Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Image and Signal Processing (CIISP 2007)

A Recurrent Functional-Link-Based Neural Fuzzy System and Its Applications

Cheng-Hung Chen*, Cheng-Jian Lin, *Member*, *IEEE*, and Chin-Teng Lin, *Fellow*, *IEEE*

***Abstract*—In this paper, a recurrent functional-link-based neural fuzzy system (RFLNFS) is proposed for prediction of time sequence and skin color detection. The proposed RFLNFS model uses functional link neural network as the consequent part of fuzzy rules. The RFLNFS model can generate the consequent part of a nonlinear combination of the input variables. The recurrent network is embedded in the RFLNFS by adding feedback connections in the second layer, where the feedback units act as memory elements. An online learning algorithm, which consists of structure learning and parameter learning, is also presented. Finally, the RFLNFS is applied to two simulations. The simulation results of the dynamic system modeling have shown that the RFLNFS model can solve the temporal problem and the RFLNFS model has superior performance than other models.**

I. Introduction

FOR a dynamic system, the output is a function of past input or past output or both, identification of this system is not as direct as a static system, and to deal with temporal problem of dynamic system, the recurrent neural network and the recurrent neural fuzzy system have been attracting great interest. Hence, for nonlinear system processing, the most commonly used model is the neural network or neural fuzzy system. If a feedforward network is adopted for this task, then we should know the number of delayed input and output in advance, and feed these delayed input and output as a taped line to the network input [1]. The problem of this approach is that the exact order of the dynamic system is usually unknowns. To solve this problem, interest in using recurrent networks for processing dynamic system has been stably growing in recent years [2]-[5].

In this paper, a recurrent functional-link-based neural fuzzy system (RFLNFS) is proposed. The RFLNFS is a recurrent multiplayer connectionist network for fuzzy reasoning and can be constructed from a set of fuzzy rules. The RFLNFS model, which combines neural fuzzy network with functional link neural network (FLNN) [6], is designed improve the accuracy of functional approximation. Each fuzzy rule that corresponds to a FLNN consists of functional expansion of the input variables. The orthogonal polynomials and linearly independent functions are adopted as functional link neural network bases. At the same time, 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. An online learning algorithm, consisting of structure learning and parameter learning, is proposed to construct the RFLNFS model automatically. The structure learning algorithm determines whether or not to add a new node which satisfies the fuzzy partition of the input variables. The parameter learning algorithm is a recursive learning algorithm based on the ordered derivative scheme [7]. The proposed RFLNFS model has three characteristics. First, the consequent of the fuzzy rules is a nonlinear combination of the input variables. This study uses the functional link neural network to the consequent part of the fuzzy rules. Second, the online learning algorithm can automatically construct the RFLNFS model. Third, the RFLNFS model can solve temporal problems effectively.

C. H. Chen and C. T. Lin are with the Dept. of Electrical and Control Engineering, National Chiao-Tung University, Hsinchu 300, Taiwan, R.O.C.

C. J. Lin is with the Dept. of Computer Science and Information Engineering, Chaoyang University of Technology, No.168, Jifong E. Rd., Wufong Township, Taichung County 41349, Taiwan, R.O.C.

* Corresponding author. (E-mail: chchen.ece93g@nctu.edu.tw)

II. Functional Link Neural Networks

The functional link neural network is a single layer network in which the need for hidden layers is eliminated. While the input variables generated by the linear links of neural networks are linearly weighted, the functional link acts on an element of input variables by generating a set of linearly independent functions, which are suitable orthogonal polynomials for a functional expansion, and then evaluating these functions with the variables as the arguments. Therefore, the FLNN structure considers trigonometric functions. For example, for a two-dimensional input $\mathbf{X} = [x_1 \ x_2]^T$, enhanced data are obtained using trigonometric functions as $\Phi = [1, x_1, sin(\pi x_1), cos(\pi x_1), ..., x_2, sin(\pi x_2), cos(\pi x_2), ...]^T$. Thus, the input variables can be separated in the enhanced space [8]. In the FLNN structure with reference to Fig. 1, a set of basis functions Φ and a fixed number of weight parameters $\mathbf{W}$ represent $f_{\mathbf{W}}(x)$. The theory behind the FLNN for multidimensional function approximation has been discussed elsewhere [6] and is analyzed below. Consider a set of basis functions $\mathrm{B} = \{\phi_k \in \Phi(A)\}_{k \in \mathrm{K}}$, $\mathrm{K} = \{1, 2, ...\}$ with the following properties; 1) $\phi_1 = 1$, 2) the subset $\mathrm{B}_j = \{\phi_k \in \mathrm{B}\}_{k=1}^{M}$ is a linearly independent set, meaning that if $\sum_{k=1}^{M} w_k \phi_k = 0$,

then $w_k = 0$ for all $k = 1, 2, ..., M$, and 3) $sup_j \left[\sum_{k=1}^{j} \left\| \phi_k \right\|_A^2 \right]^{1/2} < \infty$.

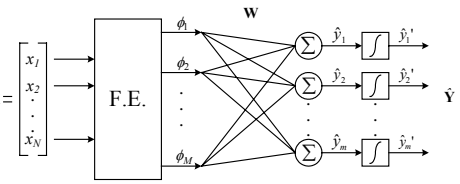

Fig. 1. Structure of FLNN.

Let $\mathrm{B}_M = \{\phi_k\}_{k=1}^{M}$ be a set of basis functions to be considered, as shown in Fig. 1. The FLNN comprises *M* basis functions $\{\phi_1, \phi_2, ..., \phi_M\} \in \mathrm{B}_M$. The linear sum of the *j*th node is given by

$$\hat{y}_j = \sum_{k=1}^{M} w_{kj} \phi_k(\mathbf{X}) \tag{1}$$

where $\mathbf{X} \in \mathrm{A} \subset \Re^N$, $\mathbf{X} = [x_1, x_2, ..., x_N]^T$ is the input vector and $\mathbf{W}_j = [w_{j1}, w_{j2}, ..., w_{jM}]^T$ is the weight vector associated with the *j*th output of the FLNN. $\hat{\mathbf{y}}_j$ denotes the local output of the FLNN structure and the consequent part of the *j*th fuzzy rule in the RFLNFS model. Thus, Eq.(1) can be expressed in matrix form as $\hat{\mathbf{y}}_j = \mathbf{W}_j \Phi$, where $\Phi = [\phi_1(x), \phi_2(x), ..., \phi_N(x)]^T$ is the basis function vector, which is the output of the functional expansion block. The *m*-dimensional linear output may be given by $\hat{\mathbf{y}} = \mathbf{W}\Phi$, where $\hat{\mathbf{y}} = [\hat{y}_1, \hat{y}_2, ..., \hat{y}_m]^T$, *m* denotes the number of functional link bases, which equals the number of fuzzy rules in the RFLNFS model, and $\mathbf{W}$ is a (*m* by *M*)-dimensional weight matrix of the FLNN given by $\mathbf{W} = [w_1, w_2, ..., w_m]^T$. The *j*th output of the FLNN is given by $\hat{y}_j' = \rho(\hat{y}_j)$, where the nonlinear function $\rho(\cdot) = tanh(\cdot)$. Thus, the *m*-dimensional output vector is given by

$$\hat{\mathbf{Y}} = \rho(\hat{y}) = f_{\mathbf{W}}(x) \tag{2}$$

where $\hat{\mathbf{Y}}$ denotes the output of the FLNN. In the RFLNFS model, the functional link bases do not exist in the initial state, and the amount of functional link bases generated by the online learning algorithm is consistent with the number of fuzzy rules. Section IV details the online learning algorithm.

III. Structure of RFLNFS Model

This section describes the recurrent functional-link-based neural fuzzy system (RFLNFS). Figure 2 presents the structure of the proposed RFLNFS model. The RFLNFS realizes a fuzzy if-then rule in the following form.

Rule-j: IF h_{1j} is A_{1j} and h_{2j} is A_{2j} ... and h_{ij} is A_{ij} ... and h_{Nj} is A_{Nj}

$$\begin{aligned} \text{THEN } \hat{y}_j &= \sum_{k=1}^{M} w_{kj} \phi_k \\ &= w_{1j}\phi_1 + w_{2j}\phi_2 + ... + w_{Mj}\phi_M \end{aligned} \tag{3}$$

where for *i*=1,2,…,*N*; $h_{ij} = u_i^{(1)}(t) + u_{ij}^{(2)}(t-1) \cdot \theta_{ij}$; $\hat{y}_j$ is local output variables; A_{ij} is the linguistic term of the precondition part with Gaussian membership function; *N* is the number of input variables; w_j is the link weight of the local output; ϕ_k is the basis trigonometric function of the input variables; *M* is the number of basis function, and *Rule-j* is the *j*th fuzzy rule. That is, 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 a dynamic fuzzy inference system.

Next, we shall introduce the operation functions of the nodes in each layer of the RFLNFS model are described. In the following description, $u^{(l)}$ denotes output of a node in the *l*th 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.

$$u_i^{(1)} = x_i \tag{4}$$

Layer 2 (Input Term Node): Nodes in this layer correspond to one linguistic label of the input variables in Layer1 and a unit of memory, i.e., the membership value specified the degree to which an input value and a unit of memory belongs a fuzzy set is calculated in Layer 2. The Gaussian membership function, the operation performed in Layer 2 is

$$u_{ij}^{(2)} = exp\left(-\frac{[h_{ij} - m_{ij}]^2}{\sigma_{ij}^2} \right) \tag{5}$$

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

$$h_{ij}(t) = u_i^{(1)}(t) + u_{ij}^{(2)}(t-1) \cdot \theta_{ij} \tag{6}$$

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

Layer 3 (Rule Node): Nodes in this layer represent the preconditioned part of a fuzzy logic rule. They receive one-dimensional membership degrees of the associated rule from the nodes of a set in layer 2. Here, the product operator described above is adopted to perform the IF-condition matching of the fuzzy rules. As a result, the output function of each inference node is

$$u_j^{(3)} = \prod_i u_{ij}^{(2)} \tag{7}$$

where the $\prod_i u_{ij}^{(2)}$ of a rule node represents the firing strength of its corresponding rule.

Layer 4 (Consequent Node): Nodes in this layer are called consequent nodes. The input to a node in layer 4 is the output from layer 3, and the other inputs are nonlinear combinations of input variables from a functional link neural network,

where the nonlinear combination function has not used the function $tanh(\cdot)$, as shown in Fig. 2. For such a node,

$$u_j^{(4)} = u_j^{(3)} \cdot \sum_{k=1}^{M} w_{kj}\phi_k \tag{8}$$

where w_{kj} is the corresponding link weight of functional link neural network and ϕ_k is the functional expansion of input variables. The functional expansion uses a trigonometric polynomial basis function, given by $[x_1\ sin(\pi x_1)\ cos(\pi x_1)\ x_2\ sin(\pi x_2)\ cos(\pi x_2)]$ for two-dimensional input variables. Therefore, M is the number of basis functions, $M = 3 \times N$, where N is the number of input variables.

Layer 5 (Output Node): Each node in this layer corresponds to a single output variable. The node integrates all of the actions recommended by layers 3 and 4 and acts as a defuzzifier with,

$$y = u^{(5)} = \frac{\sum_{j=1}^{R} u_j^{(4)}}{\sum_{j=1}^{R} u_j^{(3)}} = \frac{\sum_{j=1}^{R} u_j^{(3)} \sum_{k=1}^{M} w_{kj}\phi_k}{\sum_{j=1}^{R} u_j^{(3)}} = \frac{\sum_{j=1}^{R} u_j^{(3)} \hat{y}_j}{\sum_{j=1}^{R} u_j^{(3)}} \tag{9}$$

where R is the number of fuzzy rules, and y is the output of the RFLNFS model.

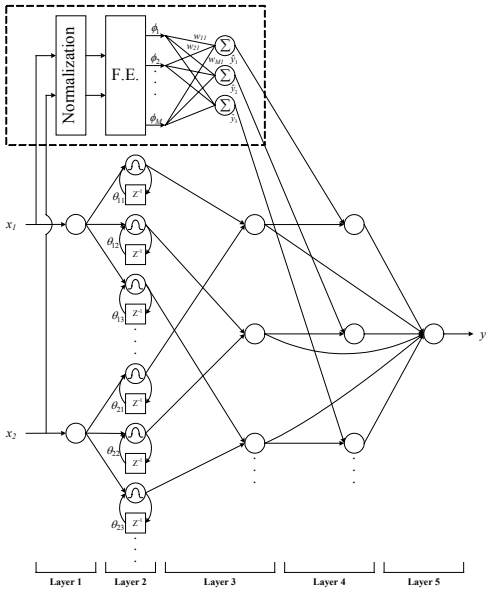

Fig. 2. Structure of the proposed RFLNFS.

IV. The Online Learning Algorithm

This section presents an online learning algorithm for constructing the RFLNFS model. 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, and the weights of the feedback.

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.

A. Structure Learning Phase

The first step in the structure learning is to determine whether or not to extract a new rule from training data as well as the number of fuzzy sets on 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. For each incoming pattern x_i the strength a rule is fired can be interpreted as the degree the incoming pattern belongs to the corresponding cluster. For computational efficiency, we can use compensatory operation of the firing strength obtained from $\prod_i u_{ij}^{(2)}$ directly as this degree measure

$$F_j = \prod_i u_{ij}^{(2)} \tag{10}$$

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

$$F_{max} = \max_{1 \le j \le R_{(t)}} F_j \tag{11}$$

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 decays during the learning process. Once a new rule is generated, the next step is to assign initial mean, variance, weight of feedback for the new membership function, and the corresponding link weight for consequent part. Since our goal is to minimize an objective function and the mean, variance, weight of feedback, and weight of consequent part are all adjustable later in the parameter learning phase. Hence, the mean, variance, weight of feedback, and weight of consequent part for the new rule are set as follow:

$$m_{ij}^{(R_{(t+1)})} = x_i \tag{12}$$

$$\sigma_{ij}^{(R_{(t+1)})} = \sigma_{init} \tag{13}$$

$$\theta_{ij}^{(R_{(t+1)})} = random[-1,1] \tag{14}$$

$$w_{kj}^{(R_{(t+1)})} = random[-1,1] \tag{15}$$

where x_i is the new input variable and σ_{init} init is a prespecified constant. The whole algorithm for the generation of new fuzzy rules as well as of 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 $\sigma_{i1}=\sigma_{init}$,
weight of feedback $\theta_{i1}=random$,
weight of consequent part $w_{kl}=random$
where σ_{init} is a perspecified 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 mean $m_{ij}^{R_{(t+1)}}=x_i$, variance $\sigma_{ij}^{R_{(t+1)}}=\sigma_{init}$,
weight of feedback $\theta_{ij}^{R_{(t+1)}}=random$,
weight of consequent part $w_{kj}^{R_{(t+1)}}=random$
where σ_{init} is a perspecified constant.}
}

B. 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. Because the RFLNFS is a dynamic system with feedback connections, the learning algorithm used in adaptive fuzzy systems [8] cannot be applied to it directly. Also, due to the online learning property of the RFLNFS, the offline learning algorithms for the recurrent neural networks, like backpropagation through time and time-dependent recurrent backpropagation [9], cannot be applied here. Instead, the ordered derivative [7], 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:

$$E(t+1)=\frac{1}{2}[y(t+1)-y^d(t+1)]^2 \tag{16}$$

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 RFLNFS are adjusted such that the error defined in Eq. (16) 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 5: There is no parameter to be adjusted in this layer. Only the error term needs to be computed and propagated. The error term is derived by

$$\delta^{(5)}(t+1)=-\frac{\partial E}{\partial y}(t+1)=y^d(t+1)-y(t+1). \tag{17}$$

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

$$\begin{aligned}\delta^{(4)}(t+1)&=-\frac{\partial E}{\partial u_j^{(4)}}(t+1)=\left[-\frac{\partial E}{\partial y}(t+1)\right]\left[\frac{\partial y(t+1)}{\partial u_j^{(4)}(t)}\right]\\&=\delta^{(5)}(t+1)\cdot\left(1\Big/\sum_{j=1}^{R}u_j^{(3)}(t)\right).\end{aligned} \tag{18}$$

And the weight of consequent part is updated by the amount

$$\begin{aligned}\Delta w_{kj}(t+1)&=-\frac{\partial E}{\partial w_{kj}}(t+1)=\left[-\frac{\partial E(t+1)}{\partial u_j^{(4)}(t)}\right]\left[\frac{\partial u_j^{(4)}}{\partial w_{kj}}(t)\right]\\&=\delta^{(4)}(t+1)\cdot\left(u_j^{(3)}(t)\varphi_k(t)\right).\end{aligned} \tag{19}$$

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

$$\begin{aligned}w_{kj}(t+1)&=w_{kj}(t)+\eta_w\Delta w_{kj}(t+1)\\&=w_{kj}(t)+\eta_w\delta^{(4)}(t+1)\cdot\left(u_j^{(3)}(t)\varphi_k(t)\right)\end{aligned} \tag{20}$$

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

$$\begin{aligned}\delta^{(3)}(t+1)&=-\frac{\partial E}{\partial u_j^{(3)}}(t+1)=\left[-\frac{\partial E(t+1)}{\partial u_j^{(4)}(t)}\right]\left[\frac{\partial u_j^{(4)}}{\partial u_j^{(3)}}(t)\right]\\&=\delta^{(4)}(t+1)\cdot\left(\sum_{k=1}^{M}w_{kj}(t)\varphi_k(t)\right).\end{aligned} \tag{21}$$

Layer 2: The error term is calculated as follows:

$$\begin{aligned}\delta^{(2)}(t+1)&=-\frac{\partial E}{\partial u_{ij}^{(2)}}(t+1)=\left[-\frac{\partial E(t+1)}{\partial u_j^{(3)}(t)}\right]\left[\frac{\partial u_j^{(3)}}{\partial u_{ij}^{(2)}}(t)\right]\\&=\delta^{(3)}(t+1)\cdot\left(\prod_{l=1}u_{ij}^{(2)}(t)\right)\end{aligned} \tag{22}$$

where l is the lth dimension. The update mean is

$$\begin{aligned}\Delta m_{ij}(t+1)&=-\frac{\partial E}{\partial m_{ij}}(t+1)=\left[-\frac{\partial E(t+1)}{\partial u_{ij}^{(2)}(t)}\right]\left[\frac{\partial u_{ij}^{(2)}}{\partial m_{ij}}(t)\right]\\&=\delta^{(2)}(t+1)u_{ij}^{(2)}(t)\left\{\frac{2[h_{ij}-m_{ij}]}{\sigma_{ij}^{2}}(t)\right\}\left\{\frac{\partial h_{ij}}{\partial m_{ij}}(t)-1\right\}\end{aligned} \tag{23}$$

where

$$\frac{\partial h_{ij}}{\partial m_{ij}}(t)=\theta_{ij}(t)u_{ij}^{(2)}(t-1)\cdot\left\{\frac{-2[h_{ij}-m_{ij}]}{\sigma_{ij}^{2}}(t-1)\right\}\cdot\left(\frac{\partial h_{ij}}{\partial m_{ij}}(t-1)-1\right). \tag{24}$$

The updated variance is

$$\begin{aligned}\Delta\sigma_{ij}(t+1)&=-\frac{\partial E}{\partial\sigma_{ij}}(t+1)=\left[-\frac{\partial E(t+1)}{\partial u_{ij}^{(2)}(t)}\right]\left[\frac{\partial u_{ij}^{(2)}}{\partial\sigma_{ij}}(t)\right]\\&=\delta^{(2)}(t+1)u_{ij}^{(2)}(t)\left\{\frac{2[h_{ij}-m_{ij}][(h_{ij}-m_{ij})-\sigma_{ij}(\partial h_{ij}/\partial\sigma_{ij})]}{\sigma_{ij}^{3}}(t)\right\}\end{aligned} \tag{25}$$

where

$$\frac{\partial h_{ij}}{\partial\sigma_{ij}}(t)=\theta_{ij}(t)u_{ij}^{(2)}(t-1)\cdot\left\{\frac{2[h_{ij}-m_{ij}][(h_{ij}-m_{ij})-\sigma_{ij}(\partial h_{ij}/\partial\sigma_{ij})]}{\sigma_{ij}^{3}}(t-1)\right\}. \tag{26}$$

The updated weight of the feedback is

$$\begin{aligned}\Delta\theta_{ij}(t+1)&=-\frac{\partial E}{\partial\theta_{ij}}(t+1)=\left[-\frac{\partial E(t+1)}{\partial u_{ij}^{(2)}(t)}\right]\left[\frac{\partial u_{ij}^{(2)}}{\partial\theta_{ij}}(t)\right]\\&=\delta^{(2)}(t+1)u_{ij}^{(2)}(t)\left\{\frac{-2[h_{ij}-m_{ij}]}{\sigma_{ij}^{2}}(t)\right\}\left\{\frac{\partial h_{ij}}{\partial\theta_{ij}}(t)\right\}\end{aligned} \tag{27}$$

where

$$\frac{\partial h_{ij}}{\partial \theta_{ij}}(t) = \theta_{ij}(t) u_{ij}^{(2)}(t-1) \left\{ \frac{-2[h_{ij} - m_{ij}]}{\sigma_{ij}^2}(t-1) \right\} \left\{ \frac{\partial h_{ij}}{\partial \theta_{ij}}(t-1) \right\} + u_{ij}^{(2)}(t-1). \quad (28)$$

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

$$m_{ij}(t+1) = m_{ij}(t) + \eta_m \Delta m_{ij}(t+1), \quad (29)$$

$$\sigma_{ij}(t+1) = \sigma_{ij}(t) + \eta_\sigma \Delta \sigma_{ij}(t+1), \quad (30)$$

$$\theta_{ij}(t+1) = \theta_{ij}(t) + \eta_\theta \Delta \theta_{ij}(t+1) \quad (31)$$

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.

V. Illustrative Examples

A. Prediction of Time Sequence

To clearly verify if the proposed RFLNFS can learn the temporal relationship, a simple time sequence prediction problem found in [10] is used for test in the following example.

The test bed used is shown in Fig. 3(a). This is an "8" shape made up of a series with 12 points which are to be presented to the network in the order as shown. The RFLNFS is 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: point 5 and point 11. The RFLNFS 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 RFLNFS contains only two input nodes, which are activated with the two dimensional coordinate of the current point, and two output nodes, which represent the two dimensional coordinate of the predicted point. After training, a root-mean-square (RMS) error of 0.000237 is achieved, and the predicted values with 12 fuzzy logic rules (σ_{init}=0.08) of RFLNFS are shown in Fig. 3(b). Simulation results show that we can obtain perfect prediction capability.

Recently, Lee and Teng [4] proposed a model, called recurrent fuzzy neural network (RFNN) architecture, for learning and tuning a fuzzy predictor. Our model is similar to RFNN except layer three. The layer three of the RFNN performed product operator while the layer three of our model performed compensatory operator. For initializing parameters of the RFNN model, the rule number should be given in advance. But, the users need not give it any *a priori* knowledge or even any initial information for our proposed model. We also applied the RFNN model [4] and a traditional (non-recurrent) fuzzy neural network (FNN) [11] to this time prediction problem. Fig. 3(c) shows the prediction results using the RFNN model [4]. In this figure, the RFNN also obtain prediction capability, but some time prediction points cannot be matched exactly. Fig. 3(d) shows that a feedforward fuzzy neural network cannot predict successfully. From the simulation results shown in Fig. 2(d), we can see that the FNN is inappropriate for time sequence prediction because of its static mapping. To give a clear understanding of this performance comparison with the RFNN [4] and FNN [11] on the same problem is made in Table 1. The results show that the proposed RFLNFS model is able to maintain a smaller RMS error than other methods.

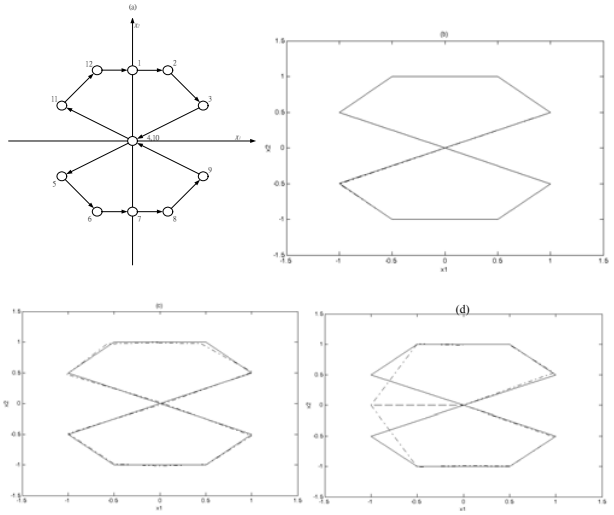

Fig. 3. Simulation results of time sequence prediction. (a) Test bed for the next sample prediction experiment in Example 1. (b) Results of prediction using the RFLNFS. (c) Results of prediction using the RFNN. (d) Results of prediction using the FNN.

Table 1: Performance comparison of various existing models.

	RFLNFS	RFNN [4]	FNN [11]
Rule numbers	**12**	12	12
RMS error	**0.000237**	0.0063	0.1758
Epochs	**1000**	1000	1000

B. Skin Color Detection

Skin color detection from images is a key problem in human computer interaction studies and pattern recognition research. We exploit one database, which is the California Institute of Technology face database (CIT) via http://www.vision.caltech.edu/Image_Datasets/faces/, including 450 color images, each of size 320*240pixels, containing 27 different people and a variety of lighting, backgrounds and facial expressions. The three input dimensions (Y, Cb, and Cr) are used in this experiment and we choose 6000 training data and 6000 testing data. We exploit the CIT database to produce both the training data and the testing data. Moreover, the 6000 skin and non-skin pixels training data in the color images are randomly chosen, testing data uses the same method.

The experiment calculated the training and testing accuracy rates (e.g. best, worst and average situations) by the RFLNFS and the RFNN model. The comparison result with various existing models for the CIT database is tabulated in Tables 2.

The color images from the CIT database are shown in Figs.

4. The well-trained network would generate binary outputs (1/0 for skin/non-skin) to detect the facial region. Fig. 5 and 6 demonstrate the results of skin color detection by RFNN and RFLNFS, respectively. Experimental result has shown that the performance of the RFLNFS is superior to the RFNN.

Fig. 4. Original of California Institute of Technology (CIT) face database.

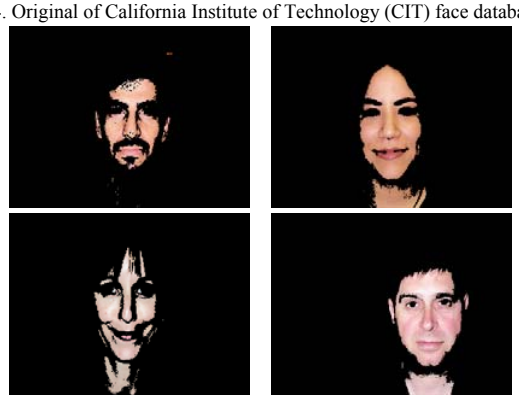

Fig. 5. Test result of 3-dimension input by the RFLNFS.

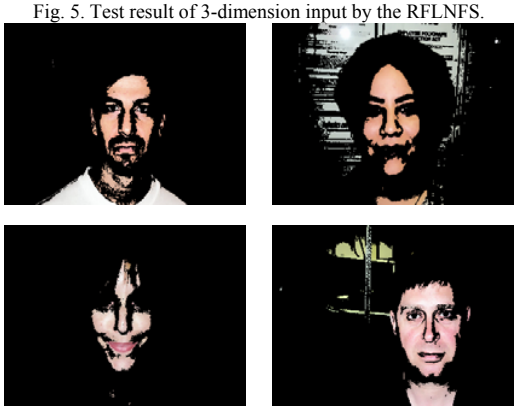

Fig. 6. Test result of 3-dimension input by RFNN.

VI. Conclusion

A recurrent functional-link-based neural fuzzy system (RFLNFS) is proposed in this paper. The RFLNFS model is able to form the consequent part of a nonlinear combination of the input variables to be approximated more effectively. A recurrent network that solves temporal problems is embedded in the RFLNFS by adding feedback connections in the second layer, where the feedback units act as memory elements. An online learning algorithm is proposed to perform the structure learning and the parameter learning. Finally, the proposed RFLNFS model also obtains better simulation results than other existing models in some circumstances.

Table 2: Performance comparison with the RFLNFS and RFNN for CIT database

	RFLNFS		RFNN [4]	
Training Data	**6000**		6000	
Accuracy Rate (Training)	**Best**	**Worst**	Best	Worst
	94.48%	**87.93%**	84.55%	83.53%
Avg(Training)	**91.17%**		71.68%	
Accuracy Rate (Testing)	**Best**	**Worst**	Best	Worst
	93.27%	**83.73%**	80.67%	61.5%
Avg(Testing)	**88.02%**		66.62%	

Acknowledgment

This research was sponsored by Department of Industrial Technology, Ministry of Economic Affairs, R.O.C. under the grant 95-EC-17-A-02-S1-029.

References

[1] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. on Neural Networks*, vol. 1, no.1, pp. 4-27, 1990.

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

[3] 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.

[4] C. H. Lee and C. C. Teng, "Identification and control of dynamic systems using recurrent fuzzy neural networks," *IEEE Trans. on Fuzzy Systems*, vol. 8, no. 4, pp. 349-366, Aug. 2000.

[5] C. J. Lin and C. C. Chin, "Prediction and identification using wavelet-based recurrent fuzzy neural networks," *IEEE Trans. on Syst., Man, Cybernet*, vol. 34, pp. 2144-2154, 2004.

[6] J. C. Patra and R. N. Pal, "A functional link artificial neural network for adaptive channel equalization," *Signal Process*, vol. 43, pp. 181-195, May 1995.

[7] P. Werbos, "New tools for prediction and analysis in the behavior sciences," Ph.D. dissertation, Harvard Univ., Cambridge, MA, 1974.

[8] L. X. Wang, *Adaptive fuzzy systems and control*, Englewood Cliffs, NJ: Prentice-Hall, 1994.

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

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

[11] C. T. Chao, T. J. Chen, and C. C. Teng, "Simplification of fuzzy-neural systems using similarity analysis," *IEEE Trans. Syst., Man, Cybernet*, vol. 26, no. 2, pp. 344-354, 1996.