newly incoming pattern. Find

$$J = \arg \max_{1\le j\le c(t)} F^j(\mathbf{x}) \tag{13}$$

where $c(t)$ is the number of existing rules at time $t$. If $F^J \le \overline{F}(t)$, then a new rule is generated where $\overline{F}(t) \in (0, 1)$ is a prespecified threshold that decays during the learning process. Once a new rule is generated, the next step is to assign initial centers and widths of the corresponding membership functions. Since our goal is to minimize an objective function and the centers and widths are all adjustable later in the parameter learning phase, it is of little sense to spend much time on the assignment of centers and widths for finding a perfect cluster. Hence, we can simply set

$$\mathbf{m}_{[c(t)+1]} = \mathbf{x} \tag{14}$$

$$D_{[c(t)+1]} = \frac{-1}{\beta} \cdot \operatorname{diag}\left[\frac{1}{\ln(F^J)} \cdots \frac{1}{\ln(F^J)}\right] \tag{15}$$

according to the first-nearest-neighbor heuristic [10] where $\beta \ge 0$ decides the overlap degree between two clusters. Similar methods are used in [18], [19] for the allocation of a new radial basis unit. However, in [18] the degree measure doesn't take the width $D$ into consideration. In [19], the width of each unit is kept at a prespecified constant value, so the allocation result is, in fact, the same as that in [18]. In the SONFIN, the width is taken into account in the degree measure, so for a cluster with larger width (meaning a larger region is covered), fewer rules will be generated in its vicinity than a cluster with smaller width. This is a more reasonable result. Another disadvantage of [18] is that another degree measure (the Euclid distance) is required, which increases the computation load.

After a rule is generated, the next step is to decompose the multidimensional membership function formed in (14) and (15) to the corresponding 1-D membership function for each input variable. For the Gaussian membership function used in the SONFIN, the task can be easily done as

$$e^{-[D_i(\mathbf{x}-\mathbf{m}_i)]^T[D_i(\mathbf{x}-\mathbf{m}_i)]} = \prod_j e^{-[(x_j-m_{ij})^2/\sigma_{ij}^2]} \tag{16}$$

where $m_{ij}$ and $\sigma_{ij}$ are, respectively, the projected center and width of the membership function in each dimension. To reduce the number of fuzzy sets of each input variable and to avoid the existence of highly similar ones, we should check the similarities between the newly projected membership function and the existing ones in each input dimension. Before going to the details on how this overall process works, let us consider the similarity measure first. Since bell-shaped membership functions are used in the SONFIN, we use the formula of the similarity measure of two fuzzy sets with bell-shaped membership functions derived previously in [11]. Suppose the fuzzy sets to be measured are fuzzy sets $A$ and $B$ with membership function $\mu_A(x) = \exp\{-(x - m_1)^2/\sigma_1^2\}$ and $\mu_B(x) = \exp\{-(x - m_2)^2/\sigma_2^2\}$, respectively. Assume $m_1 \ge m_2$ as in [11], we can compute $|A \cap B|$ by

$$\begin{aligned}|A\cap B| = {} & \frac{1}{2}\frac{h^2[m_2 - m_1 + \sqrt{\pi}(\sigma_1 + \sigma_2)]}{\sqrt{\pi}(\sigma_1 + \sigma_2)} \\ & + \frac{1}{2}\frac{h^2[m_2 - m_1 + \sqrt{\pi}(\sigma_1 - \sigma_2)]}{\sqrt{\pi}(\sigma_2 - \sigma_1)} \\ & + \frac{1}{2}\frac{h^2[m_2 - m_1 - \sqrt{\pi}(\sigma_1 + \sigma_2)]}{\sqrt{\pi}(\sigma_1 - \sigma_2)}\end{aligned} \tag{17}$$

where $h(x) = \max\{0, x\}$. So the approximate similarity measure is

$$E(A, B) = \frac{|A\cap B|}{|A\cup B|} = \frac{|A\cap B|}{\sigma_1\sqrt{\pi} + \sigma_2\sqrt{\pi} - |A\cap B|} \tag{18}$$

where we use the fact that $|A| + |B| = |A \cap B| + |A \cup B|$.

Let $\mu(m_i, \sigma_i)$ represent the Gaussian membership function with center $m_i$ and width $\sigma_i$. The whole algorithm for the generation of new fuzzy rules as well as fuzzy sets in each

input variable is as follows. Suppose no rules are existent initially:

IF $\mathbf{x}$ is the first incoming pattern THEN do
*PART 1.* { Generate a new rule
with center $\mathbf{m}_1 = \mathbf{x}$
width $D_1 = \text{diag}\left(\frac{1}{\sigma_{\text{init}}}, \cdots, \frac{1}{\sigma_{\text{init}}}\right)$
where $\sigma_{\text{init}}$ is a prespecified constant.
After decomposition, we have $n$ one-dimensional membership functions, with $m_{1i} = x_i$ and $\sigma_{1i} = \sigma_{\text{init}}$, $i = 1 \cdots n$.
}
ELSE for each newly incoming $\mathbf{x}$, do
*PART 2.* { find $J = \arg\max_{1 \le j \le c(t)} F^j(\mathbf{x})$,
IF $F^J \ge \overline{F}_{\text{in}}(t)$
do nothing
ELSE
$\{c(t+1) = c(t) + 1$
generate a new fuzzy rule, with
$\mathbf{m}_{c(t+1)} = \mathbf{x}$, $D_{c(t+1)} =$
$\frac{-1}{\beta} \cdot \text{diag}\left[\frac{1}{\ln(F^J)} \cdots \frac{1}{\ln(F^J)}\right]$.
After decomposition, we have
$m_{\text{new}-\mathbf{i}} = x_i$, $\sigma_{\text{new}-\mathbf{i}} = -\beta \cdot \ln(F^J)$, $i = 1 \cdots n$.
Do the following fuzzy measure for each input variable $i$:
{degree$(i, t) \equiv$
$\max_{1 \le j \le k_i} E[\mu(m_{\text{new}-\mathbf{i}}, \sigma_{\text{new}-\mathbf{i}})$
$\mu(m_{ji}, \sigma_{ji})]$,
where $k_i$ is the number of partitions of the $i$th input variable.
IF degree$(i, t) \le \rho(t)$
THEN adopt this new membership function, and set $k_i = k_i + 1$
ELSE set the projected membership function as the closest one. }
}

In the above algorithm, the threshold $\overline{F}_{\text{in}}$ determines the number of rules generated. For a higher value of $\overline{F}_{\text{in}}$, more rules are generated and, in general, a higher accuracy is achieved. $\overline{F}_{\text{out}}$ determines the number of output clusters generated and a higher value of $\overline{F}_{\text{out}}$ will result in a higher number of output clusters. $\rho(t)$ is a scalar similarity criterion which is monotonically decreasing such that higher similarity between two fuzzy sets is allowed in the initial stage of learning. For the output space partitioning, the same measure in (13) is used. Since the criterion for the generation of a new output cluster is related to the construction of a rule, we shall describe it together with the rule construction process in learning process $B$ below.

It is mentioned in Section II-B that we can enhance the performance of the SONFIN by incorporating a transformation matrix $R$ into the structure. To construct the transformation matrix, if we have no *a priori* knowledge about it, we can simply set the matrix to be an identity one initially for a new generated rule. The identity assignment means that the transformed rule is the same as the original one (without transformation) initially and the influence of the transformation starts when afterward parameter learning is performed. However, if we have *a priori* knowledge about the transformation matrix, e.g., from the distribution of the input data as shown in Fig. 3, we can incorporate this transformation into the rule initially.

### B. Construction of Fuzzy Rules

As mentioned in learning process 1), the generation of a new input cluster corresponds to the generation of a new fuzzy rule, with its precondition part constructed by the learning algorithm in process 1). At the same time, we have to decide the consequent part of the generated rule. Suppose a new input cluster is formed after the presentation of the current input–output training pair ($\mathbf{x}$, $\mathbf{d}$); then the consequent part is constructed by the following algorithm:

IF there are no output clusters
do { *PART 1* in Process A,
with $\mathbf{x}$ replaced by $\mathbf{d}$ }
ELSE
do {
find $J = \arg\max_j F^j(\mathbf{x})$
IF $F^J \ge \overline{F}_{\text{out}}(t)$
connect input cluster $c(t+1)$ to the existing output cluster $J$
ELSE
generate a new output cluster
connect input cluster $c(t+1)$ to the newly generated output cluster.
}.

The algorithm is based on the fact that different preconditions of different rules may be mapped to the same consequent fuzzy set. Since only the center of each output membership function is used for defuzzification, the consequent part of each rule may simply be regarded as a singleton. Compared to the general fuzzy rule-based models with singleton output where each rule has its own individual singleton value, fewer parameters are needed in the consequent part of the SONFIN, especially for the case with a large number of rules.

### C. Optimal Consequent Structure Identification

Up until now, the SONFIN contains fuzzy rules in the form of (1). Even though such a basic SONFIN can be used directly for system modeling, a large number of rules are necessary for modeling sophisticated systems under a tolerable modeling accuracy. To cope with this problem, we adopt the spirit of TSK model [21] into the SONFIN. In the TSK model, each consequent part is represented by a linear equation of the input variables. It is reported in [20] that the TSK model can model a sophisticated system using a few rules. However, even for the TSK model, if the dimension of the input or output space is high, the number of terms used in the linear equation is large even though some terms are, in fact, of little significance. Hence, instead of using the linear combination of all the input variables as the consequent part, only the most significant input

variables are used as the consequent terms of the SONFIN. The significant terms will be chosen and added to the network incrementally any time when the parameter learning cannot improve the network output accuracy any more during the on-line learning process. To find the significant terms used in the consequent part, we shall first discuss some strategies that can be used on-line for this purpose before present our own approach.

*1) Sensitivity Calculation Method [26]:* This is a network pruning method based on the estimation of the sensitivity of the global cost function to the elimination of each connection. To use this method, all the input variables are used in the linear equation of the consequent part. After a period of learning, the coefficients of the linear equation are ordered by decreasing sensitivity values so that the least significant terms in the linear equation can be efficiently pruned by discarding the last terms on the sorted list. One disadvantage of this method is that the correlation between candidate terms is not detected. Hence, after a term is removed the remaining sensitivities are not necessarily valid for the pruned network and the whole sensitivity estimation process should be performed from the beginning again.

*2) Weight Decay Method [26]:* This is also a network pruning method. Like strategy 1, to adopt this method all input variables are used in the linear equation of the consequent part initially. By adding a penalty term to the cost function, the weights of the fewer significant terms will decay to zero under backpropagation learning. The disadvantage of this method is that only the backpropagation learning algorithm can be used and, thus, the computation time is quite long for a weight to decay to zero. Besides, the terms with higher weights are not necessarily the most significant ones and, thus, this method usually chooses more terms than necessary.

*3) Competitive Learning:* The link weight (coefficient of consequent linear equation) $a_{ji}$ can be considered as an indicator of the correlation strength between the input variables and output variables in the consequent part. The competitive learning rule can thus be used to update $a_{ji}$ [10]. After competitive learning, the terms with larger weights are kept and those with smaller ones are eliminated. As in strategy 1, no correlation between the inputs is considered, so the result is not optimal.

In the choice of the significant terms participated in the consequent part, since the dependence between the candidates $u_i^{(5)} \cdot x_j$ and the desired output $y_m$ is linear [$y_m = \sum_i u_i^{(5)}(\sum_j a_{ji}^m x_j)$], we can consider the training sequences $u_i^{(5)}x_j(1), u_i^{(5)}x_j(2), \cdots, u_i^{(5)}x_j(t)$ and $y_m(1), y_m(2), \cdots, y_m(t)$ as vectors and find the correlation between $\hat{\mathbf{x}}_j = u_i^{(5)}[x_j(1), \cdots, x_j(t)]^T$ and $[y_m(1), \cdots, y_m(t)]^T$. The correlation between two vectors $\hat{\mathbf{x}}_j$ and $\mathbf{y}$ is estimated by the cosine value of their angle $\theta$, $\mathrm{Deg}(j) = \cos^2(\theta) = (\hat{\mathbf{x}}_j^T\mathbf{y})^2/(\hat{\mathbf{x}}_j^T\hat{\mathbf{x}}_j)(\mathbf{y}^T\mathbf{y})$. If vectors $\hat{\mathbf{x}}_j$ and $\mathbf{y}$ are dependent, then $\mathrm{Deg}(j) = 1$, otherwise if $\hat{\mathbf{x}}_j$ and $\mathbf{y}$ are orthogonal then $\mathrm{Deg}(j) = 0$. The main idea of the choice scheme is as follows. Suppose we have chosen $k-1$ vectors from $n$ (the number of input variables) candidates to form a space $P_{k-1} = \mathbf{p}_1 \oplus \mathbf{p}_2 \oplus \cdots \oplus \mathbf{p}_{k-1}$. To find the next important vector from the remaining $n-k+1$ vectors, we first project each of the remaining $n-k+1$ vectors to the null space of $P_{k-1}$, find the correlation value $Deg$ between the $n-k+1$ projected vectors and $\mathbf{y}$, then choose the maximum one which is the $k$th important term of the $n$ candidates, and finally set $P_k = \mathbf{p}_1 \oplus \mathbf{p}_2 \oplus \cdots \oplus \mathbf{p}_k$. Here, $\mathbf{p}_1 = \hat{\mathbf{x}}_0$ is the vector formed by the essential singleton values. To find the projected vector $\mathbf{p}_k$, the Gram–Schmidt orthogonalization procedure [27] is adopted as

$$\alpha_{ik} = \frac{\mathbf{p}_i^T\hat{\mathbf{x}}_k}{\mathbf{p}_i^T\mathbf{p}_i} \tag{19}$$

$$\mathbf{p}_k = \hat{\mathbf{x}}_k - \sum_{i=1}^{k-1}\alpha_{ik}\mathbf{p}_i. \tag{20}$$

If there are $c$ rules, then we have $cn$ candidate vectors, a large number that may lead to high computation load in the calculation of the projected vectors in the above. To reduce the computation cost and to keep the parallel-processing advantage assumed in fuzzy rule-based systems, the terms in the consequent part are selected independently for each rule; that is, the projection operation is done only for the $n$ candidate vectors in each rule, not for other rules. This computation process is based upon the local property of a fuzzy rule-based system, so the vectors from other rules usually have less influence than the vectors in the same rule and are ignored for computational efficiency.

For on-line learning, to calculate the correlation degree, we have to store all the input/output sequences before these degrees are calculated. To do this, the size of memory required is of order $O(Nnt+Mt)$, where $N$, $n$, and $M$ are the number of rules, input, and output variables, respectively. Hence, the memory requirement is huge for large $t$. To cope with this problem, instead of storing the input–output sequences, we store the correlation values only. Let $C^i_{x_jy_m}$ denote the correlation between the sequence $u_i^{(5)}x_j$ and $y_m$, $C_{y_my_m}$ the auto correlation of the sequence $y_m$ and $C^i_{x_jx_p}$ the correlation between the sequence $u_i^{(5)}x_j$ and $u_i^{(5)}x_p$. For each incoming data, these values are on-line calculated, respectively, for each rule $i$ by

$$\begin{aligned} C^i_{x_jy_m}(t+1) = {} & C^i_{x_jy_m}(t) + \Gamma(t)[u_i^{(5)}(t+1)x_j(t+1) \\ & \cdot y_m(t+1) - C^i_{x_jy_m}(t)], \end{aligned} \tag{21}$$

$$\begin{aligned} C_{y_my_m}(t+1) = {} & C_{y_my_m}(t) + \Gamma(t)[y_m(t+1) \\ & \cdot y_m(t+1) - C_{y_my_m}(t)] \end{aligned} \tag{22}$$

$$\begin{aligned} C^i_{x_jx_p}(t+1) = {} & C^i_{x_jx_p}(t) + \Gamma(t)[u_i^{(5)}(t+1)x_j(t+1) \\ & \cdot u_i^{(5)}(t+1)x_p(t+1) - C^i_{x_jx_p}(t)] \end{aligned} \tag{23}$$

where $j, p = 0, \cdots, n$, $m = 1, \cdots, M$, and $C^i_{x_jy_m}(0)$, $C_{y_my_m}(0)$, and $C^i_{x_jx_p}(0)$ are initially equal to zero. For normal correlation computation, $\Gamma(t) = 1/(t+1)$ is used, but for computation efficiency and for changing environment where the recent calculations dominate, a constant value, say $0 < \Gamma < 1$, can be used. Using the stored correlation values in (21) and (22), we can compute the correlation values and choose the significant ones. The

algorithm for this computation is described as follows. In the following, $K$ denotes the number of terms to be selected from the $n$ candidates, $s_1, \cdots, s_{k-1}$, denotes the terms already selected by the algorithm, and $s_0 = 0$ denotes the essential singleton term $x_0$ for each rule.

*Projection-Based Correlation Measure Algorithm*

For each rule $I$ and output variable $j$ do

{

For $k = 1 \cdots K$

For $i = 1 \cdots n$, $i \neq s_1, \cdots, s_{k-1}$

Compute

$$\alpha(m, i) = \frac{A(m,i)}{B(m,m)}, \quad 0 \le m \le k-1 \tag{24}$$

$$E(i, j) = C^I_{x_i y_j} - \sum_{m=0}^{k-1} \alpha(m, i) E(s_m, j) \tag{25}$$

$$G(i, i) = C^I_{x_i x_i} - 2 \sum_{m=0}^{k-1} \alpha(m, i) A(m, i) + \sum_{m=0}^{k-1} \sum_{q=0}^{k-1} \alpha(m, i)\alpha(q, i) B(m, q) \tag{26}$$

$$Deg_k(i) = \frac{E^2(i,j)}{G(i,i) C_{y_j y_j}} \tag{27}$$

where

$$E(s_0, j) = C^I_{x_0 y_j} \tag{28}$$

$$A(m, i) = C^I_{x_{s_m} x_i} - \sum_{\ell=0}^{m-1} \alpha(\ell, s_m) A(\ell, i) \tag{29}$$

$$B(m, q) = \begin{cases} C^I_{x_0 x_0}, & m = q = 0 \\ G(s_m, s_m), & m = q \neq 0 \\ A(q, s_m) - \sum_{\ell=0}^{m-1} \alpha(\ell, s_m) \cdot B(\ell, q), & 0 \le q \le m-1 \\ B(q, m), & 0 \le m \le q-1. \end{cases} \tag{30, 31}$$

Then find $s_k \in \{1, 2, \cdots, n\}$ such that

$$\mathrm{Deg}_k(s_k) = \max_{i=1,\cdots n; i \neq s_1, \cdots, i \neq s_{k-1}} [\mathrm{Deg}_k(i)]. \tag{32}$$

The procedure is terminated at the $K$th step when

$$\mathrm{Deg}_K(s_K) \le \gamma \tag{33}$$

where $0 \le \gamma \le 1$ is the tolerable dependence degree and $K$ terms are added to the consequent part of the rule.

The consequent structure identification scheme in the SONFIN is a kind of node growing method in neural networks. For the node growing method, in general, there is a question of when to perform node growing. The criterion used in the SONFIN is by monitoring the learning curve. When the effect of parameter learning diminished (i.e., the output error does not decrease over a period of time), then it is the time to apply the above algorithm to add additional terms to the consequent part. Some other methods for selecting significant consequent terms are proposed in [2] and [24]. In [2], the forward selection of variables (FSV) method is proposed. In this method, different combinations of candidate consequent terms (input variables) are chosen for test. Finally, the one resulting in the minimal error is chosen. This method requires repeated tests and is inefficient. In [24], the OLS learning algorithm is used to select the significant terms as well as their corresponding coefficients. To use this method, a block of measured data should be prepared and the input space should be partitioned in advance and kept unchanged. This is not suitable for on-line operation. Moreover, to store the input/output sequence, the memory required is of order $O(nt + Mt)$ for each rule.

### D. Parameter Identification

After the network structure is adjusted according to the current training pattern, the network then enters the parameter identification phase to adjust the parameters of the network optimally based on the same training pattern. Notice that the following parameter learning is performed on the whole network after structure learning, no matter whether the nodes (links) are newly added or are existent originally. The idea of backpropagation is used for this supervised learning. Considering the single-output case for clarity, our goal is to minimize the error function

$$E = \tfrac{1}{2}[y(t) - y^d(t)]^2 \tag{34}$$

where $y^d(t)$ is the desired output and $y(t)$ is the current output. For each training data set, starting at the input nodes, a forward pass is used to compute the activity levels of all the nodes in the network to obtain the current output $y(t)$. Then, starting at the output nodes, a backward pass is used to compute $\partial E/\partial w$ for all the hidden nodes. Assuming that $w$ is the adjustable parameter in a node (e.g., $a_{ji}$, $m_{ij}$, and $\sigma_{ij}$ in the SONFIN), the general update rule used is

$$\Delta w \propto -\frac{\partial E}{\partial w} \tag{35}$$

$$w(t+1) = w(t) + \eta\left(-\frac{\partial E}{\partial w}\right) \tag{36}$$

where $\eta$ is the learning rate and

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial(\text{activation function})} \frac{\partial(\text{activation function})}{\partial w} = \frac{\partial E}{\partial a} \frac{\partial a}{\partial w}. \tag{37}$$

To show the learning rules, we shall show the computations of $\partial E/\partial w$ layer by layer, and start the derivation from the output nodes.

*Layer 6:* There is no parameter to be adjusted in this layer. Only the error signal $[\delta_i^{(6)}]$ needs to be computed and propagated. The error signal $\delta_i^{(6)}$ is derived by

$$\delta_i^{(6)} = -\frac{\partial E}{\partial a^{(6)}} = y^d(t) - y(t). \tag{38}$$

*Layer 5:* Using (8) and (37), the update rule for $a_{ji}$ is

$$-\frac{\partial E}{\partial a_{ji}} = -\frac{\partial E}{\partial a^{(6)}} \frac{\partial a^{(6)}}{\partial a^{(5)}} \frac{\partial a^{(5)}}{\partial a_{ji}} \tag{39}$$

and

$$\frac{\partial a^{(6)}}{\partial a^{(5)}} = 1 \tag{40}$$

$$\frac{\partial a^{(5)}}{\partial a_{ji}} = x_j \sum_i u_i^{(5)} \tag{41}$$

where the summation is over the number of links from Layer 4 for the $i$th node. Hence, the parameter $a_{ji}$ is updated by

$$a_{ji}(t+1) = a_{ji}(t) + \eta[y^d(t) - y(t)]x_j \sum_i u_i^{(5)} \tag{42}$$

where $x_0 = 1$. In addition to the LMS-like algorithm in (42), to improve the learning speed, the RLS learning algorithm [30] can be used instead in Layers 5 and 6 (as below)

$$P(t+1) = \frac{1}{\lambda}\left[P(t) - \frac{P(t)\mathbf{u}^{(5)^T}(t+1)\mathbf{u}^{(5)}(t+1)P(t)}{\lambda + \mathbf{u}^{(5)^T}(t+1)P(t)\mathbf{u}^{(5)}(t+1)}\right] \tag{43}$$

$$\boldsymbol{a}(t+1) = \boldsymbol{a}(t) + P(t+1)\mathbf{u}^{(5)}(t+1)[y^d(t) - y(t)] \tag{44}$$

where $0 < \lambda \leq 1$ is the forgetting factor, $\mathbf{u}^{(5)}$ is the current input vector, $\boldsymbol{a}$ is the corresponding parameter vector, and $P$ is the covariance matrix. The initial parameter vector $\boldsymbol{a}(0)$ is determined in the structure learning phase and $P(0) = \sigma I$ where $\sigma$ is a large positive constant. To cope with changing environment, in general, $0.9 < \lambda < 1$ is used. Also, to avoid the unstable effect caused by a small $\lambda$, we may reset $P(t)$ as $P(t) = \sigma I$ after a period of learning.

No matter which tuning method (LMS or RLS) is used in Layers 5 and 6, the error propagated to the preceding layer is

$$\begin{aligned}\delta^{(5)} &= -\frac{\partial E}{\partial a^{(5)}} \\ &= y^d(t) - y(t).\end{aligned} \tag{45}$$

*Layer 4:* As in Layer 6, only the error signal need be computed in this layer. According to (5) and (37), this error signal can be derived by

$$\begin{aligned}\delta_i^{(4)} &= -\frac{\partial E}{\partial a^{(4)}} \\ &= \frac{\partial E}{\partial a^{(5)}} \frac{\partial a^{(5)}}{\partial a^{(4)}}\end{aligned} \tag{46}$$

where

$$\frac{\partial a^{(5)}}{\partial a^{(4)}} = \sum_j a_{ji} x_j. \tag{47}$$

If there are multiple outputs, then the error signal becomes $\delta_i^{(4)} = \sum_k \delta_k^{(4)}$ where the summation is performed over the consequents of a rule node; that is, the error of a rule node is the summation of the errors of its consequents.

*Layer 3:* As in Layer 4, only the error signal need be computed in this layer

$$\delta_i^{(3)} = -\frac{\partial E}{\partial a^{(3)}} \tag{48}$$

$$= -\sum_j \frac{\partial E}{\partial a_j^{(4)}} \frac{\partial a_j^{(4)}}{\partial a_i^{(3)}} \tag{49}$$

where

$$\frac{\partial a_j^{(4)}}{\partial a_i^{(3)}} = \begin{cases} \dfrac{\sum_i a_i^{(3)} - a_j^{(3)}}{\left[\sum_{i=1}^{c} a_i^{(3)}\right]^2}, & \text{if } j = i \\ \dfrac{-a_j^{(3)}}{\left[\sum_{i=1}^{c} a_i^{(3)}\right]^2}, & \text{if } j \neq i. \end{cases} \tag{50}$$

So, we have

$$\delta_i^{(3)} = \sum_j \delta_j^{(4)} \frac{\partial a_j^{(4)}}{\partial a_i^{(3)}}. \tag{51}$$

*Layer 2:* Using (3) and (37), the update rule of $m_{ij}^{(2)}$ is derived as in the following:

$$-\frac{\partial E}{\partial m_{ij}^{(2)}} = \frac{\partial E}{\partial a^{(3)}} \sum_k \frac{\partial a^{(3)}}{\partial a_k^{(2)}} \frac{\partial a_k^{(2)}}{\partial m_{ij}^{(2)}} \tag{52}$$

where

$$\frac{\partial a^{(3)}}{\partial a_k^{(2)}} = \frac{a^{(3)}}{a_k^{(2)}} \tag{53}$$

$$\frac{\partial a_k^{(2)}}{\partial m_{ij}^{(2)}} = \begin{cases} a_k^{(2)} \dfrac{2(x_i - m_{ij})}{\sigma_{ij}^2}, & \text{if term node } j \text{ is connected to rule node } k \\ 0, & \text{otherwise.} \end{cases} \tag{54}$$

So, the update rule of $m_{ij}^{(2)}$ is

$$m_{ij}^{(2)}(t+1) = m_{ij}^{(2)}(t) - \eta \frac{\partial E}{\partial m_{ij}^{(2)}}. \tag{55}$$

Similarly, using (3) and (37), the update rule of $\sigma_{ij}^{(2)}$ is derived as

$$-\frac{\partial E}{\partial \sigma_{ij}^{(2)}} = \frac{\partial E}{\partial a^{(3)}} \sum_k \frac{\partial a^{(3)}}{\partial a_k^{(2)}} \frac{\partial a_k^{(2)}}{\partial \sigma_{ij}^{(2)}} \tag{56}$$

where

$$\frac{\partial a_k^{(2)}}{\partial \sigma_{ij}^{(2)}} = \begin{cases} a_k^{(2)} \dfrac{2(x_i - m_{ij})^2}{\sigma_{ij}^3}, & \text{if term node } j \text{ is connected to rule node } k \\ 0, & \text{otherwise.} \end{cases} \tag{57}$$

So, the update rule of $\sigma_{ij}^{(2)}$ is

$$\sigma_{ij}^{(2)}(t+1) = \sigma_{ij}^{(2)}(t) - \eta \frac{\partial E}{\partial \sigma_{ij}^{(2)}} \tag{58}$$

If the transformation matrix $R$ introduced in Section II-B is used, then for rule $i$ we have

$$\begin{aligned} a^{(3)}(f) &= e^{-[D_i R_i(\mathbf{x}-\mathbf{m}_i)]^T [D_i R_i(\mathbf{x}-\mathbf{m}_i)]} \\ &= e^{-\mathbf{s}^T \mathbf{s}} \\ &= \phi(\mathbf{s}) \end{aligned} \tag{59}$$

where $\mathbf{s} = D_i R_i(\mathbf{x}-\mathbf{m}_i)$. Then the update rules for $\mathbf{m}_i$, $\boldsymbol{\sigma}_i = (\sigma_{i1} \cdots \sigma_{in})$ and $R_i$ are

$$-\frac{\partial E}{\partial \mathbf{m}_i} = -\frac{\partial E}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial \mathbf{m}_i} \tag{60}$$

$$-\frac{\partial E}{\partial \boldsymbol{\sigma}_i} = -\frac{\partial E}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial \boldsymbol{\sigma}_i} \tag{61}$$

$$-\frac{\partial E}{\partial R_i} = -\frac{\partial E}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial R_i} \tag{62}$$

where

$$\frac{\partial a^{(3)}}{\partial \mathbf{m}_i} = R_i^T D_i \phi'(\mathbf{s}) \text{ and } \phi'(\mathbf{s}) = \frac{d\phi(\mathbf{s})}{d\mathbf{s}} \tag{63}$$

$$\frac{\partial a^{(3)}}{\partial \boldsymbol{\sigma}_i} = D_i^2 \text{diag}\,[R_i(\mathbf{x}-\mathbf{m}_i)]\phi'(\mathbf{s}) \tag{64}$$

$$\frac{\partial a^{(3)}}{\partial R_i} = -D_i \phi'(\mathbf{s})(\mathbf{x}-\mathbf{m}_i)^T. \tag{65}$$

Hence, we have

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) - \eta \frac{\partial E}{\partial \mathbf{m}_i} \tag{66}$$

$$\boldsymbol{\sigma}_i(t+1) = \boldsymbol{\sigma}_i(t) - \eta \frac{\partial E}{\partial \boldsymbol{\sigma}_i} \tag{67}$$

$$R_i(t+1) = R_i(t) - \eta \frac{\partial E}{\partial R_i}. \tag{68}$$

If the role $R_i$ played is the rotation operation, then after the updating in (68), $R_i$ may no longer satisfy the constraint $R_i^T R_i = I$. To modify $R_i$ to satisfy this constraint after tuning, the orthogonalization algorithm introduced in [29] can be adopted.

## IV. Simulations

To verify the performance of the SONFIN, several examples are presented in this section. These examples cover the areas of control, communication, and signal processing.

*Example 1—Identification of the Dynamic System:* In this example, the SONFIN is used to identify a dynamic system. The identification model has the form

$$\begin{aligned} \hat{y}(k+1) = \hat{f}[&u(k), u(k-1), \cdots, u(k-p+1) \\ &y(k), y(k-1), \cdots, y(k-q+1)]. \end{aligned} \tag{69}$$

Since both the unknown plant and the SONFIN are driven by the same input, the SONFIN adjusts itself with the goal of causing the output of the identification model to match that of the unknown plant. Upon convergence, their input–output relationship should match. The plant to be identified is guided by the difference equation

$$y(k+1) = \frac{y(k)}{1+y^2(k)} + u^3(k). \tag{70}$$

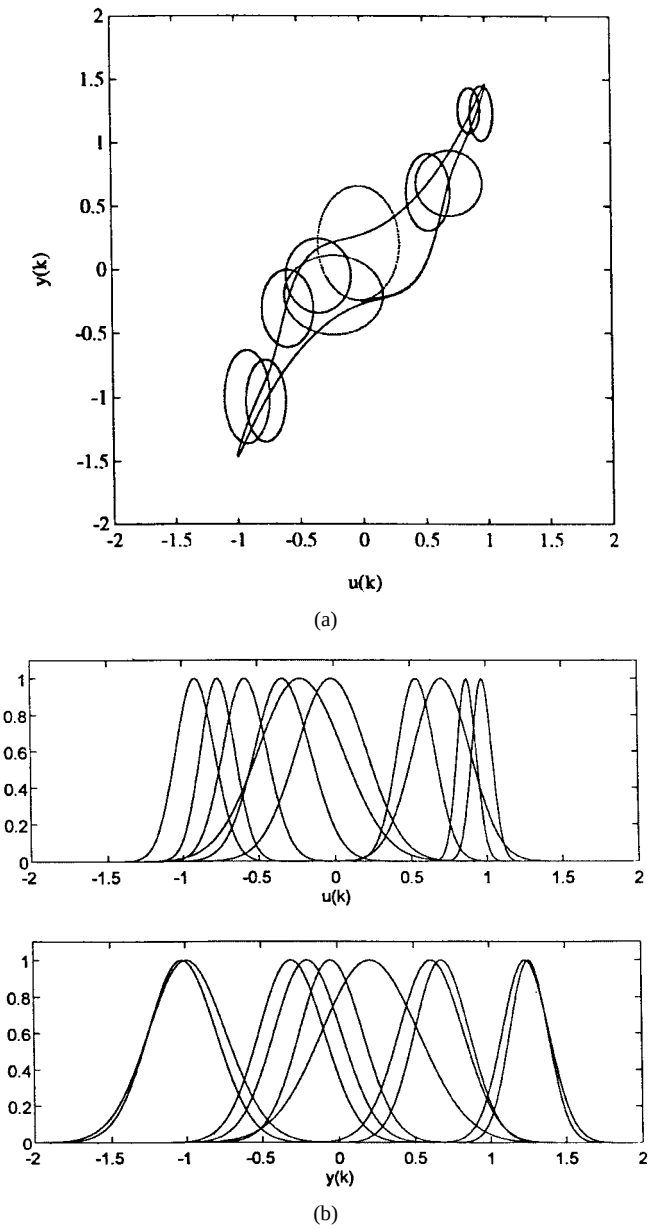



Fig. 4. Simulation results of the SONFIN without performing fuzzy measure on the membership functions of each input variable in Example 1. (a) The input training patterns and the final assignment of rules. (b) The distribution of the membership functions on the $u(k)$ and $y(k)$ dimensions.

The output of the plant depends nonlinearly on both its past values and inputs, but the effects of the input and output values are additive. In applying the SONFIN to this identification problem, the learning rate $\eta = 0.005$, $\overline{F}_{\text{in}} = 0.03$, $\overline{F}_{\text{out}} = 0.04$, and $\beta = 0.37$ are chosen, where $\overline{F}_{\text{in}}$ and $\overline{F}_{\text{out}}$ are the threshold parameters used in the input and output clustering processes, respectively. At first, we use the SONFIN without performing fuzzy measure on the membership functions of each input variable, so the number of fuzzy sets of each input variable is equal to the number of rules. The training patterns are generated with $u(k) = \sin(2\pi k/100)$. The training is performed for 50 000 time steps, where the consequent part is tuned by the LMS algorithm. After training, ten input and five output clusters are generated. Fig. 4(a) illustrates the

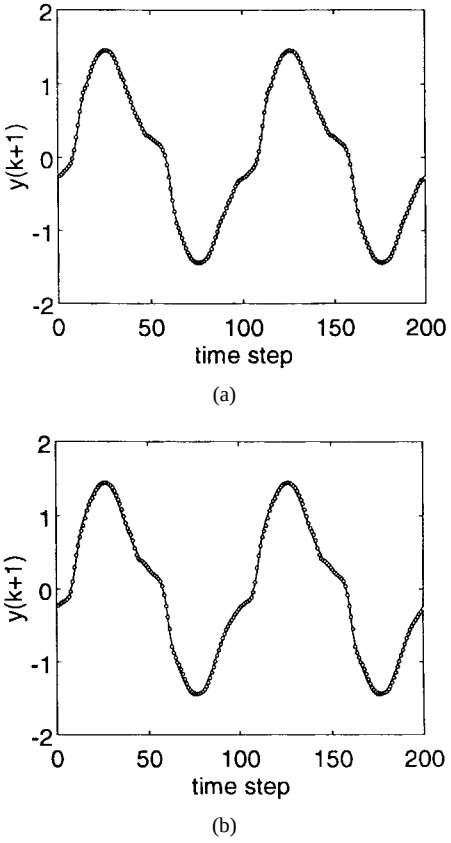



Fig. 5. Simulation results of the SONFIN in Example 1 where the dotted line denotes the output of the SOFIN and the solid line denotes the actual output. (a) Result without fuzzy measure performed. (b) Result with fuzzy measure performed.

distribution of the training patterns and the final assignment of fuzzy rules (i.e., distribution of input membership functions) in the $[u(k), y(k)]$ plain. In Fig. 4(a) and succeeding similar figures, the boundary of each ellipse represents a rule with firing strength $1/e$. The input data not covered by the ellipse are the data with a maximum corresponding firing strength less than $1/e$ but higher than $\overline{F}_{in}$, so no additional rules have to be generated to cover them. The corresponding membership functions on the $u(k)$ and $y(k)$ dimensions are shown in Fig. 4(b). From Fig. 4(b), we can see that some membership functions have high similarity degrees and some of them can be eliminated. Fig. 5(a) shows the outputs of the plant and the identification model after 50 000 time steps. In this figure, the outputs of the SONFIN are presented as the dotted curve while the plant outputs are presented as the solid curve. Since perfect identification result is achieved with the basic SONFIN [i.e., the SONFIN with fuzzy rules in the form of (1)], no additional terms need to be added to the consequent part.

In the above simulation, the parameter $\overline{F}_{\text{in}}$, $\overline{F}_{\text{out}}$, and $\beta$ need to be selected in advance. To give a clear understanding of the influence of these parameters on the structure and performance of the SONFIN, different values of them are tested. The generated network structure and corresponding root mean square (rms) errors are listed in Table I. From Table I, we can see that in certain ranges of the parameters, the rms error has no much change. A higher value of $\overline{F}_{\text{in}}$ results in a lower rms error at the cost of larger rule number. Since the

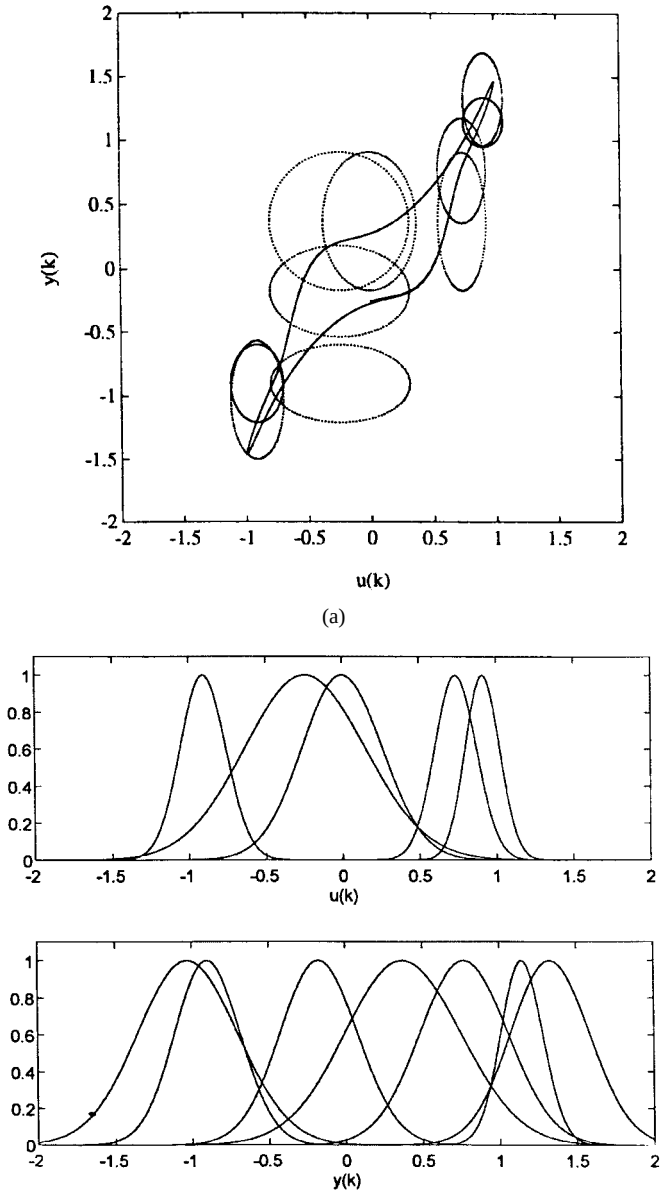



Fig. 6. Simulation results of the SONFIN with fuzzy measure performed on the membership functions of each input variable in Example 1. (a) The input training patterns and the final assignment of rules. (b) The distribution of the membership functions on the $u(k)$ and $y(k)$ dimensions where less membership functions are generated.

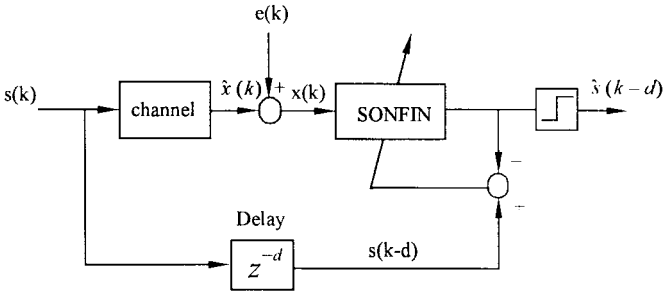



Fig. 7. The use of the SONFIN as an adaptive equalizer.

desired output values change smoothly, a low value of $\overline{F}_{\text{out}}$ resulting in small output clusters is enough. Since a higher value of $\beta$ means higher overlapping between rules and a

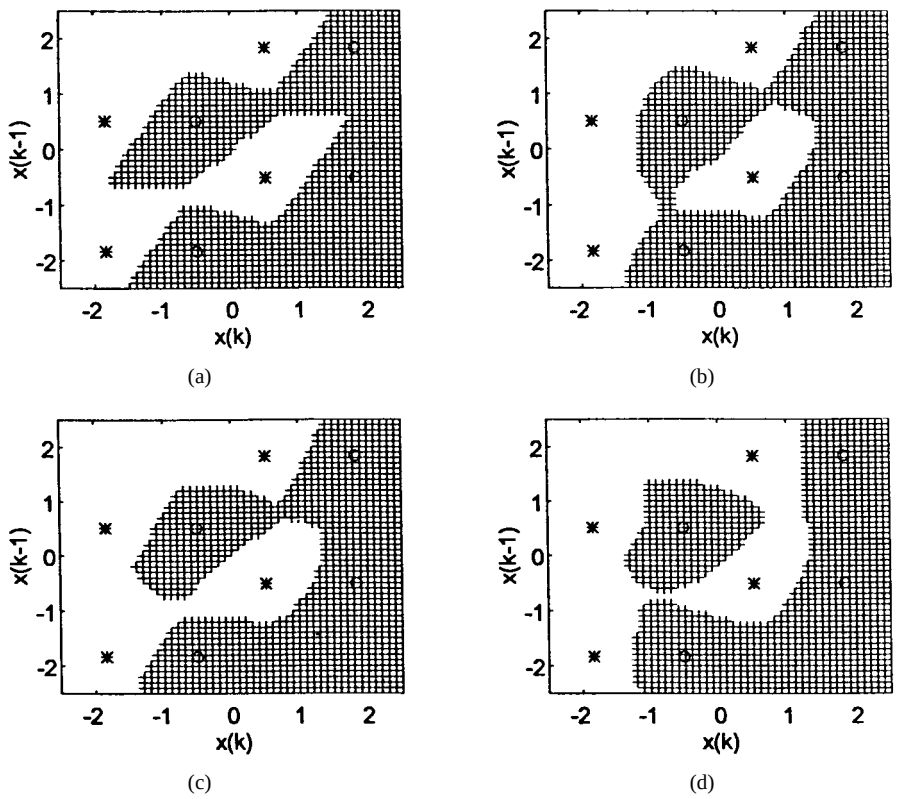



Fig. 8. Decision regions in Example 2. (a) Optimal decision region. (b) Decision region of network A. (c) Decision region of network B. (d) Decision region of network C.

TABLE I
INFLUENCES OF THE PARAMETERS. (a) $\overline{F}_{\text{out}}$, (b) $\overline{F}_{\text{in}}$, (c) $\overline{\beta}$ ON THE STRUCTURE OF THE SONFIN AND ITS CORRESPONDING RMS ERROR

| $F_{out}$* | 0.001 | 0.03 | 0.1 | 0.2 | 0.3 | 0.4 |
|---|---|---|---|---|---|---|
| (rule, output cluster) | (10, 3) | (10, 4) | (10, 5) | (10, 6) | (10, 8) | (10, 10) |
| RMS error | 0.031 | 0.014 | 0.013 | 0.013 | 0.018 | 0.022 |

* $F_{in}$ = 0.03, β = 0.37, ρ = 1

(a)

| $F_{in}$* | 0.0001 | 0.002 | 0.01 | 0.02 | 0.03 | 0.04 |
|---|---|---|---|---|---|---|
| (rule, output cluster) | (3, 3) | (4, 4) | (6, 3) | (7, 3) | (10, 4) | (13, 4) |
| RMS error | 0.065 | 0.042 | 0.024 | 0.024 | 0.016 | 0.014 |

* $F_{out}$ = 0.03, β = 0.37, ρ = 1

(b)

| β * | 0.33 | 0.47 | 0.57 | 0.77 | 0.87 | 0.97 |
|---|---|---|---|---|---|---|
| (rule, output cluster) | (15, 5) | (6, 3) | (5, 3) | (5, 3) | (5, 3) | (4, 2) |
| RMS error | 0.017 | 0.022 | 0.024 | 0.036 | 0.042 | 0.054 |

* $F_{in}$ = 0.03, $F_{out}$ = 0.03, ρ = 1

(c)

larger initial width is assigned to each rule, fewer rules are generated and larger rms error is obtained. From the table, we see that with the same structure a lower value of $\beta$ performs better on the matching task than a higher one.

To reduce the number of membership functions generated in each dimension, we have also applied the fuzzy-measure method on each input dimension during the learning process. The same training task as above is done and $\rho = 0.3$ is chosen. After 50 000 time steps of training, again, ten input and five output clusters are generated, but the number of fuzzy sets on the $u(k)$ and $y(k)$ dimensions are five and seven, respectively. Fig. 6(a) shows the distribution of the generated input clusters in the $[u(k), y(k)]$ plain. Fig. 6(b) shows the fuzzy sets on the $u(k)$ and $y(k)$ dimensions. The identification result is shown in Fig. 5(b), a result very similar to Fig. 5(a), while fewer fuzzy sets (input membership functions) are needed in total. The same structure is obtained when $\rho = 0.4$ is chosen. If $\rho = 0.5$ is chosen, then the number of fuzzy sets on the $u(k)$ and $y(k)$ dimensions are eight and ten, respectively.

*Example 2—Nonlinear Channel Equalization:* Nonlinear channel equalization is a technique used to combat some imperfect phenomenon (mainly refers to intersymbol interference in the presence of noise) in high-speed data transmission over channels like the high-speed modems [30]. The structure of the system is shown in Fig. 7. The transmitted input signal $s(k)$ is a sequence of statistically independent random binary symbols taking values of zero or one with equal probability. If $\hat{x}$ denotes the output of the channel, then the channel function can be described as

$$\hat{x}(k) = f[s(k), s(k-1), \cdots, s(k-N)]. \tag{71}$$

At the receiving end, a channel noise $e(k)$ presents and the observed signal $x(k)$ is

$$x(k) = \hat{x}(k) + e(k). \tag{72}$$

The task of the equalizer is to reconstruct the transmitted signal $s(k-d)$ from the observed information sequence $x(k), x(k-1), \cdots, x(k-N+1)$ (where $d$ and $N$ denote the

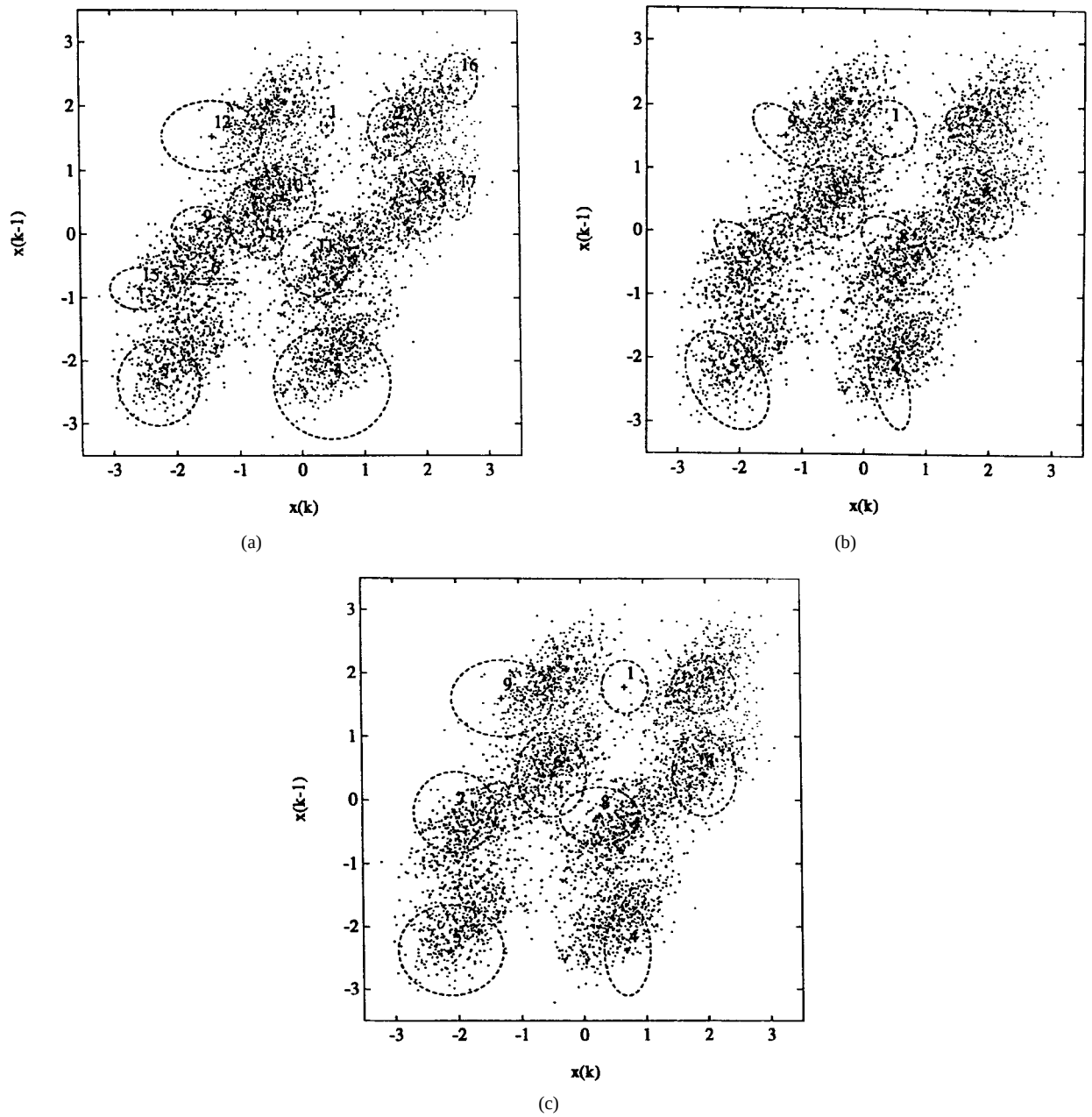



Fig. 9. The input training patterns and the final assignment of rules in Example 2, where the labeled number of each rule denotes the generation order. (a) Rule assignment in network A where 17 rules are generated. (b) Rule assignment in network B where nine rules are generated with transformation operation incorporated in each rule. (c) Rule assignment in network C where nine rules are generated.

lag and order, respectively) such that greater speed and higher reliability can be achieved. Suppose the channel function is

$$x(k) = \hat{O}(k) + 0.1\hat{O}^3(k) \tag{73}$$

where $\hat{O}(k) = 0.5s(k) + s(k-1)$ and noise $e(k)$ is zero-mean colored Gaussian distributed with $E[e^2(k)] = 0.2$ and $E[e(k)e(k-1)] = 0.1$. Suppose $d = 0$ and $N = 2$. The optimal decision boundary is plotted in Fig. 8(a). Choosing $\eta = 0.005$, $\beta = 0.25$, $\overline{F}_{\text{in}} = 0.015$, $\overline{F}_{\text{out}} = 1$, and $\rho = 1$ (i.e., no fuzzy measure performed on each input dimension), we trained the SONFIN for each on-line incoming training pattern and stopped the training at $k = 5000$. Since the desired output is either 1 or $-1$, there are only two clusters centered at $m_0 = 1$ and $m_1 = -1$ in the output space. Hence, during the training we tune the parameters in the precondition part only and keep the two consequent parameters $m_0$ and $m_1$ unchanged. After training, 17 clusters (rules) are generated. Fig. 9(a) illustrates the distribution of the 5000 training patterns as well as the generated clusters. The decision region of the trained SONFIN (network $A$) is shown in Fig. 8(b). To see the actual bit-error rate (BER), a realization of $10^6$ points of the sequence $s(k)$ and $e(k)$ are used to test the BER of the trained network. The resulting BER curve of the network is shown in Fig. 10, denoted by "o."

To reduce the number of rules generated, we incorporate the transformation matrix $R_i$ for each rule into the SONFIN. Initially, the transformation matrix $R_i$ is set as an identity matrix for each rule. Choosing $\eta = 0.005$, $\overline{F}_{in} = 0.002$, $\overline{F}_{out} = 1$,

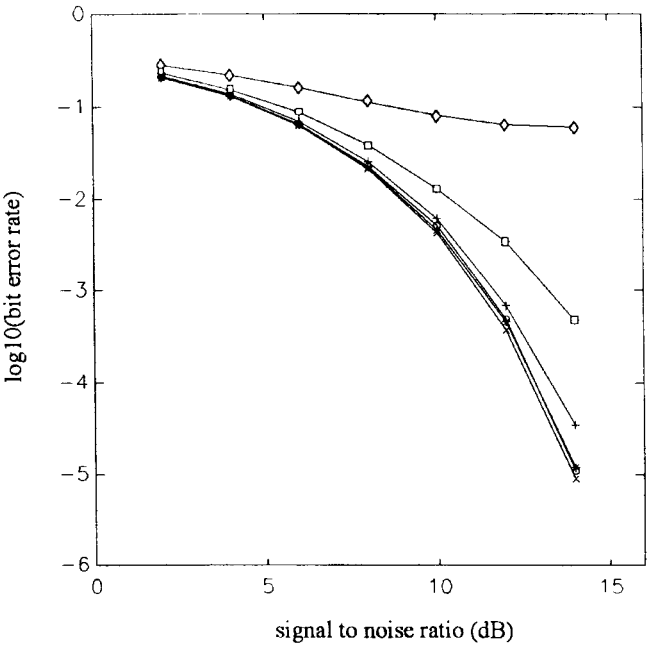



Fig. 10. Comparison of bit-error-rate curves for the optimal equalizer ("×"), network A ("∘"), network B ("∗"), network C ("+"), network D ("◇"), and network E ("□") in Example 2.

$\rho = 0.5$ and doing the same training task as above, we obtain the generated rules (clusters) shown in Fig. 9(b), where only nine rules are generated and the numbers of membership functions in the $x(k)$ and $x(k-1)$ dimensions are seven and six, respectively. The decision region of this SONFIN (network B) is shown in Fig. 8(c) and its BER is shown in Fig. 10, denoted by "∗." The performance is similar to that of network A, but fewer rules are generated. Using the same structure of network B except that no transformation matrix $R_i$ is incorporated, we have a third SONFIN (network C) with the generated rule distribution shown in Fig. 9(c). The decision region is shown in Fig. 8(d) and the BER curve is shown in Fig. 10 denoted by "+." A worse result than that of network B is obtained, verifying that a higher accuracy is achieved with the incorporation of $R_i$. To give a more clear view on the effect of $R_i$, two networks (D and E) with the same structure as network B are tested. For network D, no parameter learning is performed and the resulting BER is shown in Fig. 10 denoted by "◇." For network E, all parameters are fixed during learning except the transformation matrix $R_i$. The resulting BER is shown in Fig. 10 denoted by "□." Contribution of $R_i$ can be seen more clearly from this comparison.

In the above SONFIN's, only two values—"1" and "−1"—are assigned to the consequent part of each rule after structure learning. For comparison, we test other three networks A′, B′, and C′ using the same number of rules and membership functions as those of networks A, B, and C, respectively, except that each rule has its own corresponding singleton value in the consequent part, which is adjustable during the learning process. The training task is the same as that for networks A, B, and C. After training, the decision regions of networks A′, B′, and C′ are shown in Fig. 11 and the BER curves are shown in Fig. 12 where "×," "∗," and

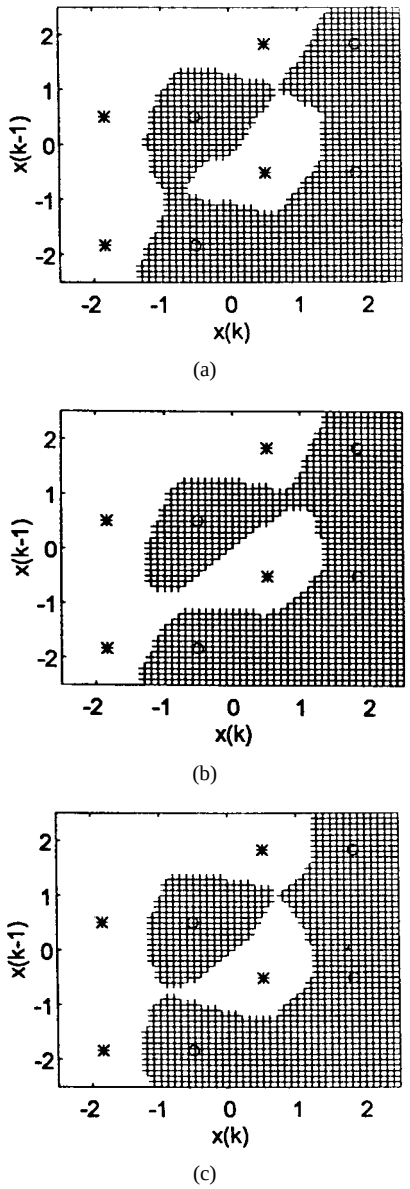



Fig. 11. Decision region in Example 2. (a) Decision region of network A′. (b) Decision region of network B′. (c) Decision region of network C′.

"∘" curves denote the BER curves of network A′, B′, and C′, respectively. The results are similar to those obtained from network A, B, and C, even though more parameters are used in the consequent parts of network A′, B′, and C′.

*Example 3—Water Bath Temperature Control:* The objective of this example is to control the temperature of a water bath at specified temperatures between 25–80 °C using the SONFIN. To achieve this purpose, the SONFIN is trained to learn the inverse dynamic model of the water bath temperature control system. The trained SONFIN is then configured as a direct controller to the system. Learning an inverse dynamic model is currently one of the most viable techniques in the application of neural networks for control. The system configuration is shown in Fig. 13, where $Z^{-i}$ represents $i$ time steps delay and $y_r$ is the desired output of the plant. By applying random inputs within a desired range to the plant, the corresponding outputs are gathered to form a set of training

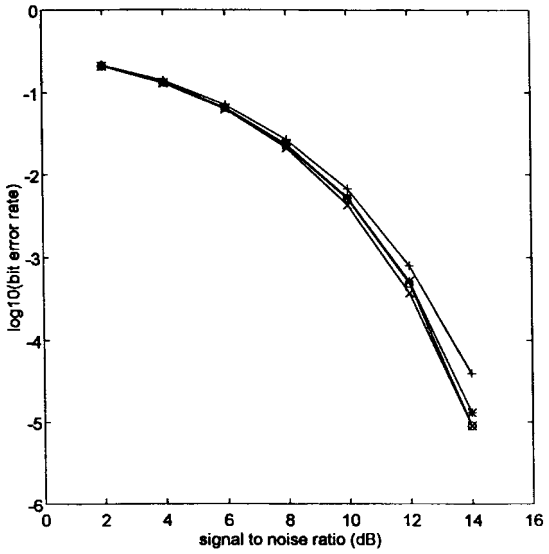



Fig. 12. Comparison of BER curves for the optimal equalizer ("×"), network A′ ("∘"), network B′ ("∗"), and network C′ ("+") in Example 2.

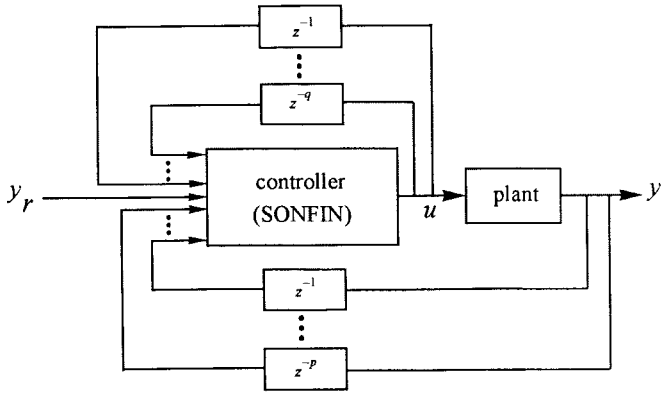



Fig. 13. The direct inverse control configuration where the SONFIN is used as the controller.

data. After training, the SONFIN is used to produce control signals as a function of the desired plant output. The water bath plant used here is governed by

$$y(k+1) = a(T_s)y(k) + \frac{b(T_s)}{1+e^{0.5y(k)-r}}u(k) + [1-a(T_s)]Y_o \tag{74}$$

where

$$a(T_s) = e^{-\alpha T_s} \tag{75}$$

$$b(T_s) = \frac{b}{a}(1-e^{-\alpha T_s}). \tag{76}$$

The system parameters used in this example are $a = 1.001\,51e^{-4}$, $b = 8.679\,73e^{-3}$, $r = 40.0$, and $Y_o = 25.0\,(^\circ\mathrm{C})$, which were obtained from a real water bath plant in [31]. The plant input $u(k)$ is limited between zero and five and the sampling period is $T_s = 60$. To generate the training data, the temperature control system is operated in an open-loop fashion and 50 random signals $u$ are injected directly to the plant described by (74). The 50 generated patterns are used to train the SONFIN, with $[y(k), y(k+1)]$ being the input and $u(k)$ the desired output.

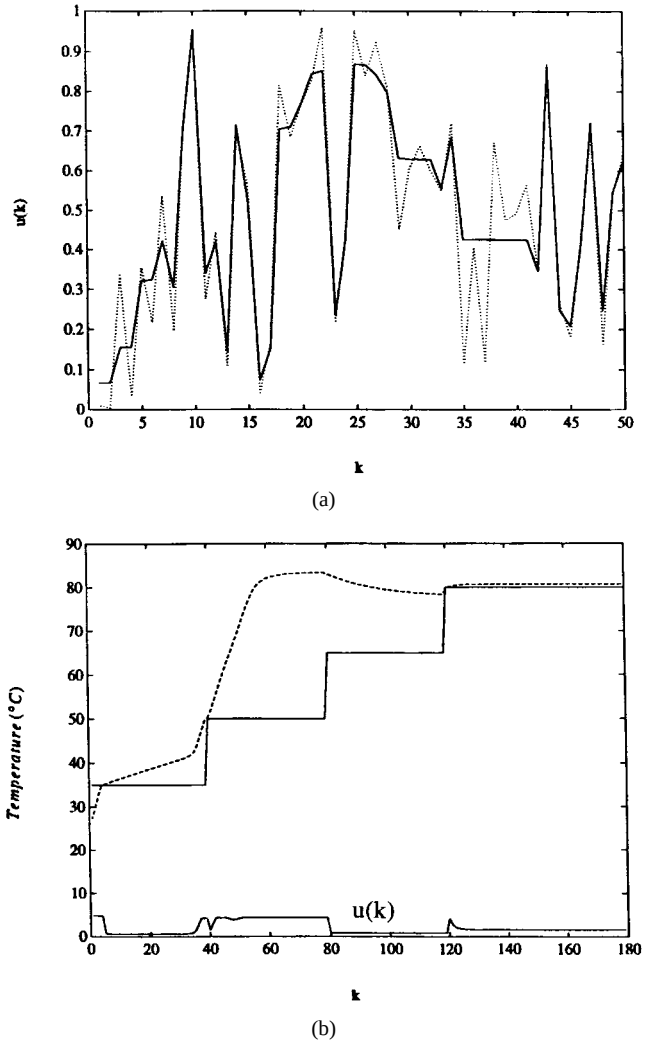



Fig. 14. Simulation results of using the SONFIN with only a singleton in the consequent part of each rule in Example 3. (a) The actual output (denoted as the solid line) and the desired output (denoted as the dotted line). (b) The desired temperature (denoted as the solid line) $\hat{y}_r(k)$ and the controlled temperature $y(k)$ (denoted as the dotted line).

Since the desired outputs change sharply even for similar inputs in this example, we shall first show that the general fuzzy rules with separate output singleton for each rule (i.e., a basic SONFIN) cannot handle this task, even though a large number of rules are used. The learning rate $\eta = 0.0001$, $\beta = 0.7$, $\overline{F}_{\text{in}} = 0.3$, $\overline{F}_{\text{out}} = 1$, and $\rho = 1$ are chosen to train a basic SONFIN. The consequent parameters are tuned by the LMS algorithm. After 1000 epochs of training, the learning result is shown in Fig. 14(a) where 23 rules are generated. The temperate control result is shown in Fig. 14(b), indicating a failure control. Next, we use fewer rules but add additional terms to the consequent part (i.e., we use the TSK-type rules). Since only two input variables are used in this example, both input variables are used in the consequent part of each rule. The parameters used for learning are $\eta = 0.005$, $\beta = 0.7$, $\overline{F}_{\text{in}} = 0.02$, and $\rho = 0.3$. The consequent parameters are tuned by the RLS algorithm with $\lambda = 0.9$. After five

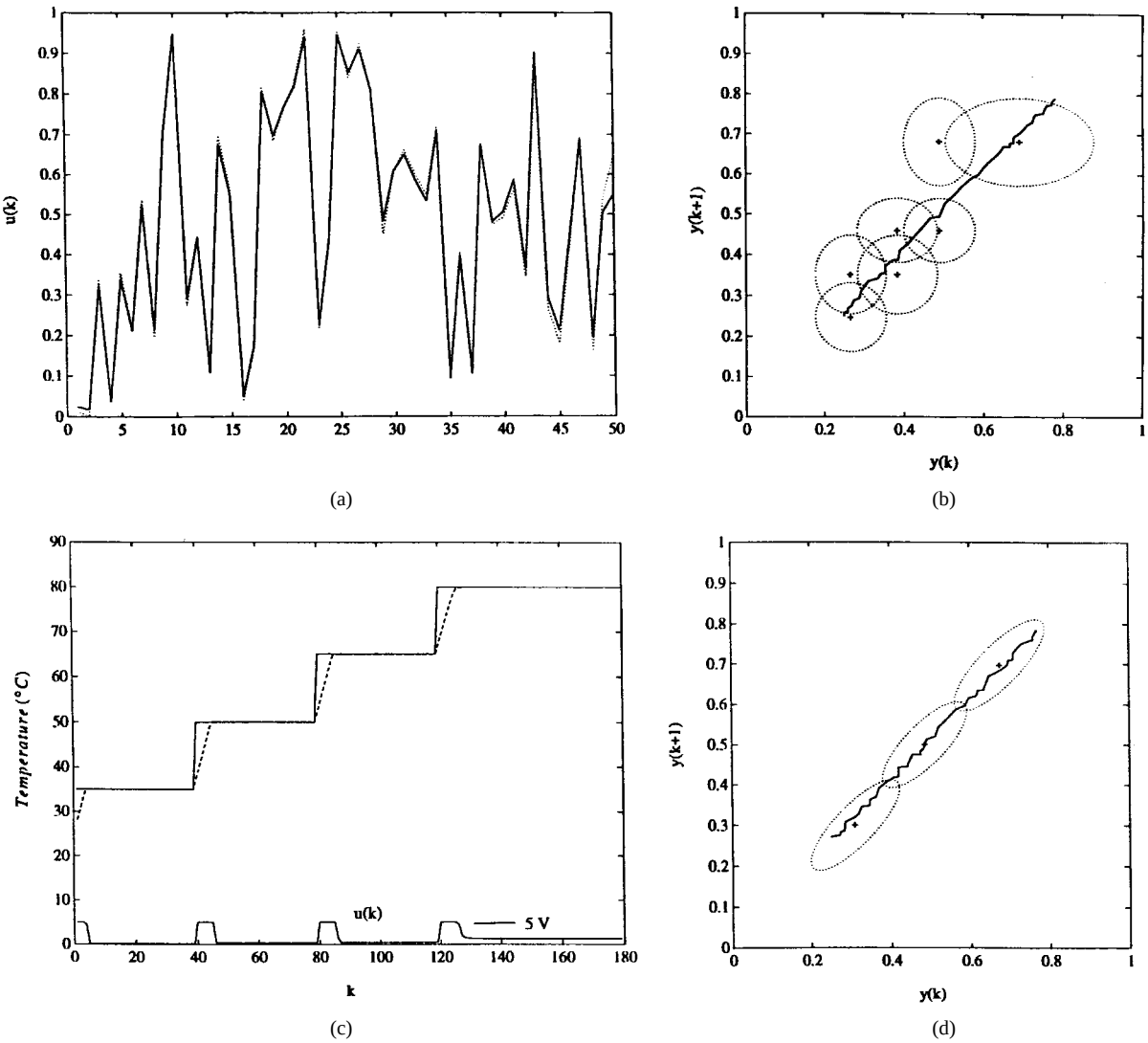



Fig. 15. Simulation results of using the SONFIN with a linear equation in the consequent part of each rule in Example 3. (a) The actual output (denoted as the solid line) and the desired output (denoted as the dotted line). (b) The input training patterns and the final assignment of rules. (c) The desired temperature $y_r(k)$ (denoted as the solid line) and the controlled temperature $y(k)$ (denoted as the dotted line). (d) The input training patterns and the final assignment of rules with linear transformations $R_i$ incorporated. (e) The desired temperature $y_r(k)$ (denoted as the solid line) and the controlled temperature $y(k)$ (denoted as the dotted line).

epochs of training, the learning result is shown in Fig. 15 (a). Fig. 15(b) illustrates the distribution of the training patterns and the final assignment of fuzzy rules in the $[y(k), y(k+1)]$ plain. The number of generated rules is seven and the numbers of fuzzy sets in the $y(k)$ and $y(k+1)$ dimensions are four and four, respectively. The control result is shown in Fig. 15(c), indicating a perfect control. To further reduce the number of rules generated, a linear transformation of the input variables $R_i$ is incorporated and is set as a rotation of 45° initially based on the observation of the input data distribution. With the choice of $\eta = 0.005$, $\beta = 0.5$, and $\overline{F}_{\text{in}} = 0.005$, only three rules are generated [see Fig. 15(d)] and a perfect control is achieved, as shown in Fig. 15(e).

*Example 4—Prediction of the Chaotic Time-Series:* In the above examples, the problems to be solved are either simple or have low-dimension inputs, so a basic SONFIN or the SONFIN with all input variables in the consequent part is used. In this example, we shall show a more complex problem which has high-dimension inputs and the SONFIN with output terms selected via the projection-based correlation measure is used. The performance of the SONFIN will be compared to that of other approaches at the end of this example.

Let $p(k)$, $k = 1, 2, \cdots$ be a time series. The problem of time-series prediction can be formulated as: given $p(k-N+1)$, $p(k-N+2)$, $\cdots$, $p(k)$, we are to determine $p(k+\ell)$, where $N$ and $\ell$ are fixed positive integers, i.e., determine a mapping from $[p(k-N+1), p(k-N+2), \cdots, p(k)] \in \Re^N$ to $[p(k+\ell)] \in \Re$. In this example, the Mackey–Glass chaotic time series is generated from the following delay differential equation:

$$\frac{dx(t)}{dt} = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1x(t) \tag{77}$$

where $\tau > 17$. In our simulation, $\tau = 30$ is chosen. The values of $N$ and $\ell$ are chosen as $N = 9$ and $\ell = 1$ in this simulation, i.e., nine point values in the series are used to predict the value

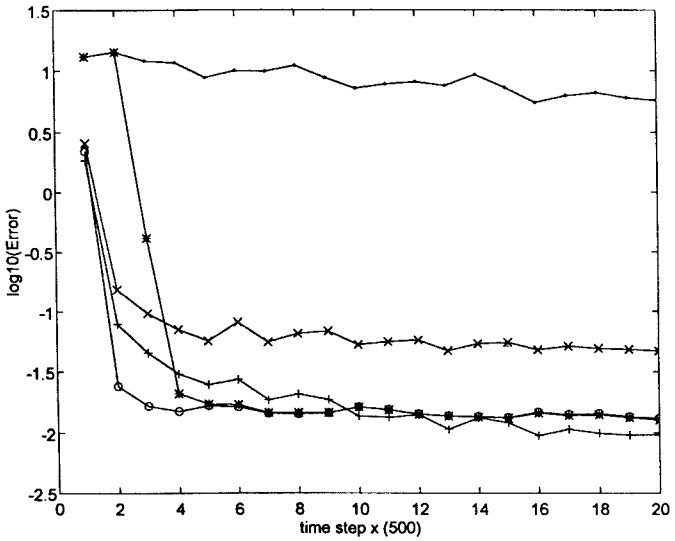



Fig. 16. The error curves for different types of networks in Example 4 during on-line learning where each point represents the value of $\log 10$(error sum over 500 points).

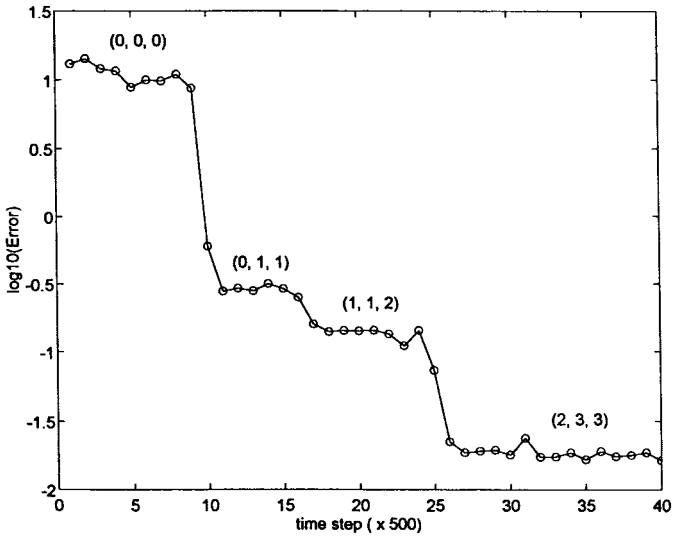



Fig. 17. The error curve during on-line learning in Example 4 where $(-, -, -)$ represents the associated number of additional terms added to the existing three rules, respectively, and each point represents the value of $\log 10$(error sum over 500 points).

of the next time point. The learning parameters $\eta = 0.005$, $\beta = 0.6$, $\overline{F}_{\text{in}} = 0.002$, $\overline{F}_{\text{out}} = 0.9$ and $\rho = 0.6$ are chosen. The consequent part is updated by the RLS algorithm with $\lambda = 0.995$.

At first, the on-line learning is performed on the basic SONFIN. Three input and three output clusters are generated during the learning process. The learning curve [$\log_{10}$(error sum over 500 points)] is shown in Fig. 16, denoted by "$\cdot$." The final error does not satisfy our requirement. Instead of using more rules to meet the requirement, we plan to add some additional terms into the consequent part of the basic model. Since the dimension of the input space is high, the consequent structure identification scheme introduced in Section III-C is used and $\gamma = 5 \times 10^{-5}$ is chosen. This on-line identification scheme is performed at the time the error curve stops descending. The resulting learning curve, denoted as "$*$" is shown in Fig. 16 where the terms are added until the 1000th time steps. A total of 15 terms (seven, five, and three for the three rules, respectively) are added to the consequent part. To test the significance of the selected terms, we assume the terms are existent once the corresponding rules are generated. The resulting learning curve is shown in Fig. 16 denoted as "$\circ$." It is observed that the "$*$" curve and the "$\circ$" curve match after 2500 time steps. For comparison, the same number of consequent terms are used in another SONFIN (i.e., seven, five, and three terms for the three rules, respectively) with the terms randomly selected. The learning curve for this SONFIN is shown in Fig. 16, denoted as "$\times$" showing a worse result. Moreover, the network with all the input variables used in the consequent part (30 terms in total) is also used for comparison. The resulting learning curve is shown in Fig. 16, denoted as "$+$." This result is similar to that (the "$*$" curve) of the network with only 18 significant terms in the consequent part.

Instead of adding a set of terms into the consequent part at once (as we did in the above), we can add the consequent terms stage by stage. This can be done by setting a larger $\gamma$ value

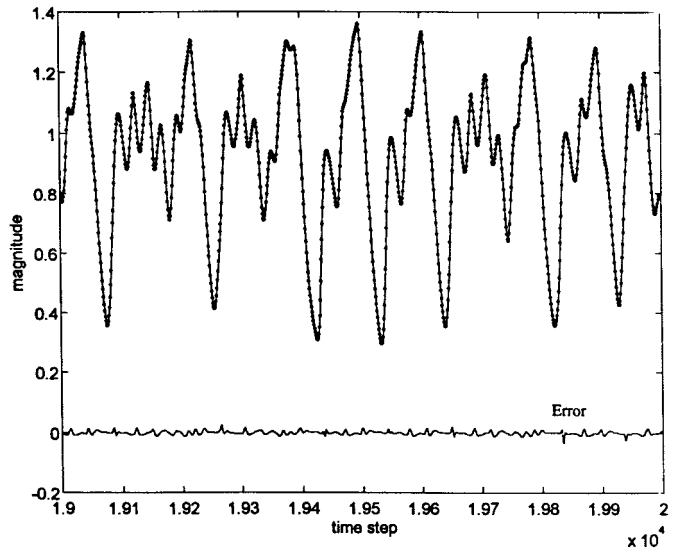



Fig. 18. The desired values (denoted as the solid line) and the predicted values (denoted as the circle line) in Example 4. The difference between the desired and actual values is also shown in the figure, which is denoted as the solid line below the two magnitude curves.

in the beginning and then decreasing $\gamma$ gradually during the consequent structure identification process until $\gamma$ is smaller than a prespecified meaningful value or the accuracy satisfies the requirement. By setting $\gamma = 0.005$ initially and $\gamma = \gamma/10$ during the consequent structure identification process, we obtain the learning curve in Fig. 17 where each sharp drop in the curve is caused by the addition of additional terms to the linear equation of the consequent part. After 19 000 time steps, eight terms are added to the consequent part and the prediction result is shown in Fig. 18 where the predicted values of the SONFIN are presented as a dotted curve and the actual values as a solid curve. The difference between the actual and predicted values is also shown in Fig. 18, which is presented as a solid curve below the two magnitude curves.

TABLE II
PERFORMANCE COMPARISON OF VARIOUS RULE GENERATION METHODS ON THE TIME SERIES PREDICTION PROBLEM

| | SONFIN | | FALCON-ART | | Wang&Mendel | Data distribution | Kosko(AVQ) without backpropagation | | Kosko (AVQ) with backpropagation | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | UCL | DCL | UCL | | DCL | |
| Rule number (200 training data) | 4 (0,0,0,0)* | 4 (2,5,2,3)* | 22 | 30** | 121 | 118 | 100 | 100 | 22 | 100 | 22 | 100 |
| RMS error | 0.07 | 0.018 | 0.08 | 0.04 | 0.08 | 0.08 | 0.17 | 0.2 | 0.16 | 0.09 | 0.17 | 0.09 |

* associated number of additional terms added to the existing 4 rules
** 700 training data are used

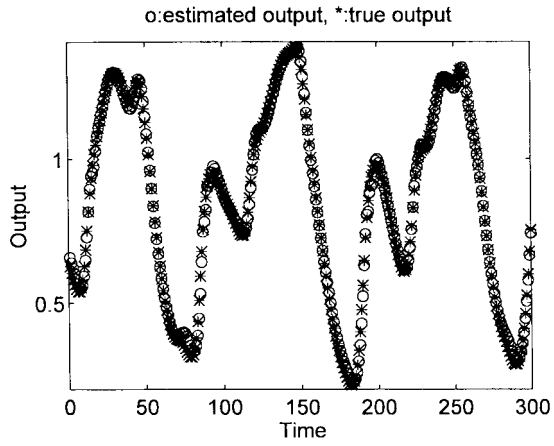



Fig. 19. Simulation results of the time series prediction from $x(701)$ to $x(1000)$ using the SONFIN with four rules when 200 training data [from $x(501)$ to $x(700)$] are used.

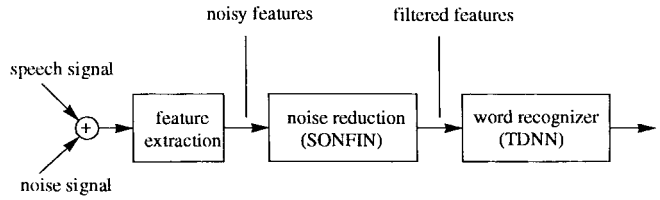



Fig. 20. The structure of the noisy speech recognizer using the SONFIN.

TABLE III
THE RECOGNITION RATES UNDER CLEAN AND NOISY ENVIRONMENT AT DIFFERENT SNR VALUES WITHOUT USING THE NOISE REDUCTION NETWORK IN EXAMPLE 5

| SNR | 0 | 6 | 12 | 18 | clean |
|---|---|---|---|---|---|
| recognition rate | 20 | 34 | 48 | 57 | 99 |

To test the generalization ability and compare the performance of the SONFIN with other methods that can generate rules from numerical data, the same chaotic time series training and testing data in [32] are used. In [32], 200 points of the series from $x(501)$ to $x(700)$ are used as training data and the succeeding 300 points from $x(701)$ to $x(1000)$ are used as testing data. After off-line training on the 200 points using the SONFIN, four rules are generated and the additional terms added to the four rules are 2, 5, 2, and 3, respectively. Fig. 19 shows the prediction of the chaotic time series from $x(701)$ to $x(1000)$ where the predictions of the SONFIN are denoted as "$\star$," and the true values as "$\circ$." The rms error over the 700 predicted points is 0.018. If no additional terms are added to the consequent part, a rms error of 0.07 is achieved.

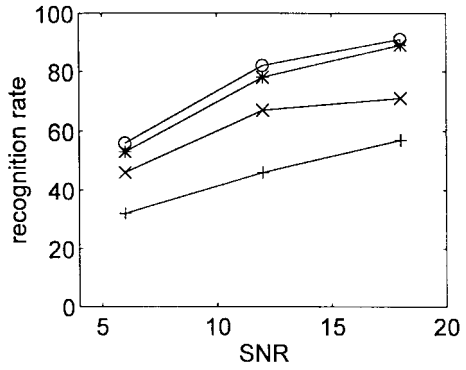
To give a clear understanding of the performance of the SONFIN, the ART-based fuzzy adaptive learning control network (FALCON-ART) and other approaches discussed in [32] are compared. These approaches include: Wang and Mendel's approach [17], [33] based upon direct matching; the data distribution method, which generates fuzzy rules according to the training data distribution in the input–output product space; the generation of fuzzy associative memory (FAM) rules based on adaptive vector quantization (AVQ) algorithms which contain unsupervised competitive learning (UCL) and differential competitive learning (DCL), proposed by Kosko [5]; and the combination of the UCL (DCL)–AVQ and backpropagation algorithms method. The generated rule number as well as rms errors of these approaches are listed in Table II. As to the detailed construction schemes and actual predicted outputs of these models, the reader is referred to [32]. From Table II, we find that the SONFIN not only needs much fewer rules and membership functions but also achieves much smaller rms error.

*Example 5—Noisy Speech Recognition:* A well-performed speech recognition system under noise-free conditions usually show marked degradation in performance when background noise is present. To overcome this problem, the SONFIN is used in this example as a noise reduction network in the cepstral domain. The SONFIN here can be considered to perform a nonlinear mapping from a noisy feature space to a noise-free feature space [34]. The architecture of the enhancement recognition system is shown in Fig. 20. In this example, the database contains ten isolated Mandarin digits "0," $\cdots$, "9." They were spoken by the same speaker, with 30 noise-free repetitions for each word. Among these 30 repetitions, ten are used for training, ten for cross validation during the training, and the left ten for testing. The time delay neural network (TDNN) is used as the recognizer. The features extracted are the cepstral coefficients with order 12 for each frame and 20 constant frames are used for each word. The noisy speech is generated by adding white Gaussian noise $e(n)$ to the clean speech with different signal-to-noise ratio (SNR).

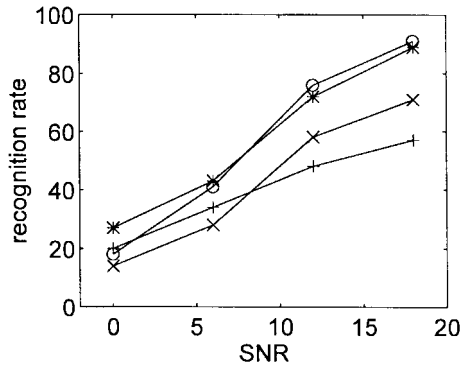
Without the noise reduction network, the recognition rates under clean and noisy environment at different SNR values are listed in Table III. In training, the SONFIN as a noise reduction network at a specific SNR value, the 12 noisy cepstral features of each frame are used as the inputs and the corresponding 12 noise-free cepstral features as the desired outputs. All the 100 words in the training set are used for training. The parameters used for learning are $\eta = 0.0005, \beta =$

TABLE IV
THE NUMBER OF RULES GENERATED, TOTAL NUMBER OF MEMBERSHIP FUNCTIONS, AND THE NUMBER OF CONSEQUENT PARAMETERS FOR DIFFERENT SNR VALUES AND MODES IN EXAMPLE 5

| SNR / parameter / model | SNR = 6 | | | SNR = 12 | | | SNR = 18 | | |
|---|---|---|---|---|---|---|---|---|---|
| | rule | fuzzy sets | output terms | rule | fuzzy sets | output terms | rule | fuzzy sets | output terms |
| model A | 17 | 100 | 204 | 16 | 110 | 192 | 14 | 87 | 168 |
| model B | 17 | 100 | 912 | 16 | 110 | 803 | 14 | 87 | 785 |
| model C | 17 | 100 | 2652 | 16 | 110 | 2496 | 14 | 87 | 2184 |

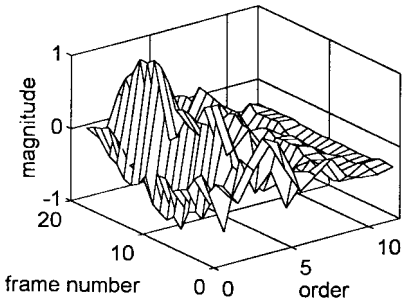



(a)

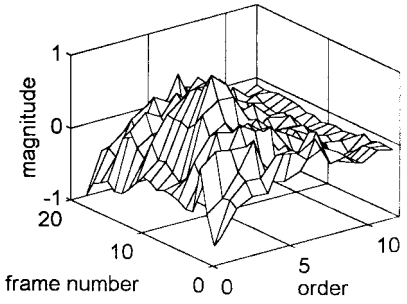



(b)

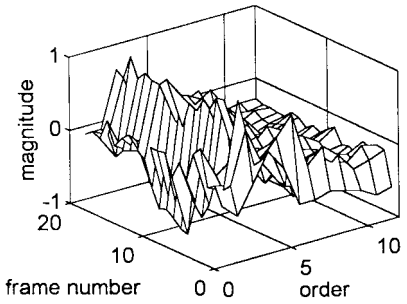
Fig. 21. The recognition rates on testing data by using the noise reduction network A ("×"), B ("∗"), C ("∘"), and without using the noise reduction network ("+") in Example 5. (a) Results for the network trained at SNR $= 0, 6, 12,$ and 18. (b) The generalization ability test for the network trained at SNR $= 18$.




(a)




(b)




(c)

Fig. 22. Cepstral features of speech signal in Example 5. (a) The clean cepstral features of word "0." (b) The noisy cepstral features wrongly recognized as word "7." (c) The filtered cepstral features by using network B, recognized as word "0" correctly.

$0.9$, $\overline{F}_{\text{in}} \approx 0.26$, $\overline{F}_{\text{out}} = 0.95$, and $\rho = 0.8$. The consequent part is tuned by the RLS algorithm with $\lambda = 0.999$. Using the above parameters, three types of SONFIN models are used for comparison. Model A is the basic model with the consequent part of each rule being a singleton value. Model B is constructed by adding some significant terms to the consequent part of model A by the consequent structure identification scheme described in Section III-C, with $\gamma \approx 0.009$. Model C is a general TSK model whose consequent part is a linear combination of all the input variables. Each model is trained at SNR $= 6, 12,$ and 18, respectively. In training these models, only about eight epochs are needed via cross validation test. The number of rules generated, total number of membership functions and the number of consequent parameters for different SNR values and models are listed in Table IV. The corresponding noise reduction effects on the test data for the models trained at each specific SNR value are shown in Fig. 21(a) where symbols "+," "∗," and "∘" denote the recognition rates of model A, B, and C, respectively, and the symbol "×" denotes the recognition rate without the noise reduction network. To see the generalization ability for a model trained at a specific SNR value, we may test it on the speech signals with different SNR values. The noise reduction network trained at SNR $= 18$ is illustrated. After trained at SNR $= 18$, the performance of the noise reduction network is tested at SNR $= 0, 6, 12,$ and 18 and is shown in Fig. 21(b).

Comparing the performance of model B and C, we find that their noise reduction effects are very similar, but the number of parameters used in the consequent part of model B is only one third of that used in model C. To see the training result, we use model B for illustration. The clean features for word "0" are shown in Fig. 22(a) and the extracted noisy features at SNR $= 12$ are shown in Fig. 22(b), which is wrongly recognized as word "7." The filtered features by using model B are shown in Fig. 22(c), which results in correct recognition.

## V. Conclusion

A neural fuzzy inference network SONFIN, with on-line self-constructing capability, is proposed in this paper. The SONFIN is a general connectionist model of a fuzzy logic system, which can find its optimal structure and parameters automatically. Both the structure and parameter identification schemes are done simultaneously during on-line learning, so the SONFIN can be used for normal operation at any time as learning proceeds without any assignment of fuzzy rules in advance. A novel network construction method for solving the dilemma between the number of rules and the number of consequent terms is developed. The number of generated rules and membership functions is small even for modeling a sophisticated system. As a summary, the SONFIN can always find itself an economic network size, and the learning speed as well as the modeling ability are all appreciated. Simulations in different areas including control, communication, and signal processing have demonstrated the on-line learning capability of the SONFIN.

## References


[1] S. Horikawa, T. Furuhashi, and Y. Uchikawa, "On fuzzy modeling using fuzzy neural networks with the backpropagation algorithm," *IEEE Trans. Neural Networks*, vol. 3, pp. 801–806, Sept. 1992.

[2] K. Tanaka, M. Sano, and H. Watanabe, "Modeling and control of carbon monoxide concentration using a neuro-fuzzy technique," *IEEE Trans. Fuzzy Syst.*, vol. 3, pp. 271–279, Aug. 1995.

[3] Y. Lin and G. A. Cunningham, "A new approach to fuzzy-neural system modeling," *IEEE Trans. Fuzzy Syst.*, vol. 3, pp. 190–197, May 1995.

[4] M. Sugeno and T. Yasukawa, "A fuzzy-logic-based approach to qualitative modeling," *IEEE Trans. Fuzzy Syst.*, vol. 1, pp. 7–31, Feb. 1993.

[5] B. Kosko, *Neural Networks and Fuzzy Systems.* Englewood Cliffs, NJ: Prentice-Hall, 1992.

[6] C. T. Lin, *Neural Fuzzy Control Systems with Structure and Parameter Learning.* New York: World Scientific, 1994.

[7] C. T. Lin and C. S. G. Lee, *Neural Fuzzy Systems: A Neural-Fuzzy Synergism to Intelligent Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1996.

[8] J. S. Jang, "ANFIS: Adaptive-network-based fuzzy inference system," *IEEE Trans. Syst., Man, Cybern.*, vol. 23, pp. 665–685, May 1993.

[9] C. T. Sun, "Rule-based structure identification in an adaptive-network-based fuzzy inference," *IEEE Trans. Fuzzy Syst.*, vol. 2, pp. 64–73, Feb. 1994.

[10] C. T. Lin and C. S. G. Lee, "Neural-network-based fuzzy logic control and decision system," *IEEE Trans. Comput.*, vol. 40, pp. 1320–1336, Dec. 1991.

[11] ———, "Reinforcement structure/parameter learning for neural-network-based fuzzy logic control systems," *IEEE Trans. Fuzzy Syst.*, vol. 2, pp. 46–63, Feb. 1994.

[12] C. J. Lin and C. T. Lin, "Reinforcement learning for ART-based fuzzy adaptive learning control networks," accepted for publication in *IEEE Trans. Neural Networks*, vol. 7, pp. 709–731, May 1996.

[13] C. T. Lin, "A neural fuzzy control system with structure and parameter learning," *Fuzzy Sets Syst.*, vol. 70, pp. 183–212, 1995.

[14] L. X. Wang and J. M. Mendel, "Generating fuzzy rules by learning from examples," *IEEE Trans. Syst., Man, Cybern.*, vol. 22, no. 6, pp. 1414–1427, Nov./Dec. 1992.

[15] H. Ishibuchi, K. Nozaki, N. Yamamoto, and H. Tanaka, "Selecting fuzzy if-then rules for classification problems using genetic algorithms," *IEEE Trans. Fuzzy Syst.*, vol. 3, pp. 260–270, Aug. 1995.

[16] L. X. Wang and J. M. Mendel, "Fuzzy basis functions, universal approximation, and orthogonal least-squares learning," *IEEE Trans. Neural Networks*, vol. 3, pp. 807–814, Sept. 1992.

[17] L. X. Wang, *Adaptive Fuzzy Systems and Control.* Englewood Cliffs, NJ: Prentice-Hall, 1994.

[18] J. Platt, "A resource allocating network for function interpolation," *Neural Computat.*, vol. 3, pp. 213–225, 1991.

[19] J. Nie and D. A. Linkens, "Learning control using fuzzified self-organizing radial basis function network," *IEEE Trans. Fuzzy Syst.*, vol. 40, pp. 280–287, Nov. 1993.

[20] M. Sugeno and K. Tanaka, "Successive identification of a fuzzy model and its applications to prediction of a complex system," *Fuzzy Sets Syst.*, vol. 42, no. 3, pp. 315–334, 1991.

[21] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its applications to modeling and control," *IEEE Trans. Syst., Man, Cybern.*, vol. 15, pp. 116–132, Jan. 1985.

[22] C. C. Lee, "Fuzzy logic in control systems: Fuzzy logic controller—Part II," *IEEE Trans. Syst., Man, Cybern.*, vol. 20, pp. 419–435, Mar./Apr. 1990.

[23] H. R. Berenji and P. Khedkar, "Learning and tuning fuzzy logic controllers through reinforcements," *IEEE Trans. Neural Networks*, vol. 3, no. 5, pp. 724–740, Sept. 1992.

[24] L. Wang and R. Langari, "Building Sugeno-type models using fuzzy discretization and orthogonal parameter estimation techniques," *IEEE Trans. Fuzzy Syst.*, vol. 3, pp. 454–458, Nov. 1995.

[25] E. H. Ruspini, "Recent development in fuzzy clustering," *Fuzzy Set and Possibility Theory.* New York: North Holland, 1982, pp. 113–147.

[26] R. Reed, "Pruning algorithms—A survey," *IEEE Trans. Neural Networks*, vol. 4, pp. 740–747, Sept. 1993.

[27] B. Noble and J. W. Daniel, *Applied Linear Algebra*, 3rd ed. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[28] C. T. Sun and J. S. Jang, "A neuro-fuzzy classifier and its applications," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, San Francisco, CA, Mar. 1993, vol. I, pp. 94–98.

[29] B. Philippe, "An algorithm to improve nearly orthonormal sets of vectors on a vector processor," *SIAM J. Alg. Disc. Meth.* vol. 8, no. 3, pp. 396–403, July 1987.

[30] B. Widrow and S. D. Stearns, *Adaptive Signal Processing.* Englewood Cliffs, NJ: Prentice-Hall, 1985.

[31] J. Tanomaru and S. Omatu, "Process control by on-line trained neural controllers," *IEEE Trans. Indust. Electrons*, vol. 39, pp. 511–521, Dec. 1992.

[32] C. J. Lin and C. T. Lin, "An ART-based fuzzy adaptive learning control network," *IEEE Trans. Fuzzy Syst.*, to be published.

[33] L. X. Wang and J. M. Mendel, "Generating fuzzy rules by learning from examples," *IEEE Trans. Syst., Man, Cybern.*, vol. 22, no. 6, pp. 1414–1427, Nov./Dec. 1992.

[34] H. B. D. Sorensen, "A cepstral noise reduction multilayer neural network," in *Proc. Int. Conf. Acoust., Speech, Signal Processing*, New York, May, 1991, pp. 933–936.


**Chia-Feng Juang** received the B.S. degree in control engineering from the National Chiao-Tung University, Taiwan, R.O.C., in 1993. He is currently working toward the Ph.D. degree in the Department of Control Engineering at the same university.

His current research interests are neural networks, learning systems, fuzzy control, noisy speech recognition, and signal processing.

**Chin-Teng Lin** received the B.S. degree in control engineering from the National Chiao-Tung University, Taiwan, R.O.C., in 1986, and the M.S.E.E. and Ph.D. degrees in electrical engineering from Purdue University, West Lafayette, IN, in 1989 and 1992, respectively.

Since August 1992, he has been with the College of Electrical Engineering and Computer Science, National Chiao-Tung University, Hsinchu, Taiwan, R.O.C., where he is currently an Associate Professor of Control Engineering. He is the coauthor of *Neural Fuzzy Systems—A Neuro-Fuzzy Synergism to Intelligent Systems* (Englewood Cliffs, NJ: Prentice-Hall, 1996) and the author of *Neural Fuzzy Control Systems with Structure and Parameter Learning* (New York: World Scientific, 1994). His current research interests are fuzzy systems, neural networks, intelligent control, human-machine interface, and video and audio processing.

Dr. Lin is a member of Tau Beta Pi, Eta Kappa Nu, the IEEE Computer Society, the IEEE Robotics and Automation Society, and the IEEE Systems, Man, Cybernetics Society.