A Recurrent Self-Organizing Neural Fuzzy Inference Network

Chia-Feng Juang and Chin-Teng Lin, *Member, IEEE*

A recurrent self-organizing neural fuzzy inference network (RSONFIN) is proposed in this paper. The RSONFIN is inherently a recurrent multilayered connectionist network for realizing the basic elements and functions of *dynamic* fuzzy inference, and may be considered to be constructed from a series of dynamic fuzzy rules. The temporal relations embedded in the network are built by adding some feedback connections representing the memory elements to a feedforward *neural fuzzy* network. Each weight as well as node in the RSONFIN has its own meaning and represents a special element in a fuzzy rule. There are no hidden nodes (i.e., no membership functions and fuzzy rules) initially in the RSONFIN. They are created on-line via concurrent structure identification (the construction of dynamic fuzzy if–then rules) and parameter identification (the tuning of the free parameters of membership functions). The structure learning together with the parameter learning forms a fast learning algorithm for building a small, yet powerful, dynamic neural fuzzy network. Two major characteristics of the RSONFIN can thus be seen: 1) the recurrent property of the RSONFIN makes it suitable for dealing with temporal problems and 2) no predetermination, like the number of hidden nodes, must be given, since the RSONFIN can find its optimal structure and parameters automatically and quickly. Moreover, to reduce the number of fuzzy rules generated, a flexible input partition method, the aligned clustering-based algorithm, is proposed. Various simulations on temporal problems are done and performance comparisons with some existing recurrent networks are also made. Efficiency of the RSONFIN is verified from these results.

***Index Terms*— Context node, dynamic fuzzy inference, feedback term node, ordered derivative, projection-based correlation measure.**

I. Introduction

PROBLEM solving using neural fuzzy network approach is becoming a popular research topic in these years [1]–[3]. Many characteristics of the neural fuzzy network contribute to this phenomenon. Some of them are, as compared to the general neural networks, faster convergence speed, and smaller network size. Moreover, the neural fuzzy network approach automates the design of fuzzy rules and makes the combinational learning of numerical data as well as expert knowledge expressed as fuzzy if–then rules possible. In contrast to the pure neural network or fuzzy system, 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 [1], [4], [5]. However, a major drawback of the existing neural fuzzy networks is that their application domain is limited to static problems due to their inherent *feedforward* network structure. Inefficiency occurs for temporal problems. Hence a *recurrent* neural fuzzy network capable for solving temporal problems is in need.

A recurrent neural network, which naturally involves dynamic elements in the form of feedback connections used as internal memories, has been attracting great interest in the past few years [6], [7]. Unlike the feedforward neural network whose output is a function of its current inputs only and is limited to static mapping, recurrent neural networks perform dynamic mapping. Recurrent networks are needed for problems where there exists at least one system state variable which cannot be observed. Most of the existing recurrent neural networks are obtained by adding trainable temporal elements to feedforward neural networks (like multilayer perceptron networks [6] and radial basis function networks [8], [9]) to make the output history-sensitive. Like feedforward neural networks, these networks function as black boxes; we do not know the meaning of each weight and node in these networks. Recently, the concept of incorporating fuzzy logic into a recurrent network is proposed in some papers [10]–[17]. Since the neural fuzzy networks have so many advantages over the feedforward neural networks as mentioned above, it seems worth constructing a recurrent network based on a neural fuzzy network. In this paper, we shall propose such a *recurrent neural fuzzy network*. The proposed network will possess the same advantages over the pure recurrent neural networks, and extend the application domain of the normal neural fuzzy networks to temporal problems.

The recurrent neural fuzzy network proposed in this paper is called recurrent self-organizing neural fuzzy inference network (RSONFIN). The RSONFIN expands the basic ability of a neural fuzzy network to cope with temporal problems via the inclusion of some internal memories, called *context elements*. In the perspective of fuzzy logic, these context elements are expressed in the form of internal fuzzy reasoning. More clearly, with these context elements, the network performs the following reasoning:

Rule i: IF $x_1(t)$ is A_{i1} and $\cdots$ and $x_n(t)$ is $A_{\mathbf{in}}$ and $h_i(t)$ is G
THEN $y_1(t+1)$ is B_{i1} and $h_1(t+1)$ is w_{1i} and $\cdots$ and $h_m(t+1)$ is w_{mi}

Manuscript received January 13, 1998; revised November 23, 1998 and February 15, 1999. This work was supported by the R.O.C. National Science Council under Grant NSC87-2213-E-009-146.

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

Publisher Item Identifier S 1045-9227(99)05486-7.

where x_i is the input variable, y_1 is the output variable, A_{i1}, A_{in}, G, and B_{i1} are fuzzy sets, h_i is the internal variable, w_{1i} and w_{mi} are fuzzy singletons, and n and m are the numbers of input and internal variables, respectively. The dynamic reasoning implies that the inference output $y_1(t+1)$ is affected by the internal variable $h_i(t)$, and the current internal output $h_i(t+1)$ is a function of previous output value $h_i(t)$; i.e., the internal variable h_i itself forms the dynamic reasoning.

To reduce the network design effort, the automatic adaptation of the network topology is a tendency [18]–[22]. In contrast to other neural fuzzy networks, where the network structure is fixed and the rules should be assigned in advance, there are no rules initially in the RSONFIN; all of them are constructed during on-line learning. Two learning phases, the structure as well as parameter learning phases, are used to accomplish this task. The structure learning phase is responsible for the generation of fuzzy if–then rules as well as the judgment of feedback configuration, and the parameter learning phase for the tuning of free parameters of each dynamic rule (such as the shapes and positions of membership functions and the singleton values). In the structure learning phase, since the way the input space is partitioned strongly affects the number of rules generated, an efficient partition scheme to reduce the number of rules is required. Although many partition methods have been proposed [1], some drawbacks still exist in these methods. The grid-type partition method encounters the problem of exponential growth of rules as the dimension of the input space increases. A flexible partition method, the clustering-based approach which clusters the input training vectors in the input space, dose reduce the rule number, but increases the number of fuzzy sets on each input dimension. For these clustering-based methods [23]–[25], the number of fuzzy sets on each input dimension is in general equal to the number of fuzzy rules. This usually produces unnecessary fuzzy sets as shown in Fig. 1(a). In this paper, an aligned-clustering-based partition scheme is proposed. This scheme partitions the input space in a flexible way and a fuzzy measure scheme is performed on each input dimension to eliminate the unnecessary terms during on-line learning. The partitioned space is like the one shown in Fig. 1(b) indicating that both the numbers of rules and membership functions are reduced. For the consequent-part identification in structure learning, a clustering-based scheme is proposed. Based on this scheme the context elements are on-line generated, and then the whole network is constructed. For parameter learning, a recursive learning algorithm is developed based on the ordered derivative scheme [37]. This algorithm can tune the free parameters in the preconditions and consequents of fuzzy rules, and weights of feedback connections simultaneously to minimize an output error function. All of these processes are done on-line, so the network can be used for normal operation at any time as learning proceeds.

Overall, the advantages of the RSONFIN against other recurrent network models [6]–[17] are summarized as follows: 1) Unlike other recurrent network models where the network structure is a normal neural network and functions as a black box, the RSONFIN is a fuzzy inference network. Each node and weight has its own meaning and functions as an element in a fuzzy reasoning process. 2) For most recurrent network models, the user has to specify the network structure in advance. However, for the RSONFIN, no preassignment of the network structure is required, since the RSONFIN can on-line construct itself automatically. 3) As will be shown in Section IV, the RSONFIN is characterized by small network size and fast learning speed.

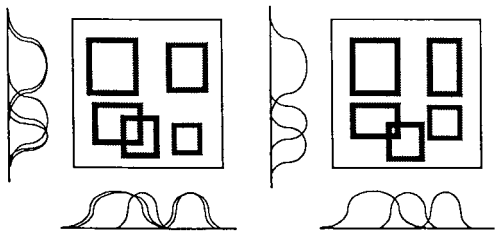


Fig. 1. Fuzzy partitions of a two-dimensional input space: (a) clustering-based partitioning and (b) proposed aligned clustering-based partitioning.

This paper is organized as follows. Section II describes the structure and functions of the RSONFIN. The on-line structure/parameter learning algorithm for the RSONFIN is presented in Section III, which contains four parts: the input–output space partitioning, fuzzy rule construction, feedback structure identification, and parameter learning. In Section IV, the RSONFIN is applied to solve several dynamic problems including the time sequence prediction, nonlinear infinite impulse response (IIR) [42] filtering, dynamic identification, and dynamic plant control. Comparisons with some existing recurrent neural networks and nonrecurrent neural fuzzy networks are also made. Finally, conclusions are summarized in the last section.

II. Structure of the RSONFIN

In this section, the structure of the RSONFIN shown in Fig. 2 is introduced. The RSONFIN consists of nodes, each of which has some finite fan-in of connections from other nodes and some fan-out of connections to other nodes. Basically, it is a five-layered neural fuzzy network embedded with dynamic feedback connections (the feedback layer in Fig. 2) that bring the temporal processing ability into a feedforward neural fuzzy network. To give a clear understanding of the network structure, the function of the node in each layer is described below. In the following descriptions, the symbol $u_i^{(k)}$ denotes the ith input of a node in the kth layer; correspondingly, the symbol $a^{(k)}$ denotes the node output in layer k.

Layer 1: No computation is done in this layer. Each node in this layer is called an input linguistic node and corresponds to one input variable. The node only transmits input values to the next layer directly. That is

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

Layer 2: Nodes in this layer are called input term nodes, each of which corresponds to one linguistic label (small, large, etc.) of an input variable. Each node in this layer calculates the membership value specifying the degree to which an input

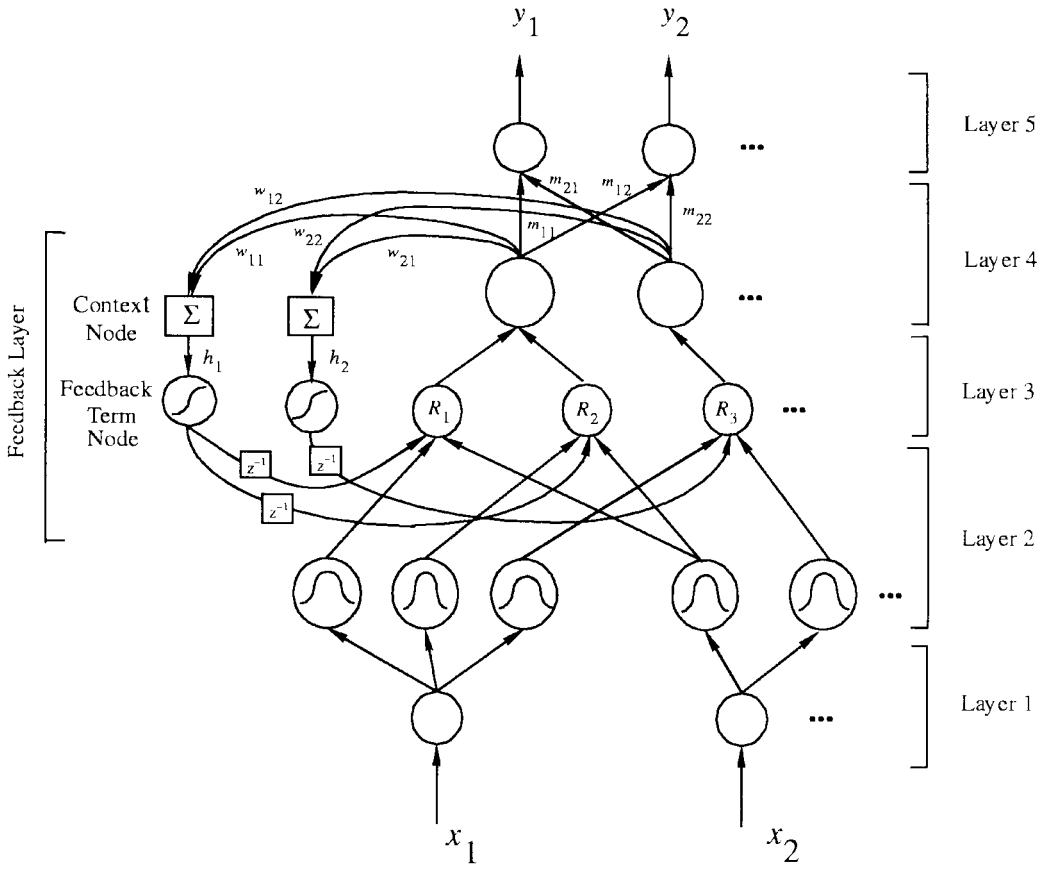


Fig. 2. Structure of the proposed recurrent self-organizing neural fuzzy inference network (RSONFIN).

value belongs to a fuzzy set. A local membership function is used in this layer. There are many qualified candidates for the types of membership functions, such as triangular, trapezoidal, or Gaussian membership functions. Here, a Gaussian membership function is employed. The reason is that a multidimensional Gaussian membership function can be easily decomposed into the product of one-dimensional membership functions. With this choice, the operation performed in this layer is

$$a^{(2)} = \exp\left\{-\frac{\left(u_i^{(2)} - m_{ij}\right)^2}{\sigma_{ij}^2}\right\} \tag{2}$$

where m_{ij} and σ_{ij} are, respectively, the center and the width of the Gaussian membership function of the jth term of the ith input variable x_i.

Layer 3: Nodes in this layer are called rule nodes. A rule node represents one fuzzy logic rule and performs precondition matching of a rule. The fan-in of a fuzzy node comes from two sources: one from layer 2 and the other from the feedback layer. The former represents the rule's spatial firing degree, and the latter the rule's temporal firing degree. We use the following AND operation on each rule node to integrate these fan-in values

$$a^{(3)} = a^{(6)} \cdot \prod_i u_i^{(3)} = a^{(6)} \cdot e^{-[D_i(\mathbf{x}-\mathbf{m}_i)]^T[D_i(\mathbf{x}-\mathbf{m}_i)]} \tag{3}$$

where $D_i = \text{diag}(1/\sigma_{i1}, 1/\sigma_{i2}, \cdots, 1/\sigma_{\text{in}})$, $\mathbf{m}_i = (m_{i1}, m_{i2}, \cdots, m_{\text{in}})^T$, and $a^{(6)}$ is the output of the feedback term node which will be described in the feedback layer part in this section. Obviously, the output $a^{(3)}$ of a rule node represents the firing strength of its corresponding rule. The fuzzy AND operation used in (3) is the algebraic product in fuzzy theory [1]. The adoption of this operation is for computational convenience, especially in deriving the learning algorithm for the RSONFIN. Also this operation transforms a set of one-dimensional Gaussian membership functions into a multidimensional one as stated in the last paragraph [also see (12)]. The algebraic product was also used in other neural fuzzy networks [2], [25], [48] as the fuzzy AND operator.

Layer 4: This layer is called the consequent layer and the nodes in this layer are called output term nodes. Each output term node represents a multidimensional fuzzy set (described by a multidimensional Gaussian function) obtained during the clustering operation in structure learning phase. Only the center of each Gaussian membership function is delivered to the next layer for the local mean of maximum (LMOM) defuzzification operation [26], so the width is used for output clustering only. Of course, we may use other types of defuzzification operation [e.g., the center of area (COA) operation] where the width is used. Since the network behavior has only a little change whether the width is used or not, especially after our learning procedure, only the center is propagated to the next layer according to the LMOM

defuzzification operation for simplicity. This simplification also comes from the nature of the LMOM defuzzifier [26].

Different nodes in layer 3 may be connected to a same node in this layer, meaning that the same consequent is specified for different rules. The function of each output term node performs the following fuzzy OR operation:

$$a^{(4)} = \sum_i u_i^{(4)} \tag{4}$$

to integrate the fired rules which have the same consequent part. The above fuzzy OR operation is a modified bounded sum operation in fuzzy theory [1]. Again, its use is for computational convenience. For the same reason, it was also used in other neural fuzzy networks [26], [48]. Although the use of simple summation as the fuzzy OR operation in (4) would give values larger than one, which are strictly speaking not fuzzy by definition, succeeding normalization in (5) makes the summation contributing to each output term smaller than or equal to one. Notice that the fuzzy OR operation in (4) may cause subsequent defuzzifications to be lopsided toward those output terms having many rules in normal fuzzy inference systems. However, this lopsided effect can be reduced or even avoided by designing proper learning scheme for a neural fuzzy network. This problem has been attacked by our learning algorithm derived in the next section. Especially, the overlapping test in (9) can help to find the well-behaved fuzzy rules which cause little lopsided effect.

Layer 5: Each node in this layer is called an output linguistic node and corresponds to one output linguistic variable. This layer performs the defuzzification operation. The nodes in this layer together with the links attached to them accomplish this task. The function performed in this layer is

$$y_j = a^{(5)} = \frac{\sum_i u_i^{(5)} \hat{m}_{ji}}{\sum_i u_i^{(5)}} \tag{5}$$

where $u_i^{(5)} = a_i^{(4)}$ and $\hat{m}_{ji}$, the link weight, is the center of the membership function of the ith term of the jth output linguistic variable.

Feedback Layer: This layer calculates the value of the internal variable h_i and the firing strength of the internal variable to its corresponding membership function, where the firing strength contributes to the matching degree of a rule node in layer 3. As shown in Fig. 2, two types of nodes are used in this layer, the square node named as *context node* and the circle node named as *feedback term node*, where each context node is associated with a feedback term node. The number of context nodes (and thus the number of feedback term nodes) are the same as that of output term nodes in layer 4. Each context node and its associated feedback term node corresponds to one output term node. The inputs to a context node are from all the output term nodes, and the output of its associated feedback term node is fed to the rule nodes whose consequent is the output term node corresponded to this context node. The context node functions as a defuzzifier

$$h_j = \sum_i a_i^{(4)} w_{ji} \tag{6}$$

where the internal variable h_j is interpreted as the inference result of the hidden (internal) rule, and w_{ji} is the link weight from the ith node in layer 4 to the jth internal variable. The link weight, w_{ji}, represents a fuzzy singleton in the consequent part of a rule, and also a fuzzy term of the internal variable h_j. For an internal variable, fuzzy singleton instead of fuzzy membership function is used as its fuzzy term; a fuzzy membership function on an internal variable does not make much sense in the network due to the use of LMOM defuzzification operation, where only the center of the Gaussian membership function is used. This is different from the situation for the input and output linguistic variables, where the widths of fuzzy membership functions are used for clustering the input and desired output training data. In (6), the simple weighted-sum is calculated [27], [28]. Instead of using the weighted-sum of each rule's outputs as the inference result, the conventional average weighted-sum, $h_j = \sum_i a_i^{(4)} w_{ji} / \sum_i a_i^{(4)}$, can also be used [28], [29].

As to the feedback term node, unlike the case in the space domain where a local membership function is used, a global membership function is adopted on the universe of discourse of the internal variable to simplify network structure and meet the global property of the temporal history. Here, the global property means that for a cluster in the space domain its history path (memorized by the internal variables) can be anywhere in the space at different time, so a global membership function, which covers the universe of discourse of the internal variable, is used to rank the influence degree each internal variable contributes to a rule. In this paper, the membership function $f(u) = 1/(1+e^{-u})$ is used for each internal variable. With this choice, the feedback term node evaluates the output by

$$a^{(6)} = \frac{1}{1+e^{-h_i}}. \tag{7}$$

This output is connected to the rule nodes in layer 3, which connect to the same output term node in layer 4. The outputs of feedback term nodes contain the firing history of the fuzzy rules.

With the aforementioned node functions in each layer, the RSONFIN realizes the following dynamic fuzzy reasoning:

Rule i: IF $x_1(t)$ is A_{i1} and $\cdots$ and $x_n(t)$ is $A_{\mathbf{in}}$ and $h_i(t)$ is G
THEN $y_1(t+1)$ is B_{i1} and $y_2(t+1)$ is B_{i2} and $h_1(t+1)$ is w_{1i} and $\cdots$ and $h_m(t+1)$ is w_{mi}

where x_i is the input variable, y_i is the output variable, A_{i1}, $A_{\mathbf{in}}$, G, B_{i1}, and B_{i2} are fuzzy sets, h_i is the internal variable, w_{1i} and w_{mi} are fuzzy singletons, and n and m are the numbers of input and internal variables, respectively. To give a clear understanding of the dynamic reasoning, we decompose the above fuzzy rule into two parts [10], the external rule and the internal rule, both of which form a hierarchical relation.

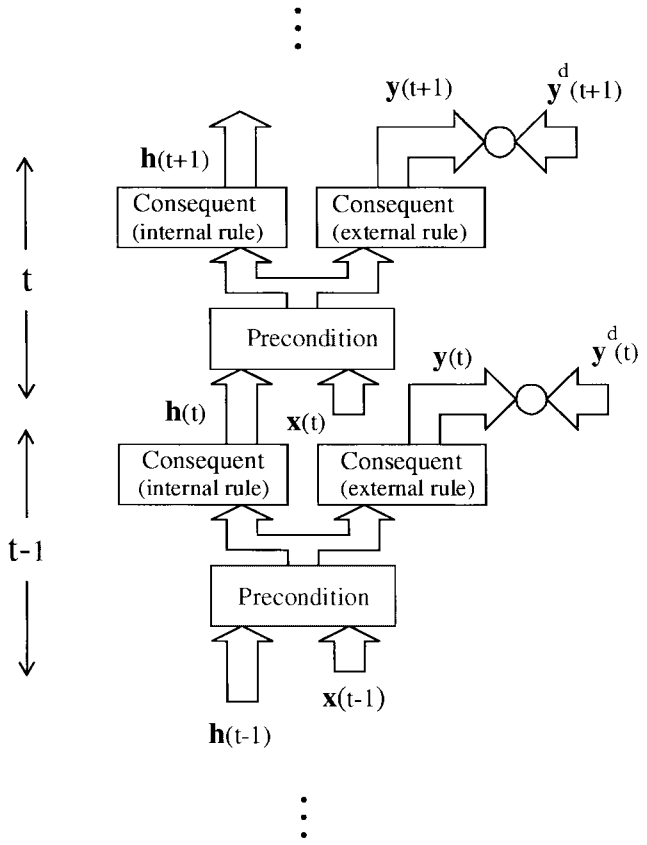


Fig. 3. Function block diagram of the RSONFIN unfolded with time.

The external rule realizes the following reasoning:

Rule i: IF $x_1(t)$ is A_{i1} and $\cdots$ and $x_n(t)$ is A_{in} and $h_i(t)$ is G
THEN $y_1(t+1)$ is B_{i1} and $y_2(t+1)$ is B_{i2}

where the outputs are functions of the internal variables acting as internal memories. The internal variables themselves constitute a dynamic internal rule as

Rule i: IF $x_1(t)$ is A_{i1} and $\cdots$ and $x_n(t)$ is A_{in} and $h_i(t)$ is G
THEN $h_1(t+1)$ is w_{1i} and $\cdots$ and $h_m(t+1)$ is w_{mi}.

The hierarchical and temporal relationship between the internal and external fuzzy rules can be easily recognized if we unfold the RSONFIN in the time domain as shown in Fig. 3, where a rule is fired by the outputs of rules which were fired one time step ahead. This figure is obtained by copying the RSONFIN at each time step, and then properly connecting the outputs of a RSONFIN at time $t-1$ to the inputs of the next RSONFIN at time t, where all copies of the RSONFIN's are identical one another in network parameters and structure.

III. Learning Algorithms for the RSONFIN

Two phases of learning, structure and parameter learning, are used concurrently for constructing the RSONFIN. The structure learning includes the preconditions, consequents, and feedback structure identification of dynamic fuzzy if–then rules. 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. The consequent structure identification is to decide whether a new membership function should be generated for the output variable based on clustering. As to the feedback structure identification, the main task is to decide the number of internal variables with its corresponding feedback fuzzy terms and the connections of these terms to each rule. For the parameter learning, based upon supervised learning, an ordered derivative learning algorithm is derived to update the free parameters in the RSONFIN. The RSONFIN 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 is no rule (i.e., no node in the network except the input–output linguistic nodes) in the RSONFIN 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) feedback structure identification;
4) parameter identification.

In the above, processes 1)–3) belong to the structure learning phase and process 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. Even though the precondition part of a rule in the RSONFIN includes the external inputs which represent the spatial information and the internal variable values which represent the temporal information, only the spatial information is used for clustering due to its local mapping property.

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 spatial firing strength derived in (3) directly as this degree measure

$$F^i(\mathbf{x}) = \prod_i u_i^{(3)} = e^{-[D_i(\mathbf{x}-\mathbf{m}_i)]^T[D_i(\mathbf{x}-\mathbf{m}_i)]} \tag{8}$$

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

$$J = \arg \max_{1 \le j \le c(t)} F^j(\mathbf{x}) \tag{9}$$

where $c(t)$ is the number of existing rules at time t. If $F^J \leq \overline{F}_{\text{in}}(t)$, then a new rule is generated, where $\overline{F}_{\text{in}}(t) \in (0, 1)$ 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

$$\mathbf{m}_{(c(t)+1)} = \mathbf{x} \tag{10}$$

$$D_{(c(t)+1)} = \frac{-1}{\beta} \cdot \text{diag}(1/\ln(F^J) \cdots 1/\ln(F^J)) \tag{11}$$

according to the first-nearest-neighbor heuristic [30], where $\beta \geq 0$ decides the overlap degree between two clusters. Similar methods are used in [31] and [33] for the allocation of a new radial basis unit. However, in [31], the degree measure does not take the width D into consideration. In [33], the width of each unit is kept at a prespecified constant value, so the allocation result is, in fact, the same as that in [31]. In the RSONFIN, 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 [31] 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 (10) and (11) to the corresponding one-dimensional membership functions for each input variable. For the Gaussian membership function used in the RSONFIN, the task can be easily done as

$$e^{-[D_i(\mathbf{x}-\mathbf{m}_i)]^T[D_i(\mathbf{x}-\mathbf{m}_i)]} = \prod_j \exp\left(-\frac{(x_j - m_{ij})^2}{\sigma_{ij}^2}\right) \tag{12}$$

where m_{ij} and σ_{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 redundant fuzzy sets, we should check the similarities between them in each input dimension. Since bell-shaped membership functions are used in the RSONFIN, we use the formula of the similarity measure of two fuzzy sets with bell-shaped membership functions derived previously in [34]. The detailed algorithm for the input space partitioning can be found in the Appendix.

For the output space partitioning, the same measure in (9) 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.

B. Construction of Fuzzy Rules

As mentioned in learning process A, 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 learning process A and the feedback structure identification scheme to be described below in learning process C. 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 $(\mathbf{x}, \mathbf{d})$. 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 $J = \arg\max_j F^j(\mathbf{d})$,
    IF $F^J \geq \overline{F}_{\text{out}}(t)$
    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 [35], where each rule has its own individual singleton value, fewer parameters are needed in the consequent part of the RSONFIN, especially for the case with a large number of rules.

C. Feedback Structure Identification

In learning process B, the number of generated clusters in the consequent part is problem dependent. The number of output clusters is large for complex problems and is small for simple ones. Naturally, in the feedback layer, more internal variables are required for more complex problems. Knowing this relationship (i.e., the increment of internal variables as well as output clusters for solving a more complex problem), for simplicity, we simply set the number of internal variables equal to the number of output clusters in the consequent part (i.e., the number of output term nodes in layer 4). With this setting, for each output cluster, the corresponding internal variable is used to record the temporal history that should participate in the precondition part of that output cluster. Hence during the on-line learning, an internal variable (and thus a context node) is created once an output cluster is created. The fan-in of the context node comes from all the nodes in layer 4), with the link weight assigned with a small random value in $[-1, 1]$ initially. This assignment is to make the initial value of internal variable h_i (i.e., input of the global membership function G) locate in the sensitive region of G. Thus, a quick parameter learning can be reached at the beginning. After an internal variable is generated (meaning a context node is created), the next step is to decide its effect on each rule node. As mentioned in Section II, only a global membership function is assigned to each internal variable and acts as the feedback term node of the corresponding context node. Of course, we can cover the universe of discourse of the internal variable by

some local membership functions, but this makes the network structure large and complex. When the firing degree of each internal variable to its corresponding membership function is calculated, we should next decide which rules the firing degree is acted on. In other words, we should decide which rule nodes in layer 3 a generated feedback term node should be connected to. In general, each rule has its own corresponding internal variable, which is to memorize the firing history of the rule. But for the rules that have the same consequent part (i.e., being connected to the same output term node), the same internal variable is assigned to these rules. Thus each internal variable can memorize the history that an output cluster is mapped by its attached fuzzy rules. With this connection method (i.e., the feedback term node is connected to the rule that maps to the corresponding output cluster it memorizes), we can effectively reduce the parameter number in the feedback layer.

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. Since the RSONFIN is a dynamic system with feedback connections, the learning algorithm used in the feedforward radial basis function networks [32] or adaptive fuzzy systems [36] cannot be applied to it directly. Also, due to the on-line learning property of the RSONFIN, the off-line learning algorithms for the recurrent neural networks, like backpropagation through time and time-dependent recurrent backpropagation [1], cannot be applied here. Instead, the ordered derivative [37], which is a partial derivative whose constant and varying terms are defined using an ordered set of equations, is used to derive our learning algorithm. The ordered set of equations are described in Section II in each layer and are summarized in (14)–(17). Considering the single-output case for clarity, our goal is to minimize the error function

$$E(t+1) = \tfrac{1}{2}(y_j(t+1) - y_j^d(t+1))^2 \tag{13}$$

where $y_j^d(t+1)$ is the desired output and $y_j(t+1)$ is the current output. For each training pattern, 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 $y_j(t+1)$. In the following, dependency on time t will be omitted unless emphasis on temporal relationships is required.

Summarizing the node functions defined in Section II, the function performed by the network is

$$y_j(t+1) = \frac{\sum_i u_i^{(5)} \hat{m}_{ji}}{\sum_i u_i^{(5)}} \tag{14}$$

$$u_i^{(5)} = \sum_k a_k^{(6)} u_{ik}^{(4)} = \sum_k a_k^{(6)} \prod_i u_{ki}^{(3)} = \sum_k a_k^{(6)} \exp\left\{ -\sum_i \left(\frac{x_i(t) - m_{ik}}{\sigma_{ik}} \right)^2 \right\} \tag{15}$$

where

$$a_k^{(6)} = \frac{1}{1 + e^{-h_k}} \tag{16}$$

and

$$h_k(t) = \sum_\ell w_{k\ell} a_\ell^{(4)}(t-1) = \sum_\ell w_{k\ell} a_\ell^{(6)}(t-1) \cdot \exp\left\{ -\sum_j \left(\frac{x_j(t-1) - m_{j\ell}}{\sigma_{j\ell}} \right)^2 \right\}. \tag{17}$$

With the above formula and the error function defined in (13), we can derive the update rules for the free parameters in the RSONFIN as follows.

- Update rule of $\hat{m}_{ji}$ (the center of the output membership function).

The update rule of $\hat{m}_{ji}$ is

$$\hat{m}_{ji}(t+1) = \hat{m}_{ji}(t) - \eta \frac{\partial^+ E}{\partial \hat{m}_{ji}}(t+1) \tag{18}$$

where

$$\frac{\partial^+ E}{\partial \hat{m}_{ji}}(t+1) = (y_j(t+1) - y_j^d(t+1)) \frac{u_i^{(5)}}{\sum_i u_i^{(5)}}. \tag{19}$$

- Update rule of m_{pq} (the center of the membership function in the precondition part).

The update rule of m_{pq} is

$$m_{pq}(t+1) = m_{pq}(t) - \eta \frac{\partial^+ E}{\partial m_{pq}}(t+1). \tag{20}$$

The value of $(\partial^+ E / \partial m_{pq})(t+1)$ is computed by

$$\frac{\partial^+ E}{\partial m_{pq}}(t+1) = (y_j(t+1) - y_j^d(t+1)) \sum_k \cdot \frac{\partial y_j(t+1)}{\partial a_k^{(3)}(t)} \frac{\partial^+ a_k^{(3)}}{\partial m_{pq}}(t) \tag{21}$$

where

$$\frac{\partial y_j(t+1)}{\partial a_k^{(3)}(t)} = \frac{\hat{m}_{jk} - y_j(t+1)}{\sum_k a_k^{(3)}(t)} \tag{22}$$

$$\frac{\partial^+ a_k^{(3)}(t)}{\partial m_{pq}} = \frac{\partial^+}{\partial m_{pq}} \left\{ a_k^{(6)} \cdot \exp\left\{ -\sum_i \left(\frac{x_i(t) - m_{ik}}{\sigma_{ik}} \right)^2 \right\} \right\} \tag{23}$$

$$= \frac{\partial^+ a_k^{(6)}}{\partial m_{pq}} \cdot \mu_k + a_k^{(6)} \cdot \mu_k \cdot 2 \frac{x_p(t) - m_{pq}}{\sigma_{pk}^2} \cdot \delta \tag{24}$$

where

$$\delta = \begin{cases} 1, & \text{if the membership function with center } m_{pq} \\ & \text{is in the precondition part of rule } k \\ 0, & \text{otherwise} \end{cases} \tag{25}$$

and

$$\mu_k = \exp\left\{-\sum_i \left(\frac{x_i(t)-m_{ik}}{\sigma_{ik}}\right)^2\right\}.$$

The partial derivative $\partial^+ a_k^{(6)}/\partial m_{pq}$ is calculated as

$$\begin{aligned}\frac{\partial^+ a_k^{(6)}}{\partial m_{pq}}(t) &= a_k^{(6)}(t)\Big(1-a_k^{(6)}(t)\Big)\frac{\partial^+ h_k}{\partial m_{pq}}(t) \\ &= a_k^{(6)}(t)\Big(1-a_k^{(6)}(t)\Big) \\ &\quad \cdot \left\{\sum_\ell w_{k\ell}\left[\frac{\partial^+ a_\ell^{(6)}}{\partial m_{pq}}(t-1)\mu_\ell(t-1)\right.\right. \\ &\quad + \mu_\ell(t-1)a_\ell^{(6)}(t-1) \\ &\quad \left.\left. \cdot\ 2\,\frac{x_p(t-1)-m_{pq}}{\sigma_{p\ell}^2}\cdot\hat{\delta}\right]\right\}\end{aligned} \tag{26}$$

where

$$\hat{\delta} = \begin{cases} 1, & \text{if the membership function with center } m_{pq} \\ & \text{is in the precondition part of the rule that} \\ & \text{is connected to node } \ell \text{ in layer 4} \\ 0, & \text{otherwise.} \end{cases} \tag{27}$$

- Update rule of σ_{pq} (the width of the membership function in the precondition part).

The update rule of σ_{pq} is

$$\sigma_{pq}(t+1) = \sigma_{pq}(t) - \eta\,\frac{\partial E}{\partial \sigma_{pq}}(t+1) \tag{28}$$

where

$$\begin{aligned}\frac{\partial E}{\partial \sigma_{pq}}(t+1) &= (y_j(t+1)-y_j^d(t+1))\sum_k \\ &\quad \cdot \frac{\partial y_j(t+1)}{\partial a_k^{(3)}(t)}\,\frac{\partial^+ a_k^{(3)}}{\partial \sigma_{pq}}(t).\end{aligned} \tag{29}$$

The partial derivative $\partial^+ a_k^{(3)}/\partial \sigma_{pq}$ is

$$\frac{\partial^+ a_k^{(3)}}{\partial \sigma_{pq}} = \frac{\partial^+ a_k^{(6)}}{\partial \sigma_{pq}}\cdot\mu_k + a_k^{(6)}\cdot\mu_k\cdot 2\,\frac{(x_p(t)-m_{pq})^2}{\sigma_{pk}^3}\cdot\delta \tag{30}$$

where

$$\delta = \begin{cases} 1, & \text{if the membership function with width } \sigma_{pq} \\ & \text{is in the precondition part of rule } k \\ 0, & \text{otherwise} \end{cases} \tag{31}$$

and the partial derivative $\partial^+ a_k^{(6)}/\partial \sigma_{pq}$ is

$$\frac{\partial^+ a_k^{(6)}}{\partial \sigma_{pq}}(t) = a_k^{(6)}(t)\Big(1-a_k^{(6)}(t)\Big)\frac{\partial^+ h_k}{\partial \sigma_{pq}}(t) \tag{32}$$

$$\begin{aligned}&= a_k^{(6)}(t)\Big(1-a_k^{(6)}(t)\Big) \\ &\quad \cdot \left\{\sum_\ell w_{k\ell}\left[\frac{\partial^+ a_\ell^{(6)}}{\partial m_{pq}}(t-1)\mu_\ell(t-1)\right.\right. \qquad (33) \\ &\quad + \mu_\ell(t-1)a_\ell^{(6)}(t-1) \\ &\quad \left.\left. \cdot\ 2\,\frac{(x_p(t-1)-m_{pq})^2}{\sigma_{p\ell}^3}\cdot\hat{\delta}\right]\right\} \qquad (34)\end{aligned}$$

where

$$\hat{\delta} = \begin{cases} 1, & \text{if the membership function with width } \sigma_{pq} \\ & \text{is in the precondition part of the rule that is} \\ & \text{connected to node } \ell \text{ in layer 4} \\ 0, & \text{otherwise.} \end{cases} \tag{35}$$

- Update rule of w_{pq} (the memory weight parameter in the feedback layer).

The update rule of w_{pq} is

$$w_{pq}(t+1) = w_{pq}(t) - \eta_w\,\frac{\partial E}{\partial w_{pq}}(t+1) \tag{36}$$

where

$$\begin{aligned}\frac{\partial E}{\partial w_{pq}}(t+1) &= (y_j(t+1)-y_j^d(t+1))\sum_k \\ &\quad \cdot \frac{\partial y_j(t+1)}{\partial a_k^{(3)}(t)}\,\frac{\partial^+ a_k^{(3)}}{\partial w_{pq}}(t)\end{aligned} \tag{37}$$

and

$$\frac{\partial^+ a_k^{(3)}}{\partial w_{pq}} = \frac{\partial^+}{\partial w_{pq}}\Big(a_k^{(6)}\cdot\mu_k\Big) = \frac{\partial^+ a_k^{(6)}}{\partial w_{pq}}\cdot\mu_k \tag{38}$$

$$= a_k^{(6)}\Big(1-a_k^{(6)}\Big)\frac{\partial^+ h_k}{\partial w_{pq}}\cdot\mu_k \tag{39}$$

where

$$\begin{aligned}\frac{\partial^+ h_k}{\partial w_{pq}}(t) &= \frac{\partial^+}{\partial w_{pq}}\left(\sum_\ell w_{k\ell}a_\ell^{(4)}(t-1)\right) \\ &= a_q^{(4)}(t-1)\delta_{kp} + \sum_\ell w_{k\ell}\,\frac{\partial^+ a_\ell^{(6)}}{\partial w_{pq}} \\ &\quad \cdot (t-1)\mu_\ell(t-1).\end{aligned} \tag{40}$$

Hence, we have the following recursive form:

$$\begin{aligned}\frac{\partial^+ a_k^{(6)}}{\partial w_{pq}}(t) &= a_k^{(6)}(t)\Big(1-a_k^{(6)}(t)\Big)\left[(a_q^{(4)}(t-1)\delta_{kp}\right. \\ &\quad \left. + \sum_\ell w_{k\ell}\,\frac{\partial^+ a_\ell^{(6)}}{\partial w_{pq}}(t-1)\mu_\ell(t-1)\right].\end{aligned} \tag{41}$$

The values $\partial^+ a^{(6)}/\partial m$, $\partial^+ a^{(6)}/\partial \sigma$, and $\partial^+ a^{(6)}/\partial w$ are equal to zero initially and are reset to zero after a period

time to avoid the accumulation of too far away errors. Note that two different learning constants are used in the above equations—η_w for the tuning of memory weight w, and η for the remaining parameters. Except the memory weight parameter w which is assigned randomly initially, the other parameters all have good initial values assigned during the structure learning phase. Owning to this good initial assignment, the convergence of these parameters is usually faster than that of the weight parameter w. To increase the learning speed of temporal relationship (i.e., tuning of the weight parameter w), we may set the learning constant η_w several times larger than η, so the convergence speed of all parameters is about the same. The learning algorithm derived above is used in the following examples. Notice that according to the real time recurrent learning (RTRL) scheme [38], we can also obtain the same parameter learning rules for the RSONFIN. Of course, other existing on-line learning algorithms [39], [40] for tuning the weights of recurrent neural networks can be possibly adopted for tuning the RSONFIN, too.

IV. Simulations

To verify the performance of the RSONFIN for temporal problems, several examples are presented in this section and performance comparisons with some existing recurrent neural networks as well as feedforward networks are also made. The examples illustrated here include the problem of time-series prediction, nonlinear IIR filtering, dynamic plant identification and control. In the following simulations, the parameters β is set as 0.6 in the RSONFIN learning algorithm. The number of training epochs chosen for the RSONFIN in each example is determined based on the desired accuracy. In each example, we first train the compared network(s) extensively to get its (their) best performance (i.e., smallest convergence error). We then use this error as the training goal for our RSONFIN to achieve. Once the RSONFIN has been trained to achieve the same accuracy as the compared network(s), it is trained continuously for some more epochs to achieve even higher accuracy. The number of these additional training epochs is chosen heuristically; we stop the training any time when the results have demonstrated that the RSONFIN can achieve higher accuracy in fewer training epochs (time steps) than the compared counterparts. In short, the training of RSONFIN is stopped once its high learning efficiency has been demonstrated.

A. Time Sequence Prediction

To clearly verify if the proposed RSONFIN can learn the temporal relationship, a simple sequence prediction problem used in [41] is used for test in the following example.

Example 1: The test bed used is shown in Fig. 4(a). This is an "8" shape made up with a series of 12 points which are to be presented to the network in a given order as shown. The network is asked to predict the succeeding point for every presented point. Obviously, this task cannot be accomplished by a static network because the point at coordinate (0, 0) has two successors: point 5 and point 11. The network must decide the successor of (0, 0) based on its predecessor; if the predecessor is 3, then the successor is 5, whereas if the predecessor is 9, the successor is 11.

In applying the RSONFIN to this prediction problem, the learning rates $\eta = \eta_w = 0.075$, $\overline{F}_{\text{in}} = 0.1$, $\rho = 0.75$, and $\overline{F}_{\text{out}} = 0.1$ are chosen. The network contains only two input linguistic nodes which are activated with the two-dimensional coordinate of the current point, and two output nodes whose values represent the two-dimension coordinate of the predicted point. The training was run for only 1600 epochs. After training, six input clusters (rules) and four output clusters are generated. The numbers of fuzzy sets on $x_1(k)$ and $x_2(k)$ dimensions are 5 and 4, respectively. The number of internal variables is equal to the number of output clusters, 4. Fig. 4(b) illustrates the distribution of training patterns and the final assignment of fuzzy rules (i.e., distribution of input membership functions) in the $[x_1(k), x_2(k)]$ plain. Since the region covered by a Gaussian membership function is unbounded, in Fig. 4(b) and the succeeding similar figures in this paper, 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 $\overline{F}_{\text{in}}$. Hence these data are in fact covered by the input clusters. The predicted values are shown in Fig. 4(c). The learned one-dimensional membership functions on $x_1(k)$, $x_2(k)$, $x_1(k+1)$, and $x_2(k+1)$ are shown in Fig. 5. To give a clear understanding of this performance, comparison with the block-structured recurrent network [41] on the same problem is made in Table I. This comparison shows that fewer network parameters and learning epochs are required for the RSONFIN, whereas a smaller error is achieved. We also used a traditional (nonrecurrent) neural fuzzy network to solve this time sequence prediction problem. The number of parameters used in the feedforward neural fuzzy network is the same as that in the RSONFIN. The prediction result after training is shown in Fig. 4(d) verifying that a feedforward neural fuzzy network fed with current point $[x_1(k), x_2(k)]$ as input only cannot do the prediction successfully. To solve the problem with the feedforward neural fuzzy network, we need to feed four points $x_1(k-1)$, $x_2(k-1)$, $x_1(k)$, and $x_2(k)$ to it. With three rules, 36 parameters, and 1600 epochs of training, an accuracy of MSE $= 0.163$ was achieved. Only when five rules (60 parameters in total) were used, a better result (MSE $= 0.02$) was achieved by the feedforward neural fuzzy network.

B. Adaptive Noise Cancellation

Adaptive noise cancellation is concerned with the enhancement of noise-corrupted signal and is based on the availability of a primary input source and an auxiliary (reference) input source located at the noise field which contains no or little desired signal as shown in Fig. 6. In Fig. 6, the primary input source contains the desired signal $s(k)$ corrupted by a noise signal $n(k)$, which is a filtered version of the noise source $r(k)$. The auxiliary input source receives the noise source directly and the measured value is used as input to an adaptive filter.

The principle of the adaptive noise cancellation techniques is to adaptively process (by adjusting the filter's weights) the

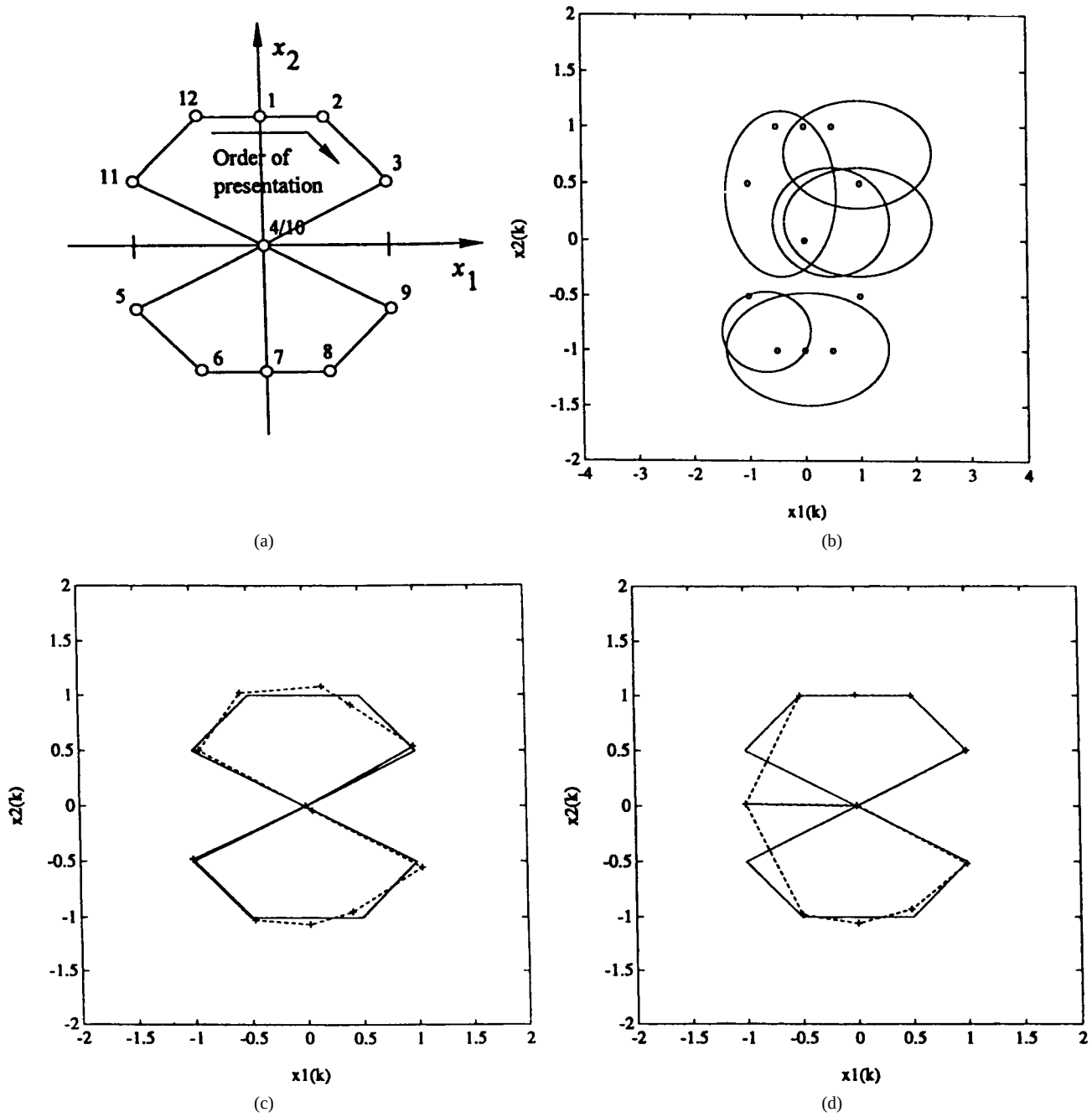


Fig. 4. (a) Test bed for the next sample prediction experiment in Example 1. (b) The input training patterns and the final assignment of rules. (c) Results of prediction using the RSONFIN after 800 training epochs. (d) Results of prediction using the nonrecurrent neural fuzzy network after 800 training epochs.

reference noise $r(k)$ to generate a replica of $n(k)$, and then subtract the replica of $n(k)$ from the primary input $x(k) = s(k) + n(k)$ to recover the desired signal $s(k)$. We denote the replica of $n(k)$, i.e., the adaptive filter output, as process $y(k)$. To show how the system works, we shall follow what is derived in [42]. In [42], the assumptions that $s(k)$, $n(k)$, and $r(k)$ are stationary zero-mean processes, $s(k)$ is uncorrelated with $n(k)$ and $r(k)$, and $n(k)$ and $r(k)$ are correlated, are made. Also, the reference input source is situated in such a position that it detects only the noise and not the signal $s(k)$. Here, another constraint that process $y(k)$ is uncorrelated with process $s(k)$ is added due to the use of nonlinear adaptive filters. From Fig. 6, we have

$$e(k) = s(k) + n(k) - y(k). \tag{42}$$

By squaring and taking expectation on both sides, we can obtain

$$E[e^2(k)] = E[s^2(k)] + E[(n(k) - y(k))^2]. \tag{43}$$

Our objective is to minimize $E[(n(k) - y(k))^2]$. Observing (43), we can see that this objective is equivalent to minimizing $E[e^2(k)]$, and when $E[(n(k) - y(k))^2] = E[(n(k) - f(r(k))^2]$ approaches zero, the remaining error $e(k)$ is in fact the desired signal $s(k)$, where $f(\cdot)$ represents the function of the nonlinear adaptive filter.

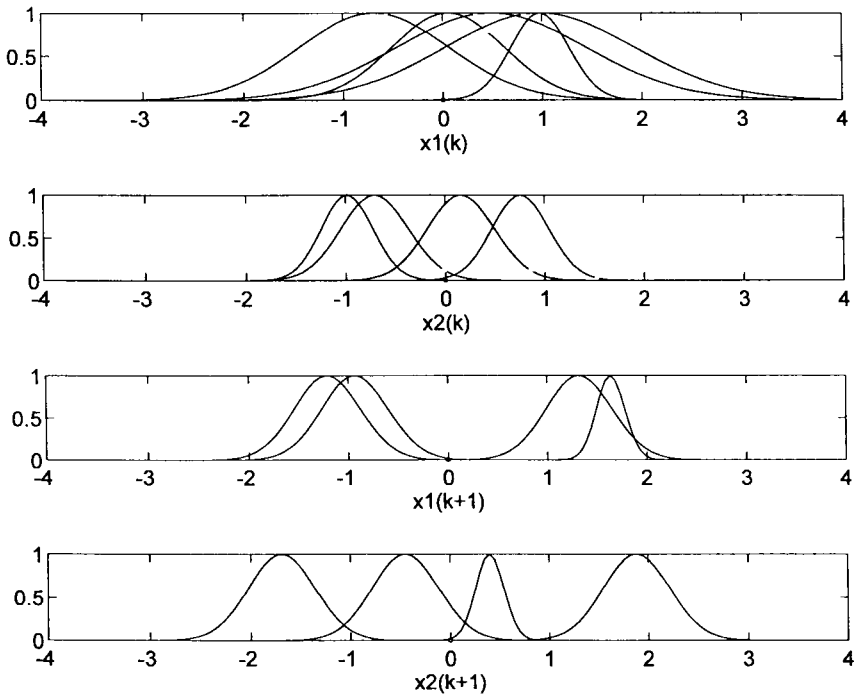


Fig. 5. The distribution of learned membership functions on the $x_1(k)$, $x_2(k)$, $x_1(k+1)$, and $x_2(k+1)$ dimensions in Example 1.

TABLE I
COMPARISONS OF THE RSONFIN WITH EXISTING RECURRENT NETWORKS, WHERE THE ERROR IN EXAMPLES 3 AND 4 IS THE MEAN SQUARE ERROR BETWEEN THE PLANT AND REFERENCE OUTPUTS OVER 1000 TIME STEPS OF THE TEST SIGNAL

Examples	Example 1		Example 3		Example 4	
Network structure	RSONFIN	block-structured recurrent network [41]	RSONFIN	memory neural network (6:1) [45]	RSONFIN	memory neural network (6:1) [45]
Network parameter	38	48	30	81	77	131
Training time	1600 epochs	20000 epochs	9000 time steps	62000 time steps	11000 time steps	77000 time steps
Mean square error	0.0272	0.0467	0.0441	0.0752	y_{p1}=0.0124 y_{p2}=0.0197	y_{p1}=0.0186 y_{p2}=0.0327

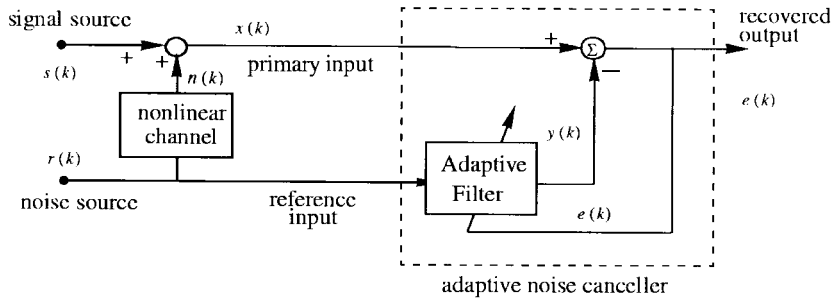


Fig. 6. Adaptive noise cancellation system in Example 2.

Traditionally, the design of an adaptive filter for the aforementioned noise cancelling is based on a linear filter, which may work well only for linear channel. If the channel is nonlinear, a neural network can be used. In the following example, we assume the channel is of nonlinear IIR type, and thus a nonlinear recurrent filter is required.

Example 2: Consider the case where the primary input signal is a sequence of Mandarin digits speech, and the noise signals are from the NOISEX-92 database [52]. Assume that the relation between noise source $r(k)$ and corrupting noise $n(k)$ is a dynamic nonlinear function

$$\begin{aligned} n(k) = {} & 0.25n(k-1) + 0.1n(k-2) \\ & + 0.08r(k-5)n(k-2) + 0.5r(k-4) \\ & + 0.1r(k-5)^2 + 0.3r(k-5) - 0.2r(k-6). \end{aligned} \quad (44)$$

Suppose the noise source r is fed to the filter input directly.

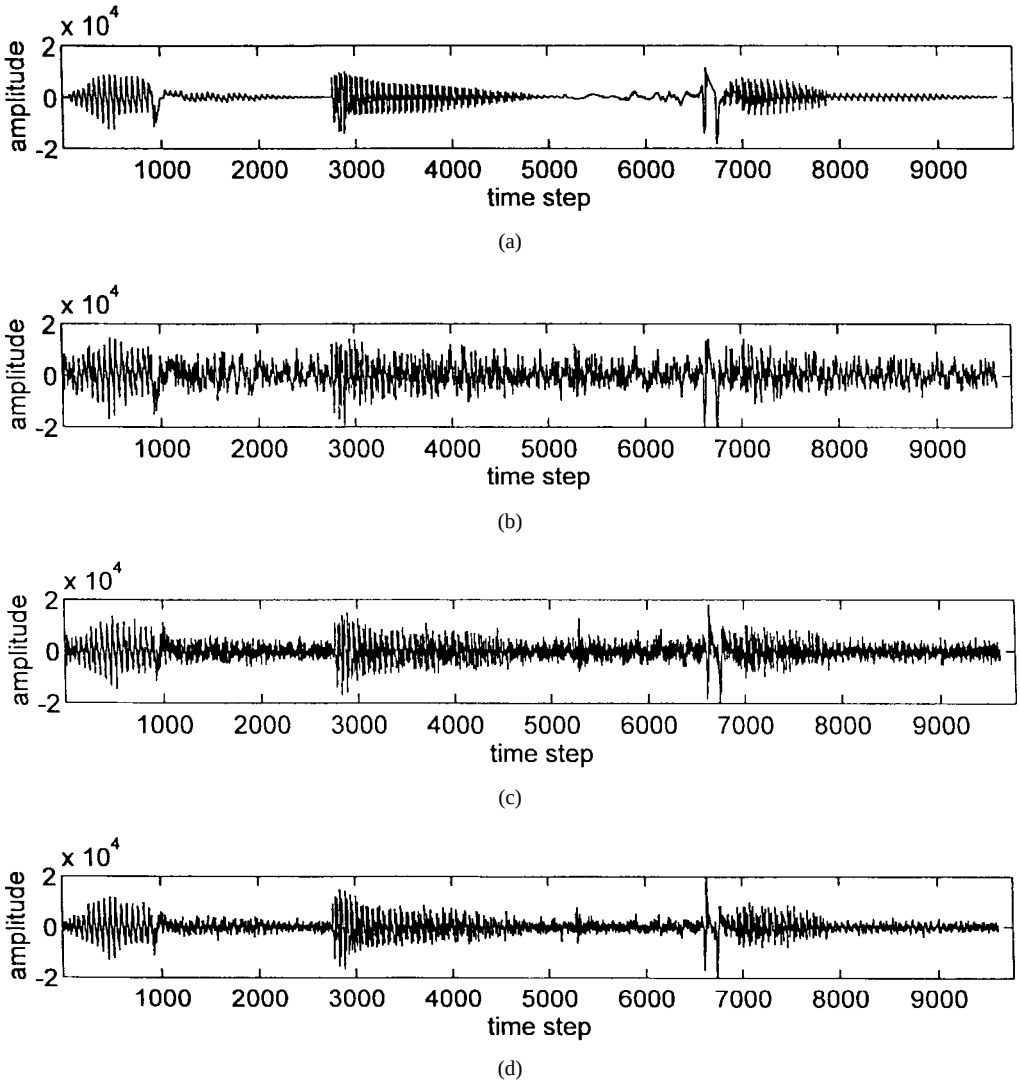


Fig. 7. (a) The original speech signal $s(k)$. (b) The corrupted speech signal $x(k)$. (c) The recovered speech signal by the feedforward fuzzy adaptive filter. (d) The recovered speech signal by the RSONFIN.

The adaptive filter is implemented by the RSONFIN. Only the currently received noise signal $r(k)$ is used as the input to the RSONFIN. The noise signal used is the noise on the floor of a car factory. A word utterance, the digit "0," is recorded, and training is performed on this word in advance. The initial parameters of the RSONFIN are set as $\overline{F}_{\text{in}} = 0.35$, $\overline{F}_{\text{out}} = 0.8$, $\eta = 0.03$, and $\eta_w = 6\eta$. Ten epochs of training are performed, and four input clusters (rules), and four output clusters and internal variables are generated. Afterwards, other speech signals are spoken, and the RSONFIN is on-line tuned to recover the speech signal. The original speech signal $s(k)$ is shown in Fig. 7(a). The measured noisy speech signal $x(k)$ is shown in Fig. 7(b), where SNR ≈ -0.6 dB. The recovered signal during on-line filtering is shown in Fig. 7(d), where SNR ≈ 4.5 dB. The obtained dynamic fuzzy rules after on-line learning are:

Rule 1) IF $r(k)$ is $\mu(-0.26, 0.45)$ and $h_1(k)$ is G, THEN $y(k)$ is $\mu(0.05, 0.3)$ and $h_1(k+1)$ is -0.45 and $h_2(k+1)$ is -0.47 and $h_3(k+1)$ is 0.21 and $h_4(k+1)$ is 1.07.

Rule 2) IF $r(k)$ is $\mu(0.53, 0.75)$ and $h_2(k)$ is G, THEN $y(k)$ is $\mu(0.74, 0.13)$ and $h_1(k+1)$ is -0.39 and $h_2(k+1)$ is 1.23 and $h_3(k+1)$ is 0.56 and $h_4(k+1)$ is -1.47.

Rule 3) IF $r(k)$ is $\mu(0.68, 0.07)$ and $h_3(k)$ is G, THEN $y(k)$ is $\mu(0.66, 0.35)$ and $h_1(k+1)$ is -0.54 and $h_2(k+1)$ is 0.84 and $h_3(k+1)$ is 0.54 and $h_4(k+1)$ is -1.08.

Rule 4) IF $r(k)$ is $\mu(-0.22, 1.38)$ and $h_4(k)$ is G, THEN $y(k)$ is $\mu(-0.85, 0.35)$ and $h_1(k+1)$ is -0.68 and $h_2(k+1)$ is 1.02 and $h_3(k+1)$ is 0.42 and $h_4(k+1)$ is -0.32.

In the above rules, h_1, h_2, h_3, and h_4 are the generated internal variables, $\mu(m_i, \sigma_i)$ represents a Gaussian membership function with center m_i and width σ_i, and G is the global membership function stated previously in Section II.

For comparison, a feedforward filter, the fuzzy adaptive filter [43], with ten rules is applied to the same problem. Different numbers of variables, including $[r(k), r(k-1)]$, $[r(k), r(k-1), r(k-2)]$, and $[r(k), r(k-1), r(k-2), r(k-$

3)], are used as inputs to the fuzzy adaptive filter for comparison. The enhanced SNR's by the fuzzy adaptive filter with different orders of inputs are −0.2, 0.5, and two, respectively. Even with only one input and a smaller filter size (32 parameters in total), the RSONFIN performs better than the feedforward fuzzy adaptive filter which needs 90 parameters in total if four inputs are used. By further increasing the input dimension of the fuzzy adaptive filter to cover more delays of the input variables, we can obtain a better result. The resulting network size is, however, quite large and this is not an economic approach. The recovered speech by the fuzzy adaptive filter with input $[r(k), r(k-1), r(k-2), r(k-3)]$ is shown in Fig. 7(c).

C. Dynamic Identification

The systems to be identified here are dynamic systems whose outputs are functions of past inputs and past outputs as well. This dynamic identification problem is more complicated than the static one because the identification model to be used (e.g., artificial neural network) should have some internal memory. Although we can model the system with a memoryless feedforward network by feeding all the necessary past inputs and outputs of the system as explicit inputs to the network, some drawbacks still exist with this method. One of them is that the exact order of past values for feeding into the network is unknown in practice. Other drawbacks and detailed discussions can be found in [44] and [45].

The dynamic systems to be identified in the following examples are from [46] and are also used in [45]. As shown in [45], the main reason for using these dynamic systems is that they provide fairly complex nonlinear functions and all of them are known to be stable in the bounded input bounded output (BIBO) [49] sense. Moreover, the use of these models makes the comparison of the RSONFIN with the memory neural network proposed in [45] easier.

Example 3—Single Input Single Output (SISO) [50] Identification: The plant to be identified in this example is guided by the difference equation

$$y_p(k+1) = f(y_p(k), y_p(k-1), y_p(k-2), u(k), u(k-1)) \tag{45}$$

where

$$f(x_1, x_2, x_3, x_4, x_5) = \frac{x_1 x_2 x_3 x_5 (x_3 - 1) + x_4}{1 + x_3^2 + x_2^2}. \tag{46}$$

Here the current output of the plant depends on three previous outputs and two previous inputs. In [46], a feedforward neural network with five input nodes for feeding the appropriate past values of y_p and u is used. In our case, only two values, $y_p(k)$ and $u(k)$, are fed to the RSONFIN and the output $y_p(k+1)$ is determined. The identification system, a series-parallel model [46], is shown in Fig. 8. In training the RSONFIN, we use only 9000 time steps and, similar to the inputs used in [45], the input is an *independent and identically distributed* (i.i.d.) uniform sequence over [−2, 2] for about half of the training time and a single sinusoid signal given by $1.05 \sin(\pi k/45)$ for the remaining training time. In applying the RSONFIN to this dynamic identification problem, the learning rates $\eta = \eta_w = 0.055$, $\rho = 0.8$, $\overline{F}_{\text{in}} = 0.2$, and $\overline{F}_{\text{out}} = 0.3$ are chosen. After training, five input clusters (rules) and three output clusters and internal variables are generated and the numbers of fuzzy sets on $u(k)$ and $y(k)$ are four and five, respectively. Fig. 9 illustrates the distribution of some of the training patterns and the final assignment of the rules in the $[u(k), y(k)]$ plain. The distribution of the formed clusters in the input–output spaces might not be perfect from the data clustering point of view. This is due to the parameter learning process which tunes the mean and width of each cluster at each time step for minimizing the output error function. The membership functions on the $y(k+1)$ dimension are shown

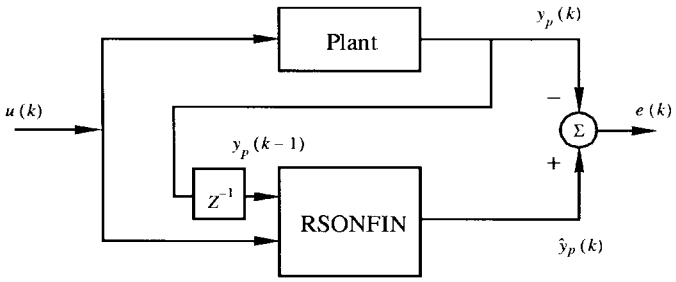


Fig. 8. Series-parallel identification model with the RSONFIN in Example 3. The current input into the plant and the most recent output of the plant are fed into the network. The error $e(k)$ is used for training the network parameters.

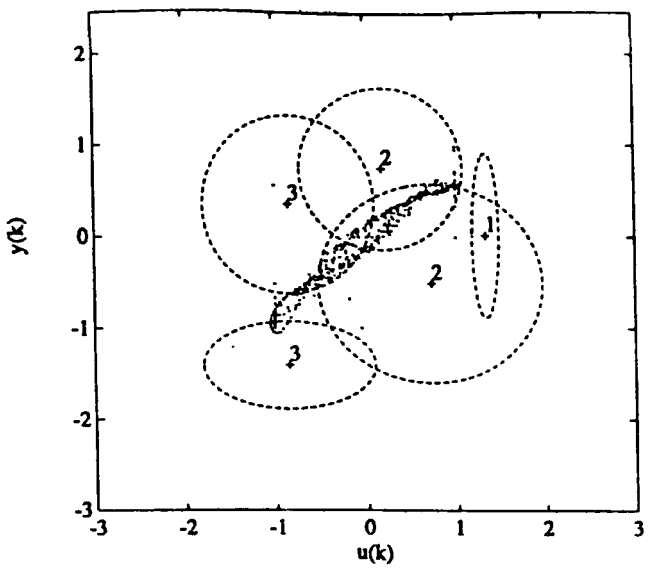


Fig. 9. The input training patterns and the final assignment of rules in Example 3. The index in the center of each cluster denotes the output cluster it maps to.

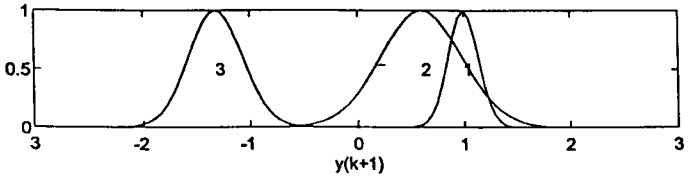


Fig. 10. The distribution of learned membership functions on the $y(k+1)$ dimension in Example 3.

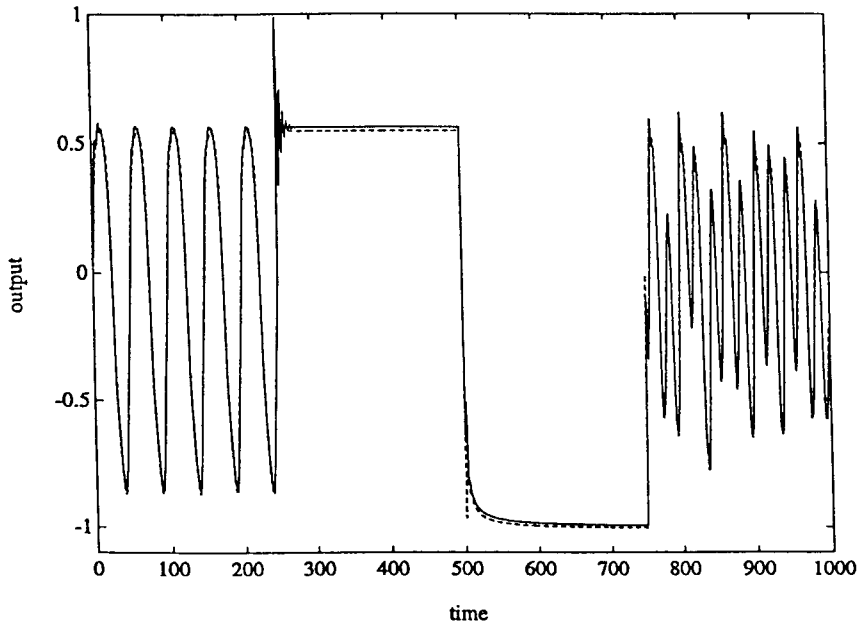


Fig. 11. Outputs of the SISO plant (solid curve) and model RSONFIN (dotted curve) in Example 3.

in Fig. 10. The index in the center of each cluster shows the output fuzzy set each rule maps to.

To see the identified result, the following input as used in [45] is adopted for test

$$\begin{aligned} u(k) &= \sin(\pi k/25) && k < 250 \\ &= 1.0, && 250 \le k < 500 \\ &= -1.0 && 500 \le k < 750 \\ &= 0.3\sin(\pi k/25) + 0.1\sin(\pi k/32) \\ &\quad + 0.6\sin(\pi k/10) && 750 \le k < 1000. \end{aligned}$$

Fig. 11 shows the outputs of the plant (denoted as a solid curve) and the RSONFIN (denoted as a dotted curve) for the test input. Detailed performance comparison of the RSONFIN and the memory neural network [45] for this identification task is given in Table I. In [46], a feedforward neural network with five inputs $y_p(k)$, $y_p(k-1)$, $y_p(k-2)$, $u(k)$, $u(k-1)$ is applied to the same problem. The numbers of parameters and training steps used are 310 and 100 000, respectively. This feedforward neural network achieved the accuracy of MSE $= 0.049$, which was close to the MSE value achieved by the RSONFIN, but the former needed much more parameters and training time steps than the latter.

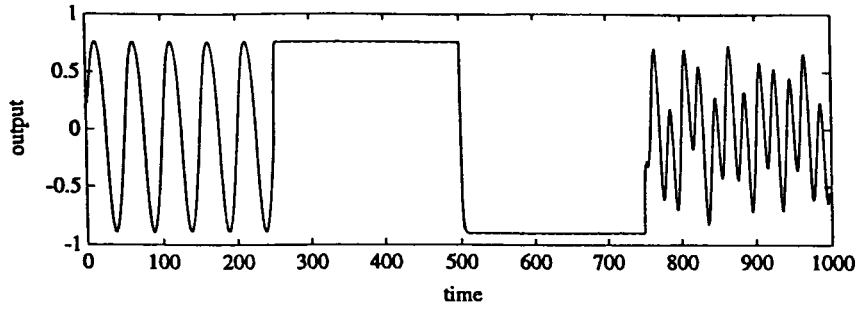
Example 4—Multiple-Input–Multiple-Output (MIMO) [50] Identification: The MIMO plant to be identified in this example is the same as that used in [45] and [46]. This plant has two inputs and two outputs, so there are four input nodes and two output nodes in the RSONFIN. The plant is specified by

$$y_{p1}(k+1) = 0.5\left[\frac{y_{p1}(k)}{1+y_{p2}^2(k)} + u_1(k-1)\right],$$
$$y_{p2}(k+1) = 0.5\left[\frac{y_{p1}(k)y_{p2}(k)}{1+y_{p2}^2(k)} + u_2(k-1)\right].$$

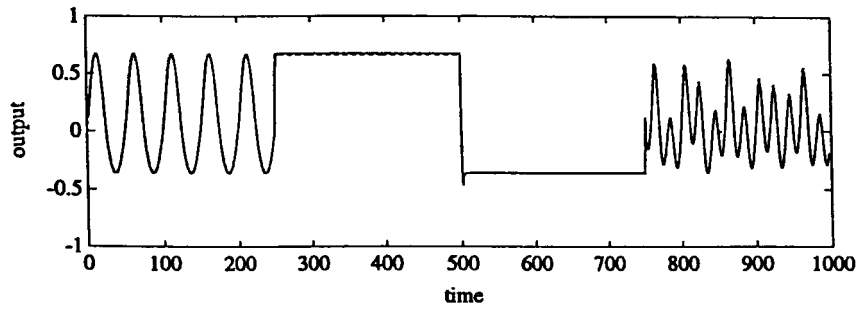
During the training phase, 11 000 time steps are used and the input sequence is the same as that used in Example 3, where about half of the training time the input is an i.i.d. uniform sequence over $[-2, 2]$ and for the remaining training time, the input is a single sinusoid signal given by $\sin(\pi k/45)$. After training, seven input clusters (rules), five output clusters, and internal variables are generated. The numbers of fuzzy sets on $u_1(k)$, $y_1(k)$, $u_2(k)$, and $y_2(k)$ are 6, 5, 4, 6, respectively. The identified results are shown Fig. 12. As in Example 3, performance comparison of the RSONFIN and the memory neural network for the same identification task is made in Table I. Through the comparisons made in Examples 3 and 4, we find that the RSONFIN needs fewer training time steps and network parameters, and achieves higher accuracy than the memory neural network.

D. Plant Control

For the plant control problem here, we focus on the control of dynamic plants. Two distinct neural control approaches, the direct and indirect control, have been used to control a plant adaptively [47]. The indirect control usually requires an identified model for the plant and the design of the controller is based on the backpropagation of the controlled output error through the identified model to train the controller. It has been pointed out in [45] that if the plant involves significant delay, then this training method encounters difficulty when the plant is identified with a feedforward neural network. For this reason, the use of a recurrent identification model is suggested. As to the direct control, we consider the direct inverse control, which is applied when the controlled plant is reversible. To learn the inverse of a dynamic plant we have to feed the exact order of inputs to the controller if a feedforward neural network is used, which may not be possible in practice. Hence a recurrent network controller is suitable. In the following example, we shall use the direct inverse control technique



(a)



(b)

Fig. 12. Outputs of the MIMO plant (solid curve) and model RSONFIN (dotted curve) in Example 4. (a) The first output $y_{p1}(k+1)$. (b) The second output $y_{p2}(k+1)$.

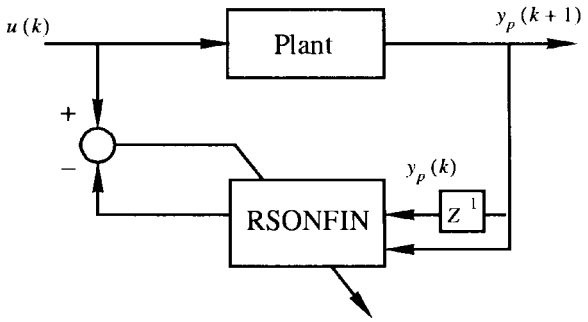


Fig. 13. Block diagram of direct inverse control with off-line learning by the RSONFIN in Example 5.

based on the RSONFIN to control a dynamic plant which is reversible.

Example 5: The controlled plant is the same as that used in [45] and [46] and is given by

$$y_p(k+1) = 0.35\left[\frac{y_p(k)y_p(k-1)(y_p(k)+2.5)}{1+y_p^2(k)+y_p^2(k-1)} + u(k)\right]. \quad (47)$$

The reference model is a second order linear system given by

$$y_m(k+1) = 0.6y_m(k) + 0.2y_m(k-1) + 0.1r(k). \quad (48)$$

The block diagram for the off-line learning of the controller is shown in Fig. 13. During off-line learning, 4000 time steps are used and for half of these time steps the input is an i.i.d. uniform sequence over $[-1, 1]$, and for the remaining time steps the input is $r(k) = \sin(\pi k/45)$. By applying these inputs to the plant, their corresponding outputs are obtained and a set of training patterns $(y_p(k), y_p(k+1); u(k))$ can thus be obtained. During training, the inputs to the RSONFIN controller are $y_p(k)$ and $y_p(k+1)$, and the desired output is the injected input $u(k)$ to the plant. The learning parameters $\eta = \eta_w = 0.025$, $\rho = 0.8$, $\overline{F}_{\text{in}} = 0.25$, and $\overline{F}_{\text{out}} = 0.8$ are chosen. Seven input clusters, four output clusters, and internal variables are generated. The numbers of generated fuzzy sets on $y_p(k+1)$ [or $y_m(k+1)$ during control phase] and $y_p(k)$

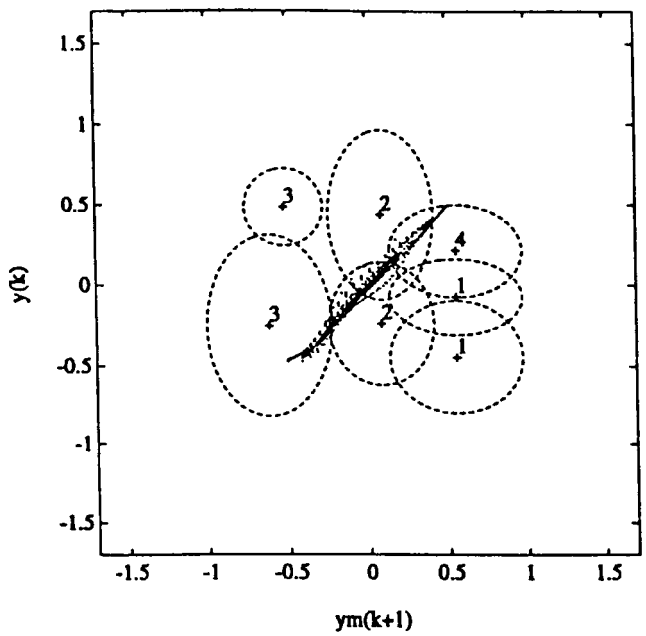


Fig. 14. The input training patterns and the final assignment of rules in Example 5. The index in the center of each cluster denotes the output cluster it maps to.

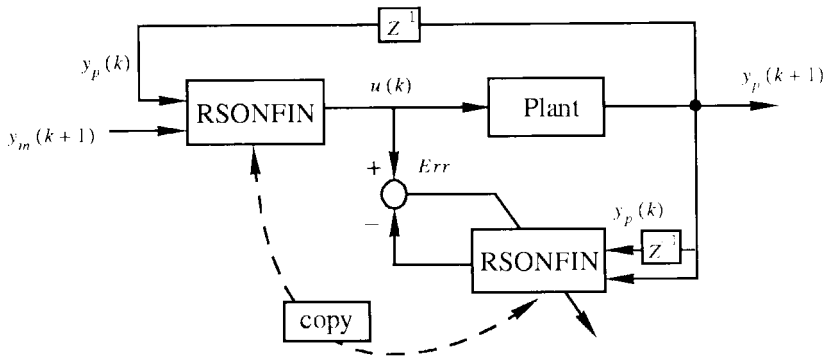


Fig. 15. Block diagram of direct inverse control with on-line learning by the RSONFIN in Example 5.

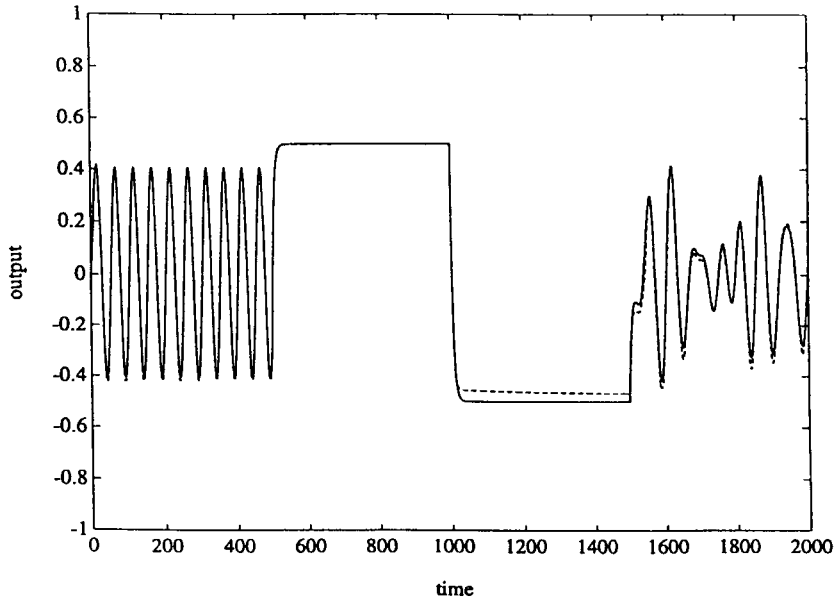


Fig. 16. The reference output (solid curve) and the RSONFIN controlled output (dotted curve) in Example 5.

are 4 and 7, respectively. Fig. 14 illustrates the distribution of some of the training patterns and the final assignment of fuzzy rules in the $[y_m(k+1), y_p(k)]$ plain. After off-line training, the RSONFIN is operated as a direct controller as shown in Fig. 15 with $y_m(k+1)$ and $y_p(k)$ being the inputs. During the control phase, on-line learning is also performed as shown in Fig. 15. To test the adaptive controller, as in [45], we use the following reference input:

$$\begin{aligned} r(k) &= \sin(\pi k/25), && k < 500 \\ &= 1.0 && 500 \le k < 1000 \\ &= -1.0 && 1000 \le k < 1500 \\ &= 0.3\sin(\pi k/25) + 0.4\sin(\pi k/32) \\ &\quad + 0.3\sin(\pi k/10) && k \ge 1500. \end{aligned}$$

The controlled result is shown in Fig. 16. In [46], a different control scheme is used, but for this control scheme much information of the plant (like the order of the plant and the plant type) must be known in advance, which might not be known in practice. In [45], an indirect control configuration is used to control the same plant using the memory neural network. Since different control configurations are used, it's difficult to compare the performance of these two networks. However, the controlled result of the RSONFIN appears to be superior to that of the memory neural network in [45] for the reversible plant.

V. Conclusions

An RSONFIN with on-line self-organizing learning capability is proposed in this paper. Basically, this network is constructed by expanding the powerful ability of a neural fuzzy network to deal with temporal problems. The RSONFIN itself realizes dynamic fuzzy reasoning by creating recursive fuzzy rules, which are generated automatically and optimally during on-line operation via concurrent structure and parameter learning. The structure identification process proposed in this paper can effectively reduce the rule number and network size, and the derived order-derivative-based parameter learning algorithm can optimally tune the parameters on both the feedforward and feedback connections. The RSONFIN can be used for normal operation at any time as learning proceeds without any assignment of fuzzy rules in advance. Simulations

in several temporal problems have demonstrated the high learning efficiency of the RSONFIN. As a contrast, the role played by the RSONFIN in the recurrent neural network domain is parallel to the role played by the feedforward neural fuzzy network in the feedforward neural network domain. The former networks are good at dynamic mapping, while the latter networks are for static mapping. Both of the RSONFIN and the feedforward neural fuzzy network have the same advantages (such as fast learning, small network size, being easy to incorporate expert knowledge) over their pure neural network counterparts.

APPENDIX

Let $\mu(m_i, \sigma_i)$ represent the Gaussian membership function with center m_i and width σ_i, and $E(A, B)$ represent the fuzzy measure of two fuzzy sets A and B. The whole algorithm for the generation of new fuzzy rules as well as fuzzy sets for each input variable is as follows. Suppose no rule exists 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
$D_1 = \text{diag}(1/\sigma_{\text{init}}, \cdots, 1/\sigma_{\text{init}})$,
where σ_{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 pattern $\mathbf{x}$, do
PART 2. { find $J = \arg\max_{1 \le j \le c(t)} F^j(\mathbf{x})$,
IF $F^J \ge \overline{F}_{\text{in}}(t)$
after a period of time
perform fuzzy measure and eliminate
unnecessary membership functions
ELSE
{ $c(t+1) = c(t) + 1$,
generate a new fuzzy rule, with
$\mathbf{m}_{c(t+1)} = \mathbf{x}$, $D_{c(t+1)} = (-1/\beta) \cdot \text{diag}$
$(1/\ln(F^J) \cdots 1/\ln(F^J))$.
After decomposition, we have
$m_{\text{new}-i} = x_i$, $\sigma_{\text{new}-i} = -\beta \cdot \ln(F^J)$,
$i = 1 \cdots n$.
Do the following fuzzy measure for each
input variable:
$\{degree(i, t) \equiv \max_{1 \le j \le k_i}$
$E[\mu(m_{\text{new}-i}, \sigma_{\text{new}-i}), \mu(m_{ji}, \sigma_{ji})]$,
where k_i is the number of partitions
of the ith input variable.
IF $degree(i, t) \le \rho$,
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, ρ is a scalar similarity criterion; higher similarity between two fuzzy sets is allowed for larger ρ.

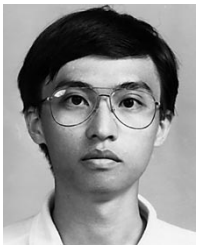
REFERENCES

[1] C. T. Lin and C. S. G. Lee, *Neural Fuzzy Systems: A Neural-Fuzzy Synergism to Intelligent Systems*. Englewood CLiffs, NJ: Prentice-Hall, May 1996.
[2] J. S. Jang, "ANFIS: Adaptive-network-based fuzzy inference system," *IEEE Trans. Syst., Man, Cybern.*, vol. 23, pp. 665–685, May 1993.
[3] 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.
[4] B. Kosko, *Neural Networks and Fuzzy Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1992.
[5] C. T. Lin, *Neural Fuzzy Control Systems with Structure and Parameter Learning*. Singapore: World, 1994.
[6] J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Computation*. Reading, MA: Addison-Wesley, 1991, ch. 7.
[7] C. L. Giles, G. M. Kuhn, and R. J. Williams, "Dynamic recurrent neural networks: Theory and applications," *IEEE Trans. Neural Networks*, vol. 5, pp. 153–156, Mar. 1994.
[8] S. A. Billings and C. F. Fung, "Recurrent radial basis function networks for adaptive noise cancellation," *Neural Networks*, vol. 8, no. 2, pp. 273–290, 1995.
[9] T. Miyoshi, H. Ichihashi, S. Okamoto, and T. Hayakawa, "Learning chaotic dynamics in recurrent RBF network," *IEEE ICNN*, pp. 588–593.
[10] V. Gorrini and H. Bersini, "Recurrent fuzzy systems," in *Proc. IEEE Int. Conf. Fuzzy Systems*, 1994, vol. 1, pp. 193–198.
[11] J. Grantner and M. Patyra, "Synthesis and analysis of fuzzy logic finite state machine models," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, 1994, vol. 1, pp. 205–210.
[12] E. Khan and F. Unal, "Recurrent fuzzy logic using neural networks," *Advances in Fuzzy Logic, Neural Network, and Genetic Algorithms*, T. Furuhashi, Ed., *Lecture Notes in Artificial Intelligence*. Berlin, Germany: Springer Verlag, 1995.
[13] E. B. Kosmatopoulos and M. A. Christodoulou, "Recurrent neural networks for approximation of fuzzy dynamical systems," *Int. J. Intell. Contr. Syst. (Special Issue Neural Networks Fuzzy Syst. Closed-Loop Applicat.)*, vol. 1, no. 2, pp. 223–233, 1996.
[14] E. Kosmatopoulos and M. Christodoulou, "Structural properties of gradient recurrent high-order neural networks," *IEEE Trans. Circuits Syst.*, 1995.
[15] E. Kosmatopoulos, M. Polycarpou, M. Christodoulou, and P. Ioannou, "High-order neural networks for identification of dynamic systems," *IEEE Trans. Neural Networks*, vol. 6, pp. 422–431, 1995.
[16] C. W. Omlin, K. K. Thornber, and C. L. Gilies, "Fuzzy finite state automata can be deterministically encoded into recurrent neural networks," *IEEE Trans. Fuzzy Syst.*, vol. 6, pp. 76–89, 1998.
[17] F. Unal and E. Khan, "A fuzzy finite state machine implementation based on a neural fuzzy system," in *IEEE Int. Conf. Fuzzy Syst.*, 1994, vol. 3, pp. 1749–1754.
[18] M. Mézard and J. P. Nadal, "Learning in feedforward layered networks: The tiling algorithm." *J. Phys.*, vol. 22, pp. 2191–2204, 1989.
[19] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Advances in Neural Information Processing Systems II*, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1990, pp. 524–532.
[20] T. C. Lee, *Structure Level Adaptation for Artificial Neural Networks*. Boston, MA: Kluwer, 1991.
[21] G. Martinelli, F. M. Mascioli, and G. Bei, "Cascade neural network for binary mapping," *IEEE Trans. Neural Networks*, vol. 4, pp. 148–150, 1993.
[22] C. L. Giles, D. Chen, G. Z. Sun, H. H. Chen, Y. C. Lee, and M. W. Goudreau, "Constructive learning of recurrent neural networks: Limitations of recurrent cascade correlation and a simple solution," *IEEE Trans. Neural Networks*, vol. 6, pp. 829–836, 1995.
[23] C. J. Lin and C. T. Lin, "Reinforcement learning for ART-based fuzzy adaptive learning control networks," *IEEE Trans. Neural Networks*, vol. 7, pp. 709–731, May 1996.
[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," in *Fuzzy Set and Possibility Theory*. New York: North Holland, 1982, pp. 113–147.
[26] H. R. Berenji and P. Khedkar, "Learning and tuning fuzzy logic controllers through reinforcements," *IEEE Trans. Neural Networks*, vol. 3, pp. 724–740, 1992.

[27] H. Takagi and I. Hayashi, "NN-driven fuzzy reasoning," *Int. J. Approximate Reasoning*, vol. 5, no. 3, pp. 191–212, 1991.
[28] J. S. Roger Jang and C. T. Sun, "Functional equivalence between radial basis function networks and fuzzy inference system," *IEEE Trans. Neural Networks*, vol. 4, pp. 156–159, 1993.
[29] T. Takagi and M. Sugeno, "Derivation of fuzzy control rules from human operator's control actions," in *Proc. IFAC Symp. Fuzzy Inform., Knowledge Representation, Decision Anal.*, July 1983, pp. 55–60.
[30] 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, 1991.
[31] J. Platt, "A resource allocating network for function interpolation," *Neural Comput.*, vol. 3, pp. 213–225, 1991.
[32] J. Moody and C. J. Darken, "Fast learning in networks of locally-tuned processing units," *Neural Comput.*, vol. 1, pp. 281–294, 1989.
[33] 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.
[34] C. T. Lin and C. S. G. Lee, "Reinforcement structure/parameter learning for neural-network-based fuzzy logic control systems," *IEEE Trans. Fuzzy Syst.*, vol. 2, pp. 46–63, Feb. 1994.
[35] 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.
[36] L. X. Wang, *Adaptive Fuzzy Systems and Control.* Englewood Cliffs, NJ: Prentice-Hall, 1994.
[37] P. Werbos, "Beyond regression: New tools for prediction and analysis in the behavior sciences," Ph.D. dissertation, Harvard Univ., Cambridge, MA, Aug. 1974.
[38] R. J. Williams and D. Zipser, "A learning algorithm for continually running recurrent neural networks," *Neural Comput.*, vol. 1, no. 2, pp. 270–280, 1989.
[39] B. A. Pearlmutter, "Gradient calculations for dynamic recurrent neural networks: A survey," *IEEE Trans. Neural Networks*, vol. 6, pp. 1212–1228, 1995.
[40] S. W. Piché, "Steepest descent algorithms for neural-network controllers and filters," *IEEE Trans. Neural Networks*, vol. 5, pp. 198–212, 1994.
[41] S. Santini, A. D. Bimbo, and R. Jain, "Block-structured recurrent neural networks," *Neural Networks*, vol. 8, no. 1, pp. 135–147, 1995.
[42] B. Widrow and S. D. Stearns, *Adaptive Signal Processing.* Englewood Cliffs, NJ: Prentice-Hall, 1985.
[43] L. X. Wang and J. M. Mendel, "Fuzzy adaptive filters with application to nonlinear channel equalization," *IEEE Trans. Fuzzy Syst.*, vol. 1, pp. 161–170, 1993.
[44] R. J. Williams, "Adaptive state representation and estimation using recurrent connectionist networks," in *Neural Networks for Control*, W. T. Miller, R. S. Sutton, and P. J. Werbos, Eds. Cambridge, MA: MIT Press, 1990.
[45] P. S. Sastry, G. Santharam, and K. P. Unnikrishnan, "Memory neural networks for identification and control of dynamic systems," *IEEE Trans. Neural Networks*, vol. 5, pp. 306–319, 1994.
[46] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Network*, vol. 1, pp. 4–27, 1990.
[47] K. S. Narendra and A. M. Annaswamy, *Stable Adaptive Systems.* Englewood Cliffs, NJ: Prentice-Hall, 1989.
[48] S. S. Horikawa, T. Furuhashi, and Y. Uchikawa, "On fuzzy modeling using fuzzy neural networks with the back-propagation algorithm," *IEEE Trans. Neural Networks*, vol. 3, pp. 801–806, 1992.
[49] F. M. Callier and C. A. Desoer, *Linear System Theory.* New York: Springer-Verlag, 1992.
[50] C. L. Phillips and H. T. Nagle, *Digital Control System.* Englewood Cliffs, NJ: Prentice-Hall, 1995.
[51] L. N. Teow and K. F. Loe, "An effective learning method for max–min neural networks," in *Proc. 15th Int. Joint Conf. Artificial Intell., NIL'97*, 1997, pp. 1134–1139.
[52] A. Varga, H. J. M. Steeneken, M. Tomlinson, and D. Jones, "The NOISEX-92 study on the effect of additive noise automatic speech recognition," Description of RSG.10 and Esprit SAM experiment and database, DRA Speech Research Unit, Malvern, U.K.

Chia-Feng Juang received the B.S. degree in control engineering from the National Chiao-Tung University, Hsinchu, Taiwan, R.O.C., in 1993. He is currently working toward the Ph.D. degree in the Department of Electrical and 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 (S'88–M'91) received the B.S. degree in control engineering from the National Chiao-Tung University, Hsinchu, 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 a Professor of Electrical and Control Engineering. He also serves as the Deputy Dean of the Research and Development Office of the National Chiao-Tung University since 1998. His current research interests are fuzzy systems, neural networks, intelligent control, human-machine interface, and video and audio processing. 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* (Singapore: World, 1994). He has published more than 30 journal papers in the areas of neural networks and fuzzy systems.

Dr. Lin is a member of Tau Beta Pi and Eta Kappa Nu. He is also a member of the IEEE Computer Society, the IEEE Robotics and Automation Society, and the IEEE Systems, Man, Cybernetics Society. He has been the Executive Council Member of Chinese Fuzzy System Association (CFSA) since 1995, and the Supervisor of Chinese Automation Association since 1998. He was the Vice Chairman of IEEE Robotics and Automation Taipei Chapter in 1996 and 1997. He won the Outstanding Research Award granted by National Science Council (NSC), Taiwan, in 1997, and the Outstanding Electrical Engineering Professor Award granted by the Chinese Institute of Electrical Engineering (CIEE) in 1997.