A TSK-Type Recurrent Fuzzy Network for Dynamic Systems Processing by Neural Network and Genetic Algorithms

Chia-Feng Juang, *Member, IEEE*

***Abstract*—In this paper, a TSK-type recurrent fuzzy network (TRFN) structure is proposed. The proposal calls for a design of TRFN by either neural network or genetic algorithms depending on the learning environment. Set forth first is a recurrent fuzzy network which develops from a series of recurrent fuzzy if–then rules with TSK-type consequent parts. The recurrent property comes from feeding the internal variables, derived from fuzzy firing strengths, back to both the network input and output layers. In this configuration, each internal variable is responsible for memorizing the temporal history of its corresponding fuzzy rule. The internal variable is also combined with external input variables in each rule's consequence, which shows an increase in network learning ability. TRFN design under different learning environments is next advanced. For problems where supervised training data is directly available, TRFN with supervised learning (TRFN-S) is proposed, and neural network (NN) learning approach is adopted for TRFN-S design. An online learning algorithm with concurrent structure and parameter learning is proposed. With flexibility of partition in the precondition part, and outcome of TSK-type, TRFN-S has the admirable property of small network size and high learning accuracy. As to the problems where gradient information for NN learning is costly to obtain or unavailable, like reinforcement learning, TRFN with Genetic learning (TRFN-G) is put forward. The precondition parts of TRFN-G are also partitioned in a flexible way, and all free parameters are designed concurrently by genetic algorithm. Owing to the well-designed network structure of TRFN, TRFN-G, like TRFN-S, also is characterized by a high learning accuracy property. To demonstrate the superior properties of TRFN, TRFN-S is applied to dynamic system identification and TRFN-G to dynamic system control. By comparing the results to other types of recurrent networks and design configurations, the efficiency of TRFN is verified.**

***Index Terms*—Control, identification, recurrent neural network, reinforcement learning.**

I. Introduction

PROBLEMS in dealing with dynamic systems are encountered in many areas, such as control, communication, and pattern recognition. In the control area, we usually face the problem of dynamic system identification and control. Since for a dynamic system, the output is a function of past output or past input or both, identification and control of this system is not as straightforward as a static system. For nonlinear system processing, the most commonly used model is the neural or neural fuzzy network. If a feedforward network is adopted for this task, then we should know the number of delayed input and output in advance, and feed these delayed input and output as a taped line to the network input [1]. The problem of this approach is that the exact order of the dynamic system is usually unknown. Besides, the usage of the long tapped delay input will increase the input dimension and will result in a large network size. To deal with this problem, interest in using recurrent networks for processing dynamic systems has been steadily growing in recent years, and a number of recurrent models have been proposed [2]–[9]. Some of them are Elman [2] and Jordan's [3] networks, which are feedforward multilayer perceptron networks with an extra set of context nodes for copying the delayed states of the hidden or output nodes back to the network input; and the fully recurrent neural network [4], where all nodes are fully connected. Other more different types are the memory neuron network [7], where each neuron has associated with it a memory neuron whose single scalar output summarizes the history of past activation of that unit; the high-order neural network [8], where high-order recurrent connections between each neuron are included; and the recurrent radial basis function network [9], where the past output values of a radial basis function network are fed back to both the network input and output nodes. By inspecting the structure of the above networks, we may find that their recurrent properties are achieved by involving internal memory in the form of feedback connections to existing networks, such as feedforward multilayer perceptron networks and radial basis function network. In feedforward network structure, the performance of a neural fuzzy network has been shown to be better than a neural network, and several neural fuzzy networks have been proposed [10]–[16]. Based on this observation, design of a recurrent network from a feedforward fuzzy network structure should be a better choice.

For fuzzy networks, several types of them have been proposed depending on the types of fuzzy if-then rules and fuzzy reasoning employed. Two usually types are the Mamdani-type and TSK-type fuzzy networks. For a Mamdani-type fuzzy network, the minimum fuzzy implication is used in fuzzy reasoning and each rule is of the following form:

Rule i: IF $x_1(t)$ is A_{i1} And $\cdots$ And $x_n(t)$ is A_{in}
Then $y(t+1)$ is B_i

Manuscript received March 20, 2001; revised June 22, 2001 and August 14, 2001. This work was supported by the National Science Council, Republic of China, under Grant number NSC 89-2218-E-235-001.

The author is with the Department of Electrical Engineering, National Chung Hsing University, Taichung, 402 Taiwan, R.O.C.

Publisher Item Identifier S 1063-6706(02)02970-3.

where x is the input variable, y is the output variable, A and B are fuzzy sets. For TSK-type fuzzy network, the consequence of each rule is a function input linguistic variable. The general adopted function is a linear combination of input variables plus a constant term, and each rule is of the following form:

$$\text{Rule i: IF } x_1(t) \text{ is } A_{i1} \text{ And } \cdots \text{ And } x_n(t) \text{ is } A_{\mathbf{in}} \quad \text{Then } y(t+1) = c_i + \sum_j a_{ij} x_j(t).$$

The final network output is a weighted average of each rule's output. Some results on the research direction of designing a recurrent network from a fuzzy network have been proposed. One category focusses on the combination of the fuzzy finite-state machine with recurrent neural networks [17]–[19]. For example, in [18], the fuzzy finite machine is encoded into a recurrent network, and in [19], a neural fuzzy network is implemented as a fuzzy finite machine. Another category focusses on embedding the recurrent structure into a feedforward fuzzy network [20]–[22]. In [20], the concept of recurrent fuzzy network is proposed. In [21], a recurrent neuron-fuzzy network is proposed. The structure of the network is similar to the recurrent radial basis function network mentioned above. In [22], the authors provide for a recurrent self-organizing neural fuzzy inference network (RSONFIN) with online supervised learning ability. The rules in RSONFIN are of ordinary Mamdani-type fuzzy rule. In [14], [15], where several static mapping problems are performed, it has been shown that if a feedforward TSK-type fuzzy network is used, the performance in network size and learning accuracy is superior to those of Mamdani-type fuzzy network. It seems to be more efficient, based on these results, to include the TSK-type fuzzy rules into the design of recurrent fuzzy network. With this motivation, a TSK-type recurrent fuzzy network, the TRFN-S, is proposed for a supervised learning environment with available gradient information. To design TRFN-S under this learning environment, since the gradient information is available, the neural network learning approach is adopted.

For the aforementioned network, all design work is based upon supervised learning. In dynamic system identification, where the precise input–output pattern is available, these network design algorithms may handle the situation. However, for other problems, such as dynamic system control, where precise control input–output training patterns are unavailable or expensive to collect, a new learning algorithm or design configuration is required. As to time-delayed plant control, one generally adopted controller design approach is the generalized predictive control (GPC) [24]. GPC is presented based as originally upon a linear model, so it is not suitable for nonlinear plant control. To cope with this problem, some nonlinear controller model designs based on GPC are proposed [25]–[27]. Most of these belong to fuzzy model based predictive control. In this model, a fuzzy controller with the consequence of linear GPC form is designed. Parameter design algorithm in linear GPC is applied to this model. The drawback of this model is that we should know in advance the order of input and output terms of the linear GPC model in the fuzzy consequence. Other controller design approaches for dynamic systems based upon supervised learning are the direct inverse, direct and indirect adaptive control [1]. For direct inverse control, the control configuration fails when the inverse of the controlled plant is nonexistent. This is true for most dynamic plants. For direct adaptive control, we should know the form of the controlled plant. For an unknown plant, this approach cannot be applied. For indirect adaptive control, the controlled plant should first be identified and then a controller is designed based on this identification network. Controller design based upon this configuration is complex, and a good control performance is achieved only if a high precision identification model is obtained. Although some fuzzy neural networks have been proposed and applied to dynamic system control, there are still disadvantages in these network structures and the controller design configurations are mainly based on the above mentioned methods. In [21], a recurrent neuro-fuzzy model is put forward as a way to built prediction model for nonlinear process, and based on this model a predictive controller is designed by GPC. For this recurrent neural fuzzy model, the recurrent property is achieved by modifying the consequence of each fuzzy rule to be a linear model in AutoRegressive with eXogenous (ARX) inputs form. The disadvantage of this model is that we need to know the order of both control input and network output to participate in the ARX model. For the proposed TRFN in this paper, we solve this problem by feeding back the firing strength of each rule. This way, only the current control input and system state are fed to network input, and the past values can be memorized by feedback structure. In [23], a recurrent fuzzy neural network (RFNN) is proposed. In RFNN, the recurrent property is achieved by feeding the output of each membership function back to itself, so each membership value is only influenced by its previous value. In contrast to this local feedback structure, in TRFN, a global feedback structure is adopted. The outputs of all rule nodes, the firing strengths, are fed back and summed, so each rule's firing strength depends not only on its previous value but also on others. We will show by simulation that with the global feedback structure, TRFN can achieve better performance than the local feedback structure in RFNN. In [23], RFNN is applied to dynamic plant control, and the controller is designed by direct and indirect adaptive control methods mentioned above. In [22], RSONFIN constructed by Mamdani type fuzzy if–then rules is also applied to plant control based on direct inverse control which works only when the inverse of the plant exists.

In contrast to the above supervised learning-based controller design, several controller design configurations have been put forth [28]–[34]. Among them, one efficient way is design by genetic algorithms (GAs). GAs don't require or use derivative information, the most appropriate applications are problems where gradient information is unavailable or costly to obtain. Reinforcement learning is one example of such a domain. In reinforcement learning, agents learn from signals that provide some measure of performance and which may be delivered after a sequence of decisions have been made. In GAs, the only feedback used by the algorithm is information about the relative performance of different individuals and may be applied to reinforcement problems where the evaluative signals contain relative performance information [31], [32]. Several results

from designing recurrent neural network with GAs have been proposed. In [35], GA is used for training a fully connected recurrent neural network. In [36], there is an evolutionary algorithm that acquires both the structure and weights for recurrent neural networks. The scheme for [37] has a 2-D GA, and in [38] there is a cellular GA with learning ability for training recurrent neural network. Basically, these papers focus on the development of new GAs for the design of existing recurrent network structures. Besides the GA itself, another factor that may influence a GA-based recurrent network performance is the structure of the designed recurrent network. Although both the aforementioned newly proposed recurrent neural and neural fuzzy network structure do achieve a better performance than old ones under supervised learning, it is not necessarily true with GA. In this paper, in contrast to TRFN-S, the TRFN design with GA (TRFN-G) is proposed and applied to dynamic system control. For TRFN-G, the spatial and temporal fuzzy rules that constitute the TRFN are designed concurrently. In contrast to the simple GA with roulette wheel selection [54] and traditional tournament selection [45], a different approach, the tournament selection combined with elitist reproduction and crossover strategy is adopted for TRFN-G design. Besides, the spatial input is partitioned according to flexible methods, as compared to the grid-type partition methods encountered in earlier GA-based fuzzy rules design approaches [44]–[47]. This way, TRFN-G can achieve a good performance with only a few rule numbers, and we only need to assign the number of fuzzy rules in TRFN-G before proceeding to GA. In contrast to the fixed network structure in TRFG-G during design, some works on structure optimization using GA are proposed. In [48], GA is applied to determine the rule number in a fuzzy network. In [34], GA is applied to select the significant input variables to participate in the consequence of TSK-type fuzzy rules. These works are applied to feedforward fuzzy network design. For recurrent fuzzy networks, we may adopt these algorithms. However, to perform structure optimization, different GAs should be used owing to different network structures. With fixed structure in TRFN-G and compared recurrent networks, we can design each by the same GA and demonstrate the structure superiority of TRFN-G. We will show by simulations that the TRFN structure not only achieves a good performance for TRFN-S, but also does for TRFN-G. Comparisons of TRFN-G to its recurrent neural network counterpart, designed by the same GA, and to other dynamic system control configurations, will verify this.

The proposed TRFN is constructed from a series of fuzzy if-then rules, with the consequence of each rule being of TSK-type fuzzy reasoning. Inputs to the network precondition part include external variables and internal variables derived from the fuzzy firing strengths, and the consequence is a linear combination of them. A design of TRFN by neural networks under supervised learning, identified as "TRFN-S," and a design by genetic algorithm, identified as "TRFN-G," are proposed in this paper. TRFN-S is applied to dynamic system identification and TRFN-G to dynamic system control. Advantages of TRFN for these two kinds of problems will verify the advantages of TRFN for dealing with dynamic systems processing.

This paper is organized as follows. Section II describes the structure of the TSK-type recurrent fuzzy network. Section III sets forth the supervised learning algorithm including structure and parameter learning for online generation of the recurrent fuzzy rules based on neural network learning algorithms. In Section IV, design of TRFN by GA for learning environments where gradient information is unavailable or costly to obtain is proposed. In Section V, TRFN is used for solving dynamic system processing problems, where TRFN-S and TRFN-G are applied to dynamic system identification and control, respectively. Finally, conclusions are drawn in Section VI.

II. Structure of the TSK-Type Recurrent Fuzzy Network (TRFN)

In this section, structure of the TRFN (as shown in Fig. 1) is introduced. A network with two external inputs and a single output is considered here for convenience. This six-layered network realizes a recurrent fuzzy network of the following form:

Rule 1: IF $x_1(t)$ is A_{11} and $x_2(t)$ is A_{12} and $h_1(t)$ is G
THEN $y(t+1)$ is $a_{10}+a_{11}x_1(t)+a_{12}x_2(t)$ $+a_{13}h_1(t)$ and $h_1(t+1)$ is w_{11} and $h_2(t+1)$ is w_{21}.

Rule 2: IF $x_1(t)$ is A_{21} and $x_2(t)$ is A_{22} and $h_2(t)$ is G
THEN $y(t+1)$ is $a_{20}+a_{21}x_1(t)+a_{22}x_2(t)$ $+a_{23}h_2(t)$ and $h_1(t+1)$ is w_{12} and $h_2(t+1)$ is w_{22}

where A and G are fuzzy sets, w and a are the consequent parameters for inference output h and y, respectively. The consequent part for the external output y is of TSK-type and is a linear combination of the external input variables x and internal variables h, plus a constant.

In Fig. 1, a network constructed by the above two rules is shown. There are two external input variables x and single output y. Accordingly, TRFN has two nodes in layer 1 and one node in layer 5, respectively. Nodes in layer 1 are input nodes. Nodes in layer 2 are called input term nodes and act as membership functions to express the input fuzzy linguistic variables. Two types of membership functions are used in this layer. For the external variable x, a local membership function, the Gaussian membership function is adopted. For the internal variable h, a global membership function, the sigmoid function is adopted. Each internal variable has a single corresponding fuzzy set. Each node in layer 3 is called a rule node. The number of rule nodes in this layer is equal to the number of fuzzy sets corresponding to each external linguistic input variable. Nodes in layer 4 are called consequent nodes. Each rule node has a corresponding consequent node which performs a weighted linear combination of the input variables x and h plus a constant. Nodes in layer 5 are called context nodes and perform defuzzification operation. The number of internal variables h in this layer is equal to the rule nodes. In layer 6, the node is called a defuzzification node.

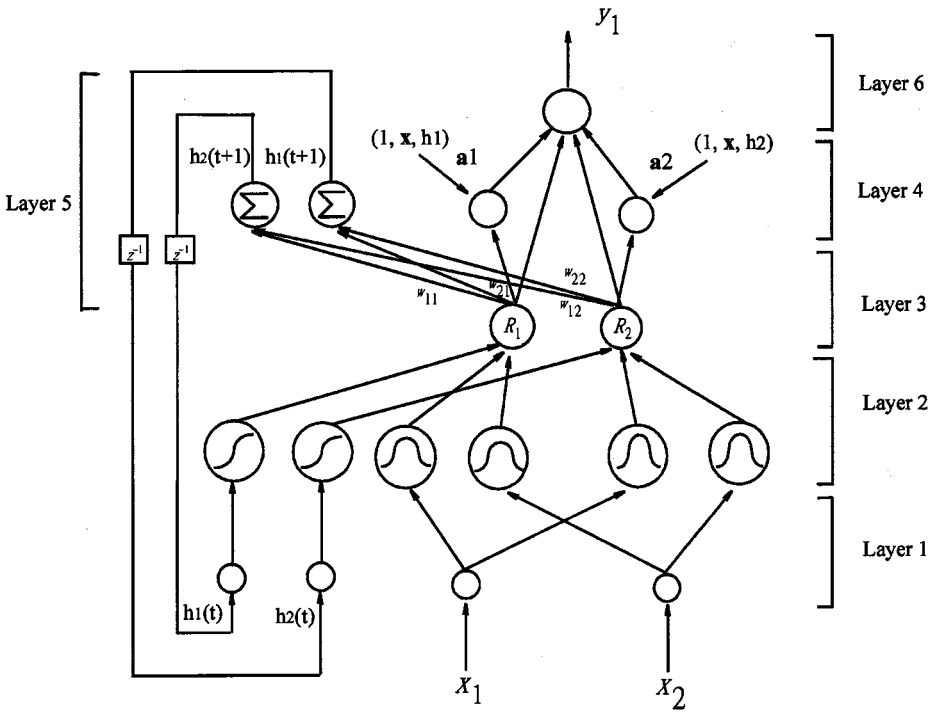

Fig. 1. Structure of the TRFN.

To give a clear understanding of the mathematical function of each node, we will describe function of TRFN layer by layer. For notation convenience, the net input to the ith node in layer k is denoted by $u_i^{(k)}$ and the output value by $O_i^{(k)}$.

Layer 1: No function is performed in this layer. The node only transmits input values to layer 2.

Layer 2: As described above, two types of membership functions are used in this layer. For external input x_j, the following Gaussian membership function is used:

$$O_i^{(2)} = \exp\left\{-\frac{\left(u_j^{(2)} - m_{ij}\right)^2}{\sigma_{ij}^2}\right\} \quad \text{and} \quad u_j^{(2)} = O_j^{(1)} \tag{1}$$

where m_{ij} and σ_{ij} are, respectively, the center and the width of the Gaussian membership function of the ith term of the jth input variable x_j. For internal variable h_i, the following sigmoid membership function is used:

$$O_i^{(2)} = \frac{1}{1+\exp\left\{-u_i^{(2)}\right\}} \quad \text{and} \quad u_i^{(2)} = O_i^{(5)}. \tag{2}$$

Links in layer 2 are all set to unity.

Layer 3: The output of each node in this layer is determined by fuzzy AND operation. Here, the product operation is utilized to determine the firing strength of each rule. The function of each rule is

$$O_i^{(3)} = \prod_{j=1}^{n+1} O_j^{(2)} = \frac{1}{1+\exp\left\{-O_i^{(5)}\right\}} \cdot \exp\left\{-\left(\sum_{j=1}^{n} \frac{\left(O_j^{(1)} - m_{ij}\right)^2}{\sigma_{ij}^2}\right)\right\} \tag{3}$$

where n is the number of external inputs. The link weights are all set to unity.

Layer 4: Nodes in this layer perform a linear summation. The mathematical function of each node i is

$$O_i^{(4)} = \sum_{j=0}^{n+1} a_{ij} u_j^{(4)} = a_{i0} + \sum_{j=1}^{n} a_{ij} x_j + a_{in+1} h_i \tag{4}$$

where n is the number of external input variables, and $a_{ij}, j = 0, \ldots, n+1$, are the parameters to be tuned. Links from this layer to layer 6 are all equal to unity.

Layer 5: The context node functions as a defuzzifier for the fuzzy rules with inference output h. The link weights represent the singleton values in the consequent part of the internal rules. The simple weighted sum [49] is calculated in each node

$$h_i = O_i^{(5)} = \sum_{j=1}^{r} O_j^{(3)} w_{ij} \tag{5}$$

As in Fig. 1, the delayed value of h_i is fed back to layer 1 and acts as an input variable to the precondition part of a rule. Each rule has a corresponding internal variable h and is used to decide the influence degree of temporal history to the current rule.

Layer 6: The node in this layer computes the output signal y of the TRFN. The output node together with links connected to it act as a defuzzifier. The mathematical function is

$$y = O^{(6)} = \frac{\sum_{j=1}^{r} O_j^{(3)} O_j^{(4)}}{\sum_{j=1}^{r} O_j^{(3)}} \tag{6}$$

III. TRFN With Supervised Learning (TRFN-S)

In this section, we present a two-phase learning scheme for TRFN. The task of constructing the TRFN is divided into two subtasks: structure learning and parameter learning. The objective of the structure learning is to decide the number of fuzzy rules, initial location of membership functions, and initial consequent parameters. On the contrary, the objective of parameter learning is to tune the free parameters of the constructed network to an optimal extent. Details of the two learning algorithms are described as follows.

A. Structure Learning

Since there are no rules initially in TRFN, the first task in structure learning is to decide when to generate a new rule. Clustering on the external input, which represents the spatial information, is used as the criterion. The idea of clustering approach in [15] is adopted. Geometrically, a rule corresponds to a cluster in the input space. The spatial firing strength can be regarded as the degree the incoming pattern belongs to the corresponding cluster. An input data x with higher firing strength means its spatial location is nearer the cluster center than those with smaller strengths. Based on this concept, the spatial firing strength

$$F^i(x) = \prod_{k=1}^{n} O_k^{(2)} = \exp\left\{-\left(\sum_{j=1}^{n}(x_j - m_{ij})^2\sigma_{ij}^2\right)\right\} \in [0,1] \quad (7)$$

is used as the criterion to decide if a new fuzzy rule should be generated. For the first incoming data $\mathbf{x}(0)$, a new fuzzy rule is generated, with the center and width of Gaussian membership function assigned as

$$m_{1i} = x_i(0) \quad \text{and} \quad \sigma_{1i} = \sigma_{\text{init}}, \quad \text{for } i = 1, \ldots, n \quad (8)$$

where n is the number of external input variables, and σ_{init} is a prespecified value that determines the initial width of the first cluster. For succeeding incoming data $\mathbf{x}(t)$, find

$$I = \arg \max_{1 \le i \le r(t)} F^i(\mathbf{x}) \quad (9)$$

where $r(t)$ is the number of existing rules at time t. If $F_J \le F_{\text{in}}(t)$, then a new rule is generated, where $F_{\text{in}}(t) \in (0,1)$ is a pre-specified threshold that decays during the learning process. In this paper, $F_{\text{in}}(t) = F_{\text{in}}(t-1)(1 - t/C)$ is used, where C is a constant value that controls the decay speed. For a more complex learning problem, a larger rule number is required, and so a higher initial threshold $F_{\text{in}}(0)$ should be set in advance. 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 makes little sense to spend much time on the assignment of the centers and widths for finding a perfect cluster. Hence, we can simply set

$$m_{(r(t)+1)i} = x_i(t), \quad \text{and} \quad \sigma_{(r(t)+1)i} = \beta \cdot \sum_{j=1}^{n} \frac{(x_j - m_{Ij})^2}{\sigma_{Ij}^2} \quad (10)$$

for $i = 1, \ldots, n$, according to the first-nearest-neighbor heuristic [11], where $\beta \ge 0$ decides the overlap degree between two clusters. Both of the parameters F_{in} and β decide the number of rules to be generated. In applying TRFN-S to different problems, we may specify one parameter with the same value, and alter the other one to find a best network structure. For example, in this paper, the parameter $F_{\text{in}}(0)$ is set to 0.01 for different examples, and the best structure of TRFN-S is decided by β. With the same value of F_{in}, a higher value of β means a higher overlapping degree, and so fewer rules are generated. The number of fuzzy sets in each external input dimension is equal to the number of fuzzy rules. To further reduce the number of fuzzy sets in each dimension, we may perform the similarity measure checking between two neighboring fuzzy sets as in [15] and eliminate the redundant ones. Since a recurrent structure is used, the external input dimension contains current system state only, so the input dimension is usually small. Furthermore, the inclusion of the TSK-type consequence can significantly reduce the rule number. Owing to these two reasons, chances that two fuzzy sets are highly overlapped are small, and the fuzzy similarity measure process is omitted for design simplification.

Once a new rule is newly generated during the presentation of $(\mathbf{x}(t), y(t)))$ data, generation of the corresponding consequent node in layer 4 and context node in layer 5 follows. The initial constant value a_{i0} connected to layer 4 is set to $y(t)$, and the other a_{ij} parameters are assigned as small random signals in $[-0.05, 0.05]$ initially. For the newly generated context node, its fan-in comes from all the existing rule nodes in layer 3. The initial link weights w are set as random values in $[-1, 1]$ to make the initial values of internal variables h locate in the sensitive region of membership function G. This way, a quick parameter learning can be reached at the beginning. The output, h, of the new context node is fed back as input in the precondition part of the newly generated rule. With this setting, each rule has its own memory elements for memorizing the temporal firing strength history. Via repeating the above process for every incoming training data, a new recurrent rule is generated, one after another, and a whole TRFN is constructed finally.

B. Parameter Learning

The objective of parameter learning is to optimally adjust the free parameters of the network structure for each incoming data, whether the rules are newly generated or are existent originally. The learning process is performed concurrently with the structure learning phase. With the concurrent learning approach, learning may be performed online. If the parameter learning is performed after the structure learning phase, then we have to collect the whole training data in advance for

network structure design. And only after the network structure is completely designed can we apply the parameter learning phase to the same training data. Owning to this reason, this approach can only be performed offline. For TRFN-S, owing to this online learning property, it may be used for normal operation any time as learning proceeds without any assignment of fuzzy rules in advance.

Considering the single output case for clarity, our goal is to minimize the error function

$$E(t+1)=\frac{1}{2}(y(t+1)-y^d(t+1))^2 \tag{11}$$

where $y^d(t+1)$ is the desired output and $y(t+1)$ is the actual output. Suppose there are n external values x_j. The parameter a_{ij} in layer 4 is updated by

$$a_{ij}(t+1)=a_{ij}(t)-\eta\frac{\partial E(t+1)}{\partial a_{ij}(t)},$$
$$\text{for } i=1,\ldots,r(t), \quad \text{and} \quad j=0,\ldots,n+1 \tag{12}$$

where

$$\frac{\partial E(t+1)}{\partial a_{ij}(t)}=(y(t+1)-y^d(t+1))\frac{O_i^{(3)}u_j^{(4)}}{\sum_{k=1}^{r(t)}O_k^{(3)}}. \tag{13}$$

For the other free parameters, including w in layer 5, m and σ in layer 2, owing to the recurrent property, the real time recurrent learning algorithm [50] is used. We will show the update rule of m only. Updated rules of w and σ can be derived the same way and are omitted. For each m_{pq}, it is updated by

$$m_{pq}(t+1)=m_{pq}(t)-\eta\frac{\partial E(t+1)}{\partial m_{pq}(t)} \tag{14}$$

and

$$\frac{\partial E(t+1)}{\partial m_{pq}(t)}=(y(t+1)-y^d(t+1))\left(\frac{O_p^{(4)}-y(t+1)}{\sum_{k=1}^{r}O_k^{(3)}(t)}\cdot\frac{\partial O_p^{(3)}}{\partial m_{pq}}(t)+\sum_{i=1}^{r}O_i^{(3)}a_{in+1}\frac{\partial O_i^{(5)}}{\partial m_{pq}}(t)\sum_{i=1}^{r}O_i^{(3)}\right) \tag{15}$$

where

$$\frac{\partial O_p^{(3)}}{\partial m_{pq}}(t)=\frac{\partial O_p^{(5)}}{\partial m_{pq}}\cdot F^p+O_p^{(5)}\cdot F^p\cdot 2\frac{x_q(t)-m_{pq}}{\sigma_{pq}^2}, \tag{16}$$

and

$$\frac{\partial O_p^{(5)}}{\partial m_{pq}}(t)=O_p^{(5)}(t)\left(1-O_p^{(5)}(t)\right)\left\{\sum_{\ell=1}^{r}w_{p\ell}\times\left[\frac{\partial O_\ell^{(5)}}{\partial m_{pq}}(t-1)F^\ell(t-1)+F^q(t-1)\times O_q^{(5)}(t-1)\cdot 2\frac{x_p(t-1)-m_{pq}}{\sigma_{pq}^2}\right]\right\}. \tag{17}$$

There are two learning constants, η and η_w, during learning. Learning constant η is used for tuning parameters a, m and σ, while η_w for parameter w. Except for parameter w, all parameters either have good initial values during structure learning [m and σ in (10)] or are tuned directly from the error function [a in (10)]. To increase the learning speed of temporal memory w, we may set $\eta_w > \eta$.

IV. TRFN Design by Genetic Algorithm (TRFN-G)

For problems where supervised training data is unavailable or expensive to obtain, such as dynamic system control, TRFN-G instead of TRFN-S is applied. The TRFN-G is proposed for solving problems where gradient information is costly to obtain when explicit credit assignment is available or to reinforcement type learning problems, where only sparse training information is required. The sparse training information in reinforcement learning is obtained from evaluation of the system performance. In GA, the only feedback that is required is a relative performance measure for each individual. Credit assignment for each individual's action may made implicitly. Since GAs can work without explicit credit assignment to individual actions, they may be regarded as a kind of reinforcement learning when only implicit information is available. In [31], approaches that apply GAs for solving reinforcement type problems are called genetic reinforcement learning.

Details of TRFN-G design are introduced in this section. The adopted GA consists of three main operators: reproduction, crossover, and mutation. Before going into the three operations, we should assign the number of rules in TRFN-G in advance. When the number of rules is assigned then the whole network structure is known. Suppose there are r rules in TRFN-G, then the numbers of local membership functions on each external input variable x, the internal variable h, and the global membership functions are all equal to r. As to the choice of the rule number, it's a little heuristic and depends on the complexity of the problems to be solved. For TRFN-G, owing to its network superiority, it can achieve a good performance with only a small rule number. After the rule number is determined, coding of the TRFN into a chromosome is next performed. A floating point coding scheme is adopted, meaning that each gene in a chromosome is represented as a floating point number. All free parameters of TRFN, including m and σ in layer 2, a in layer 4, and w in layer 5, are coded into a chromosome. To see the coding order of these parameters into the chromosome, we will take a TRFN with n external input variables and one output as an example. Suppose there are r rules in the TRFN, then the following coded order of each dynamic fuzzy rule is obtained:

Rule i: $|m_{i1}\,|\,\sigma_{i1}|\cdots|m_{\mathrm{in}}\,|\,\sigma_{\mathrm{in}}\,|\,a_{i0}|\cdots$
$|a_{in+1}\,|\,w_{1i}\,|\,w_{2i}|\cdots|w_{ri}|.$

The chromosome representing the whole network is achieved by concatenating the above coding form of rule 1 to rule r one after another. The space partition of the external input has a character of flexibility in contrast to the grid-type partition. With the flexible partition, no prepartition of each input dimension is required, and the total number of rules can be reduced. With the structure of TRFN introduced in Section II, we only need to assign the total number of fuzzy rules to TRFN-G design, and then a whole network can be constructed.

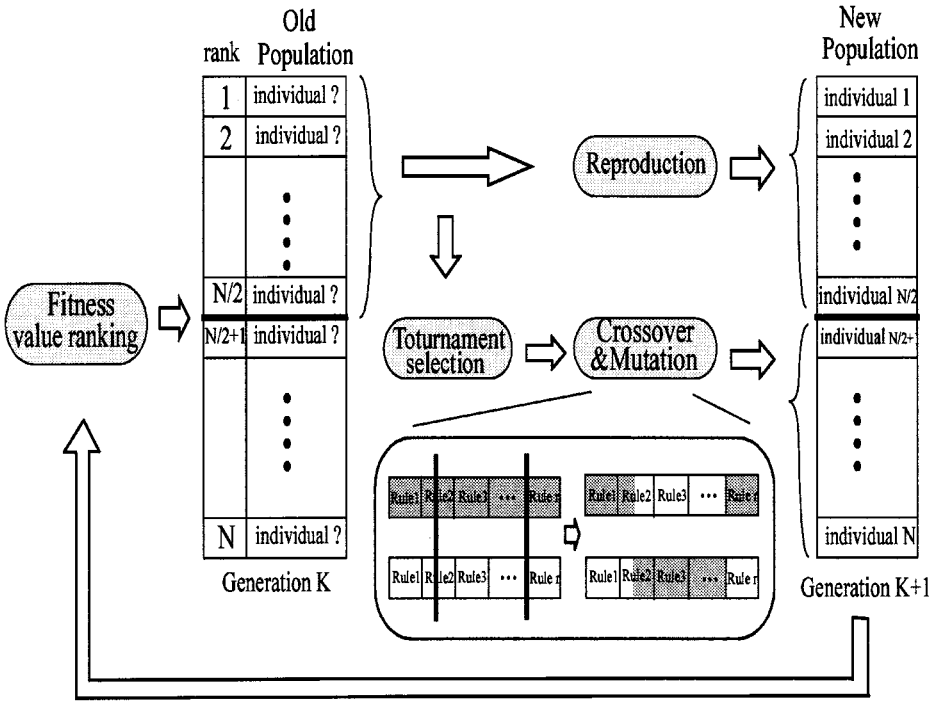

Fig. 2. Flow of the adopted GA.

Let's now see how the GA operates. With the aforementioned genotype, a population with Ps individuals is formed by random generation. During each generation, every individual in the population is applied to problem solving, and a fitness value is obtained according to its performance. For the reproduction process, the population is first sorted according to the fitness value of each individual. Based on the elitist strategy, the top-half of the best-performing individuals in the population, the elites, will advance to the next generation directly. The remaining half will be generated by performing crossover operations on individuals on the top half of the parent generation. Elitist strategy may increase the speed of domination of a population by a super individual, and thus improves the local search at the expense of a global perspective, but on balance it appears to improve GA performance [51], [52]. To overcome the disadvantage that population diversity might be lost fast by elitist strategy, in [53], a relative-based mutated reproduction method is proposed and may be incorporated into TRFN-G to further improve the performance.

After the reproduction process, TRFN-G enters the crossover process. In order to select the individuals for crossover, tournament selection instead of a simple GA roulette wheel selection [54] is performed. Also, to speed up the learning speed, tournament selection is performed only on the top-half of the best-performing individuals instead of the whole population. In our tournament selection, two individuals in the top-half of the population are selected at random, and their fitness values are compared. The individual with the highest fitness value is selected as one parent. The other parent is selected in the same way. Performing crossover on the selected parents creates the offspring. Here, two-point crossover is performed. After the operation, the top-half worst performing individuals in the population will be replaced by the newly produced offspring. For the mutation operation, it is an operator whereby the allele of a gene is altered randomly. With mutation, new genetic materials can be introduced into the population. Mutation should be used sparingly because it is a random search operator; otherwise, with high mutation rates, the algorithm will become little more than a random search. To clarify the adopted GA, flow of the algorithm is shown in Fig. 2.

In the following section, TRFN-G will be applied to dynamic control. Comparisons with recurrent neural networks designed by the same GA are made. Besides, comparisons with other control configurations, like direct inverse, indirect adaptive, and fuzzy models based on predictive control, are also performed.

V. Simulations

In this section, the TRFN is applied to two kinds of dynamic system problems, the dynamic system identification and the dynamic system control. For dynamic system identification where we can easily collect the precise input–output training data, TRFN-S is applied, while for the dynamic control problem, where the supervised input–output training data is either costly to obtain or only the reinforcement signal is available, the TRFN-G is applied.

A. Dynamic System Identification

The systems to be identified are dynamic systems whose outputs are functions of past inputs and past outputs as well. For this dynamic system identification, since a recurrent network, the TRFN-S, is used, only the current state of the system and control signal are fed as input to the network. The adopted identification configuration is a serial-parallel model shown in Fig. 3. The model is used in the following two identification problems. If a feedforward network is used, then we should know the order

of the system, and feed all the related states to the input as in [1].

Example 1: The plant to be identified in this example is guided by the following difference equation:

$$y_p(k+1) = f(y_p(k), y_p(k-1), y_p(k-2), u(k), u(k-1)) \tag{18}$$

where

$$f(x_1, x_2, x_3, x_4, x_5) = \frac{x_1 x_2 x_3 x_5 (x_3 - 1) + x_4}{1 + x_3^2 + x_2^2}. \tag{19}$$

Here, the current output of the plant depends on three previous outputs and two previous inputs. In [1], a feedforward neural network with five input nodes for feeding the appropriate past values of y_p and u is used. In [7], identification of the same plant using a recurrent neural network, the memory neural network, is put forward. Also in [20], a recurrent fuzzy system, and in [22], a recurrent neural fuzzy network, the RSONFIN, are applied to the same task. Here the TRFN-S is used. As in Fig. 3, owning to the recurrent property of TRFN-S, only the current state $y_p(k)$ and control input $u(k)$ are fed as the input. The other past values of $y_p(k)$ and $u(k)$ are not used since their influence on the output are memorized by the feedback structure. If a feedforward network structure is used, then we should know the order of system in advance and feed the appropriate past values of $y_p(k)$ and $u(k)$ to the network. In training the TRFN-S, we use only ten epoches and there are 900 time steps in each epoch. Similar to the inputs used in [7], the input is an iid uniform sequence over $[-2, 2]$ for about half of the 900 time steps and a sinusoid given by $1.05\sin(\pi k/45)$ for the remaining time. There is no repetition on these 900 training data, i.e., we have different training sets for each epoch. The learning rates $\eta = 0.4, \eta_w = 2\eta, \beta = 0.6, F_{\text{in}}(0) = 0.01$, and $F_{\text{in}}(m) = F_{\text{in}}(m-1)(1 - m/15)$, where m denotes the mth epoch, are chosen. After training, three recurrent fuzzy rules, in contrast to the thirty fuzzy rules in [20], are generated, and a root-mean-square error (RMSE) of 0.0265 is achieved. To see the identified result, the following input as used in [7] is adopted for test

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

Fig. 4 shows the outputs of the plant (denoted as a solid curve) and the TRFN-S (denoted as a dotted curve) for the test input. In Fig. 4, owing to the dynamic property of the identified plant, there is an oscillation after transient time 250. To show the effectiveness and efficiency of the recurrent part in TRFN-S, a feedforward TSK type fuzzy network is applied to the same problem. In the feedforward network, the five plant input variables $y_p(k), y_p(k-1), y_p(k-2), u(k)$, and $u(k-1)$ are fed as input. For fair comparison, the precondition part of the feedforward network is identified by the same way as TRFN-S. Form the results in Table I, we can see that performances of TRFN-S surpass the feedforward network. In this paper, the computation

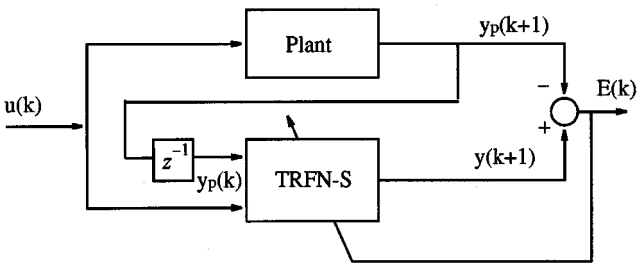

Fig. 3. Series-parallel identification model with the TRFN-S. where $E(k)$ is used for training the network parameter.

time is measured on a personal computer with Intel pentium III-600 CPU inside. Besides the comparison with feedforward fuzzy network, other types of recurrent networks are also compared. Performance of the TRFN-S is compared with memory neural network and recurrent neural fuzzy networks including RSONIN and RFNN. For memory neural network, the result proposed in [7] is adopted. For RSONFIN, there are four rules with four output clusters after training, and the learning rates are set as the same as TRFN-S. In [23], RFNN is applied to the same problem, and the same network size and training approach are simulated in the paper. Detailed comparisons of the modeling accuracy and computation time of these networks are listed in Table I for both training and test data. From the comparisons, we see that TRFN-S can achieve the highest modeling accuracy with fewer network parameters and CPU time than the compared networks.

Example 2: Consider next the following dynamic plant with longer input delays:

$$\begin{aligned} y_p(k+1) = 0.72 y_p(k) + 0.025 y_p(k-1) u(k-1) \\ + 0.01 u^2(k-2) + 0.2 u(k-3). \end{aligned} \tag{20}$$

This plant is the same as that used in [27]. The current output of the plant depends on two previous outputs and four previous inputs. As in example 1, the identification model shown in Fig. 3 is used, where only two external input values are fed to the input of TRFN-S. The training data and time steps are the same as those used in example 1. In applying the TRFN-S to this plant, the learning rates $\eta = 0.4, \eta_w = 2\eta, \beta = 0.3, F_{\text{in}}(0) = 0.01$, and $F_{\text{in}}(m) = F_{\text{in}}(m-1)(1-m/15)$, where m denotes the mth epoch, are chosen. After training, three recurrent fuzzy rules are generated. These designed three rules are

Rule 1: IF $u(k)$ is $\mu(0.0214, 0.2942)$ and $y_p(k)$ is $\mu(-0.0574, 0.0015)$ and $h_1(k)$ is G, THEN $y(k+1)$ is $-0.0234 - 0.0431u(k) + 0.0724y_p(k) - 0.0433h_1(k)$ and $h_1(k+1)$ is 0.2243 and $h_2(k+1)$ is -0.8393 and $h_3(k+1)$ is 0.6966.

Rule 2: IF $u(k)$ is $\mu(-0.9688, 0.4813)$ and $y_p(k)$ is $\mu(0.0311, 0.5031)$ and $h_2(k)$ is G, THEN $y(k+1)$ is $0.0095 + 0.0003u(k) + 1.000y_p(k) - 0.0195h_2(k)$ and $h_1(k+1)$ is -0.7092 and $h_2(k+1)$ is -0.3565 and $h_3(k+1)$ is -0.6972.

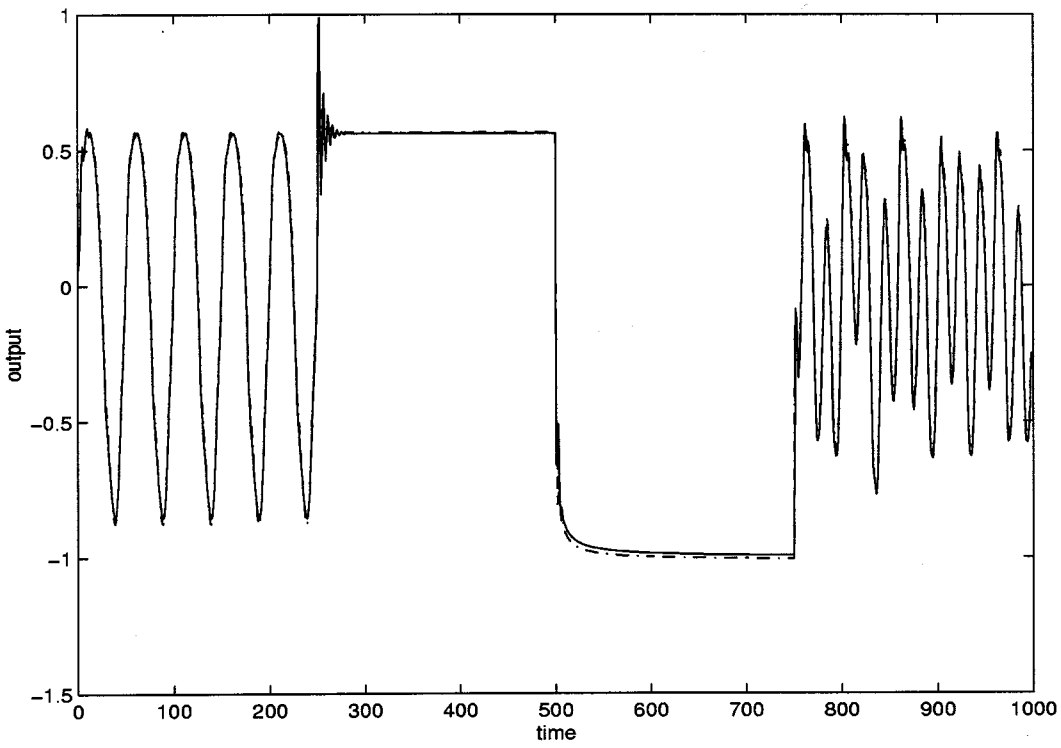

Fig. 4. Identification results of the TRFN-S in Example 1, where the dotted curve denotes the output of the TRFN-S and the solid curve denotes the actual output.

TABLE I
COMPARISONS OF THE TRFN-S WITH TSK-TYPE FEEDFORWARD NEURAL FUZZY NETWORK AND OTHER EXISTING RECURRENT NETWORKS FOR DYNAMIC SYSTEM IDENTIFICATION IN EXAMPLE 1

Example	Example 1				
Network structure	memory neural network	feedforward neural fuzzy network	RSONFIN	RFNN	TRFN-S
Network parameter	81	48	36	112	33
RMS error (train)	0.1521	0.0203	0.0248	0.0114	0.0084
RMS error (test)	0.2742	0.0521	0.0780	0.0575	0.0346
training time steps	90000	9000	9000	9000	9000
CPU time (second)	2.19	3.50	3.95	4.51	1.86

Rule 3 : IF $u(k)$ is $\mu(0.7269, 1.173)$ and $y_p(k)$ is $\mu(-0.0510, 1.0418)$ and $h_3(k)$ is G,
THEN $y(k+1)$ is $0.0087 + 0.000\,078u(k) + 1.000y_p(k) - 0.016\,611h_3(k)$ and
$h_1(k+1)$ is 0.6953 and $h_2(k+1)$ is 0.4623 and
$h_3(k+1)$ is 0.2416.

With the learned three rules, a RMSE of 0.0067 for the training data is achieved. For clarity, the network structure constructed by the above three rules is shown in Fig. 5. To test the identified result, the test signal used in Example 1 is adopted. Fig. 6 shows the outputs of the plant (denoted as a solid curve) and the TRFN-S (denoted as a dotted curve) for the test input. For performance comparison, Elman's Recurrent Neural Network (ERNN), whose feedback connections also comes from feeding the hidden states back to the input nodes as TRFN does, is simulated. The number of hidden nodes in ERNN is five, and there are 30 free parameters in total. With the same training time step and data, the resulting RMSE of ERNN for the training and test data are shown in Table II. Besides ERNN, RSONFIN is applied to the same problem by using the same training time step and data. The learning constant of RSONFIN is the same as TRFN-S. There are six rules and three output clusters in RSONFIN after training, resulting in a total number of 36 parameters. The resulting RMSE of RSONFIN for training and test data are shown in Table II. From the compared data listed in Table II, we see that the TRFN-S shows much better performance in identification accuracy than the compared two networks. For the computation time, we see that ERNN requires less computation time than TRFN-S when the same training time step is performed. However, from Table II, for ERNN to achieve the same training accuracy as TRFN-S does, a longer computation time and more network parameters are required.

B. Dynamic System Control

For the dynamic system control problem, since the precise controller input-output training data is either costly to obtain or unavailable, the GA is adopted for controller design. Although we may use some specific ways for training the controller by supervised learning, like the direct inverse and indirect adaptive control mentioned above, these methods may be applied only for some special plants. Problems with these two methods are described in Section I. Based on GA, the TRFN-G controller design is proposed. The control configuration and input–output variables of TRFN-G are shown in Fig. 7, and are applied to the following two dynamic system control problems. In the following two examples, to show the advantage of TRFN-G, its counterpart recurrent neural network, the ERNN, is compared

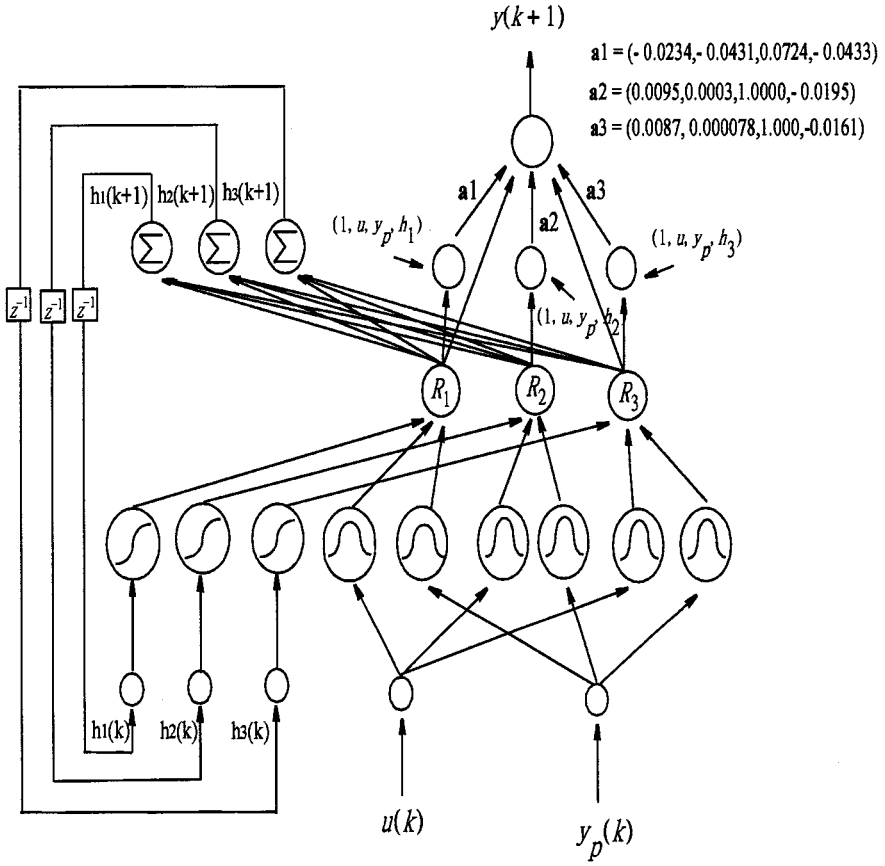

Fig. 5. Structure of the constructed TRFN-S in Example 2.

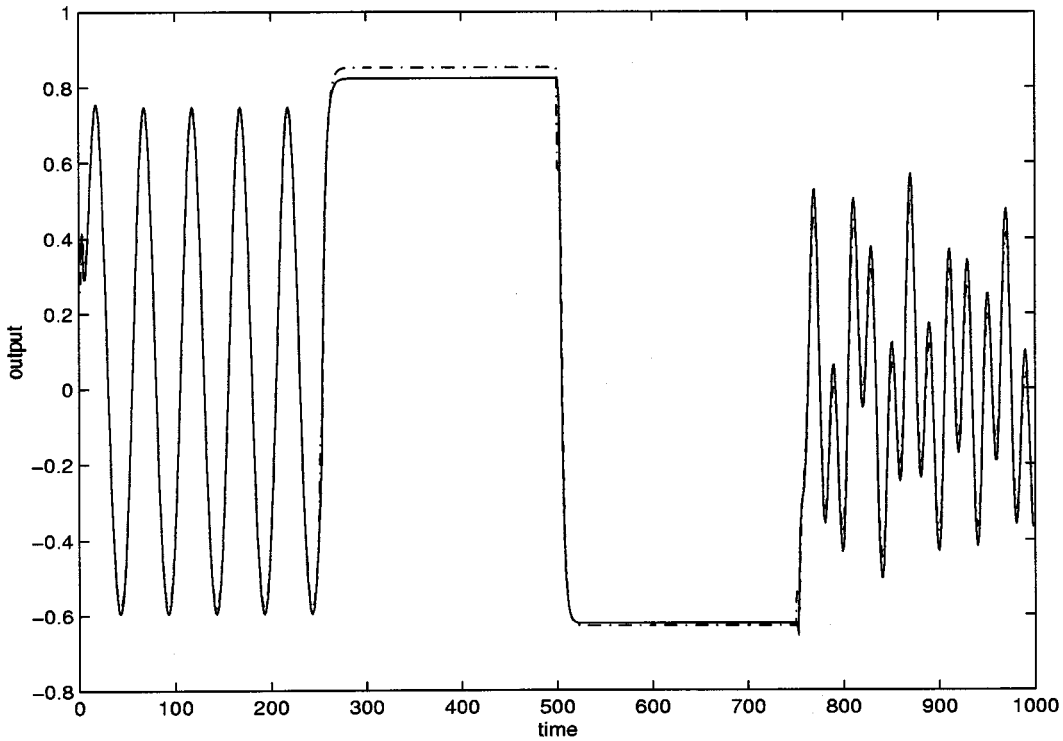

Fig. 6. Outputs of the dynamic plant (solid curve) and model TRFN-S (dotted curve) in Example 2.

with the same controller design configuration. As TRFN, the ERNN has feedback connections coming from the hidden node output. A recurrent fuzzy neural network, the RFNN, designed by the same GA is also compared. Besides the network struc-

TABLE II
COMPARISONS OF THE TRFN-S WITH OTHER EXISTING RECURRENT NETWORKS FOR DYNAMIC SYSTEM IDENTIFICATION IN EXAMPLE 2

Example	Example 2			
Network structure	ERNN	ERNN	RSONFIN	TRFN-S
Network parameter	54	54	49	33
RMS error (train)	0.036	0.0067	0.03	0.0067
RMS error (test)	0.078	0.058	0.06	0.0313
training time steps	9000	333000	9000	9000
CPU time (second)	0.83	6.26	3.2	1.98

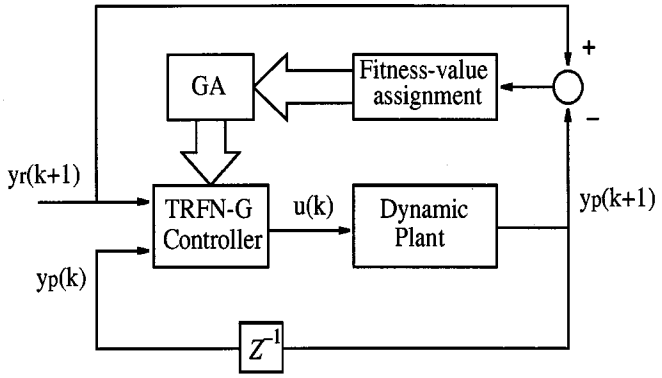

Fig. 7. Dynamic system control configuration with TRFN-G controller.

ture performance comparison, comparisons with other control configurations, including the direct inverse and indirect adaptive control, and fuzzy model-based predictive control, are also made in the following examples.

Example 3: The controlled plant is the same as that used in [1] and is given by

$$y_p(k+1)=\frac{y_p(k)y_p(k-1)(y_p(k)+2.5)}{1+y_p^2(k)+y_p^2(k-1)}+u(k). \quad (21)$$

The plant is slightly different from that used in [7] and [22] in that there is no scaling operation to the output. In designing the TRFN-G, the desired output y_r is given by the following 250 pieces of data:

$$\begin{aligned} y_r(k+1) &= 0.6y_r(k)+0.2y_r(k-1)+0.6\sin(2\pi k/45), \\ &\qquad 1\le k\le 110, \\ &= 0.6y_r(k)+0.2y_r(k-1)+0.2\sin(2\pi k/25) \\ &\quad +0.4\sin(\pi k/32), \quad 110<k\le 250. \end{aligned}$$

There are four rules in TRFN-G, resulting in 48 free parameters in total. In applying the GA, 50 chromosomes are randomly generated in a population, i.e., $P_s=50$, initially. Each chromosome contains 48 genes. The probability of mutation, P_m, is 0.01. The fitness value of each chromosome is defined as

$$\text{Fitness-value}=\frac{1}{\sum_{k=1}^{250}(y_r(k+1)-y_p(k+1))^2}. \quad (22)$$

The evolution is processed for 1500 generations and is repeated for 50 runs. The averaged best-so-far fitness value over 50 runs for each generation is shown in Fig. 8. The best and averaged RMSE error for the 50 runs after 1500 generations of training and the computation time for each run are listed in Table III. To test the performance of the designed controller, another reference input $r(k)$ is given by

$$\begin{aligned} y_r(k+1) &= 0.6y_r(k)+0.2y_r(k-1)+0.2\sin(2\pi k/25) \\ &\quad +0.4\sin(\pi k/32), \quad 250<k\le 500. \end{aligned}$$

The best and averaged control performance for the test signal over the 50 runs is also listed in Table III. To demonstrate the control result, one control performance of TRFN-G is shown in Fig. 9 for both training and test control reference output. The corresponding designed four fuzzy rules are

Rule 1: IF $y_r(k+1)$ is $\mu(0.620,0.510)$ and $y_p(k)$ is $\mu(-0.850,0.427)$ and $h_1(k)$ is G,
THEN $u(k)$ is $4.445\,54+5.327\,54y_r(k+1)+3.964\,15(k)-6.390\,46h_1(k)$ and $h_1(k+1)$ is 2.703 and $h_2(k+1)$ is 2.923 and $h_3(k+1)$ is 2.138 and $h_4(k+1)$ is -2.876.

Rule 2: IF $y_r(k+1)$ is $\mu(0.630,0.551)$ and $y_p(k)$ is $\mu(-0.090,0.354)$ and $h_2(k)$ is G,
THEN $u(k)$ is $-1.494\,50+3.075\,69y_r(k+1)-0.597\,69y_p(k)-1.749\,46h_2(k)$ and $h_1(k+1)$ is -2.899 and $h_2(k+1)$ is -1.540 and $h_3(k+1)$ is -0.523 and $h_4(k+1)$ is 2.991.

Rule 3: IF $y_r(k+1)$ is $\mu(-0.28,0.265)$ and $y_p(k)$ is $\mu(-0.890,0.521)$ and $h_3(k)$ is G,
THEN $u(k)$ is $-1.529\,50+3.857\,54y_r(k+1)-2.167\,85y_p(k)-1.428h_3(k)$ and $h_1(k+1)$ is -2.978 and $h_2(k+1)$ is 1.416 and $h_3(k+1)$ is -0.584 and $h_4(k+1)$ is -1.936.

Rule 4: IF $y_r(k+1)$ is $\mu(0.840,0.715)$ and $y_p(k)$ is $\mu(-0.570,0.435)$ and $h_4(k)$ is G,
THEN $u(k)$ is $2.1226+5.5085y_r(k+1)+0.6268y_p(k)+4.9883h_4(k)$ and $h_1(k+1)$ is 2.602 and $h_2(k+1)$ is -2.921 and $h_3(k+1)$ is 2.989 and $h_4(k+1)$ is 2.855.

The input states $y_r(k+1)$ and $y_p(k)$ are divided by 4.2 before entering the above controller. To show the effectiveness and efficiency of the recurrent part in TRFN-G, a feedforward TSK-type fuzzy network designed by the same GA is compared. For the feedforward network, all plant input variables including $u(k), y_p(k)$ and $y_p(k-1)$ are fed as network input. To have about the same network parameters as TRFN-G, the number of fuzzy rule is set to 5, resulting in a total number of 50 free parameters. The averaged best-so-far fitness values over 50 runs for each generation are shown in Fig. 8. The learned accuracy and computation time are listed in Table III for comparison. Effectiveness and efficiency of the recurrent part in TRFN-G are verified from Table III. Besides the above comparison with a feedforward fuzzy network, a recurrent neural network, the ERNN, and a recurrent fuzzy network, the RFNN, designed by the same GA are simulated. The input and output of ERNN and RFNN are the same as those of TRFN-G. For ERNN, the number of

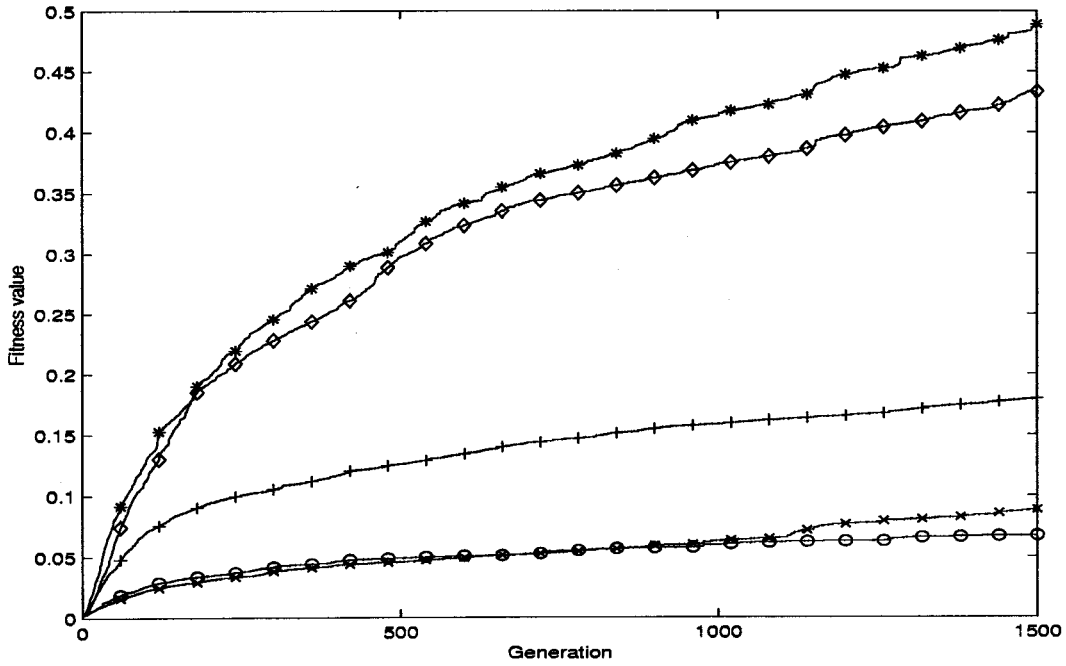

Fig. 8. The averaged best-so-far fitness values on each generation for TRFN-G (denoted as "⋆"), TRFN with traditional operation (denoted as "◇"), ERNN (denoted as "+"), feedforward TSK-type fuzzy network (denoted as "∘"), and RFNN (denoted as "×") in Example 3.

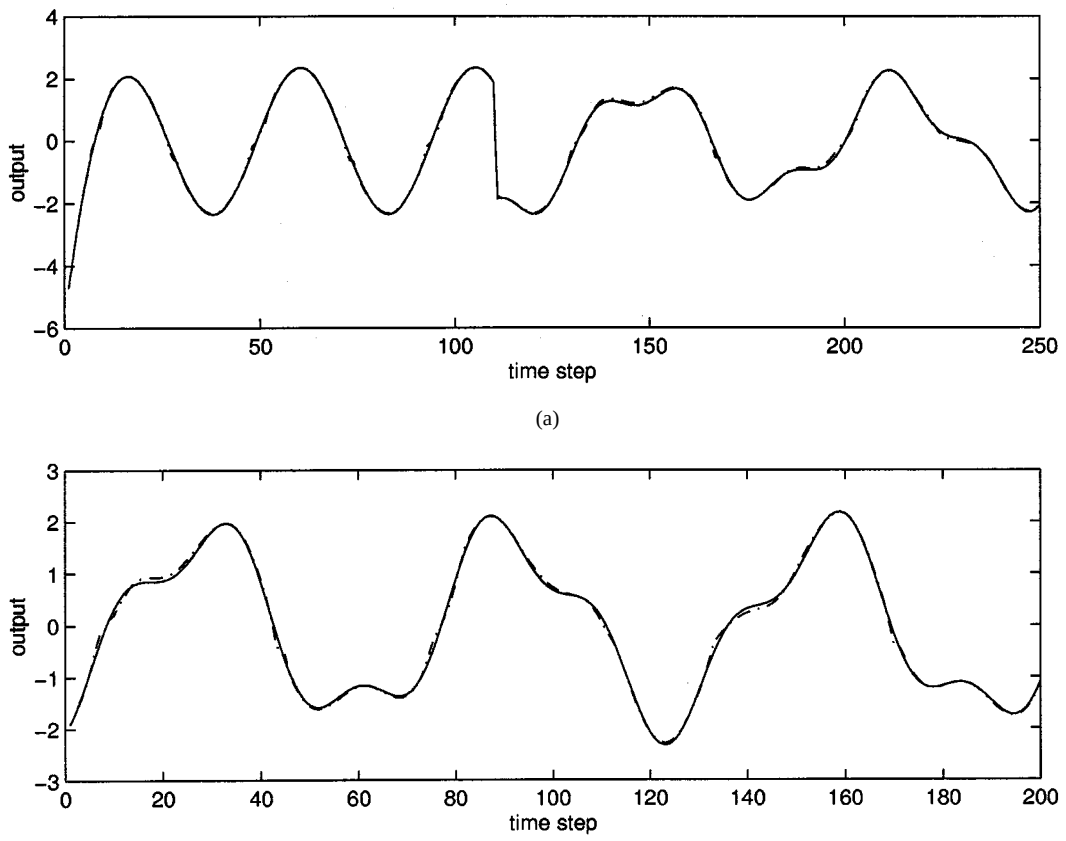

Fig. 9. The tracking performance by TRFN-G controller in Example 3 for (a) training and (b) test reference output, where the reference output is denoted as a solid curve and the actual output by a dotted curve.

hidden nodes is 6, resulting in a total number of 54 free parameters. For RFNN, the number of rules is 7, resulting in a total number of 49 free parameters. The averaged best-so-far fitness values of ERNN and RFNN over 50 runs for each generation

TABLE III
THE COMPUTATION TIME FOR EACH GA RUN AND CONTROLLED RMSE BY DIFFERENT TYPES OF NETWORKS AND DESIGN METHODS IN EXAMPLE 3

Example	Example 3								
Control configuration	RSONFIN+direct inverse control	feedforward fuzzy network+GA		ERNN+GA		RFNN+GA		TRFN-G	
Network parameter	50	50		54		49		48	
Condition	—	Mean	Best	Mean	Best	Mean	Best	Mean	Best
RMS error (train)	—	0.4172	0.1371	0.2747	0.1185	0.5540	0.0941	0.1317	0.0631
RMS error (test)	0.1661	0.4324	0.1561	0.1694	0.1069	0.3911	0.0850	0.0910	0.0536
CPU time (second)	—	830		361		908		585	

are shown in Fig. 8. The training and test results for the best and averaged rms errors over 50 runs are also listed in Table III. For RFNN, owning to its local feedback structure, a poor performance occurs near time step 110 where a sharp change at the desired control trajectory occurs. From these results, we see that an obviously better control result is achieved for TRFN-G than those achieved by ERNN and RFNN when the same GA design approach is applied. As to the computation time, from Table III, we see that ERNN requires less computation time (360 s) than TRFN-G for a run with 1200 generations. However, from Fig. 8, for TRFN-G to achieve the same accuracy as ERNN dose, only about 170 generations (96 s) are required.

In TRFN-G, the elitist crossover operation is used. In this operation, the parents for crossover are selected from the top-half best-performing individuals, the elitist. To see the performance of this operation, we compare it with the traditional operation where the parents for crossover are selected from the whole population instead of the elitist. The averaged best-so-far fitness values over 50 runs for each generation by the traditional operation are shown in Fig. 8. From the fitness value comparisons in Fig. 8, we see that the adopted elitist crossover operation achieves a better performance.

To see the performance of other control configurations for the same task, the direct inverse control by RSONFIN controller proposed in [22] is compared. There are five rules and output clusters generated after the training of RSONFIN; the total number of free parameters is 50. Since the training data is different from that of TRFN-G, only the RMSE of the above test signal is shown in Table III. In [7], an indirect adaptive control configuration using memory neural network controller is proposed. It has been shown in [22] that RSONFIN with direct inverse control can achieve a better control performance than this method, so comparison with it is omitted. From the results in Table III, we see that TRFN-G can achieve the highest control accuracy in comparison with the other two approaches.

Example 4: In this example, we will see the regulation performance of TRFN-G for a dynamic plant with longer input delays. The controlled plant used in this example is the same as in example 2. To govern a plant with long input delays like this, the usually adopted design approach is the predictive control method. Many linear controller designs for time delayed plants by model-predicted control have been proposed [55]. For the current control plant, owing to its nonlinear behavior, it has been shown in [27] that a linear predictive controller can not achieve a good performance. For this reason, in [27], a fuzzy model based predictive control approach is proposed. In this method, the controller is a RNFN as proposed in [21], and the consequence parameters in the fuzzy model is designed by GPC. However, as in linear predictive controller design, for this model we should know the order of input and output stated to participate in the consequence.

The proposed TRFN-G is applied to this time-delayed plant. The control configuration in Fig. 7 is adopted. The number of recurrent fuzzy rules in TRFN-G is set to three; thus there are 33 free parameters. For the GA, there are 50 chromosomes in the population, and each chromosome contains 33 genes. The probability of mutation, P_m, is set to 0.05. During training, the desired output, $y_r(k+1)$, is set as follows:

$$y_r(k+1)=\begin{cases}10, & \text{if } k\le 50 \text{ or } 100<k\le 150,\\ 15, & \text{if } 50<k\le 100 \text{ or } 150<k\le 200\end{cases} \tag{23}$$

and is shown in Fig. 10. in solid line. These 200 pieces of data are used for training. The fitness value is first defined by

$$\text{Fitness-value}=\frac{1}{\sum_{k=1}^{200}(y_r(k+1)-y_p(k+1))^2}.$$

The evolution is proceeded for 1200 generations and is repeated for 50 runs. The training results for the best and averaged rms errors over 50 runs are listed in Table IV, and the best control result is shown in Fig. 10. Although a small rms error can be achieved, from the result in Fig. 10, we see that there's a large steady state error on set point 10. To minimize the steady state error, a weight $W(k)$ is incorporated into the fitness value. The new fitness value is defined by

$$\text{Fitness-value}=\frac{1}{\sum_{k=1}^{200}W(k)(y_r(k+1)-y_p(k+1))^2}$$

where

$$W(k)=\begin{cases}5 & \text{for } 5+50n\le k\le 50(n+1), \quad n=0,1,2,3,\\ 1 & \text{otherwise.}\end{cases}$$

A higher weight value $W=5$ is assigned after 5 time steps when the target output changes from one set point to another. This way we can minimize the regulation error after every five time steps when the regulation point changes from one to another. The evolution is proceeded for 1200 generations and is repeated for 50 runs. The averaged best-so-far fitness value over 50 runs for each generation is shown in Fig. 11. The controlled rms error by this method is listed in Table IV. To demonstrate the control result, one control result of the TRFN-G constituted by the following three recurrent fuzzy rules:

Rule 1: IF $y_r(k+1)$ is $\mu(0.570,0.286)$ and $y_p(k)$ is $\mu(0.610,0.441)$ and $h_1(k)$ is G,
THEN $u(k)$ is $-9.9231+10.5692y_r(k+1)+7.3846y_p(k)+0.8615h_1(k)$ and $h_1(k+1)$ is -1.103 and $h_2(k+1)$ is -0.034 and $h_3(k+1)$ is -1.717.

Rule 2: IF $y_r(k+1)$ is $\mu(0.530,0.409)$ and $y_p(k)$ is $\mu(0.370,0.333)$ and $h_2(k)$ is G,
THEN $u(k)$ is $17.4333+12.1077y_r(k+1)-2.8108y_p(k)+10.8h_2(k)$ and $h_1(k+1)$ is 0.876 and $h_2(k+1)$ is -0.873 and $h_3(k+1)$ is -2.257.

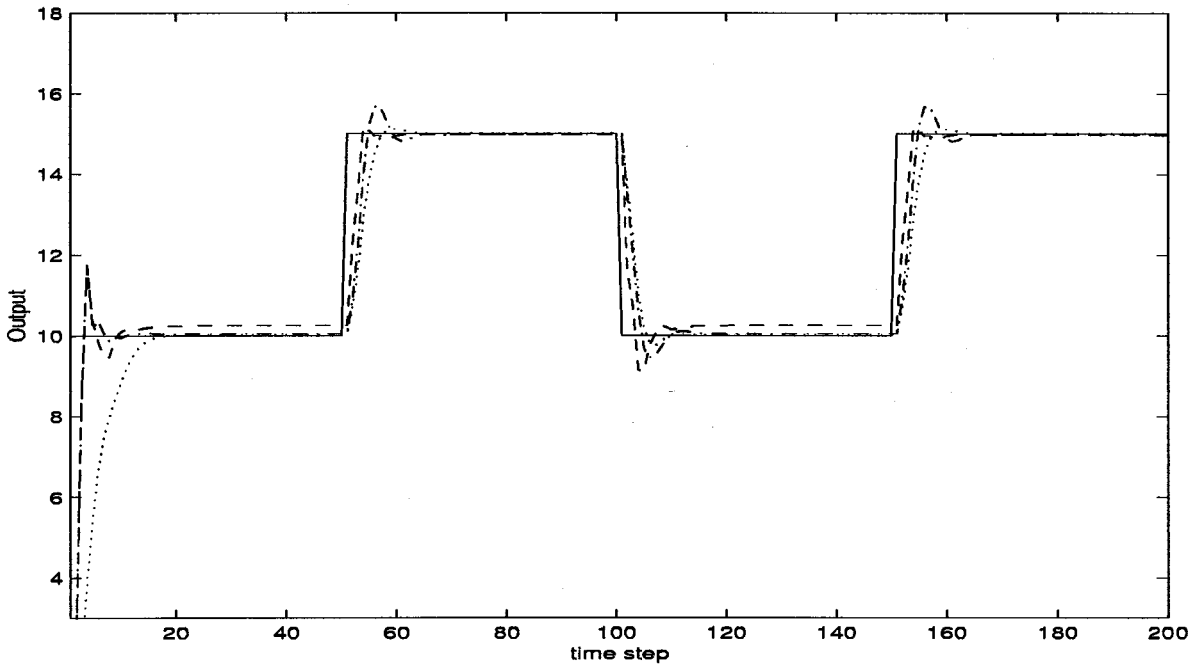

Fig. 10. The regulation performance by TRFN-G controller in Example 4 for TRFN-G designed by fitness function based on control accuracy without $W(--)$, with $W(-\cdot)$, and based on time steps until failure $(\cdots)$, where the reference output is denoted as a solid curve.

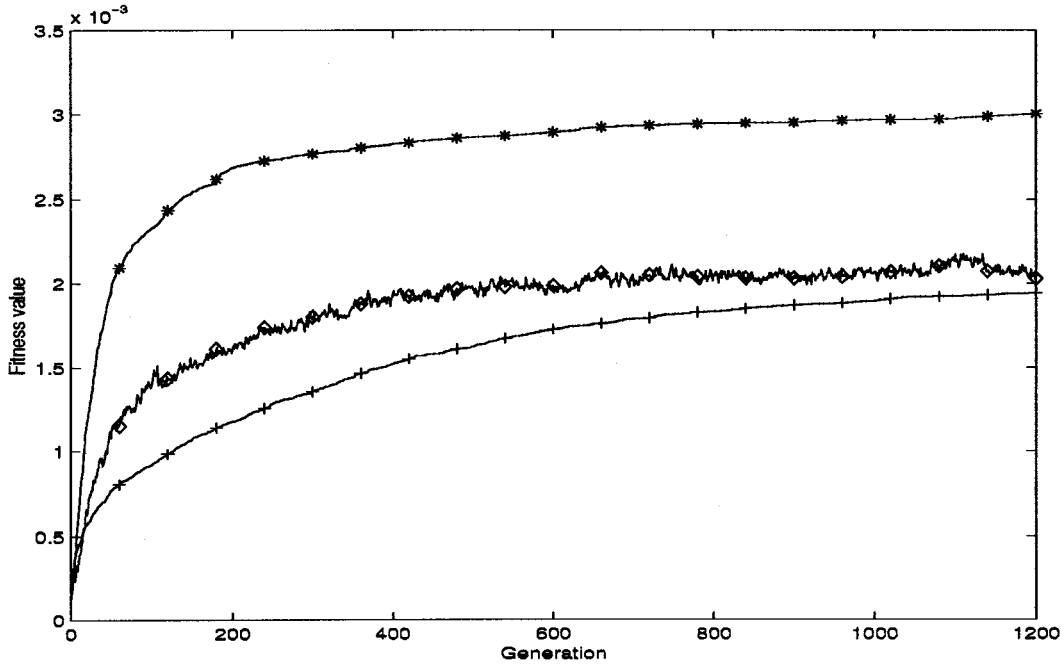

Fig. 11. The averaged best-so-far fitness values on each generation for TRFN-G (denoted as "$\star$"), TRFN designed by traditional GA with tournament selection (denoted as "$\diamond$") and ERNN (denoted as "$+$") in Example 4.

Rule 3: IF $y_r(k+1)$ is $\mu(-0.060, 0.742)$ and $y_p(k)$ is $\mu(0.100, 0.157)$ and $h_3(k)$ is G,
THEN $u(k)$ is $-11.631 + 28.4615y_r(k+1) + 19.9538y_p(k) + 28.0308h_3(k)$ and $h_1(k+1)$ is 2.381 and $h_2(k+1)$ is -0.796 and $h_3(k+1)$ is 2.02,

is shown in Fig. 10. From Fig. 10, we see that the regulation error is minimized at the cost of a higher overshoot. The states $y_r(k+1)$ and $y_p(k)$ are divided by 20 before entering the above controller. For comparison, ERNN designed by the same configuration in Fig. 7 is simulated. The number of hidden nodes in ERNN is 5, resulting in a total number of 40 free parameters. The best-so-far fitness values over 50 runs for each generation are shown in Fig. 10. The controlled rms error is shown in Table IV, too. From this result, we see that a better control result is achieved for TRFN-G than that of ERNN for the same GA design approach. In [27], a fuzzy model based predictive control approach is applied to the same regulation task. The

TABLE IV
THE COMPUTATION TIME FOR EACH GA RUN AND CONTROLLED RMSE BY DIFFERENT TYPES OF NETWORKS AND DESIGN METHODS IN EXAMPLE 4

Example	Example 4						
Control configuration	RNFN+GPC	ERNN+GA		TRFN-G without W in fitness function		TRFN-G with W in fitness function	
Network parameter		40		33		33	
Condition	—	Mean	Best	Mean	Best	Mean	Best
RMS error	1.5805	1.5046	1.4394	1.1861	1.0878	1.2353	1.1374
CPU time (second)	—	190		304		305	

network controller is an RNFN and the parameters are design by GPC method. The controlled rms error by this method is listed in Table IV. From these two comparisons, the good performance of TRFN-G is verified. In example 3, the adopted elitist crossover operation is compared to the operation where the parents for crossover are selected from the whole population. In this example, the adopted GA is compared to the traditional GA with tournament selection [45]. In the traditional method, tournament selection is performed on the whole population and the selected individual will either advance to the next generation or act as one parent for crossover depending on the crossover probability, which is set to 0.5 in this paper. The averaged fitness value for each generation by traditional GA is shown in Fig. 11. From Fig. 11, we find that the adopted elitist operation achieves a better result.

In the above simulations, we use the precise control accuracy as the fitness value during the learning process of GAs. Next, we consider TRFN-G design when only a sparse evaluation information is used. Here, a control is considered to be successful if the controlled output meets the constraint. The constraint is set as that for each reassigned set point in Fig. 11, starting from the current state and after ten times, the controlled output error should be within ± 0.2; otherwise, a failure occurs. With this performance evaluation, supervised learning cannot be applied. In reinforcement learning, the controller is designed by using the failure and success information. In TRFN-G, for each failure control, since the number of time steps until failure reveals a kind of relative importance, it is used as the fitness value of each individual. With this fitness assignment, a successful controller may be designed by TRFN-G and one successful control result is shown in Fig. 10.

VI. CONCLUSION

A TRFN is proposed in this paper. The TRFN expands the powerful ability of the TSK-type feedforward fuzzy network to deal with temporal problems. The proposal calls for design of TRFN under either neural network or genetic learning algorithms. For supervised learning, the TRFN-S designed by neural network is put forward. TRFN-S can automatically construct itself by performing online supervised structure and parameter learning concurrently. For problems where supervised training data is costly to obtain or unavailable, TRFN-G has been advanced. Owing to the supervisory network structure of TRFN, TRFN-G can achieve a high design accuracy with small network size. From simulations to dynamic system processing by neural network and genetic learning algorithms, TRFN has shown itself supervisory to its recurrent neural network counterpart. Also, in dynamic system identification, TRFN-S is compared with other types of recurrent neural or neural fuzzy networks; and in dynamic system control, TRFN-G is compared with different types of control configurations. For both problems, TRFN has shown better results than what are compared. As characterized by this supervisory property, TRFN will be applied to solve more dynamic problems in future work.

REFERENCES

[1] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Networks*, vol. 1, pp. 4–27, 1990.

[2] J. L. Elman, "Finding structure in time," *Cognit. Sci.*, vol. 14, pp. 179–211, 1990.

[3] M. I. Jordan, "Attractor dynamics and parallelism in a connectionist sequential machine," in *Proc. 8th Annual Conf. Cognitive Science Society*, Amherst, MA, 1986, pp. 531–546.

[4] B. A. Pearlmutter, "Learning state space trajectories in recurrent neural networks," in *Proc. Int. Joint Conf. Neural Networks*, vol. 2, Washingon, DC, 1989, pp. 365–372.

[5] C. L. Giles, C. B. Miller, D. Chen, H. H. Chen, G. Z. Sun, and Y. C. Lee, "Learning and extracting finite state automata with second-order recurrent neural network," *Neural Comput.*, vol. 4, pp. 395–405, May 1992.

[6] S. Santini, A. D. Bimbo, and R. Jain, "Block-structured recurrent neural networks," *Neural Networks*, vol. 8, no. 1, pp. 135–147, 1995.

[7] 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, Apr. 1994.

[8] 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, Apr. 1995.

[9] S. A. Billings and C. F. Fung, "Recurrent radial basis function networks for adaptive noise cancellation," *Neural Networks*, vol. 8, pp. 273–290, Apr. 1995.

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

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

[12] L. X. Wang, *Adaptive Fuzzy Systems and Control*. Upper Saddle River, NJ: Prentice-Hall, 1994, ch. 3.

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

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

[15] C. F. Juang and C. T. Lin, "An online self-constructing neural fuzzy inference network and its applications," *IEEE Trans. Fuzzy Syst.*, vol. 6, pp. 12–32, Feb. 1998.

[16] C. T. Lin, C. F. Juang, and J. C. Huang, "Temperature control of rapid thermal processing system using adaptive fuzzy network," *Fuzzy Sets Syst.*, vol. 103, pp. 49–65, 1999.

[17] J. Grantner and M. Patyra, "Synthesis and analysis of fuzzy logic finite state machine models," *Proc. IEEE Int. Conf. Fuzzy Systems*, vol. 1, pp. 205–210, June 1994.

[18] C. Omlin, K. Thornber, and C. Gilies, "Representation of fuzzy finite-state automata in continuous recurrent neural networks," in *Proc. IEEE Int. Conf. Neural Networks*, vol. 2, June 1996, pp. 1023–1027.

[19] F. Unal and E. Khan, "A fuzzy finite state machine implementation based on a neural fuzzy system," in *Proc. IEEE Int. Conf. Fuzzy Systems*, vol. 3, Orlando, FL, June 1994, pp. 1749–1754.

[20] V. Gorrini and H. Bersini, "Recurrent fuzzy systems," *Proc. IEEE Int. Conf. Fuzzy Systems*, vol. 1, pp. 193–198, June 1994.

[21] J. Zhang and A. J. Morris, "Recurrent neuro-fuzzy networks for nonlinear process modeling," *IEEE Trans. Neural Networks*, vol. 10, no. 2, pp. 313–326, Mar. 1999.

[22] C. F. Juang and C. T. Lin, "A recurrent self-organizing neural fuzzy inference network," *IEEE Trans. Neural Networks*, vol. 10, pp. 828–845, July 1999.

[23] C. H. Lee and C. C. Teng, "Identification and control of dynamic systems using recurrent fuzzy neural networks," *IEEE Trans. Fuzzy Syst.*, vol. 8, pp. 349–366, Aug 2000.

[24] D. W. Clarke, C. Mohtadi, and P. S. Tuffs, "Generalized predictive control part I, the basic algorithm," *Automatica*, vol. 23, no. 2, pp. 137–148, 1988.
[25] J. H. Kim, J. Y. Jeon, J. M. Yang, and H. K. Chae, "Generalized predictive control using fuzzy neural network model," in *Proc. IEEE Int. Conf. Neural Networks*, 1994, pp. 2596–2598.
[26] A. Cipriano and M. Ramos, "Fuzzy model based control for a mineral flotation plant," in *Proc. IEEE Int. Conf. Industrical Electronics, Control, Instrumentation*, 1994, pp. 1375–1380.
[27] J. H. Kim, D. T. College, A. Gun, and G. Do, "Fuzzy model based predictive control," in *Proc. IEEE Int. Conf. Fuzzy Systems*, vol. 1, Anchorage, AK, May 1998, pp. 405–409.
[28] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Trans. Syst., Man, Cyber.*, vol. SMC-13, pp. 834–846, 1983.
[29] H. R. Berenji and P. Khedkar, "Learning and tuning fuzzy logic controllers through reinforcements," *IEEE Trans. Neural Networks*, vol. 3, pp. 724–740, Sept. 1992.
[30] 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.
[31] D. Whitely, S. Dominic, R. Das, and C. W. Anderson, "Genetic reinforcement learning for neurocontrol problems," *Machine Learning*, vol. 13, pp. 259–284, 1993.
[32] D. E. Moriarty and R. Miikkulainen, "Efficient reinforcement learning through symbiotic evolution," *Machine Learning*, vol. 22, pp. 11–32, 1996.
[33] W. A. Farag, V. H. Quintana, and G. L. Torres, "A genetic-based neurofuzzy approach for modeling and control of dynamic systems," *IEEE Trans. Fuzzy Syst.*, vol. 9, pp. 756–767, Oct. 1998.
[34] C. F. Juang, J. Y. Lin, and C. T. Lin, "Genetic reinforcement learning through symbiotic evolution for fuzzy controller design," *IEEE Trans. Syst. Man, Cybern., B*, vol. 30, pp. 290–302, Apr. 2000.
[35] V. Petridis, S. Kazarlis, and A. Papaikonomou, "A genetic algorithm for training recurrent neural networks," in *Proc. Int. Joint. Conf. Neural Networks*, 1993, pp. 2706–2709.
[36] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Trans. Neural Networks*, vol. 5, pp. 54–65, Jan. 1994.
[37] Y. Sato and S. Nagaya, "Evolutionary algorithms that generate recurrent neural networks for learning chaos dynamics," in *Proc. IEEE Int. Conf. Evolutionary Computation*, 1996, pp. 144–149.
[38] K. W. C. Ku, M. W. Mark, and W. C. Siu, "Addining learning to cellular genetic algorithms for training recurrent neural networks," *IEEE Trans. Neural Networks*, vol. 10, pp. 239–252, Mar. 1999.
[39] B. A. Pearlmutter, "Gradient calculations for dynamic recurrent neural networks: A survey," *IEEE Trans. Neural Networks*, vol. 6, pp. 1212–1228, Oct.. 1995.
[40] J. Torreele, "Temporal processing with recurrent networks: An evolutionary approach," in *Proc. 4th Int. Conf. Genetic Algorithms*, 1991, pp. 555–561.
[41] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Trans. Neural Networks*, vol. 5, pp. 54–65, Feb. 1994.
[42] Y. Sato and S. Nagaya, "Evolutionary algorithms that generate recurent neural networks for learning chaos dynamics," in *Proc. IEEE Int. Conf. Evolutionary Comp.*, 1996, pp. 144–149.
[43] K. W. C. Ku, M. W. Mak, and W. C. Siu, "Adding learning to cellular genetic algorithms for training recurrent neural networks," *IEEE Trans. Neural Networks*, vol. 10, pp. 239–252, Apr.. 1999.
[44] C. L. Karr, "Design of an adaptive fuzzy logic controller using a genetic algorithm," in *Proc. 4th Int. Conf. Genetic Algorithms*, 1991, pp. 450–457.
[45] A. Homaifar and E. McCormick, "Simultaneous design of membership functions and rule sets for fuzzy controllers using genetic algorithms," *IEEE Trans. Fuzzy Syst.*, vol. 3, pp. 129–139, Apr. 1995.
[46] P. Thrift, "Fuzzy logic synthesis with genetic algorithms," in *Proc. 4th Int. Conf. Genetic Algorithms*, 1991, pp. 509–513.
[47] M. H. Lim, S. Rahardja, and B. H. Gwee, "A GA paradigm for learning fuzzy rules," *Fuzzy Sets Syst.*, vol. 82, pp. 177–186, 1996.
[48] M. A. Lee and H. Takagi, "Integrating design stages of fuzzy systems using genetic algorithms," in *Proc. IEEE Int. Conf. Fuzzy Systems*, vol. 1, San Francisco, CA, Apr. 1993, pp. 612–617.
[49] J. S. R. 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, Feb. 1993.
[50] 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.
[51] D. Thierens and D. Goldberg, "Elitist recombination: An integrated selection recombination GA," in *Proc. IEEE Int. Conf. Evolutionary Computation*, vol. 1, 1994, pp. 508–512.
[52] P. Larranaga, M. Poza, Y. Yurramenddi, R. H. Murga, and C. M. H. Kuijpers, "Structure learning of Bayesian networks by genetic algorithms: A performance analysis of control parameters," *IEEE Trans. Patt. Anal. Mach. Intell.*, vol. 18, pp. 912–926, Sept. 1996.
[53] C. F. Juang, "Construction of dynamic fuzzy if-then rules through genetic reinforcement learning for temporal problems solving," in *Proc. Joint 9th IFSA World Congress 20th NAFIPS Int. Conf.*, Vancouver, BC, Canada, July 2001, pp. 2341–2346.
[54] D. E. Goldberg, *Genetic Algorithms in Search Optimization and Machine Learning*. Reading, MA: Addison-Wesley, 1989, ch. 3.
[55] E. F. Camacho and C. Bordons, *Model Predictive Control in the Process Industry*. New York: Springer-Verlag, 1995.

Chia-Feng Juang (M'99) received the B.S. and Ph.D. degrees in control engineering from the National Chiao-Tung University, Hsinchu, Taiwan, R.O.C., in 1993 and 1997, respectively.

In 1999, he joined Chung Chou Institute of Technology, as an Assistant Professor. In 2001, he joined the National Chung Hsing University, Taichung, Taiwan, R.O.C., where he is currently an Assistant Professor of Electrical Engineering. His current research interests are neural fuzzy systems, intelligent control, evolutionary computation, and speech signal processing.

Dr. Juang is a member of the IEEE Signal Processing Society and the IEEE Robotics and Automation Society.