Available online at www.sciencedirect.com
SCIENCE DIRECT®
Fuzzy Sets and Systems 150 (2005) 307–330
FUZZY sets and systems
www.elsevier.com/locate/fss

Identification and prediction using recurrent compensatory neuro-fuzzy systems

Cheng-Jian Lin*, Cheng-Hung Chen

Department of Computer Science and Information Engineering, Chaoyang University of Technology, No.168, Jifong E. Road, Wufong Township, Taichung County 41349, Taiwan, ROC

Received 28 November 2003; accepted 21 July 2004
Available online 11 August 2004

Abstract

In this paper, a recurrent compensatory neuro-fuzzy system (RCNFS) for identification and prediction is proposed. The compensatory-based fuzzy method uses the adaptive fuzzy operations of neuro-fuzzy systems to make fuzzy logic systems more adaptive and effective. A recurrent network is embedded in the RCNFS by adding feedback connections in the second layer, where the feedback units act as memory elements. In this paper, the RCNFS model is proved to be a universal approximator. Also, an online learning algorithm is proposed which can automatically construct the RCNFS. There are no rules initially in the RCNFS. They are created and adapted as online learning proceeds through simultaneous structure and parameter learning. Structure learning is based on the degree measure and parameter learning is based on the ordered derivative algorithm. Finally, the RCNFS is used in several simulations. The simulation results of the dynamic system model have shown that (1) the RCNFS model converges quickly; (2) the RCNFS model requires a small number of tuning parameters; (3) the RCNFS model can solve temporal problems and approximate a dynamic system.

Keywords: Identification; Neuro-fuzzy systems; Compensatory; Chaotic; Recurrent neural networks

1. Introduction

Recently, neuro-fuzzy systems [1,4,9,12,17,21] have been demonstrated to be successful. Most traditional neuro-fuzzy systems (NFSs) combine the capability of neural networks to learn from processes

* Corresponding author. Fax: +886-43742375.
E-mail address: cjlin@mail.cyut.edu.tw (C.-J. Lin).

0165-0114/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.fss.2004.07.001

and the capability of interpolative reasoning under linguistic information pertaining to numerical variables. They can only be applied to parameter learning based on the ordered derivative algorithm in which the parameters of the membership functions and the connected weights are adjusted and in which the structure of the NFSs has been determined and fixed in advance [11,17]. Sugeno [21] was one of the first to propose self-learning NFSs. Fuzzy rules are automatically generated from training data. A thorough study of NFSs has been done by various researchers [1,5,12]. However, a major disadvantage of existing neuro-fuzzy systems is that their application is limited to static problems as a result of their internal feedforward network structure. Inefficiency occurs for temporal problems. Hence, a recurrent neuro-fuzzy system capable of solving temporal problems is needed.

In a dynamic system, the output is a function of past inputs or past outputs or both. Identification of this kind of system is not as direct as a static system, and to deal with temporal problems of a dynamic system, the recurrent neural network and the recurrent neuro-fuzzy system have attracted great interest. Hence, for nonlinear system processing, the most commonly used model is the neural network or the neuro-fuzzy system. If a feedforward network is adopted for this task, then we should know the number of delayed inputs and outputs in advance and feed these delayed inputs and outputs as a taped line into the network input [15].

The problem with this approach is that the exact order of the dynamic system is usually unknown. To solve this problem, recurrent networks for processing dynamic systems have been considered, and interest in its use has grown steadily in recent years [3,4,6,8,13,24]. In [3], the network is a feedforward multilayer perceptron network with an extra set of context nodes for copying the delayed states of the hidden nodes back to the network input. In [24], a recurrent neuron-fuzzy network was proposed. The structure of the network is similar to the recurrent radial basis function network mentioned above. In [6], the authors provided for a recurrent self-organizing neural fuzzy inference network (RSONFIN) with online supervised learning ability. 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 [8], a recurrent fuzzy neural network (RFNN) was proposed. In RFNN, the recurrent property is achieved by feeding the output of each membership function back to itself so that each membership value is only influenced by its previous value. In [13], the dynamic fuzzy-neural network (DFNN), consisting of recurrent TSK rules, was developed. The premise and defuzzification parts are static while the consequent parts of the fuzzy rules are recurrent neural networks with the internal feedback and time delay synapses. In [4], the TSK-type recurrent fuzzy network (TRFN) structure was proposed. The recurrent property comes from feeding the internal variables back to both the network input and output layers. The internal variables are derived from fuzzy firing strengths. The proposal calls for a design of TRFN using either neural networks or genetic algorithms, depending on the learning environment.

Many papers [3,4,6,8,13,24] have dealt with how to optimize fuzzy membership functions and how to choose an optimal defuzzification scheme for applications by using learning algorithms to adjust the parameter of fuzzy membership functions and defuzzification functions. Unfortunately, for optimizing fuzzy logic reasoning and selecting optimal fuzzy operators, only static fuzzy operators are often used to create fuzzy reasoning [25]. Because the conventional neuro-fuzzy system can only adjust fuzzy membership functions by using fixed fuzzy operations, such as Min and Max, the compensatory neuro-fuzzy system [25] with adaptive fuzzy reasoning is more effective and adaptive than the conventional neuro-fuzzy systems with non-adaptive fuzzy reasoning [12]. Therefore, an effective neuro-fuzzy system should be able not only to adaptively adjust fuzzy membership functions but also to dynamically optimize adaptive fuzzy operators.

In this paper, a recurrent compensatory neuro-fuzzy system (RCNFS) is proposed. The RCNFS is a recurrent multi-layer connectionist network for fuzzy reasoning and can be constructed from a set of fuzzy rules. In the RCNFS, the recurrent property is achieved by feeding the output of each membership function back to itself so that each membership value is influenced by its previous value. In this configuration, each internal variable is responsible for memorizing the temporal history of its membership value. At the same time, the compensatory fuzzy inference method uses adaptive fuzzy operations of neuro-fuzzy systems that can make the fuzzy logic system more adaptive and effective.

An online learning algorithm is proposed which can automatically construct the RCNFS. It consists of structure learning and parameter learning. The structure learning algorithm decides whether to add a new node which satisfies the fuzzy partition of the input data. The parameter learning algorithm is a recursive learning algorithm based on the ordered derivative scheme [23]. This algorithm can tune the free parameters in the RCNFS simultaneously to minimize an output error function.

The proposed learning algorithm has two advantages. First, it does not require a human expert's assistance, its structure is obtained from the input data. Second, it converges quickly, requires a small number of tuning parameters.

This paper is organized as follows. Section 2 describes the compensatory neuro-fuzzy system. Section 3 describes the basic structure and functions of the RCNFS. The online structure and parameter learning algorithms of the RCNFS is presented in Section 4. Next, Section 5 presents the results of the simulation of several problems. Finally, the conclusions are given in the last section.

2. The compensatory operation

Zimmermann [26] first defined the essence of compensatory operations. Zhang and Kandel [25] proposed more extensive compensatory operations based on the pessimistic operation and the optimistic operation. The pessimistic operation can map the inputs x_i to the pessimistic output by making a conservative decision for the pessimistic situation or for even the worst case. For example, $p(x_1, x_2, \ldots, x_N) = \mathrm{MIN}(x_1, x_2, \ldots, x_N)$ or Πx_i. Actually, the t-norm fuzzy operation is a pessimistic operation.

The optimistic operation can map the inputs x_i to the optimistic output by making an optimistic decision for the optimistic situation or for even the best case. For example, $p(x_1, x_2, \ldots, x_N) = \mathrm{MAX}(x_1, x_2, \ldots, x_N)$. Actually, the t-conorm fuzzy operation is an optimistic operation. The compensatory operation can map the pessimistic input x_1 and the optimistic input x_2 to make a relatively compromised decision for the situation between the worst case and the best case. For example, $c(x_1, x_2) = x_1^{1-\gamma} x_2^{\gamma}$, where $\gamma \in [0, 1]$ is called the compensatory degree. Many researchers [10,9,16,20] have used the compensatory operation in fuzzy systems successfully.

The general fuzzy if–then rule is as follows:

$$R_j\text{: IF } x_1 \text{ is } A_{1j} \text{ and} \ldots \text{and } x_n \text{ is } A_{Nj} \text{ THEN } y \text{ is } b_j, \tag{1}$$

where x_i and y are the input dimensions and output variables, respectively; A_{ij} is the linguistic term of the precondition part with membership function $\mu_{A_{ij}}$; b_j is the constant consequent; i is the input dimension, $i = 1, \ldots, N$; N is the number of existing dimensions; j is the number of rules, $j = 1, \ldots, R$; and R is the number of existing rules.

For an input fuzzy set A' in U, the jth fuzzy rule (1) can generate an output fuzzy set b'_j in v by using the sup-dot composition

$$\mu_{b'_j} = \sup_{\underline{x}\in U} [\mu_{A_{1j}\times\cdots\times A_{Nj}\to b_j}(\underline{x}, y) \bullet \mu_{A'}(\underline{x})], \tag{2}$$

where $\underline{x} = (x_1, x_2, \ldots, x_N)$. $\mu_{A_{1j}\times\cdots\times A_{Nj}}(\underline{x})$ is defined in a compensatory operation

$$\mu_{A_{1j}\times\cdots\times A_{Nj}}(\underline{x}) = (u_j)^{1-\gamma_j}(v_j)^{\gamma_j}, \tag{3}$$

where $\gamma_j \in [0, 1]$ is a compensatory degree. The pessimistic operation and the optimistic operation are as follows:

$$u_j = \prod_{i=1}^{N} \mu_{A_{ij}}(x_i), \tag{4}$$

$$v_j = \left[\prod_{i=1}^{N} \mu_{A_{ij}}(x_i)\right]^{1/N}. \tag{5}$$

For simplicity, we can rewrite

$$\mu_{A_{1j}\times\cdots\times A_{Nj}}(x) = \left[\prod_{i=1}^{N} \mu_{A_{ij}}(x_i)\right]^{1-\gamma_j+\gamma_j/N}. \tag{6}$$

Since $\mu_{A'}(x_i) = 1$ for the singleton fuzzifier and $\mu_{b'_j}(y) = 1$, according to (2) we have

$$\mu_{b'_j}(y) = \left[\prod_{i=1}^{N} \mu_{A_{ij}}(x_i)\right]^{1-\gamma_j+\gamma_j/N}. \tag{7}$$

Therefore, we can rewrite the fuzzy if–then rule as follows:

$$R_j\text{: [IF } x_1 \text{ is } A_{1j} \text{ and} \ldots \text{and } x_N \text{ is } A_{Nj}]^{1-\gamma_j+\gamma_j/N} \text{ THEN } y \text{ is } b_j. \tag{8}$$

3. The structure of the RCNFS model

In this section, we describe the RCNFS. Recently, Lee and Teng [8] proposed a model, called the RFNN architecture, for learning and tuning a fuzzy predictor. Our model is similar to the RFNN except for layer three. Layer three of the RFNN uses the product operator while layer three of our model uses the compensatory operator. For initializing parameters in the RFNN model, the rule number should be given in advance. But in our proposed model, users need not give it any a priori knowledge or even any initial information.

The structure of the RCNFS is shown in Fig. 1, which is systematized into N input variables, R-term nodes for each input variable, M output nodes, and $N \times R$ membership function nodes. The RCNFS consists of four layers and $N + (N \times R) + R + M$ nodes, where R denotes the number of existing rules.

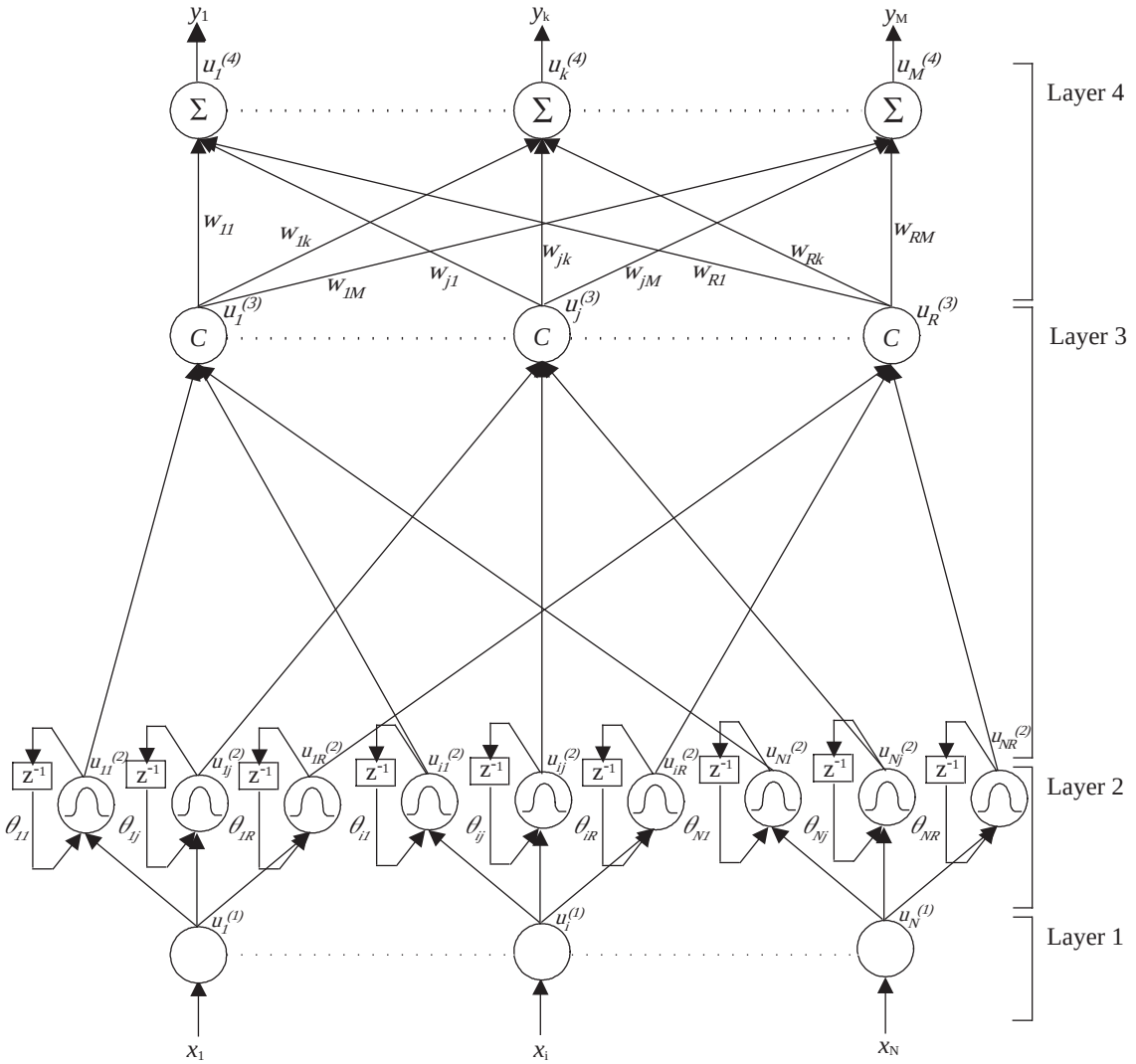


Fig. 1. Structure of the proposed RCNFS.

Nodes in layer 1 are input nodes, which represent input variables. Nodes in layer 2 are called input term nodes and act as membership functions to express the input fuzzy linguistic variables. Nodes in this layer are used to calculate Gaussian membership values. Each node in layer 3 is called a compensatory rule node. Nodes in layer 3 are equal to the number of compensatory fuzzy sets corresponding to each external linguistic input variable. Links before layer 3 represent the preconditions of the rules, and links after layer 3 represent the consequences of the rule nodes. Nodes in layer 4 are called output nodes, where each node is for an individual output of the system. The links between layer 3 and layer 4 are connected by the weighting values w_{jk}.

The RCNFS realizes a fuzzy model in the following form:

$$\text{Rule-}j\text{: [IF } h_{1j} \text{ is } A_{1j} \text{ and } h_{2j} \text{ is } A_{2j} \ldots \text{and } h_{Nj} \text{ is } A_{Nj}]^{1-\gamma+\gamma/N}, \text{ THEN } y' \text{ is } w_j, \tag{9}$$

where for $i = 1, 2, \ldots, N$, $h_{ij} = x_i + u_{ij}^{(2)}(t-1)\theta_{ij}$, y' is the output variable, A_{ij} is the linguistic term of the precondition part, w_j is the constant consequent part, N is the number of input variables, and θ_{ij} denotes the link weight of the feedback unit. In other words, the input of each membership function is the network input x_i plus the temporal term $u_{ij}^{(2)}\theta_{ij}$. Therefore, the fuzzy system, with its memory (feedback units), can be considered to be a dynamic fuzzy inference system.

Next, we shall describe the operation functions of the nodes in each layer of the RCNFS model. In the following description, $u(\mathrm{l})$ denotes the output of a node in the lth layer.

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

$$u_i^{(1)} = x_i. \tag{10}$$

Layer 2 (Input term node): Nodes in this layer correspond to one linguistic label of the input variables in Layer 1 and to a unit of memory. That is, the membership value specifying the degree to which an input value and a unit of memory belong to a fuzzy set is calculated in Layer 2. The Gaussian membership function, the operation performed in Layer 2, is

$$u_{ij}^{(2)} = \exp\left(-\frac{[h_{ij} - m_{ij}]^2}{\sigma_{ij}^2}\right), \tag{11}$$

where m_{ij} and σ_{ij} are, respectively, the mean and variance of the Gaussian membership function of the jth term of the ith input variable x_i. In addition, the inputs of this layer for discrete time t can be defined by

$$h_{ij}(t) = u_i^{(1)}(t) + u_{ij}^{(2)}(t-1)\theta_{ij}, \tag{12}$$

where $u_{ij}^{(2)}(t-1)$ denotes the feedback unit of memory, which stores the past information of the system, and where θ_{ij} denotes the link weight of the feedback unit.

Layer 3 (Compensatory rule node): Nodes in this layer represent the precondition part of one fuzzy logic rule. They receive the one-dimensional membership degrees of the associated rule from the nodes of a set in Layer 2. Here, we use the compensatory operator previously mentioned to perform IF-condition matching of fuzzy rules. As a result, the output function of each inference node is

$$u_j^{(3)} = \left(\prod_i u_{ij}^{(2)}\right)^{1-\gamma_j+(\gamma_j/N)}, \tag{13}$$

where the $\prod_i u_{ij}^{(2)}$ of a rule node represents the firing strength of its corresponding rule and where $\gamma_j \in [0, 1]$ is called the compensatory degree. When γ_j is turned, the compensatory operator becomes more adaptive.

Layer 4 (Output node): This layer acts as an output layer. The node in this layer is labeled $\sum$, and it sums all incoming signals to obtain the final inferred result

$$u_k^{(4)} = \sum_j u_j^{(3)} w_{jk}, \tag{14}$$

where the weight w_{jk} is the output action strength of the kth output associated with the jth rule, and $u_k^{(4)}$ is the kth output of the RCNFS.

Finally, the overall representation of input x and the kth output is

$$y_k(t) = u_k^{(4)}(t) = \sum_{j=1}^{R} w_{jk} \left\{ \prod_{i=1}^{N} \exp\left[- \frac{[u_i^{(1)}(t) + u_{ij}^{(2)}(t-1)\theta_{ij} - m_{ij}]^2}{\sigma_{ij}^2} \right] \right\}^{1-\gamma_j+(\gamma_j/N)}, \tag{15}$$

where $u_i^{(1)} = x_i$, $\gamma_j = c_j^2/(c_j^2 + d_j^2)$ is the compensatory degree, m_{ij}, σ_{ij}, θ_{ij}, c_j, d_j, and w_{jk} are the tuning parameters, and $u_{ij}^{(2)}(t-1) = \exp\{-[u_i^{(1)}(t-1) + u_{ij}^{(2)}(t-2)\theta_{ij} - m_{ij}]^2/(\sigma_{ij})^2\}$. Explicitly, when the RCNFS is used, the same inputs at different times yield different outputs. As above, the number of tuning parameters for the RCNFS is $R \times (N \times 3 + 2 + M)$, where N, R, and M denote the number of inputs, existing rules, and outputs, respectively. By inspecting the structure of the proposed RCNFS network, we find that the recurrent properties are achieved by involving internal memory in the form of feedback connections to the feedforward CNFS network [9].

Since a recurrent neuron has an internal feedback loop, it captures the dynamic response of a system. We show that the universal approximation characteristic of the CNFS network is extended to the RCNFS. Therefore, the proposed RCNFS can be shown to be a universal uniform approximation for continuous functions over compact sets. The detailed proof is shown in the appendix. We have the following result.

Theorem 1. *Universal approximation theorem—for any real continuous function g in a compact set $U \subset \Re^N$, and for any given arbitrary $\varepsilon > 0$, there is a model f such that*

$$\sup_{x \in U} \| f(x) - g(x) \| < \varepsilon. \tag{16}$$

Here $\| \bullet \|$ can be any norm.

This theorem shows that if the RCNFS has a sufficiently large number of compensatory fuzzy rules, then it can approximate any continuous function in $C(\Re^N)$ over a compact subset of $\Re^N$. For system over identification, the theorem means that for any given continuous output trajectory $y^d(t)$ of any nonlinear dynamic system over any compact time-interval $t \in [t_0, T]$, the output $y(t)$ of the RCNFS can approximate $y^d(t)$ uniformly with arbitrarily high accuracy.

4. The online learning algorithm

In this section, we present an online learning algorithm for constructing the RCNFS. The proposed learning algorithm consists of a structure learning phase and a parameter learning phase. Structure learning

is based on the degree used to determine the number of fuzzy rules. Parameter learning is based upon supervised learning algorithms. The ordered derivative algorithm that minimizes a given cost function adjusts the weights in the consequent part, the parameters of the membership functions, the weights of the feedback, and the compensatory degree.

Initially, there are no nodes in the network except the input–output nodes, i.e., there are no any rule nodes and memberships. They are created dynamically and automatically as learning proceeds upon receiving online incoming training data by performing the structure and parameter learning processes. The details of the structure learning phase and the parameter learning phase are described in the rest of this section.

4.1. The structure learning phase

The first step the structure learning is to determine whether to extract a new rule from the training data as well as to determine the number of fuzzy sets in the universal of discourse of each input variable, since one cluster in the input space corresponds to one potential fuzzy logic rule, with m_{ij} and σ_{ij} representing the mean and variance of that cluster, respectively. For each incoming pattern x_i rule firing strength can be regarded as the degree to which the incoming pattern belongs to the corresponding cluster.

For computational efficiency, we can use the compensatory operation of the firing strength obtained from $[\Pi u_{ij}^{(2)}]^{1-\gamma+\gamma/N}$ directly as the degree

$$F_j = \left[\prod_i u_{ij}^{(2)}\right]^{1-\gamma_j+\gamma_j/N}, \tag{17}$$

where $F_j \in [0, 1]$. According to the degree, the criterion for generating a new fuzzy rule and a new wavelet base for new incoming data is described as follows. Find the maximum degree $F_{\max}$

$$F_{\max} = \max_{1 \leqslant j \leqslant R_{(t)}} F_j, \tag{18}$$

where $R_{(t)}$ is the number of existing rules at time t. If $F_{\max} \leqslant \overline{F}$, then a new rule is generated, where $\overline{F} \in (0, 1)$ is a prespecified threshold that decays during the learning process. Once a new rule is generated, the next step is to assign the initial mean, variance, and weight of the feedback for the new membership function. Since our goal is to minimize an objective function, the mean, variance, and weight of the feedback are all adjustable later in the parameter learning phase. Hence, the mean, variance, and weight of the feedback for the new membership function are set as follows:

$$m_{ij}^{(R_{(t+1)})} = x_i, \tag{19}$$

$$\sigma_{ij}^{(R_{(t+1)})} = \sigma_{\text{init}}, \tag{20}$$

$$\theta_{ij}^{(R_{(t+1)})} = random, \tag{21}$$

$$u_{ij}^{(2)}(t-1)^{(R_{(t+1)})} = 0, \tag{22}$$

where x_i is the new input data and σ_{init} is a pre-specified constant. Since the generation of a membership function corresponds to the generation of a new fuzzy rule, the compensatory degree $c_j^{(R_{(t+1)})}$, $d_j^{(R_{(t+1)})}$, i.e., $\gamma_j^{(R_{(t+1)})} = (c_j^{(R_{(t+1)})})^2/((c_j^{(R_{(t+1)})})^2 + (d_j^{(R_{(t+1)})})^2)$, and the weight, $w_j^{(R_{(i+1)})}$, associated with a new fuzzy rule has to be decided. Generally, the weight of the feedback $\theta_j^{(R_{(t+1)})}$, the compensatory degree $c_j^{(R_{(t+1)})}$, $d_j^{(R_{(t+1)})}$, and the weight $w_j^{(R_{(t+1)})}$ are selected with random values in $[-1, 1]$.

The whole algorithm for the generation of new fuzzy rules as well as fuzzy sets in each input variable is as follows. We make the assumption that no rules initially exist:

Step 1: IF x_i is the first incoming pattern THEN do
{Generate a new rule
with mean $m_{i1} = x_i$, variance $\sigma_{i1} = \sigma_{\text{init}}$, weight of feedback $\theta_{i1} = random$,
compensatory degree $c_1 = random$, $d_1 = random$, weight $w_1 = random$
where σ_{init} is a prespecified constant.
}

Step 2: ELSE for each newly incoming x_i, do
{Find $F_{\max} = \max_{1 \leqslant j \leqslant R_{(t)}} F_j$,
IF $F_{\max} \geqslant \overline{F}$
do nothing
ELSE
{$R_{(t+1)} = R_{(t)} + 1$
generate a new rule
with mean $m_{ij}^{R_{(t+1)}} = x_i$, variance $\sigma_{ij}^{R_{(t+1)}} = \sigma_{\text{init}}$,
weight of feedback $\theta_{ij}^{R_{(t+1)}} = random$, compensatory degree $c_j^{R_{(t+1)}} = random$,
$d_j^{R_{(t+1)}} = random$, weight $w_j^{R_{(t+1)}} = random$
where σ_{init} is a prespecified constant.}
}

4.2. The parameter learning phase

After the network structure is adjusted according to the current training pattern, the network then enters the parameter learning phase to adjust the parameters of the network optimally based on the same training pattern. The learning process involves determining the minimum of a given cost function. The gradient of the cost function is computed and the parameters are adjusted along the negative gradient. Because the RCNFS is a dynamic system with feedback connections, the learning algorithm used in the feedforward radial basis function networks [14] or adaptive fuzzy systems [22] cannot be applied to it directly. Also, due to the online learning property of the RCNFS, the offline learning algorithms for the recurrent neural networks, like backpropagation through time and time-dependent recurrent backpropagation [11], cannot be applied here. Instead, the ordered derivative [23], 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. When the single output case is considered for clarity, our goal is to minimize the cost function E, which

is defined as follows:

$$E(t+1) = \frac{1}{2}[y(t+1) - y^d(t+1)]^2, \tag{23}$$

where $Y(t+1)$ is the model output and $Y^d(t+1)$ is the desired output at time $t+1$. When the ordered derivative learning algorithm is used, the free parameters of the RCNFS are adjusted such that the error defined in Eq. (23) is less than the desired threshold value after a given number of training cycles. The parameter learning algorithm based on the ordered derivative algorithm is described below:

Layer 4: The error term to be propagated is calculated as

$$\delta^{(4)}(t+1) = -\frac{\partial E}{\partial y}(t+1) = y^d(t+1) - y(t+1). \tag{24}$$

And the weight is updated by the amount

$$\begin{aligned}\Delta w_j(t+1) &= -\frac{\partial E}{\partial w_j}(t+1) = \left[-\frac{\partial E}{\partial y}(t+1)\right]\left[\frac{\partial y(t+1)}{\partial w_j(t)}\right] \\ &= \delta^{(4)}(t+1)u_j^{(3)}(t).\end{aligned} \tag{25}$$

The weight in layer 4 is updated according to the following equation:

$$\begin{aligned}w_j(t+1) &= w_j(t) + \eta_w\, \Delta w_j(t+1) \\ &= w_j(t) + \eta_w \delta^{(4)}(t+1)u_j^{(3)}(t),\end{aligned} \tag{26}$$

where the factor η_w is the learning rate parameter of the weight, and t denotes the iteration number of the jth.

Layer 3: In this layer, only the error term needs to be computed and propagated

$$\begin{aligned}\delta^{(3)}(t+1) &= -\frac{\partial E}{\partial u_j^{(3)}}(t+1) = \left[-\frac{\partial E}{\partial y}(t+1)\right]\left[\frac{\partial y(t+1)}{\partial u_j^{(3)}(t)}\right] \\ &= \delta^{(4)}(t+1)w_j(t).\end{aligned} \tag{27}$$

To eliminate the constraint $\gamma_j \in [0, 1]$, we redefine γ_j as follows:

$$\gamma_j(t+1) = \frac{c_j^2(t+1)}{c_j^2(t+1) + d_j^2(t+1)}, \tag{28}$$

$$\begin{aligned}\Delta\gamma_j(t+1) &= -\frac{\partial E}{\partial \gamma_j}(t+1) = \left[-\frac{\partial E(t+1)}{\partial u_j^{(3)}(t)}\right]\left[\frac{\partial u_j^{(3)}}{\partial \gamma_j}(t)\right] \\ &= \delta^{(3)}(t+1)\left[\frac{1}{n} - 1\right]\ln\left[\prod_i u_{ij}^{(2)}(t)\right]u_j^{(3)}(t).\end{aligned} \tag{29}$$

We have

$$c_j(t+1) = c_j(t) + \eta_c\left\{\frac{2c_j(t)d_j^2(t)}{[c_j^2(t) + d_j^2(t)]^2}\right\}\Delta\gamma_j(t+1), \tag{30}$$

$$d_j(t+1) = d_j(t) - \eta_d \left\{ \frac{2c_j^2(t)d_j(t)}{[c_j^2(t) + d_j^2(t)]^2} \right\} \Delta\gamma_j(t+1), \tag{31}$$

$$\gamma_i(t+1) = \frac{c_j^2(t+1)}{c_j^2(t+1) + d_j^2(t+1)}. \tag{32}$$

In all the above formulas, η_c and η_d are the learning rate of the parameter c_j and the parameter d_j, respectively.

Layer 2: The error term is calculated as follows:

$$\begin{aligned}\delta^{(2)}(t+1) &= -\frac{\partial E}{\partial u_{ij}^{(2)}}(t+1) = \left[-\frac{\partial E(t+1)}{\partial u_j^{(3)}(t)} \right] \left[\frac{\partial u_j^{(3)}}{\partial u_{ij}^{(2)}}(t) \right] \\ &= \delta^{(3)}(t+1) \left[1 - \gamma_j + \frac{\gamma_j}{n} \right] \left[\prod_i u_{ij}^{(2)}(t) \right]^{(-\gamma_j + (\gamma_j/n))} \left[\prod_{l \neq i} u_{lj}^{(2)}(t) \right], \end{aligned} \tag{33}$$

where l is the lth dimension. The update mean is

$$\begin{aligned}\Delta m_{ij}(t+1) &= -\frac{\partial E}{\partial m_{ij}}(t+1) = \left[-\frac{\partial E(t+1)}{\partial u_{ij}^{(2)}(t)} \right] \left[\frac{\partial u_{ij}^{(2)}}{\partial m_{ij}}(t) \right] \\ &= \delta^{(2)}(t+1) u_{ij}^{(2)}(t) \left\{ \frac{2[h_{ij} - m_{ij}]}{\sigma_{ij}^2}(t) \right\} \left\{ \frac{\partial h_{ij}}{\partial m_{ij}}(t) - 1 \right\}, \end{aligned} \tag{34}$$

where

$$\frac{\partial h_{ij}}{\partial m_{ij}}(t) = \theta_{ij}(t) u_{ij}^{(2)}(t-1) \left\{ \frac{-2[h_{ij} - m_{ij}]}{\sigma_{ij}^2}(t-1) \right\} \left(\frac{\partial h_{ij}}{\partial m_{ij}}(t-1) - 1 \right). \tag{35}$$

The updated variance is

$$\begin{aligned}\Delta\sigma_{ij}(t+1) &= -\frac{\partial E}{\partial \sigma_{ij}}(t+1) = \left[-\frac{\partial E(t+1)}{\partial u_{ij}^{(2)}(t)} \right] \left[\frac{\partial u_{ij}^{(2)}}{\partial \sigma_{ij}}(t) \right] \\ &= \delta^{(2)}(t+1) u_{ij}^{(2)}(t) \left\{ \frac{2\left[h_{ij} - m_{ij}\right]\left[(h_{ij} - m_{ij}) - \sigma_{ij}(\partial h_{ij}/\partial \sigma_{ij})\right]}{\sigma_{ij}^3}(t) \right\}, \end{aligned} \tag{36}$$

where

$$\frac{\partial h_{ij}}{\partial \sigma_{ij}}(t) = \theta_{ij}(t) u_{ij}^{(2)}(t-1) \left\{ \frac{2[h_{ij} - m_{ij}][(h_{ij} - m_{ij}) - \sigma_{ij}(\partial h_{ij}/\partial \sigma_{ij})]}{\sigma_{ij}^3}(t-1) \right\}. \tag{37}$$

The updated weight of the feedback is

$$\Delta\theta_{ij}(t+1) = -\frac{\partial E}{\partial \theta_{ij}}(t+1) = \left[-\frac{\partial E(t+1)}{\partial u_{ij}^{(2)}(t)} \right] \left[\frac{\partial u_{ij}^{(2)}}{\partial \theta_{ij}}(t) \right],$$

$$= \delta^{(2)}(t+1)u_{ij}^{(2)}(t)\left\{\frac{-2[h_{ij}-m_{ij}]}{\sigma_{ij}^2}\frac{\partial h_{ij}}{\partial\theta_{ij}}(t)\right\}, \tag{38}$$

where

$$\frac{\partial h_{ij}}{\partial\theta_{ij}}(t) = \theta_{ij}(t)u_{ij}^{(2)}(t-1)\left\{\frac{-2[h_{ij}-m_{ij}]}{\sigma_{ij}^2}\frac{\partial h_{ij}}{\partial\theta_{ij}}(t-1)\right\} + u_{ij}^{(2)}(t+1). \tag{39}$$

The mean and variance of the membership functions and the weight of the feedback in this layer are updated as follows:

$$m_{ij}(t+1) = m_{ij}(t) + \eta_m\,\Delta m_{ij}(t+1), \tag{40}$$

$$\sigma_{ij}(t+1) = \sigma_{ij}(t) + \eta_\sigma\,\Delta\sigma_{ij}(t+1), \tag{41}$$

$$\theta_{ij}(t+1) = \theta_{ij}(t) + \eta_\theta\,\Delta\theta_{ij}(t+1), \tag{42}$$

where η_m, η_σ, and η_θ are the learning rate parameters of the mean, the variance, and the weight of the feedback for the Gaussian function, respectively. The values $\partial h_{ij}/\partial m_{ij}$, $\partial h_{ij}/\partial\sigma_{ij}$, $\partial h_{ij}/\partial\theta_{ij}$ are equal to zero initially and are reset to zero after a period of time to avoid the accumulation of too far away errors.

5. Illustrative examples

The performance of the RCNFS for temporal problems was verified. Several examples and performance comparisons with other recurrent neuro-fuzzy system are presented in this section. These parameters (η_m, η_σ, η_θ, η_c, η_d, η_w, η_{init}, $\overline{F}$) are set in advance, and the number of training epochs for the RCNFS in each example was determined based on the desired accuracy.

5.1. Example 1: Prediction of time sequence

To clearly verify if the proposed RCNFS can learn temporal relationships, a simple time sequence prediction problem found in [19] was used in the following example.

The test bed that was used is shown in Fig. 2(a). The test bed was in the shape of an "8" and was made up of a series of 12 points which were to be presented to the network in the order shown. The RCNFS was asked to predict the succeeding point for every presented point.

Obviously, a static network cannot accomplish this task, since the point at coordinate (0,0) has two successors: points 5 and 11. The RCNFS 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 this example, the RCNFS contains only two input nodes, which were activated with the two dimensional coordinate of the current point, and two output nodes, which represented the two dimensional coordinate of the predicted point. A learning rate of $\eta_w = \eta_c = \eta_d = \eta_m = \eta_\sigma = \eta_\theta = 0.05$ and a prespecified threshold of $\overline{F} = 10^{-4}$ were chosen. After training, a root-mean-square (rms) error of 0.000237 was achieved. The predicted values with 12 fuzzy logic rules ($\sigma_{\text{init}} = 0.08$) of RCNFS are shown in Fig. 2(b). Simulation results show that we can obtain perfect prediction capability.

We also applied the RFNN model [8] and a traditional (non-recurrent) fuzzy neural network (FNN) [1] to this time prediction problem. Fig. 2(c) shows the prediction results using the RFNN model [8]. In the

figure, the RFNN also obtains prediction capability, but some time prediction points cannot be matched exactly. Fig. 2(d) shows that a feedforward fuzzy neural network cannot make predictions successfully. Fig. 2(e) shows the learning curves of the RCNFS model, the RFNN model, and the FNN model.

From the simulation results shown in Fig. 2(d), we can see that the FNN model is inappropriate for time sequence prediction because of its static mapping. To give a clear understanding of this performance, a

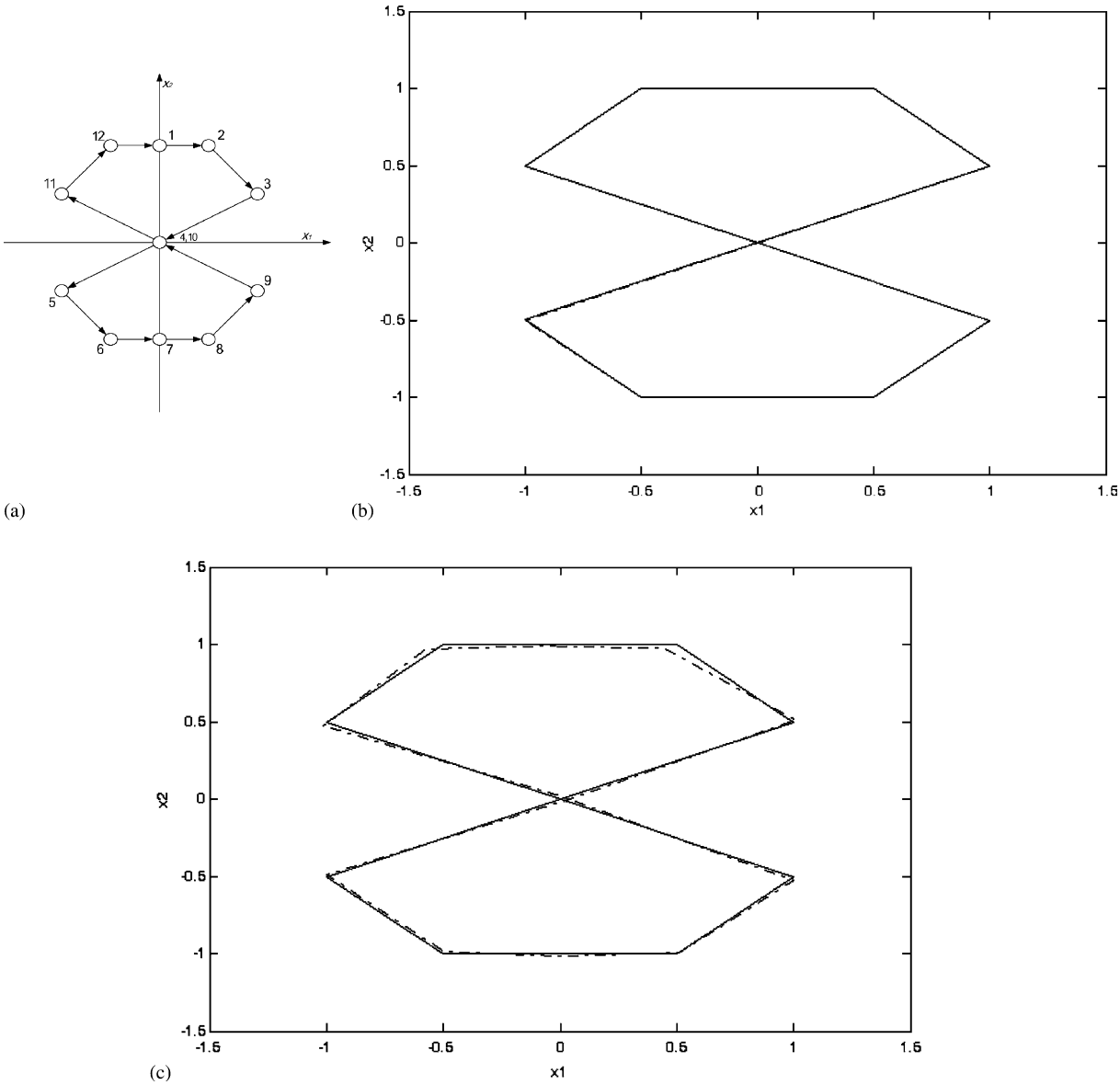
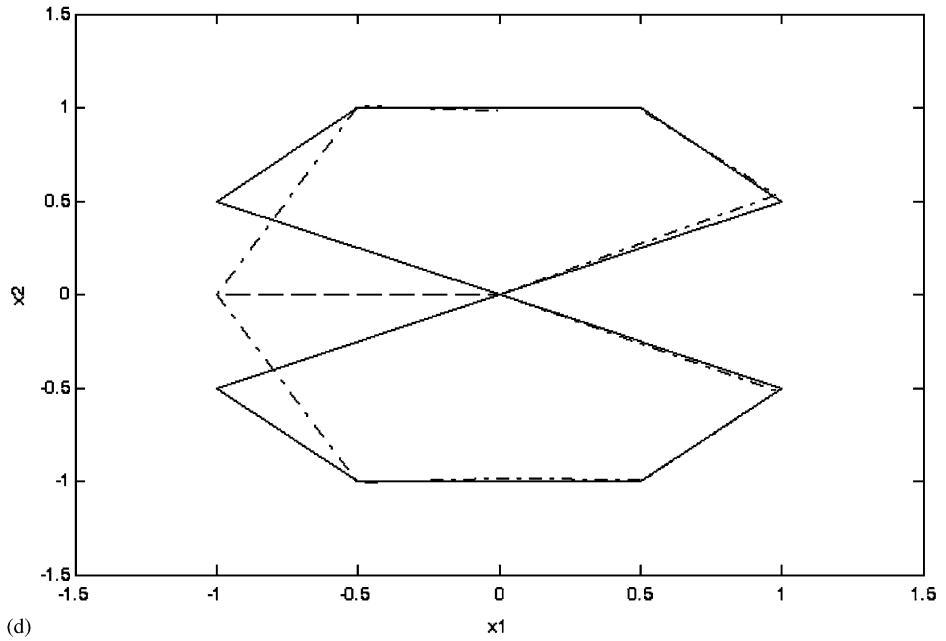
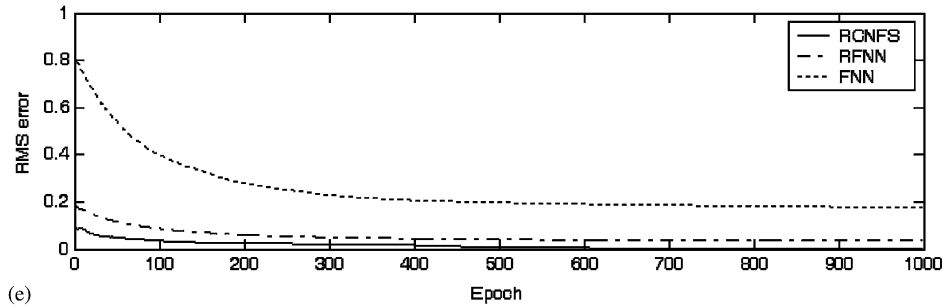


Fig. 2. Simulation results of time sequence prediction: (a) test bed for the next sample prediction experiment in Example 1, (b) results of prediction using the RCNFS after 1000 training epochs, (c) results of prediction using the RFNN after 1000 training epochs, (d) results of prediction using the FNN after 1000 training epochs, and (e) learning curves of the RCNFS, the RFNN [8] and the FNN [1].



(d)



(e)

Fig. 2. (*Continued*).

Table 1
Performance comparison of various existing models on time sequence prediction

	RCNFS	RFNN [8]	FNN [1]
Rule numbers	12	12	12
Nodes	40	40	66
Parameter	120	96	108
RMS error	0.000237	0.0063	0.1758
Epochs	1000	1000	1000

comparison between the RFNN model [8] and the FNN model [1] on the same problem is given in Table 1. Although the RCNFS needs more adjustable parameters than RFNN and FNN under the same fuzzy rules that are required, our model can obtain a smaller rms error and can converge more quickly.

5.2. Example 2: Identification of a nonlinear dynamic system

Consider the following dynamic plant with longer input delays:

$$y_p(t+1) = 0.72y_p(t) + 0.025y_p(t-1)u(t-1) + 0.01u^2(t-2) + 0.2u(t-3). \tag{43}$$

This plant is the same as that used in [7]. In our model, with only two input values, $y_p(t)$ and $u(t)$, the input values were fed to the RCNFS to determine the output $y_p(t)$. The training inputs were independent and identically distributed (i.i.d.) uniform sequence over $[-2, 2]$ for about half of the training time and a single sinusoid signal of $1.05\sin(\pi t/45)$ for the remaining training time. There is no repetition of these 900 training data; that is, we had different training sets for each epoch. The check input signal $u(t)$, the equation below, was used to determine the identification results:

$$u(t) = \begin{cases} \sin(\frac{\pi t}{25}), & 0 < t < 250, \\ 1.0, & 250 \leqslant t < 500, \\ -1.0, & 500 \leqslant t < 750, \\ 0.3\sin(\frac{\pi t}{25}) + 0.1\sin(\frac{\pi t}{32}), & \\ +0.6\sin(\frac{\pi t}{10}), & 750 \leqslant t < 1000. \end{cases} \tag{44}$$

During the training, we used only 10 epochs, with 900 time steps in each epoch. A learning rate of $\eta_w = \eta_c = \eta_d = \eta_m = \eta_\sigma = \eta_\theta = 0.05$, an initial variance of $\sigma_{\text{init}} = 0.2$, and a prespecified threshold of $\overline{F} = 10^{-4}$ were chosen. After training, the final rms error was 0.0057. Three fuzzy logic rules were generated. The three rules were

Rule 1: IF $[u(t)$ is $\mu(-0.5757, 0.5546)$ and $y(t)$ is $\mu(-0.3447, 0.4765)]^{0.6434}$ THEN $y(t+1)$ is -0.1256.
Rule 2: IF $[u(t)$ is $\mu(0.0330, 0.5011)$ and $y(t)$ is $\mu(0.0137, -0.6984)]^{0.9909}$ THEN $y(t+1)$ is 0.9516.
Rule 3: IF $[u(t)$ is $\mu(1.1974, 0.3089)$ and $y(t)$ is $\mu(0.7244, 0.6678)]^{0.3393}$ THEN $y(t+1)$ is -0.5021.

Fig. 3(a) shows the distribution with some of the training patterns and the final task of the rules in the $[u(t), y(t)]$ graph. This is due to the parameter learning process, which changes the mean and variance of each membership function at each time step to minimize the output error function. The membership functions on the $u(t)$ and $y(t)$ dimension are shown in Figs. 3(b) and 3(c). Fig. 4(a) shows the outputs of the plant and the RCNFS model. We obtained perfect identification capability. Fig. 4(b) illustrates the error between the desired output and the RCNFS output. The learning curves of the RCNFS model and the RFNN [8] model are shown in Fig. 4(c). The figure shows that our model obtained a smaller rms error and a quicker convergence since the compensatory operator was used.

We compared the performance of our model with that of other existing recurrent methods (ERNN [3], RSONFIN [6], RFNN [8], TRFN-S [4]). The comparison results are tabulated in Table 2. The results show that the proposed RCNFS model results in smaller rms error than other recurrent methods.

5.3. Example 3: Identification of a chaotic system

The discrete time Henon system was repeatedly used in the study of chaotic dynamics and was not too simple in the sense that it was of second order with one delay and two parameters [2]. This chaotic system is described by

$$y(t+1) = -Py^2(t) + Qy(t-1) + 1.0, \quad \text{for } t = 1, 2, \ldots \tag{45}$$

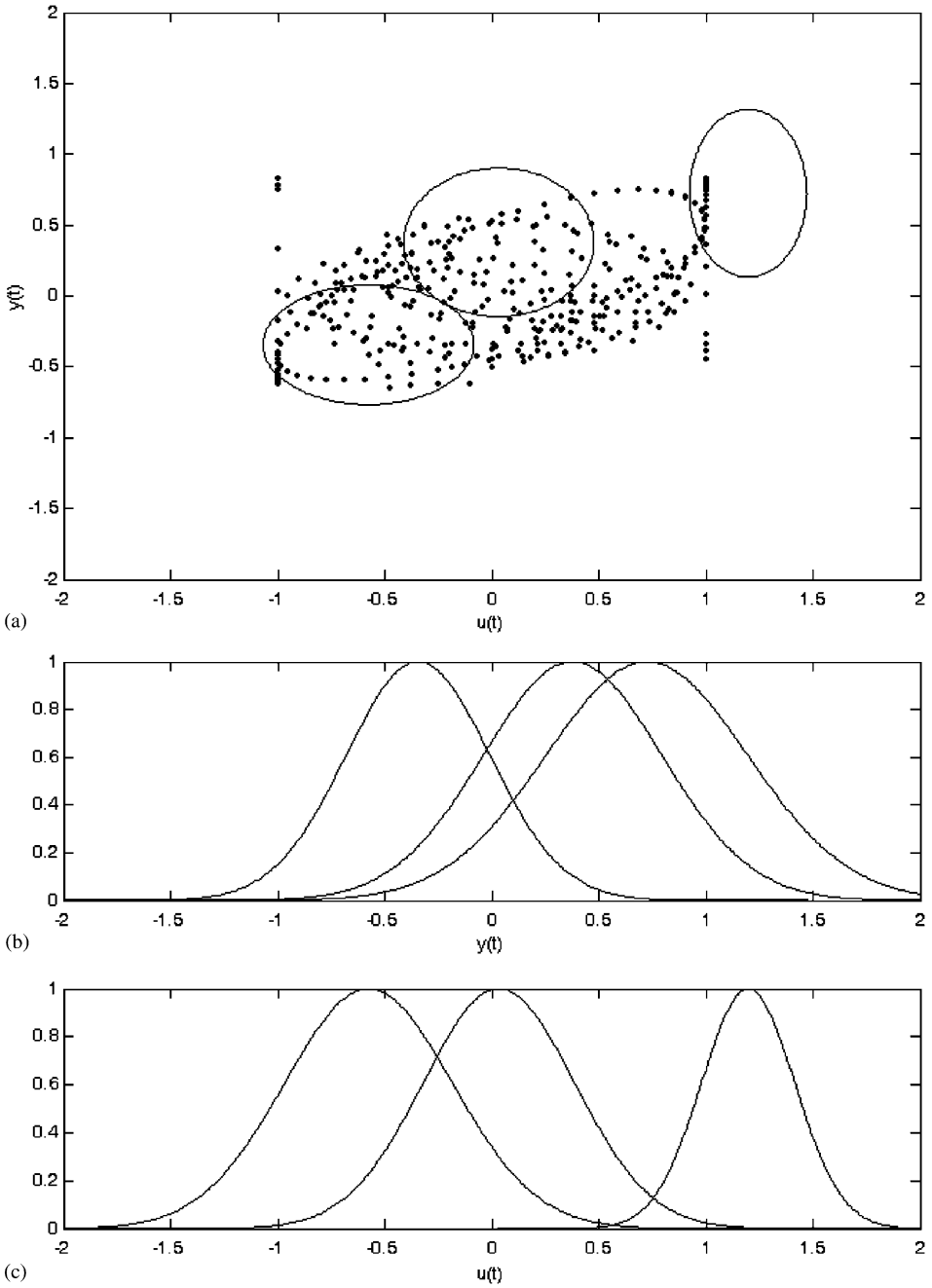


Fig. 3. Simulation results of the RCNFS on the membership functions of each input variable in Example 2. (a) The input training patterns and the final assignment of rules. (b) The distribution of the membership functions in the $y(t)$ dimensions. (c) The distribution of the membership functions in the $u(t)$ dimensions.

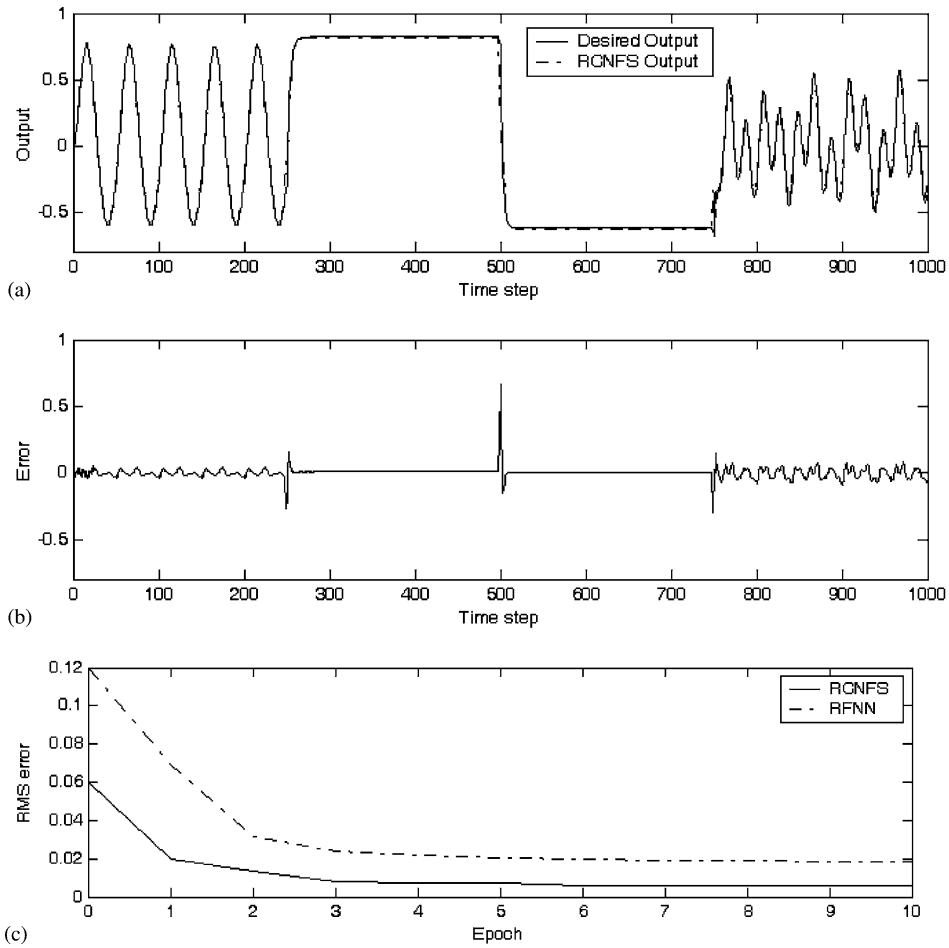


Fig. 4. Simulation results for nonlinear system identification: (a) the outputs of the plant and the RCNFS, (b) the error between the RCNFS output and the desired output, and (c) learning curves of the RCNFS and the RFNN [8].

Table 2
Performance comparison of various recurrent methods on the identification problem

	RCNFS	TRFN-S [4]	RFNN [8]	RSONFIN [6]	ERNN [3]
Parameter	27	33	21	49	54
RMS error (train)	0.0057	0.0067	0.0187	0.0300	0.0360
RMS error (test)	0.0221	0.0313	0.0402	0.0600	0.0780
Epochs	10	10	10	10	10

which, with $P = 1.4$ and $Q = 0.3$, produced a chaotic strange attractor as shown in Fig. 5(a). For this training, the input of the RCNFS was $y(t-1)$ and the output was $y(t)$. We first randomly took the training data (1000 pairs) from the system over the interval $[-1.5, 1.5]$. Then the RCNFS was used to approximate the chaotic system.

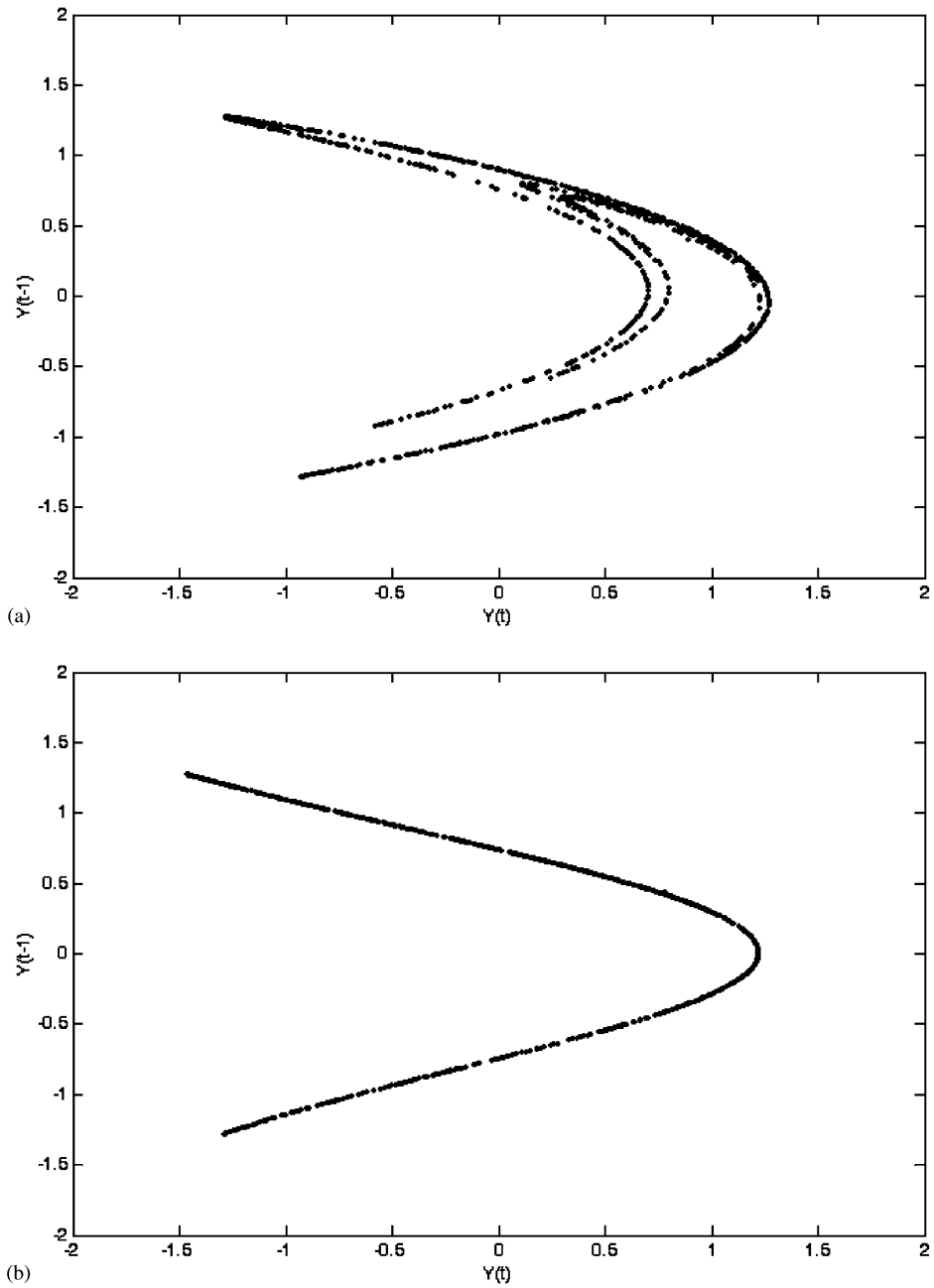


Fig. 5. Simulation results for identification of a chaotic system: (a) check data of this chaotic system, (b) result of identification using the FNN [1] for the chaotic system, and (c) result of identification using the RCNFS for the chaotic system.

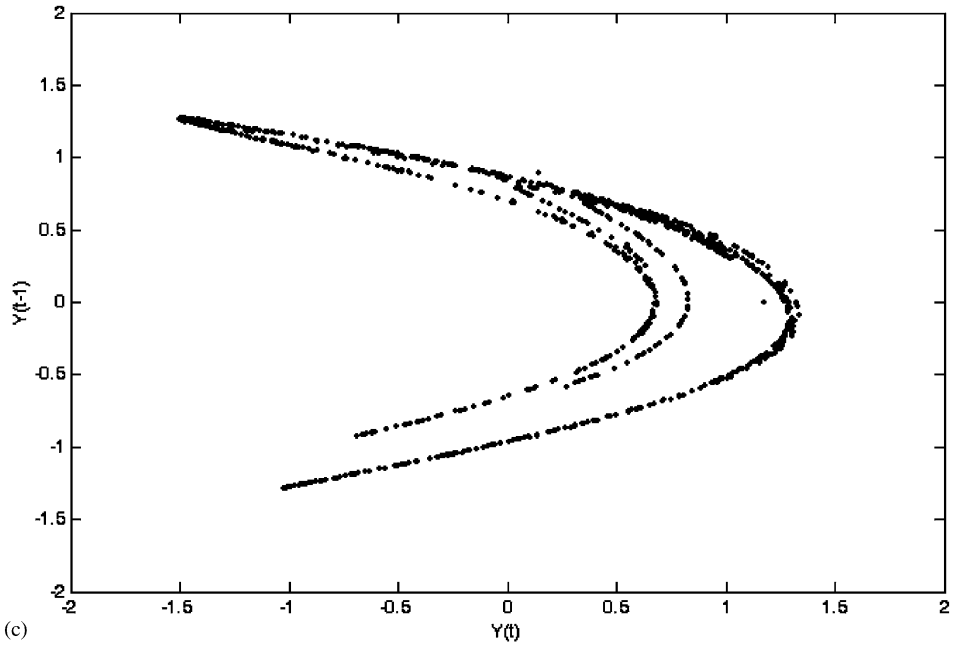


Fig. 5. (*Continued*).

Table 3
Performance comparison of various methods in the chaotic system of Example 3

	RCNFS	RFNN [8]	FNN [1]
Number of rules	9	9	9
RMS error (train)	0.0032	0.0060	0.1238
RMS error (test)	0.0035	0.0067	0.1077
Epochs	100	100	100

In applying the RCNFS to this example, we used 100 epochs for training. Here, the initial point was $[y(1), y(0)]^{\mathrm{T}} = [0.4, 0.4]^{\mathrm{T}}$. A learning rate of $\eta_w = \eta_c = \eta_d = \eta_m = \eta_\sigma = \eta_\theta = 0.05$, an initial variance of $\sigma_{\mathrm{init}} = 0.1$, and a prespecified threshold of $\overline{F} = 10^{-1}$ were used. After training, a rms error of 0.0032 was achieved, and nine fuzzy logic rules were generated.

The phase plane of this chaotic system after training for the FNN [1] and the RCNFS are shown in Figs. 5(b) and (c). In Fig. 5(b), the simulation results show that the FNN is inappropriate for dynamic chaotic systems because of its static mapping. We compared the performance of our model with [8,1]. The comparison results are tabulated in Table 3. The results show that the proposed RCNFS model is able to maintain a smaller rms error than other methods.

6. Conclusion

A recurrent compensatory neuro-fuzzy system (RCNFS) was proposed in this paper. Compensatory operators are used to optimize fuzzy logic reasoning and to select optimal fuzzy operators. Therefore, an

effective neuro-fuzzy system should be able not only to adaptively adjust fuzzy membership functions but also to dynamically optimize adaptive fuzzy operators. An online learning algorithm was proposed to perform structure learning and parameter learning. The simulation results show that the proposed learning algorithm converges quickly and requires a small number of tuning parameters.

Acknowledgements

This research is supported by the National Science Council of ROC under grant NSC 92-2213-E-324-002.

Appendix A. Proof of the Universal Approximation Theorem

Theorem 1 will be proven using the Stone–Weierstrass theorem. The structure of the proposed RCNFS is illustrated in Fig. 1. The single output of the RCNFS can be expressed as

$$y(x) = \sum_{j=1}^{R} w_j u_j^{(3)}(x), \tag{A.1}$$

where $x \in \Re^N$ is the input variable of the RCNFS,

$$u_j^{(3)}(x) = \left[\prod_{i=1}^{N} u_{ij}^{(2)}(x) \right]^{1-\gamma_j+\gamma_j/N}, \tag{A.2}$$

where

$$u_{ij}^{(2)}(x) = \exp\left(- \frac{[h_{ij}(x) - m_{ij}]^2}{\sigma_{ij}^2} \right), \tag{A.3}$$

where $h_{ij}(t) = x(t) + u_{ij}^{(2)}(t-1)\theta_{ij}$ denotes the input of layer 2, the link weight w_j is the output action strength, $m_{ij}, \sigma_{ij} \in \Re$, and Y is the family of function $y : \Re^N \to \Re$.

Theorem 2 (*Stone–Weierstrass Theorem* [18]). *Let A be a set of real continuous functions in a compact set U. If* (1) *U is an algebra. That is, if $f_1, f_2 \in A$, and $c \in \Re$, then $f_1 + f_2 \in A$, $f_1 f_2 \in A$, and $cf_1 \in A$;* (2) *A separates points in U. That is, for $x, y \in U$, $x \neq y$, there exists $f_1 \in A$ such that $f_1(x) \neq f_2(y)$; and* (3) *A vanishes at no point in U. That is, for each $x \in U$, there exists $f_1 \in A$ such that $f_1(x) \neq 0$. Then the uniform closure of A consists of all real continuous functions in U.*

Lemma 1. *Let Y be the family of y defined in Eq.* (A.1). *Then $Y \subset U$, where U is a compact set.*

Proof of Lemma 1. Here, the membership function

$$0 < \mu_{A_{ij}}(h_{ij}) = \exp\left[- \frac{(h_{ij} - m_{ij})^2}{(\sigma_{ij})^2} \right] \leqslant 1$$

and, therefore, the continuous function $u_j^{(3)}(x)$ is closed and bounded for all $x \in \Re^N$. That is, $Y \subset A$. □

Proof of Theorem 2. First, we prove that Y is an algebra. Let $f_1,\ f_2 \in Y$, such that we can write them as

$$\begin{aligned} f_1(x) &= \sum_{j_1=1}^{R_1} w1_{j_1} u1_{j_1}^{(3)}(x) \\ &= \sum_{j_1=1}^{R_1} w1_{j_1} \left[\prod_{i_1=1}^{N_1} u1_{i_1 j_1}^{(2)}(x) \right]^{1-\gamma 1_{j_1}+\gamma 1_{j_1}/N_1} \\ &= \sum_{j_1=1}^{R_1} w1_{j_1} \left[\prod_{i_1=1}^{N_1} \exp\left(-\frac{(h_{i_1 j_1}(x) - m1_{i_1 j_1})^2}{(\sigma 1_{i_1 j_1})^2} \right) \right]^{1-\gamma 1_{j_1}+\gamma 1_{j_1}/N_1}, \end{aligned} \tag{A.4}$$

$$\begin{aligned} f_2(x) &= \sum_{j_2=1}^{R_2} w2_{j_2} u2_{j_2}^{(3)}(x) \\ &= \sum_{j_2=1}^{R_2} w2_{j_2} \left[\prod_{i_2=1}^{N_2} u2_{i_2 j_2}^{(2)}(x) \right]^{1-\gamma 2_{j_2}+\gamma 2_{j_2}/N_2} \\ &= \sum_{j_2=1}^{R_2} w2_{j_2} \left[\prod_{i_2=1}^{N_2} \exp\left(-\frac{(h_{i_2 j_2}(x) - m2_{i_2 j_2})^2}{(\sigma 2_{i_2 j_2})^2} \right) \right]^{1-\gamma 2_{j_2}+\gamma 2_{j_2}/N_2}, \end{aligned} \tag{A.5}$$

where $w1_j$ and $w2_j \in \Re$, $\forall j$, and h_{ij} is a time sequence of x_i, for $i = 1, 2, \ldots, N$. That is,

$$\begin{aligned} h_{ij}(1) &= x_i(1), \\ &\vdots \\ h_{ij}(t) &= x_i(t) + u_{ij}^{(2)}(t-1)\theta, \\ &= x_i(t) + \exp\left[-\frac{(h_{ij}(t-1) - m_{ij})^2}{(\sigma_{ij})^2} \right] \theta_{ij}. \end{aligned}$$

Therefore, we have

$$f_1(x) + f_2(x) = \sum_{j_1=1}^{R_1} w1_{j_1} u1_{j_1}^{(3)}(x) + \sum_{j_2=1}^{R_2} w2_{j_2} u2_{j_2}^{(3)}(x) \tag{A.6}$$

Since $u1_j^{(3)}$ and $u2_j^{(3)}$ are Gaussian in form (this can be verified by straightforward algebraic operations), then Eq. (A.6) is in the same form as Eq. (A.1), so that $f_1 + f_2 \in Y$. Similarly, we have

$$f_1(x) f_2(x) = \sum_{j_1=1}^{R_1} w1_{j_1} u1_{j_1}^{(3)}(x) \sum_{j_2=1}^{R_2} w2_{j_2} u2_{j_2}^{(3)}(x) \tag{A.7}$$

which is also in the same form as Eq. (A.1). Hence, $f_1 f_2 \in Y$. Finally, for arbitrary $c \in \Re$

$$cf_1(x) = \sum_{j_1=1}^{R_1} cw1_{j_1} u1_{j_1}^{(3)}(x) \tag{A.8}$$

which is again in the form of Eq. (A.1). Hence, $cf_1 \in Y$. Therefore, Y is an algebra.

Next, we prove that Y separates points in U. We prove this by constructing a required f. That is, we specify $f \in Y$ such that $f(\underline{x}') \neq f(\underline{y}')$ for an arbitrarily given $\underline{x}', \underline{y}' \in U$, with $\underline{x}' \neq \underline{y}'$. We choose two fuzzy rules in the form of Eq. (9) for the fuzzy rule base. Let $\underline{x}' = (x_1', x_2', \ldots, x_N')$ and $\underline{y}' = (y_1', y_2', \ldots, y_N')$. If $x_i' \neq y_i'$, we can choose two fuzzy rules as the fuzzy rule base. Furthermore, let the Gaussian membership functions be

$$\begin{aligned}\mu_{A_{i1}}(x_i) &= \exp\left(-\frac{(h_{i1} - x_i')^2}{\sigma^2}\right) \\ &= \exp\left(-\frac{(x_i + u_{i1}^{(2)}\theta_{i1} - x_i')^2}{\sigma^2}\right),\end{aligned} \tag{A.9}$$

$$\begin{aligned}\mu_{A_{i2}}(x_i) &= \exp\left(-\frac{(h_{i2} - y_i')^2}{\sigma^2}\right) \\ &= \exp\left(-\frac{(x_i + u_{i2}^{(2)}\theta_{i2} - y_i')^2}{\sigma^2}\right).\end{aligned} \tag{A.10}$$

Then f can be expressed as

$$\begin{aligned}f = w_1 &\left[\prod_{i=1}^{N} \exp\left(-\frac{(x_i + u_{i1}^{(2)}\theta_{i1} - x_i')^2}{\sigma^2}\right)\right]^{1-\gamma_1+\gamma_1/N} \\ &+ w_2 \left[\prod_{i=1}^{N} \exp\left(-\frac{(x_i + u_{i2}^{(2)}\theta_{i2} - y_i')^2}{\sigma^2}\right)\right]^{1-\gamma_2+\gamma_2/N},\end{aligned} \tag{A.11}$$

where w_1, w_2 are the link weights. With this system, we have

$$\begin{aligned}f(\underline{x}') = w_1 &\left[\prod_{i=1}^{N} \exp\left(-\frac{(u_{i1}^{(2)}\theta_{i1})^2}{\sigma^2}\right)\right]^{1-\gamma_1+\gamma_1/N} \\ &+ w_2 \left[\prod_{i=1}^{N} \exp\left(-\frac{(x_i' + u_{i2}^{(2)}\theta_{i2} - y_i')}{\sigma^2}\right)\right]^{1-\gamma_2+\gamma_2/N},\end{aligned} \tag{A.12}$$

$$\begin{aligned}f(\underline{y}') = w_1 &\left[\prod_{i=1}^{N} \exp\left(-\frac{(y_i' + u_{i1}^{(2)}\theta_{i1} - x_i')}{\sigma^2}\right)\right]^{1-\gamma_1+\gamma_1/N} \\ &+ w_2 \left[\prod_{i=1}^{N} \exp\left(-\frac{(u_{i2}^{(2)}\theta_{i2})}{\sigma^2}\right)\right]^{1-\gamma_2+\gamma_2/N}.\end{aligned} \tag{A.13}$$

Since $\underline{x}' \neq \underline{y}'$, there must be some i such that $x_i' \neq y_i'$. Hence, $f(\underline{x}') \neq f(\underline{y}')$. Therefore, Y separates points in U.

Finally, we prove that Y vanishes at no point of U. By Eq. (A.1), $u_j^{(3)}$ is constant and not equal to zero. That is, for all $x \in \Re^N$, $u_j^{(3)}(x) > 0$. If we choose $w_j > 0$ ($j = 1, 2, \ldots, R$), then $y > 0$ for any $x \in \Re^N$. That is, any $y \in Y$ with $w_j > 0$ can serve as the required f.

In summary, the RCNFS is a universal approximator. Using the *Stone–Weierstrass theorem* and the fact that Y is a set of real, continuous in U, we have proven the theorem. □

References

[1] C.T. Chao, T.J. Chen, C.C. Teng, Simplification of fuzzy-neural systems using similarity analysis, IEEE Trans. Systems Man Cybernet. 26 (1996) 344–354.

[2] G. Chen, Y. Chen, H. Ogmen, Identifying chaotic system via a Wiener-type cascade model, IEEE Trans. Control System 17 (5) (1997) 29–36.

[3] J.L. Elman, Finding structure in time, Cognitive Sci. 14 (1990) 179–211.

[4] C.F. Juang, A TSK-type recurrent fuzzy network for dynamic systems processing by neural network and genetic algorithms, IEEE Trans. Fuzzy Systems 10 (2) (2002) 155–170.

[5] C.F. Juang, C.T. Lin, An on-line self-constructing neural fuzzy inference network and its applications, IEEE Trans. Fuzzy Systems 6 (1) (1998) 12–31.

[6] C.F. Juang, C.T. Lin, A recurrent self-organizing neural fuzzy inference network, IEEE Trans. Neural Networks 10 (4) (1999) 828–845.

[7] J.H. Kim, D.T. College, A. Gun, G. Do, Fuzzy model based predictive control, Proceedings of the IEEE International Conference on Fuzzy Systems, 1998, pp. 405–409.

[8] C.H. Lee, C.C. Teng, Identification and control of dynamic systems using recurrent fuzzy neural networks, IEEE Trans. Fuzzy Systems 8 (4) (2000) 349–366.

[9] C.J. Lin, C.H. Chen, Nonlinear system control using compensatory neuro-fuzzy networks, IEICE Trans. Fundam. Electron. Comm. Comput. Sci. E 86-A (9) (2003) 2309–2316.

[10] C.J. Lin, W.H. Ho, A pseudo-Gaussian-based compensatory neural fuzzy system, IEEE International Conference on Fuzzy Systems, 2003, pp. 214–220.

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

[12] C.J. Lin, C.T. Lin, An ART-based fuzzy adaptive learning control network, IEEE Trans. Fuzzy Systems 5 (4) (1997) 477–496.

[13] P.A. Mastorocostas, J.B. Theocharis, A recurrent fuzzy-neural model for dynamic system identification, IEEE Trans. Systems Man Cybernet. 32 (2) (2002) 176–190.

[14] J. Moody, C.J. Darken, Fast learning in networks of locally-tuned processing units, Neural Comput. 1 (1989) 281–294.

[15] K.S. Narendra, K. Parthasarathy, Identification and control of dynamical systems using neural networks, IEEE Trans. Neural Networks 1 (1990) 4–27.

[16] C.S. Ouyang, S.J. Lee, An improved learning algorithm for rule refinement in neuro-fuzzy modeling, Third International Conference Knowledge-Based Intelligent Information Engineering Systems, 1999, pp. 238–241.

[17] S. Paul, S. Kumar, Subsethood-product fuzzy neural inference system (SuPFuNIS), IEEE Trans. Neural Networks 13 (3) (2002) 579–599.

[18] W. Rudin, Principles of Mathematical Analysis, 3rd ed., McGraw-Hill, New York, 1976.

[19] S. Santini, A.D. Bimbo, R. Jain, Block-structured recurrent neural networks, Neural Networks 8 (1) (1995) 135–147.

[20] H. Seker, D.E. Evans, N. Aydin, E. Yazgan, Compensatory fuzzy neural networks-based intelligent detection of abnormal neonatal cerebral doppler ultrasound waveforms, IEEE Trans. Inform. Technol. Biomed. 5 (3) (2001) 187–194.

[21] T. Takagi, M. Sugeno, Fuzzy identification of systems and its applications to modeling and control, IEEE Trans. Systems Man Cybernet. 15 (1985) 116–132.

[22] L.X. Wang, Adaptive Fuzzy Systems and Control, Prentice-Hall, Englewood Cliffs, NJ, 1994.
[23] P. Werbos, Beyond regression: new tools for prediction and analysis in the behavior sciences, Ph.D. Dissertation, Harvard University, Cambridge, MA, 1974.
[24] J. Zhang, A.J. Morris, Recurrent neuro-fuzzy networks for nonlinear process modeling, IEEE Trans. Neural Networks 10 (2) (1999) 313–326.
[25] Y.Q. Zhang, A. Kandel, Compensatory neurofuzzy systems with fast learning algorithms, IEEE Trans. Neural Networks 9 (1) (1998) 83–105.
[26] H.J. Zimmermann, P. Zysno, Latent connective in human decision, Fuzzy Sets and Systems 4 (1980) 31–51.