Dynamic System Identification Using a Recurrent Compensatory Fuzzy Neural Network

Chi-Yung Lee, Cheng-Jian Lin*, Cheng-Hung Chen, and Chun-Lung Chang

Abstract: This study presents a recurrent compensatory fuzzy neural network (RCFNN) for dynamic system identification. The proposed RCFNN uses a compensatory fuzzy reasoning method, and has feedback connections added to the rule layer of the RCFNN. The compensatory fuzzy reasoning method can make the fuzzy logic system more effective, and the additional feedback connections can solve temporal problems as well. Moreover, an online learning algorithm is demonstrated to automatically construct the RCFNN. The RCFNN initially contains no rules. The rules are created and adapted as online learning proceeds via simultaneous structure and parameter learning. Structure learning is based on the measure of degree and parameter learning is based on the gradient descent algorithm. The simulation results from identifying dynamic systems demonstrate that the convergence speed of the proposed method exceeds that of conventional methods. Moreover, the number of adjustable parameters of the proposed method is less than the other recurrent methods.

Keywords: Chaotic, compensatory operator, fuzzy neural networks, identification, recurrent networks.

1. INTRODUCTION

In artificial intelligence, fuzzy models can be implemented using fuzzy neural networks. Fuzzy neural networks have emerged as a highly effective approach to solving many engineering problems [1-5]. However, fuzzy neural networks are restricted in that their application domain is limited to static problems owing to their internal feedforward network structure, causing inefficiency for temporal problems. Consequently, a recurrent fuzzy neural network must be developed to solve temporal problems.

For a dynamic system, the output is a function of past inputs or past outputs or both; identification of dynamic systems is less direct than for static systems. To address temporal problems involving dynamic systems, the conventionally adopted model is a neural network [6] or a fuzzy neural network [2-5]. If a feedforward network is adopted for this task, the number of delayed inputs and outputs should be known in advance. However, this approach is limited in that the precise order of the dynamic system is generally unknown. Recurrent networks for processing dynamic systems can be used to solve this problem. These networks have received increasing interest in recent years [7-9]. Juang and Lin [7] proposed a recurrent self-organizing neural fuzzy inference network (RSONFIN), which uses Mamdani-type fuzzy rule, with online supervised learning ability. Lin and Wai [8] applied the recurrent-fuzzy -neural network (RFNN), which involves dynamic elements in the form of feedback connections as internal memories. A TSK-type recurrent fuzzy network is proposed for a supervised learning environment (TRFN-S) with available gradient information by Juang [9]. However, recurrent networks deal with fuzzy membership functions and defuzzification schemes for applications by using learning algorithms to adjust the parameters of fuzzy membership functions and defuzzification functions. Unfortunately, for optimal fuzzy logic rea-

Manuscript received January 15, 2007; revised January 21, 2008; accepted April 18, 2008. Recommended by Editorial Board member Sung-Kwun Oh under the direction of Editor Young-Hoon Joo. This work was supported in part by the National Science Council, Taiwan, R.O.C., under Grant NSC 95-2221-E-390-040-MY2, and in part by Study of Intelligence Fuzzy Controller in cooperation with the Mechanical and Systems Research Laboratories, Industrial Technology Research Institute, Taiwan, R.O.C., under Grant 6301XS3210.

Chi-Yung Lee is with Dept. of Computer Science and Information Engineering, Nankai Institute of Technology, Nantou, Taiwan 542, R.O.C. (e-mail: cylee@nkc.edu.tw).

Cheng-Jian Lin is with the Dept. of Computer Science and Engineering, National Chin-Yi University of Technology, Taichung County, Taiwan 411, R.O.C. (e-mail: cjlin@nuk.edu.tw).

Cheng-Hung Chen is with the Dept. of Electrical and Control Engineering, National Chiao-Tung University, Hsinchu, Taiwan 300, R.O.C. (e-mail: chchen.ece93g@nctu.edu.tw).

Chun-Lung Chang is with the Mechanical and Systems Research Laboratories, Industrial Technology Research Institute, Hsinchu, Taiwan 310, R.O.C. (e-mail: VincentChang@itri.org.tw).

* Corresponding author.

soning and fuzzy operators, only static fuzzy operators are widely used for fuzzy reasoning. Because the conventional fuzzy neural network can only adjust fuzzy membership functions by using fixed fuzzy operations, for example Min and Max, intuitively it is not adaptive for a complete fuzzy system to employ an unchangeable pair of fuzzy operators. Zimmermann [10] was the first to define the essence of compensatory operations. Zhang and Kandel [11] proposed more extensive compensatory operations based on the pessimistic and optimistic operations. Since the compensatory parameters in a fuzzy neural network are physically meaningful, these compensatory parameters can be initialized using a heuristic algorithm to accelerate system training. The compensatory fuzzy neural network [14] with adjustable fuzzy reasoning is more effective than the conventional fuzzy neural network with nonadjustable fuzzy reasoning [3]. Recently, many researchers [12-15] have successfully applied the compensatory operation to fuzzy systems. Therefore, this study adopts a compensatory fuzzy neural network that can not only adaptively adjust fuzzy membership functions but also dynamically adjust the compensatory operators.

This study presents a recurrent compensatory fuzzy neural network (RCFNN). The RCFNN is a recurrent multilayer connectionist network for fuzzy reasoning and can be constructed from a set of fuzzy rules. Simultaneously, the compensatory fuzzy inference method uses adaptive fuzzy reasoning of fuzzy neural networks to increase the adaptability of the fuzzy logic system. An on-line learning algorithm is also proposed to automatically construct the RCFNN. The proposed learning algorithm includes structure learning and parameter learning. The structure learning algorithm determines whether to add a new node to satisfy the fuzzy partition of the input data. Moreover, the gradient descent learning algorithm is used to tune the free parameters in the RCFNN to minimize an output cost function.

The proposed RCFNN model possesses four advantages. First, it does not require assistance from a human expert, and its structure is obtained from the input data. Second, it adopts compensatory operators. Because the conventional fuzzy neural network can only adjust fuzzy membership functions by using fixed fuzzy operations such as, *MIN* and *MAX*, the RCFNN model with adjustable fuzzy reasoning is more effective than the conventional fuzzy neural network. Third, it converges rapidly, and requires few tuning parameters. Fourth, it can effectively address temporal problems when identifying a dynamic system.

2. COMPENSATORY OPERATION

Zhang and Kandel [11] recently proposed compensatory operations based on the pessimistic and optimistic operations. The pessimistic operation could map the inputs x_i to the pessimistic output by making a conservative decision regarding the pessimistic situation or for even the worst case situation. For example, $p(x_1, x_2, ..., x_N)=MIN(x_1, x_2, ..., x_N)$ or $\Pi\, x_i$. In fact, the *t*-norm fuzzy operation is a pessimistic operation.

The optimistic operation can map the inputs xi to the optimistic output by making an optimistic decision regarding the optimistic situation or even the best case. For example, $o(x_1, x_2, ..., x_N)=MAX(x_1, x_2, ..., x_N)$. In fact, the *t*-conorm fuzzy operation is an optimistic operation. The compensatory operation can map the pessimistic input x1 and the optimistic input x_2 to make a relatively compromised decision for the situation between the worst and best cases. For example, $c(x_1, x_2) = x_1^{1-\gamma} x_2^{\gamma}$, where $\gamma \in [0,1]$ is called the compensatory degree.

The general fuzzy if-then rule is as follows:

$$\begin{aligned} Rule-j: \text{IF } & x_1 \text{ is } A_{1j} \text{ and } ..... \text{ and } x_N \text{ is } A_{Nj} \\ & \text{THEN } y \text{ is } w_j, \end{aligned} \tag{1}$$

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

For an input fuzzy set $A_j^{'}$ in $U \subset R$, the jth fuzzy rule (1) can generate an output fuzzy set $w_j^{'}$ in $v \subset R$ 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 $\underline{x} = (x_1, x_2, ..., x_N)$. $\mu_{A_{1j} \times \cdots \times A_{Nj}}(\underline{x})$ is defined using 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 (4) and the optimistic operation (5) are as follows:

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

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

For simplicity, the above can be rewritten as

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

Since $\mu_{A'}(x_i)=1$ for the singleton fuzzifier and $\mu_{b_j'}(y)=1$, then according to (2):

$$\mu_{b_j'}(y) = [\prod_{i=1}^{N}\mu_{A_{ij}}(x_i)]^{1-\gamma_j+\gamma_j/N} \quad (7)$$

Therefore, the fuzzy if-then rule with compensatory degree γ can be rewritten as follows:

$$\begin{aligned} Rule-j: [&\text{IF } x_1 \text{ is } A_{1j} \text{ and } \ldots\ldots \text{ and } x_N \text{ is } A_{Nj}\ ]^{\gamma_j} \\ &\text{THEN } \ y \text{ is } w_j. \end{aligned} \quad (8)$$

3. STRUCTURE OF THE RECURRENT COMPENSATORY FUZZY NEURAL NETWORK

This section describes the proposed recurrent compensatory fuzzy neural network (RCFNN). Recently, Lin and Wai [8] proposed a model, called the recurrent fuzzy neural network (RFNN) architecture. The model presented here resembles the RFNN with the only difference being in the rule layer. Layer three of the RFNN uses the product operator, while layer three of the proposed model uses the compensatory operator.

The structure of the proposed RCFNN is illustrated in Fig. 1, and is systematized into N input variables, R-term nodes for each input variable, M output nodes, and $N\times R$ membership function nodes. The RCFNN comprises four layers and $R\times(N\times2+3+M)$ parameters, where R denotes the number of existing rules. Nodes in layer 1 are input nodes, which represent input variables. Moreover, nodes in layer 2 are called linguistic nodes and act as membership functions to express the input fuzzy linguistic variables. Nodes in this layer are used to calculate the Gaussian membership values. Each node in layer 3 is called a compensatory rule node. Furthermore, nodes in layer 3 equal the number of compensatory fuzzy sets corresponding to each external linguistic input variable. Links before layer 3 represent the rule preconditions, and links after layer 3 represent the consequences of the rule nodes. Nodes in layer 4 are called output nodes, where each node represents an individual system output. The links between layers 3 and 4 are connected via the weighting values w_j. Each node in the feedback layer is embedded in the RCFNN by adding feedback connections to layer 3.

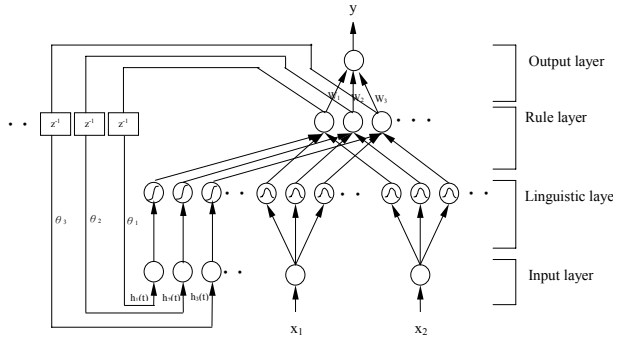


Fig. 1. Structure of the proposed RCFNN.

The structure of the RCFNN is shown in Fig. 1, in which A_{ij} by a Gaussian-type membership function, $\mu_{A_{ij}}(x_i)$, defined by

$$\mu_{A_{ij}}(x_i) = \exp\left\{-\frac{[x_i - m_{ij}]^2}{\sigma_{ij}^{\ 2}}\right\}, \quad (9)$$

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. Defining the number and the locations of the membership functions leads to a partition of the premise space $\aleph = \aleph_1 \times \ldots \times \aleph_N$.

The collection of fuzzy sets $\mathbf{A}_j = \{A_{1j},\ldots,A_{Nj}\}$ pertaining to the premise part of *Rule-j* forms a fuzzy region A_j that can be regarded as a multi-dimensional fuzzy set whose membership function is determined by

$$\mu_{A_j}(\underline{x}(t)) = \prod_{i=1}^{N}\mu_{A_{ij}}(x_i(t)). \quad (10)$$

The above equation provides the degree to which a particular input vector $x(t)$ belongs to the fuzzy region A_j. For the internal variable s_j, the following sigmoid function is used:

$$s_j = \frac{1}{1+\exp\{-h_j\}}, \quad (11)$$

where $h_j = \mu_{A_j}(\underline{x}(t-1))\cdot\theta_j$ is the feedback units acting as memory elements, and θ_j is the feedback weight. Furthermore, due to the compensatory operation of the grades of the membership functions $\mu_{A_{ij}}(x_i(t))$ in (10), for simplicity, we can rewrite it as:

$$\mu_{A_j}(\underline{x}(t)) = [\prod_{i=1}^{N}\mu_{A_{ij}}(x_i)\cdot s_j]^{1-\gamma_j+\gamma_j/N} \quad (12)$$

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

$$\begin{aligned} Rule-j: [\text{IF } & x_1(t) \text{ is } A_{1j} \text{ and .....} \\ & \text{and } x_N(t) \text{ is } A_{Nj} \text{ and } h_j(t) \text{ is } G]^{\gamma_j} \\ \text{THEN } & y'(t+1) \text{ is } w_j \text{ and } h_j(t+1) \text{ is } \theta_j, \end{aligned} \tag{13}$$

where *Rule-j* denotes the *j*th fuzzy rule, $x(t)=[x_1(t),x_2(t),\ldots,x_N(t)]^T$ is the input vector to the model at time t with $x_i(t) \in \aleph_i \subset \Re(i=1,...,N)$, $h_j(t)$ is the internal variable at time *t*, *y'*(t+1) is the jth output of the local model for Rule-j, A_{ij} and G are fuzzy sets, w_j is the consequent part of *Rule-j*, and θ_j is the fuzzy singleton. The output of the model at time *t* is $y(t)$ and is labeled $\sum$. It is the sum of all incoming signals:

$$y(t) = \sum_{j=1}^{R} \mu_{A_j}(x(t)) \cdot w_j, \tag{14}$$

where the weight w_j is the output action strength of the output associated with the *j*th rule.

Because (14) has a similar form to [16], which is a universal approximator, it is easy to prove the universal approximation theorem of (14), as follows.

For any real continuous function g in a compact set $U \subset \Re^N$ and for any given arbitrary $\varepsilon > 0$, a model f exists such that

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

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

To summarize, the overall representation of the input *x* and the output *y* is

$$y(t) = \sum_{j=1}^{R} w_j \cdot \left\{ \frac{\prod_{i=1}^{N} \exp\left[-\frac{(x_i(t) - m_{ij})^2}{\sigma_{ij}^{\,2}} \right]}{1 + \exp[-u_j^{(3)}(x_i(t-1)) \cdot \theta_j]} \right\}^{1-\gamma_j + \gamma_j / N}, \tag{15}$$

where $\gamma_j = c_j^{\,2} / \left(c_j^{\,2} + d_j^{\,2} \right)$ is the compensatory degree, m_{ij}, σ_{ij}, θ_j, c_j, d_j, and w_j are the tuning parameters, and

$$u_j^{(3)}(\underline{x}(t-1)) = \left\{ \frac{\prod_{i=1}^{N} \exp\left[-\frac{(x_i(t-1) - m_{ij})^2}{\sigma_{ij}^{\,2}} \right]}{1 + \exp[-u_j^{(3)}(x_i(t-2)) \cdot \theta_j]} \right\}^{1-\gamma_j + \gamma_j / N}. \tag{16}$$

Explicitly, using the RCFNN, the same inputs yield different outputs at different times. Obviously, (16) can be demonstrated to be a universal approximator by using the same method in [16].

4. LEARNING ALGORITHMS OF RCFNN

This section describes an on-line learning algorithm for constructing the RCFNN. The learning algorithm comprises both structure and parameter learning phases. Fig. 2 illustrates the flow diagram of the learning scheme for the RCFNN model. The structure learning algorithm is based on the measure of degree to determine the number of fuzzy rules. The parameter learning is based on supervised learning algorithms. Meanwhile, the gradient descent algorithm minimizes a given cost function by adjusting the weights in the consequent part, the weights of the feedback, the compensatory degree, and the parameters of the membership functions. Initially, the network contains no node except the input-output nodes; that is, there are no rules and memberships. The rule nodes are created dynamically and automatically as learning proceeds when online incoming training data are received and when the structure and parameter learning processes are performed. The details of the structure and parameter learning phases are described in the rest of this section.

4.1. Structure learning

Structure learning initially attempts to determine whether to extract a new rule from the training data, and also 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 center and width, respectively, of the Gaussian membership function. For each incoming pattern *x*, the strength of rule firing can be interpreted as the degree to which the incoming pattern belongs to the corresponding cluster. To improve computational efficiency, a compensatory operator of the firing strength obtained from $u_j^{(3)}(x(t))$ can be directly used as the degree measure

$$F_j = u_j^{(3)}(x(t)) = \left[\left[\prod_{i=1}^{n} u_{ij}^{(2)}(x_i) \right] \cdot s_j \right]^{1-\gamma_j + \gamma_j / N}, \tag{17}$$

where $F_j \in [0,1]$. Using this degree measure, the following criterion can be obtained for generating a new fuzzy rule for new incoming data, described as follows. Find the maximum degree F_{max}

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

where $R(t)$ is the number of existing rules at time t. If $F_{\max} \le \overline{F}$, then a new rule is generated, where $\overline{F} \in (0,1)$ is a prespecified threshold that should decay when the learning process limits the size of the RCFNN. We adopt the decay method $(1-i/N)\overline{F}$, where N is the prespecified number of training epoch and i is ith training epoch when the learning process. The threshold parameter $\overline{F}$ is an important parameter in the structure learning step. The threshold value is set to between zero and one. A low threshold value leads to the learning of coarse clusters (that is, less rules are generated), whereas a high threshold value leads to the learning of fine clusters (that is, more rules are generated). If the threshold value equals zero, all the training data belong to the same cluster in the input space. After generating a new rule, the next step is to assign initial center and width according to (19) and (20) for the new Gaussian membership function. Since the goal of this study is to minimize an objective function, and since the center and width of Gaussian membership function are all adjustable later in the parameter learning, the center and width for the new Gaussian membership function are set as follows:

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

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

where x_i denotes the new data and σ_{init} represents a prespecified constant. Therefore, the selection of the threshold value $\overline{F}$ and the prespecified constant σ_{init} critically influence the simulation results, and the threshold value is based on practical experimentation or trial-and-error testing.

Since the generation of a membership function corresponds to the generation of a new fuzzy rule, the compensatory degree $c_j^{(R_{(t+1)})}$, $d_j^{(R_{(t+1)})}$, namely, $\gamma_j^{(R_{(t+1)})} = (c_j^{(R_{(t+1)})})^2 / (c_j^{(R_{(t+1)})})^2 + (d_j^{(R_{(t+1)})})^2$, the weight of the feedback $\theta_j^{(R_{(t+1)})}$, and the weight of the link $w_j^{(R_{(t+1)})}$ associated with a new fuzzy rule have to be determined. Generally, the compensatory degree $c_j^{(R_{(t+1)})}, d_j^{(R_{(t+1)})}$, the weight of the feedback $\theta_j^{(R_{(t+1)})}$, and the weight of the link $w_j^{(R_{(t+1)})}$ are selected with random values in [-1,1].

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

Step 1: IF x_i is the first incoming pattern THEN do
{Generate a new rule
with center $m_{i1}=x_i$, width $\sigma_{i1}=\sigma_{init}$, compensatory degree c_1=random, d_1=random, weight of feedback $\theta_1 = random$, weight of link w_1=random
where σ_{init} is a prespecified constant.}

Step 2: ELSE for each newly incoming x_i, do
{Find $F_{\max} = \max_{1 \le j \le R_{(t)}} F_j$

IF $F_{\max} \ge \overline{F}$
do nothing
ELSE
{
$R(t+1) = R(t) + 1$
generate a new rule
with center $m_{ij}^{R_{(t+1)}} = x_i$, width
$\sigma_{ij}^{R_{(t+1)}} = \sigma_{init}$,
compensatory degree $c_j^{R_{(t+1)}} = random$,
$d_j^{R_{(t+1)}} = random$, weight of feedback
$\theta_j^{R_{(t+1)}} = random$, weight of link
$w_j^{R_{(t+1)}} = random$
where σ_{init} is a prespecified constant.}
}

4.2. Parameter learning body

After adjusting the network structure based on the current training pattern, the network then enters the parameter learning phase to optimize the network parameters based on the same training pattern. Notice that the following parameter learning is performed on the whole network after structure learning, regardless of whether the nodes (links) are newly added or originally existed. Since the learning process involves the determination of the vector which minimizes a given cost function, the gradient of the cost function with respect to the vector is calculated, and the vector is adjusted along the negative gradient. When the single output case is considered for clarity, the goal is to minimize the cost function $E(t)$, which is defined as

$$E(t) = \frac{1}{2}[y^d(t) - y(t)]^2, \tag{21}$$

where $y^d(t)$ denotes the desired output and $y(t)$ represents the current output for each discrete time t. During each training cycle, beginning at the input variables, a forward pass is used to calculate the activity of all the current outputs $y(t)$.

When the gradient descent learning algorithm is used, the weighting vector of the RCFNN is adjusted such that the error defined in (21) is below the desired threshold value after a given number of training cycles. The well-known gradient descent learning algorithm can be written briefly as

$$W(t+1) = W(t) + \Delta W(t) = W(t) + \eta\left(-\frac{\partial E(t)}{\partial W}\right), \tag{22}$$

where, in this case, η and W represent the learning rate and tuning parameters of the RCFNN, respectively. Let $e(t) = y(t)^d - y(t)$ and $W = [m, \sigma, \theta, c, d, w]^T$ represent the training error and weighting vector of the RCFNN, respectively. The gradient of error $E(.)$ in (21) with respect to an arbitrary weighting vector W then is

$$\frac{\partial E(t)}{\partial W} = -e(t)\frac{\partial y(t)}{\partial W}. \tag{23}$$

The error term for each layer is first computed to form recursive applications of the chain rule. The parameters in the corresponding layers then are adjusted. With the RCFNN and the cost function as defined in (21), the update rule for w_j, c_j, d_j, and θ_j can be derived, as follows:

$$w_j(t+1) = w_j(t) - \eta_w \frac{\partial E(t)}{\partial w_j}, \tag{24}$$

$$c_j(t+1) = c_j(t) - \eta_c \frac{\partial E(t)}{\partial c_j}, \tag{25}$$

$$d_j(t+1) = d_j(t) - \eta_d \frac{\partial E(t)}{\partial d_j}, \tag{26}$$

$$\theta_j(t+1) = \theta_j(t) - \eta_\theta \frac{\partial E(t)}{\partial \theta_j}, \tag{27}$$

where

$$\frac{\partial E(t)}{\partial w_j} = -e(t)\cdot u_j^{(3)}(x(t)),$$

$$\frac{\partial E(t)}{\partial c_j} = -e(t)\cdot w_j \cdot u_j^{(3)}(x(t)) \cdot \ln\left[\prod_{i=1}^{N} u_{ij}^{(2)}(x_i)\cdot s_j\right]\cdot\left(\frac{1}{N}-1\right)\cdot\left[\frac{2c(t)d^2(t)}{(c^2(t)+d^2(t))^2}\right],$$

$$\frac{\partial E(t)}{\partial d_j} = -e(t)\cdot w_j \cdot u_j^{(3)}(x(t)) \cdot \ln\left[\prod_{i=1}^{N} u_{ij}^{(2)}(x_i)\cdot s_j\right]\cdot\left(\frac{1}{N}-1\right)\cdot\left[-\frac{2c^2(t)d(t)}{(c^2(t)+d^2(t))^2}\right],$$

$$\frac{\partial E(t)}{\partial \theta_j} = -e(t)\cdot\frac{\partial y}{\partial u_j^{(3)}}\cdot\frac{\partial u_j^{(3)}}{\partial \theta_j} = -e(t)\cdot w_j \cdot \frac{\partial u_j^{(3)}}{\partial \theta_j},$$

where

$$\frac{\partial u_j^{(3)}}{\partial \theta_j} = (1-\gamma_j + \frac{\gamma_j}{N}) \cdot\left[\left(\prod_{i=1}^{N} u_{ij}^{(2)}(x_i)\right)\cdot s_j\right]^{-\gamma_j+\gamma_j/N}\cdot\frac{\partial s_j}{\partial \theta_j}$$

$$= (1-\gamma_j+\frac{\gamma_j}{N}) \cdot\left[\left(\prod_{i=1}^{N} u_{ij}^{(2)}(x_i)\right)\cdot s_j\right]^{-\gamma_j+\gamma_j/N}\cdot s_j\cdot(1-s_j)\cdot\frac{\partial h_j}{\partial \theta_j}$$

and the partial derivative $\partial h_j / \partial \theta_j$ is calculated as

$$\frac{\partial h_j}{\partial \theta_j} = u_j^{(3)}(t-1) + \theta_j \cdot \frac{\partial s_j}{\partial \theta_j}(t-1)\cdot u_{ij}^{(2)}(t-1).$$

Hence, we have the following recursive form:

$$\frac{\partial s_j}{\partial \theta_j}(t) = s_j(t)\cdot(1-s_j(t))\cdot\left[u_j^{(3)}(t-1) + \theta_j\cdot u_{ij}^{(2)}(t-1)\cdot\frac{\partial s_j}{\partial \theta_j}(t-1)\right].$$

Similarly, the update laws for m_{ij} and σ_{ij} are

$$m_{ij}(t+1) = m_{ij}(t) - \eta_m \frac{\partial E(t)}{\partial m_{ij}}, \tag{28}$$

$$\sigma_{ij}(t+1) = \sigma_{ij}(t) - \eta_\sigma \frac{\partial E(t)}{\partial \sigma_{ij}}, \tag{29}$$

where

$$\frac{\partial E(t)}{\partial m_{ij}} = -e(t)\cdot\frac{\partial y}{\partial u_j^{(3)}}\cdot\frac{\partial u_j^{(3)}}{\partial m_{ij}} = -e(t)\cdot w_j\cdot\frac{\partial u_j^{(3)}}{\partial m_{ij}},$$

$$\frac{\partial E(t)}{\partial \sigma_{ij}} = -e(t)\cdot\frac{\partial y}{\partial u_j^{(3)}}\cdot\frac{\partial u_j^{(3)}}{\partial \sigma_{ij}} = -e(t)\cdot w_j\cdot\frac{\partial u_j^{(3)}}{\partial \sigma_{ij}},$$

where

$$\frac{\partial u_j^{(3)}}{\partial m_{ij}} = \frac{\partial}{\partial m_{ij}}\left\{\left[\prod_{i=1}^{N}\exp\left[-\frac{(x_i-m_{ij})^2}{\sigma_{ij}^2}\right]\right]\cdot s_j\right\}^{1-\gamma_j+\frac{\gamma_j}{N}}$$

$$= \left(1-\gamma_j+\frac{\gamma_j}{N}\right)\cdot\left[\left(\prod_{i=1}^{N} u_{ij}^{(2)}\right)\cdot s_j\right]^{-\gamma_j+\frac{\gamma_j}{N}} \cdot\left\{\left[\prod_{i=1}^{N} u_{ij}^{(2)}\right]\cdot\left[\frac{2\cdot(x_i-m_{ij})}{\sigma_{ij}^2}\right]\cdot s_j + \frac{\partial s_j}{\partial m_{ij}}\cdot\left[\prod_{i=1}^{N} u_{ij}^{(2)}\right]\right\}$$

and the partial derivative $\partial s_j / \partial m_{ij}$ is calculated as

$$\frac{\partial s_j}{\partial m_{ij}} = s_j \cdot \left(1 - s_j\right) \cdot \frac{\partial h_j}{\partial m_{ij}} = s_j \cdot \left(1 - s_j\right) \cdot \left\{ \theta_j \cdot \left[\frac{\partial s_j}{\partial m_{ij}} (t-1) \cdot u_{ij}^{(2)}(t-1) + s_j(t-1) \cdot u_{ij}^{(2)}(t-1) \cdot \left[\frac{2 \cdot \left(x_i - m_{ij}\right)}{\sigma_{ij}^2} \right] \right] \right\}$$

and

$$\frac{\partial u_j^{(3)}}{\partial \sigma_{ij}} = \frac{\partial}{\partial \sigma_{ij}} \left\{ \left[\prod_{i=1}^{N} \exp\left[-\frac{\left(x_i - m_{ij}\right)^2}{\sigma_{ij}^2} \right] \right] \cdot s_j \right\}^{1-\gamma_j + \frac{\gamma_j}{N}}$$

$$= \left(1 - \gamma_j + \frac{\gamma_j}{N}\right) \cdot \left[\left(\prod_{i=1}^{N} u_{ij}^{(2)} \right) \cdot s_j \right]^{-\gamma_j + \frac{\gamma_j}{N}} \cdot \left\{ \left[\prod_{i=1}^{N} u_{ij}^{(2)} \right] \cdot \left[\frac{2 \cdot \left(x_i - m_{ij}\right)^2}{\sigma_{ij}^3} \right] \cdot s_j + \frac{\partial s_j}{\partial \sigma_{ij}} \cdot \left[\prod_{i=1}^{N} u_{ij}^{(2)} \right] \right\}$$

and the partial derivative $\partial s_j / \partial \sigma_{ij}$ is

$$\frac{\partial s_j}{\partial \sigma_{ij}} = s_j \cdot \left(1 - s_j\right) \cdot \frac{\partial h_j}{\partial \sigma_{ij}} = s_j \cdot \left(1 - s_j\right) \cdot \left\{ \theta_j \cdot \left[\frac{\partial s_j}{\partial \sigma_{ij}} (t-1) \cdot u_{ij}^{(2)} + s_j(t-1) \cdot u_{ij}^{(2)}(t-1) \cdot \left[\frac{2 \cdot \left(x_i - m_{ij}\right)^2}{\sigma_{ij}^3} \right] \right] \right\}.$$

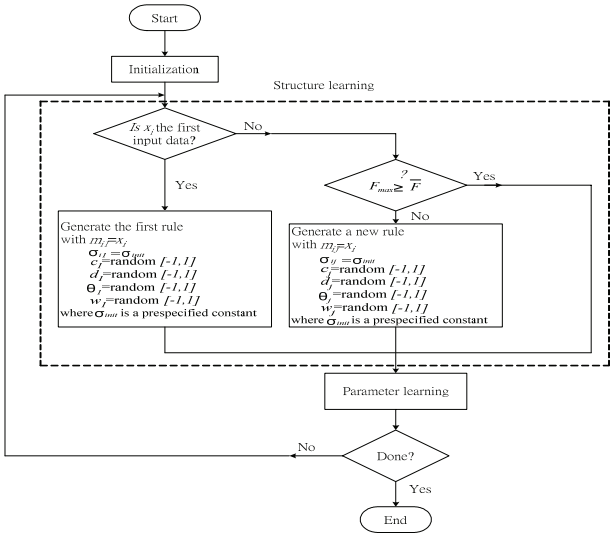


Fig. 2. Flow diagram of the structure/parameter learning for the RCFNN model.

5. SIMULATION RESULTS

This study evaluated the performance of the RCFNN for temporal problems. This section presents several examples and performance contrasts with some other recurrent fuzzy neural networks. The first problem involves identifying a nonlinear dynamic system, while the second problem involves identifying a chaotic system. In the following simulations, the parameters and number of training epochs were determined based on the desired result.

5.1. Identification of dynamic system

The systems to be identified are dynamic systems whose outputs are functions of past inputs and outputs. For this dynamic system identification, since a recurrent network is used, only the current states of the system and control signal are used as network inputs. The model is applied in the following two identification problems.

Example 1: In this example, a nonlinear plant with multiple time delays was guided using the following differential equation:

$$y_p(t+1) = f(y_p(t), y_p(t-1), y_p(t-2), u_p(t), u_p(t-1)), \tag{30}$$

where

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

In this study, the current output of the plant depends on three previous outputs and two previous inputs. In [6], the feedforward neural network, with five input nodes for feeding, the appropriate past values of y_p and u were used. In the proposed model, only two input values, $y_p(t)$ and $u(t)$, were fed to the RCFNN to determine the output $y_p(t+1)$. The training inputs were independent and had an identically distributed (i.i.d.) uniform sequence over [-2,2] for about half of the training time and a single sinusoid signal given by $1.05\sin(\pi t/45)$ for the remainder of the training time. These 900 training data were not repeated; that is, different training sets were used for each epoch. The test input signal $u(t)$, as shown in the equation below, was used to determine the identification results.

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

Ten epochs were used for training the RCFNN. The learning rate $\eta_w = \eta_c = \eta_d = \eta_m = \eta_\sigma = \eta_\Theta = 0.05$, the prespecified $\sigma_{init} = 0.2$, and the prespecified threshold $\overline{F} = 10^{-4}$ were chosen. After training, the final root-mean-square error (rms error) was 0.0011, and three fuzzy logic rules were generated. These designed three fuzzy IF-THEN rules of the recurrent compensatory fuzzy neural network with a compensatory degree γ are described below

Rule 1:

IF [$u(t)$ is μ(-0.0313,0.3149) and $y(t)$ is μ(0.1539, 0.7908) and $h(t)$ is $G]^{0.3724}$

THEN $y(t+1)$ is 0.4686 and $h(t+1)$ is 0.4686

Rule 2:

IF [$u(t)$ is μ(0.6041,0.3120) and $y(t)$ is μ(1.0319, 0.4989) and $h(t)$ is $G]^{0.7094}$

THEN $y(t+1)$ is 0.9018 and h($t+1$) is-0.5938

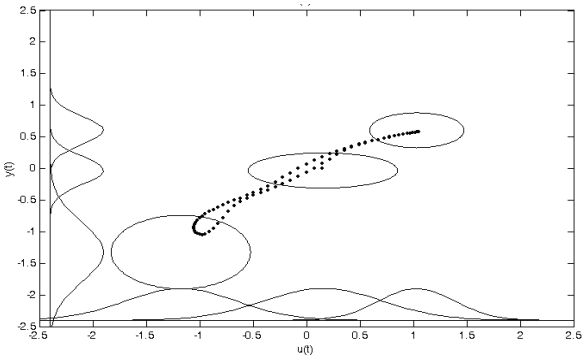
Rule 3:

IF [$u(t)$ is μ(-1.3234,0.6565) and $y(t)$ is μ(-1.1754, 0.7360) and $h(t)$ is $G]^{0.9352}$

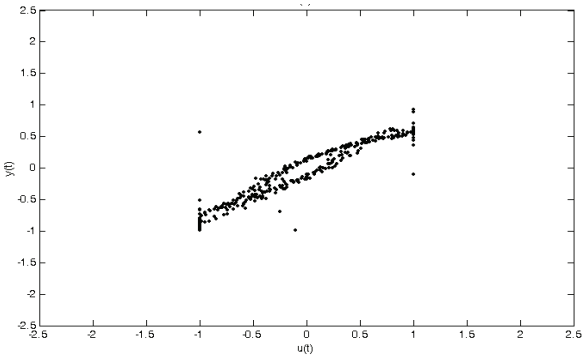
THEN $y(t+1)$ is -1.3784 and $h(t+1)$ is 0.0816

Fig. 3(a) shows the distribution of some of the training patterns and the final assignment of the rules in the $[u(t),y(t)]$ plane. This is due to the parameter learning process, which adjusts the center and width of each Gaussian membership function at each time step to minimize the output cost function. Fig. 3(a) shows the membership functions in the $u(t)$ and $y(t)$ dimensions. Moreover, Fig. 3(b) illustrates the distribution of the testing patterns on the $y(t)$ and $u(t)$ dimensions. Additionally, Fig. 3(c) shows results using the RCFNN for identification. The analytical results demonstrate that the RCFNN model can achieve perfect identification. Fig. 3(d) illustrates the error between the desired output and the RCFNN output. Moreover, Fig. 3(e) shows the learning curves of the RCFNN, RSONFIN [7], RFNN [8], and TRFN-S [9] models. In the figure, the proposed model converges faster than other models. In many practical cases, the plant being identified is noisy. Internal plant noise appears at the plant output and is commonly represented as an additive noise. In the present simulations, noise is white Gaussian distributed with $\sigma_e^2 = 0.1$. Fig. 4(a) plots the noise-corrupted signal. Moreover, Fig. 4(b) shows a trace of the recovered signal sequence.

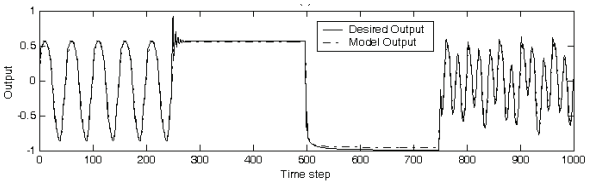
This study also compared the performance of the proposed model with that of other existing recurrent methods (RSONFIN [7], RFNN [8], TRFN-S [9]). The performance indices considered included the numbers of adjustable parameters, the rms error (training and testing), and the numbers of epochs. Table 1 lists the comparison results. As listed in Table 1, the numbers of adjustable parameters and the rms error of the RCFNN are smaller than other recurrent



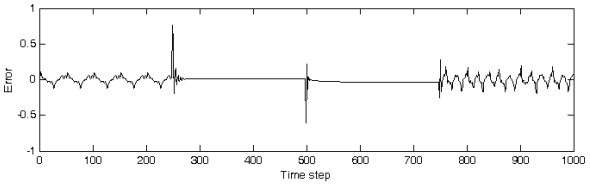
(a) The input training patterns and the final assignment of rules for the distribution of the membership functions on the $y(t)$ and $u(t)$ dimensions.



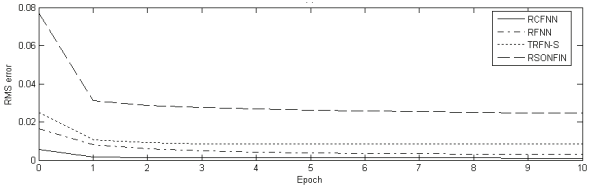
(b) The distribution of the testing patterns on the $y(t)$ and $u(t)$ dimensions.



(c) The outputs of the plant and the RCFNN.

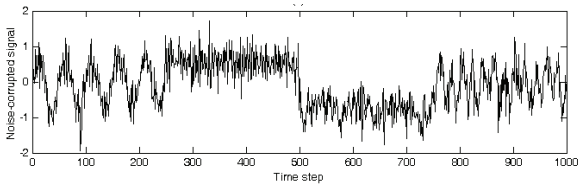


(d) The error between the RCFNN output and the desired output.

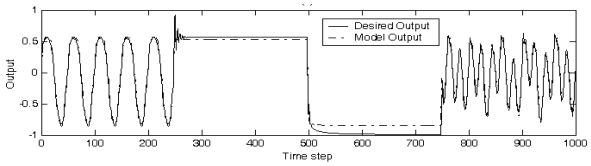


(e) The learning curves of the RCFNN, RSONFIN [7], the RFNN [8] and the TRFN-S [9].

Fig. 3. Simulation results of the RCFNN for dynamic system identification in Example1.



(a) The noise-corrupted signal.



(b) Output of the plant and the identification RCFNN for the same input.

Fig. 4. Simulation results in the noisy case of the RCFNN for dynamic system identification.

Table 1. Performance comparison of various recurrent methods with respect to the identification problem in Example 1.

	RCFNN	RSONFIN [7]	RFNN [8]	TRFN-S [9]
Parameters	24	36	24	33
RMS error (training)	0.0011	0.0248	0.0030	0.0084
RMS error (testing)	0.0019	0.0780	0.0033	0.0346
Epochs	10	10	10	10

models under the same training epochs.

Example 2: Next consider the following dynamic plant with longer input delays:

$$y_p(t+1) = 0.72y_p(t) + 0.025y_p(t-1)u(t-1) + 0.01u^2(t-2) + 0.2u(t-3). \quad (31)$$

This plant is the same as that used in [17]. The current output of the plant depends on two previous outputs and three previous inputs. As in example 1, the identification model, where only two external input values were fed to the input of RCFNN. The training data and time steps were the same as in Example 1. Ten epochs were used in training the RCFNN. The learning rate $\eta_w = \eta_c = \eta_d = \eta_m = \eta_\sigma = \eta_\Theta = 0.05$, the prespecified σ_{init} = 0.2, and the prespecified threshold $\overline{F} = 10^{-4}$ were chosen. After training, the final rms error was 0.0005, and three fuzzy logic rules were generated. These designed three rules with a compensatory degree are

Rule 1:

IF [$u(t)$ is $\mu(0.1786,0.4733)$ and $y(t)$ is $\mu(0.0115,0.2374)$ and $h(t)$ is $G]^{0.4192}$

THEN $y(t+1)$ is 0.4047 and $h(t+1)$ is 0.5621

Rule 2:

IF [$u(t)$ is $\mu(0.9473,-0.4273)$ and $y(t)$ is $\mu(1.0411,1.1604)$ and $h(t)$ is $G]^{0.9466}$

THEN $y(t+1)$ is 1.0166 and $h(t+1)$ is 0.4766

Rule 3:

IF [$u(t)$ is $\mu(-0.7305,0.3594)$ and $y(t)$ is $\mu(-0.8313,0.8858)$ and $h(t)$ is $G]^{0.9504}$

THEN $y(t+1)$ is –0.7550 and $h(t+1)$ is 0.7186

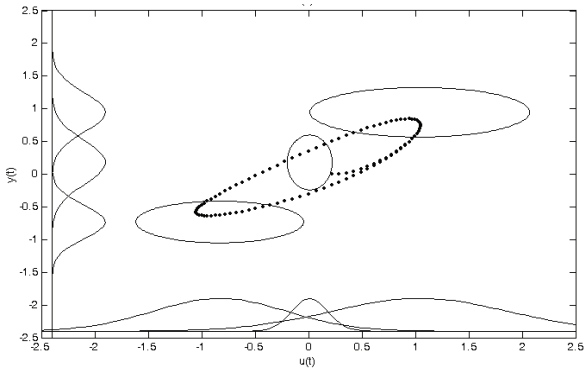
The check signal used in Example 1 is adopted for checking the identified result. Fig. 5(a) shows the distribution of input training patterns and the final task of the rules in the [$u(t)$,$y(t)$] plane. Fig. 5(a) shows the membership functions in the $u(t)$ and $y(t)$ dimensions. Moreover, Fig. 5(b) shows the distribution of the testing patterns in the $y(t)$ and $u(t)$ dimensions. Additionally, Fig. 5(c) shows the outputs of the plant and the RCFNN for the testing patterns. The analytical results also show the perfect identification capability of the RCFNN model. Moreover, Fig. 5(d) illustrates the error between the desired output and the RCFNN output. Meanwhile, Fig. 5(e) shows the learning curve of the RCFNN, the RSONFIN [7], the RFNN [8], and the TRFN-S [9]. Simultaneously, the plant additive noise was also identified by RCFNN. Fig. 6(a) illustrates the corrupted signal. Finally, Fig. 6(b) shows the recovered signal using the RCFNN after convergence.

This study compared the performance of the proposed model with that of other existing recurrent methods (RSONFIN [7], RFNN [8], and TRFN-S [9]). This study demonstrates that the proposed network outperforms its comparing rivals, and exhibits a smaller rms error for the same learning epochs. We can find that the required parameters of the proposed method are less than the parameter number of the other methods. Obviously, as shown in Table 2, the RCFNN is more effective than other existing recurrent networks.

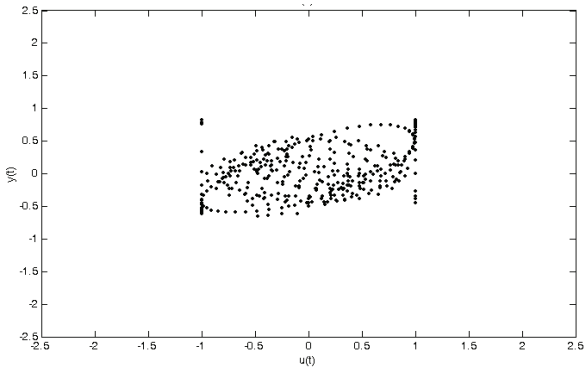
5.2. Identification of a chaotic system

This subsection discusses a typical discrete time chaotic system, which is easier to describe. Successful simulation results are then presented by training the RCFNN model. Performance comparisons with the FNN [5], RSONFIN [7], RFNN [8], and TRFN-S [9] are presented in this subsection to verify the performance of the RCFNN for temporal problems.

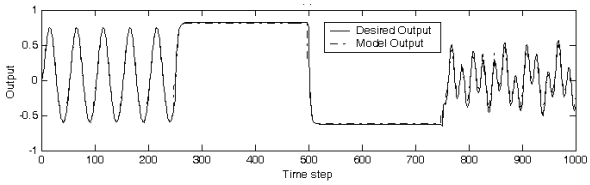
Example 3: The discrete time Henon system is repeatedly used in the study of chaotic dynamics and is not too simple in the sense that it is of the second order with one delay and two parameters [18]. This chaotic system is described by



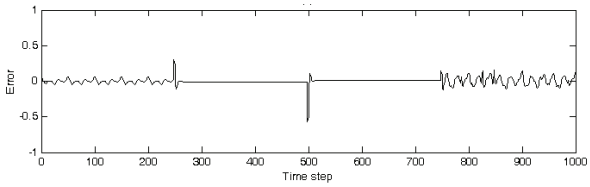
(a) The input training patterns and final assignment of rules for the distribution of the membership functions on the $y(t)$ and $u(t)$ dimensions.



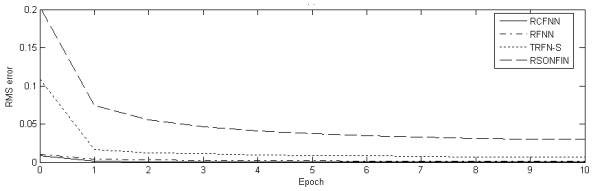
(b) The distribution of the testing patterns on the $y(t)$ and $u(t)$ dimensions.



(c) The outputs of the plant and the RCFNN.

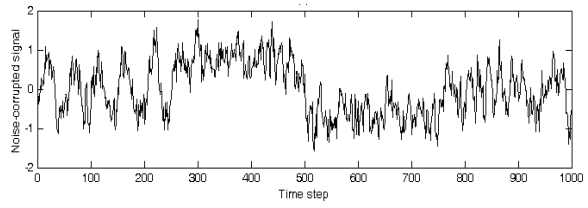


(d) The error between the RCFNN output and the desired output.

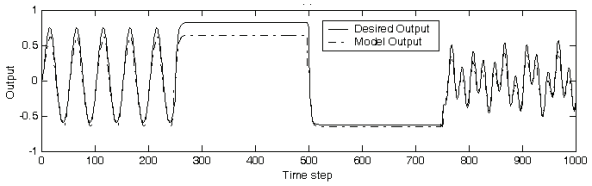


(e) Learning curve of the RCFNN, the RSONFIN [7],the RFNN [8] and the TRFN-S [9].

Fig. 5. Simulation results of the RCFNN for nonlinear system identification in Example2.



(a) The corrupted signal.



(b) Output of the plant and the identification RCFNN.

Fig. 6. Simulation results in the noisy case of the RCFNN for nonlinear system identification.

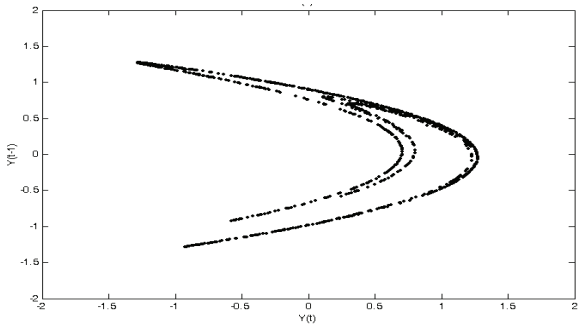
Table 2. Performance comparison of various recurrent methods with respect to the identification problem in Example 2.

	RCFNN	RSONFIN [7]	RFNN [8]	TRFN-S [9]
Parameters	24	49	24	33
RMS error (training)	0.0005	0.0300	0.0014	0.0067
RMS error (testing)	0.0015	0.0600	0.0026	0.0313
Epochs	10	10	10	10

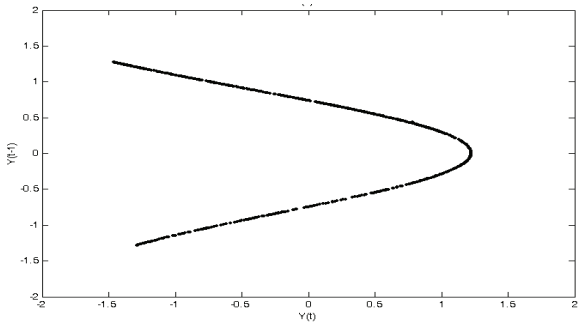
$$y(t+1) = -P \cdot y^2(t) + Q \cdot y(t-1) + 1.0, \quad for\ t = 1,2,..., \tag{32}$$

which, with P=1.4 and Q=0.3, produces a chaotic strange attractor, as shown in Fig. 7(a). For this training, the input of the RCFNN was $y(t\text{-}1)$ and the output was $y(t)$. We first randomly took the training data (1000 pairs) from a system over the interval [-1.5,1.5]. Then the RCFNN was used to approximate the chaotic system.

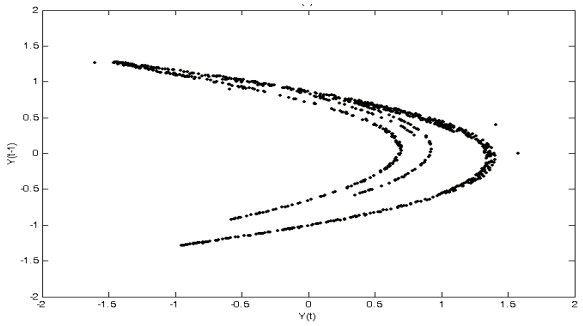
In applying the RCFNN to this example, this study used only 100 epochs. Here, the initial point was $[y(1),y(0)]^T=[0.4,0.4]^T$. The learning rate $\eta_w=\eta_c=\eta_d=\eta_m=\eta_\sigma=\eta_\Theta$=0.05, the prespecified σ_{init}=0.1, and the prespecified threshold $\bar{F}=10^{-4}$ were chosen. After training, eight fuzzy logic rules were generated. The phase plane of this chaotic system after training for the FNN [5] and the RCFNN are shown in Fig. 7(b) and Fig. 7(c), respectively. Simulation results in Fig, 7(b), indicate that the FNN is inappropriate for chaotic dynamics system because of its static mapping.



(a) The testing data of this chaotic system.



(b) Result of identification using the FNN for the chaotic system.



(c) Result of identification using the RCFNN for the chaotic system.

Fig. 7. Simulation results for identification of a chaotic system.

Table 3. Performance comparison of various methods with respect to the chaotic system identification problem in Example 3.

	RCFNN	RSONFIN [7]	RFNN [8]	TRFN-S [9]	FNN [5]
Number of rules	8	8	8	8	8
RMS error (train)	0.0034	0.0755	0.0141	0.0306	0.1338
RMS error (test)	0.0036	0.0927	0.0145	0.0341	0.1577
Epochs	100	100	100	100	100

The comparison of performance in Table 3 reveals that the rms error (training and testing) of the proposed model is smaller than the RSONFIN model [7], the RFNN model [8], the TRFN-S model [9] and the FNN model [5] under the same fuzzy rules and training epochs.

6. CONCLUSIONS

This study presents a recurrent compensatory fuzzy neural network for identifying dynamic systems. The RCFNN is a recurrent multilayered connectionist network for realizing fuzzy inference using dynamic fuzzy rules. The network comprises four layers, including two hidden layers and a feedback network. An on-line learning algorithm that consists of structure learning and parameter learning is also proposed to automatically construct the RCFNN. The structure learning algorithm is based on the measure of degree and determines whether or not to add a new node to satisfy the fuzzy partition of the input data. Moreover, the gradient descent learning algorithm is used for tuning input membership functions. Simulation results demonstrate the effectiveness of the proposed RCFNN model in resolving many temporal problems.

REFERENCES

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

[2] J. S. R. Jang, "ANFIS: Adaptive-network-based fuzzy inference system," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 23, no. 3, pp. 665-685, 1993.

[3] C. J. Lin and C. T. Lin, "Reinforcement learning for an ART-based fuzzy adaptive learning control network," *IEEE Trans. Neural Networks*, vol. 7, pp. 709-731, June 1996.

[4] C. F. Juang and C. T. Lin, "An on-line self-constructing neural fuzzy inference network and its applications," *IEEE Trans. on Fuzzy Systems*, vol. 6, no. 1, pp. 12-31, Feb. 1998.

[5] F. J. Lin, C. H. Lin, and P. H. Shen, "Self-constructing fuzzy neural network speed controller for permanent-magnet synchronous motor drive," *IEEE Trans. on Fuzzy Systems*, vol. 9, pp. 751-759, Oct. 2001.

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

[7] C. F. Juang and C. T. Lin, "A recurrent self-organizing neural fuzzy inference network," *IEEE Trans. on Neural Networks*, vol. 10, no. 4, pp. 828-845, July 1999.

[8] F. J. Lin and R. J. Wai, "Hybrid control using recurrent fuzzy neural network for linear-induction motor servo drive," *IEEE Trans. on*

Fuzzy Systems, vol. 9, no. 1, pp. 102-115, Feb. 2001.

[9] C. F. Juang, "A TSK-type recurrent fuzzy network for dynamic systems processing by neural network and genetic algorithms," *IEEE Trans. on Fuzzy Systems,* vol. 10, no. 2, pp. 155-170, Apr. 2002.

[10] H. J. Zimmermann and P. Zysno, "Latent connective in human decision," *Fuzzy Sets and Systems*, vol. 4, pp. 37-51, 1980.

[11] Y. Q. Zhang and A. Kandel, "Compensatory neurofuzzy systems with fast learning algorithms," *IEEE Trans. on Neural Networks*, vol. 9, no. 1, pp. 83-105, Jan. 1998.

[12] M. Mizumoto, "Pictorial representations of fuzzy connectives, part II: Cases of compensatory operators and self-dual operators," *Fuzzy Sets and Systems*, vol. 32, pp. 45-79, 1989.

[13] H. Seker, D. E. Evans, N. Aydin, and E. Yazgan, "Compensatory fuzzy neural networks-based intelligent detection of abnormal neonatal cerebral doppler ultrasound waveforms," *IEEE Trans. on Information Technology in Biomedicine*, vol. 5, no. 3, pp. 187-194, 2001.

[14] C. J. Lin and C. H. Chen, "Nonlinear system control using compensatory neuro-fuzzy networks," *IEICE Fundamentals on Electronics, Communications and Computer Sciences*, vol. E86-A, no. 9, pp. 2309-2316, 2003.

[15] C. J. Lin and W. H. Ho, "A pseudo-gaussian-based compensatory neural fuzzy system," *Proc. of the IEEE International Conference on Fuzzy Systems*, St. Louis, MO, USA, pp. 214-220, May 25-28, 2003.

[16] C. J. Lin and C. C. Chin, "Prediction and identification using wavelet-based recurrent fuzzy neural networks," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 34, no. 5, pp. 2144-2154, Oct. 2004.

[17] J. H. Kim and U. Y. Huh, "Fuzzy model based predictive control," *Proc. of IEEE Int. Conf. Fuzzy Systems*, vol. 1, Anchorage, AK, pp. 405-409, May 1998.

[18] G. Chen, Y. Chen, and H. Ogmen, "Identifying chaotic systems via a wiener-type cascade model," *IEEE Control Systems Magazine*, vol. 17, no. 5, pp. 29-36, Oct. 1997.

Chi-Yung Lee received the B.S. degree in Electronic Engineering from the National Taiwan University of Science and Technology, Taiwan, R.O.C., in 1981 and the M.S. degree in Industrial Education from the National Taiwan Normal University, Taiwan, R.O.C., in 1989. From August 1989 to July 2004, he was a Lecturer in the Department of Electronic Engineering, Nan-Kai Institute of Technology, Nantou, Taiwan, R.O.C. Since August 2004, he has been with the Department of Computer Science and Information Engineering, Nan-Kai Institute of Technology. Currently, he is an Associate Professor of Computer Science and Information Engineering Department, Nan-Kai Institute of Technology, Nantou, Taiwan, R.O.C. His current research interests are neural networks, fuzzy systems, intelligence control, and FPGA design.

Cheng-Jian Lin received the B.S. degree in Electrical Engineering fromTa-Tung University, Taiwan, R.O.C., in 1986 and the M.S. and Ph.D. degrees in Electrical and Control engineering from the National Chiao-Tung University, Taiwan, R.O.C., in 1991 and 1996. From April 1996 to July 1999, he was an Associate Professor in the Department of Electronic Engineering, Nan-Kai College, Nantou, Taiwan, R.O.C. From August 1999 to January 2005, he was an Associate Professor in the Department of Computer Science and Information Engineering, Chaoyang University of Technology. From February 2005 to July 2007, he was a full Professor in the Department of Computer Science and Information Engineering, Chaoyang University of Technology. Currently, he is a full Professor of Electrical Engineering Department, National University of Kaohsiung, Kaohsiung, Taiwan, R.O.C. His current research interests are soft computing, pattern recognition, intelligent control, image processing, bioinformatics, and FPGA design.

Cheng-Hung Chen was born in Kaohsiung, Taiwan, R.O.C. in 1979. He received the B.S. and M.S. degrees in Computer Science and Information Engineering from the Chaoyang University of Technology, Taiwan, R.O.C., in 2002 and 2004. He is currently working toward a Ph.D. degree in Electrical and Control Engineering at National Chiao-Tung University, Taiwan, R.O.C. His current research interests are neural networks, fuzzy systems, evolutionary algorithms, intelligent control, and pattern recognition.

Chun-Lung Chang received the B.S. degree in Automatic Control Engineering from the Feng-Chia University, Taichung, Taiwan, in 1990, and the M.S. degree in Power Mechanical Engineering from the National Tsing-Hua University, Hsinchu, Taiwan, in 1992, and the Ph.D. degree in Electrical and Control Engineering at National Chiao-Tung University, Taiwan, in 2006. He is a Researcher and Manager in Mechanical and Systems Research Laboratories, Industrial Technology Research Institute, Hsinchu, Taiwan. His current research interests are computer vision, pattern recognition, neural networks, and fuzzy inference system.