Available online at www.sciencedirect.com
SCIENCE DIRECT
ELSEVIER
Mathematical and Computer Modelling 42 (2005) 339–351
MATHEMATICAL AND COMPUTER MODELLING
www.elsevier.com/locate/mcm

A Self-Constructing Compensatory Neural Fuzzy System and Its Applications

CHENG-JIAN LIN AND CHENG-HUNG CHEN
Department of Computer Science and Information Engineering
Chaoyang University of Technology, No. 168, Jifong E. Rd., Wufong Township
Taichung County 41349, Taiwan, R.O.C.
cjlin@mail.cyut.edu.tw

(Received May 2003; accepted July 2004)

Abstract—A self-constructing compensatory neural fuzzy system (SCCNFS) for nonlinear system identification and control is proposed in this paper. The compensatory fuzzy reasoning method uses adaptive fuzzy operations of a neural fuzzy network to make the fuzzy logic system more adaptive and effective. An online learning algorithm is proposed to automatically construct the SCCNFS. The fuzzy rules are created and adapted as online learning proceeds through simultaneous structure and parameter learning. The structure learning is based on the fuzzy similarity measure and the parameter learning is based on the backpropagation algorithm. The advantages of the proposed learning algorithm are that it converges quickly and that the fuzzy rules that are obtained are more precise. The performance of SCCNFS compares excellently with other various existing models.

Keywords—Compensatory, Fuzzy similarity measure, Inverted wedge system, Backpropagation algorithm.

1. INTRODUCTION

Recently, the neural fuzzy approach to system modeling has become a popular research topic [1–10]. Moreover, the neural fuzzy method possesses the advantages of both the pure neural and the fuzzy methods; it brings the low-level learning and computational power of neural networks into fuzzy systems and incorporates the high-level human-like thinking and reasoning of fuzzy systems into neural networks.

Many papers [4–10] have dealt with optimal fuzzy membership functions and defuzzification schemes for applications by using learning algorithms to adjust the parameter of fuzzy membership functions and defuzzification functions. Unfortunately, for optimal fuzzy logic reasoning and selected optimal fuzzy operators, only static fuzzy operators are often used for fuzzy reasoning, such that the conventional neural fuzzy system can only adjust the fuzzy membership functions by using fixed fuzzy operations, such as Min and Max. The compensatory neural fuzzy system [11] with adaptive fuzzy reasoning is more effective and adaptive than the conventional

This research is supported by the National Science Council of the R.O.C. under Grant NSC 90-2213-E-324-011.

0895-7177/05/$ - see front matter Typeset by AMS-TEX
doi:10.1016/j.mcm.2004.07.017

neural fuzzy system with nonadaptive fuzzy reasoning [4]. Therefore, an effective neural 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 self-constructing compensatory neural fuzzy system (SCCNFS) is proposed. The compensatory fuzzy reasoning method uses adaptive fuzzy operations of a neural fuzzy network to make the fuzzy logic system more adaptive and effective. An online learning algorithm is proposed to automatically construct the SCCNFS. It consists of structure learning and parameter learning. The structure learning algorithm determines whether to add a new node which satisfies the fuzzy partition of the input data. The similarity measure of symmetric Gaussian membership functions is used. The backpropagation learning algorithm is then used for tuning membership functions.

The proposed learning algorithm has four advantages. First, it does not require human assistance. Second, its structure is obtained from the input data. Third, it converges quickly. Fourth, the obtained fuzzy rules are more precise than other learning algorithms.

This paper is organized as follows. Section 2 describes the basic structure and functions of the SCCNFS. The online structure and parameter learning algorithms of the SCCNFS is presented in Section 3. In Section 4, the SCCNFS is applied to solve several problems. Finally, conclusions are given in the last section.

2. THE STRUCTURE OF SCCNFS

In this section, the structure of the SCCNFS is introduced. This four-layer network [6] realizes a fuzzy model in the following form:

$$\begin{aligned} R_j : &\text{ IF } x_1 \text{ is } A_{1j} \text{ and } x_2 \text{ is } A_{2j} \ldots \text{ and } x_n \text{ is } A_{nj} \\ &\text{THEN } y' = b_j, \end{aligned} \tag{1}$$

where x_i is the input variable, y' is the output variable, A_{nj} is the linguistic term of the precondition part, b_j is the constant consequent part, and n is the number of input variables.

The structure of the SCCNFS is shown in Figure 1. The functions of the nodes in each layer of the SCCNFS model are described as follows.

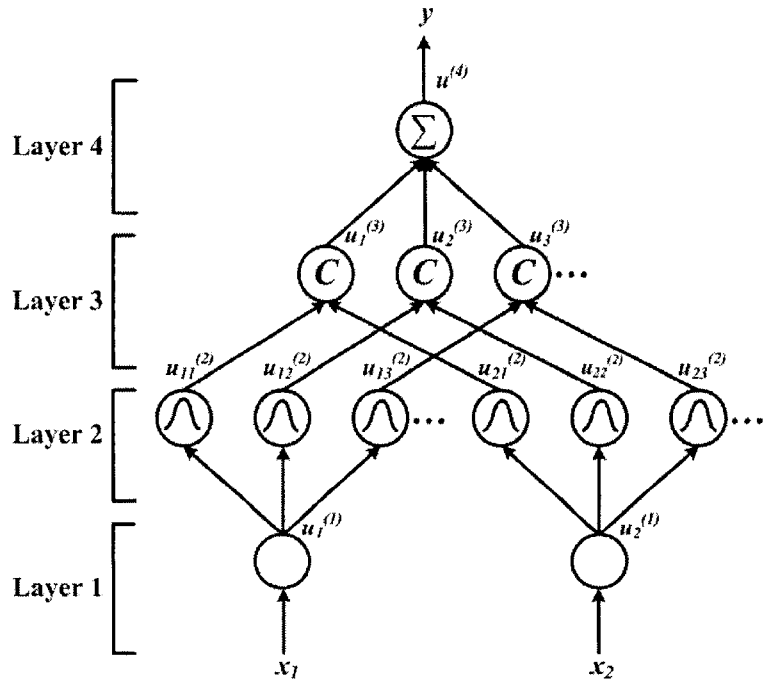

Figure 1. Structure of the proposed SCCNFS.

LAYER 1. *No computation is done in this layer. Each node in this layer is an input node, which corresponds to one input variable and only transmits input values to the next layer directly*

$$u_i^{(1)} = x_i. \tag{2}$$

LAYER 2. *Nodes in this layer correspond to one linguistic label of the input variables in Layer 1; that is, the membership value specifies the degree to which an input value belonging to a fuzzy set is calculated in Layer 2. The Gaussian membership function, the operation performed in Layer 2, is*

$$u_{ij}^{(2)} = \exp\left(-\frac{\left[u_i^{(1)} - m_{ij}\right]^2}{\sigma_{ij}^2}\right), \tag{3}$$

where m_{ij} and σ_{ij} are, respectively, the mean and variance of the Gaussian membership function of the j^{th} term of the i^{th} input variable x_i.

LAYER 3. *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 nodes of a set in Layer 2. Here, we use a compensatory fuzzy operator mentioned in [11] to perform IF-condition matching of fuzzy rules. As a result, the output function of each inference node is*

$$u_j^{(3)} = \left(\prod_i u_{ij}^{(2)}\right)^{1-r+r/R}, \tag{4}$$

where $r \in [0,1]$ is called the compensatory degree and R is number of rules. When r is tuned, the fuzzy operator becomes more adaptive.

LAYER 4. *This layer acts a defuzzifier. The single node in this layer is labeled $\sum$ and it sums all incoming signals to obtain the final inferred result*

$$u^{(4)} = \sum_j u_j^{(3)} w_j, \tag{5}$$

where the weight w_j is the output action strength associated with the j^{th} rule and $u^{(4)}$ is the output of the SCCNFS.

3. THE ONLINE HYBRID LEARNING ALGORITHM

In this section, we present an online hybrid learning algorithm for constructing the SCCNFS. The hybrid learning algorithm consists of a structure learning phase and a parameter learning phase. The structure learning phase includes both determining proper fuzzy partitions and finding fuzzy logic rules subject to two objectives: to minimize the number of rules generated and to minimize the number of fuzzy sets in the universe of discourse of each input variable. The parameter learning phase is based upon supervised learning algorithms. The backpropagation algorithm is used to minimize a given cost function to adjust the weights in the consequent part, the parameters of the membership functions, and the compensatory degree.

Initially, there are no nodes in the network except the input-output nodes; that is, there are no rule nodes and membership. They are created dynamically and automatically as learning proceeds upon receiving online incoming training data when the structure and parameter learning processes are used. The details of the structure learning phase and the parameter learning phase are described in the rest of this section.

3.1. The Structure Learning Phase

The first step in 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. For each incoming pattern x_i, the strength a rule is fired can be interpreted as the degree to which the incoming pattern belongs to the corresponding cluster. For computational efficiency, we can use the firing strength obtained from $\Pi u_{ij}^{(2)}$ directly as the degree measure

$$F_j = \prod_i u_{ij}^{(2)}, \tag{6}$$

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

$$F_{\max} = \max_{1 \le j \le R_{(t)}} F_j, \tag{7}$$

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

$$m_{ij}^{(R_{(t+1)})} = x_i, \tag{8}$$

$$\sigma_{ij}^{(R_{(t+1)})} = \sigma_{\text{init}}, \tag{9}$$

where x_i is the new data and σ_{init} is a prespecified constant.

Since the generation of a membership function corresponds to the generation of a new fuzzy rule, the weight $w_j^{(R_{(t+1)})}$ associated with a new fuzzy rule has to be determined. Generally, the weight $w_j^{(R_{(t+1)})}$ is selected randomly.

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

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 $w_1 =$ 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 $F_{\max} \ge \bar{F}$
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 $w_j^{R_{(t+1)}} =$ random

where σ_{init} is a prespecified constant.}
}

To prevent a newly generated membership function from being too similar to an existing one, the similarities between a new membership function and existing membership functions must be checked. The fuzzy similarity measure [1] determines the similarity between two fuzzy sets. If A and B are two fuzzy sets with membership functions $u_A(x) = \exp[-(x - m_1)^2/\sigma_1^2]$ and $u_B(x) = \exp[-(x - m_2)^2/\sigma_2^2]$, then the approximate fuzzy similarity measure of A and B, $E(A, B)$ [1], can be computed as follows.
Assume $m_1 \geq m_2$. Then,

$$E(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{\sigma_1\sqrt{\pi} + \sigma_2\sqrt{\pi} - |A \cap B|}, \tag{10}$$

where $|A \cap B|$ indicates the cardinality of $A \cap B$. $|A \cap B|$ can be easily computed from

$$\begin{aligned} |A \cap B| = {} & \frac{1}{2}\frac{h^2\left(m_2 - m_1 + \sqrt{\pi}\left(\sigma_1 + \sigma_2\right)\right)}{\sqrt{\pi}\left(\sigma_1 + \sigma_2\right)} \\ & + \frac{1}{2}\frac{h^2\left(m_2 - m_1 + \sqrt{\pi}\left(\sigma_1 - \sigma_2\right)\right)}{\sqrt{\pi}\left(\sigma_2 - \sigma_1\right)} \\ & + \frac{1}{2}\frac{h^2\left(m_2 - m_1 - \sqrt{\pi}\left(\sigma_1 - \sigma_2\right)\right)}{\sqrt{\pi}\left(\sigma_1 - \sigma_2\right)}, \end{aligned} \tag{11}$$

where $h(x) = \max\{0, x\}$.

The similarity measure E between a new membership function and all existing membership functions are calculated and the maximum one, $E_{\max}$, is found as follows:

$$E_{\max} = \max_{1 \leq j \leq M(t)} E\left(u\left(m_i^{(\text{new})}, \sigma_i^{(\text{new})}\right), u\left(m_{ij}, \sigma_{ij}\right)\right), \tag{12}$$

where $u(m_{ij}, \sigma_{ij})$ represents the Gaussian membership function with mean m_{ij} and standard deviation σ_{ij} and $M(t)$ is the number of membership functions of the i^{th} input variable. If $E_{\max} \leq \bar{E}$, where $\bar{E} \in (0, 1)$ is a prespecified value, then the new membership function is adopted and the number $M(t)$ is incremented

$$M(t+1) = M(t) + 1. \tag{13}$$

3.2. 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 adjusted along the negative gradient. The idea of backpagation is used for this supervised learning method. For clarity of the single output case, our goal is to minimize the cost function E, which is defined as

$$E = \frac{1}{2}\left[y - y^d\right]^2, \tag{14}$$

where y^d is the desired output and y is the current output. The parameter learning algorithm based on backpropagation is described as follows.

LAYER 4. *The error term to be propagated is calculated as*

$$\delta^{(4)} = -\frac{\partial E}{\partial y} = y^d - y. \tag{15}$$

The weight is updated by the amount

$$\Delta w_j = -\frac{\partial E}{\partial w_j} = \left[-\frac{\partial E}{\partial y}\right]\left[\frac{\partial y}{\partial w_j}\right] = \delta^{(4)} u_j^{(3)}. \tag{16}$$

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

$$\begin{aligned} w_j(t+1) &= w_j(t) + \eta_w \Delta w_j \\ &= w_j(t) + \eta_w \delta^{(4)} u_j^{(3)}, \end{aligned} \tag{17}$$

where factor η_w is the learning rate parameter of the weight and t denotes the iteration number of the j^{th}.

LAYER 3. *In this layer only, the error term needs to be computed and propagated*

$$\begin{aligned} \delta^{(3)} &= -\frac{\partial E}{\partial u_j^{(3)}} = \left[-\frac{\partial E}{\partial y}\right]\left[\frac{\partial y}{\partial u_j^{(3)}}\right] \\ &= \delta^{(4)} w_j. \end{aligned} \tag{18}$$

The following is the learning rule for the compensatory degree r. To eliminate the constraint $r \in [0,1]$, we redefine r as follows:

$$r = \frac{c^2}{c^2 + d^2}, \tag{19}$$

$$\begin{aligned} \Delta r &= -\frac{\partial E}{\partial r} = \left[-\frac{\partial E}{\partial u_j^{(3)}}\right]\left[\frac{\partial u_j^{(3)}}{\partial r}\right] \\ &= \delta^{(3)}\left[\frac{1}{R} - 1\right] \ln\left[\prod_i u_{ij}^{(2)}\right] u_{ij}^{(3)}. \end{aligned} \tag{20}$$

Then, we have

$$c(t+1) = c(t) + \eta_c \left\{\frac{2c(t)d^2(t)}{[c^2(t) + d^2(t)]}\right\} \Delta r, \tag{21}$$

$$d(t+1) = d(t) - \eta_d \left\{\frac{2c^2(t)d(t)}{[c^2(t) + d^2(t)]}\right\} \Delta r, \tag{22}$$

$$r(t+1) = \frac{c^2(t+1)}{c^2(t+1) + d^2(t+1)}. \tag{23}$$

In all the above formulas, η_c and η_d are the learning rate of parameters c and d.

LAYER 2. *The error term is calculated as follows:*

$$\begin{aligned} \delta^{(2)} &= -\frac{\partial E}{\partial u_{ij}^{(2)}} = \left[-\frac{\partial E}{\partial u_j^{(3)}}\right]\left[\frac{\partial u_j^{(3)}}{\partial u_{ij}^{(2)}}\right] \\ &= \delta^{(3)}\left[1 - r + \frac{r}{R}\right]\left[\prod_i u_{ij}^{(2)}\right]^{(-r+r/R)}\left[\prod_{l \neq i} u_{lj}^{(2)}\right], \end{aligned} \tag{24}$$

where l is the l^{th} dimension. The updated mean is

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

The updated variance is

$$\begin{aligned}\Delta\sigma_{ij} &= -\frac{\partial E}{\partial \sigma_{ij}} = \left[-\frac{\partial E}{\partial u_{ij}^{(2)}}\right]\left[\frac{\partial u_{ij}^{(2)}}{\partial \sigma_{ij}}\right] \\ &= \delta^{(2)} u_{ij}^{(2)} \left[\frac{2\left(u_i^{(1)} - m_{ij}\right)^2}{\sigma_{ij}^3}\right].\end{aligned} \tag{26}$$

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

$$m_{ij}(t+1) = m_{ij}(t) + \eta_m \Delta m_{ij}, \tag{27}$$

$$\sigma_{ij}(t+1) = \sigma_{ij}(t) + \eta_\sigma \Delta \sigma_{ij}, \tag{28}$$

where η_m and η_σ are the learning rate parameters of the mean and the variance of the Gaussian function, respectively.

4. SIMULATIONS

In this section, we compare the performance of SCCNFS with other models in two applications: identification of a nonlinear dynamic system [4] and control of an inverted wedge system [12].

Example 1: Identification of a Nonlinear Dynamic System

In this example, the SCCNFS was used to identify a dynamic system. The identification model had the form

$$\widehat{y}(k+1) = \widehat{f}[u(k), u(k-1), \ldots, u(k-p+1)y(k), y(k-1), \ldots, y(k-q+1)]. \tag{29}$$

Since both the unknown plant and the SCCNFS were driven by the same input, the SCCNFS adjusted itself with the goal of causing the output of the identification model to match that of the unknown plant. Upon convergence, their input-output relationship should match. The plant to be identified was guided by the differential equation

$$y(k+1) = \frac{y(k)}{1+y^2(k)} + u^3(k). \tag{30}$$

The output of the plant depends nonlinearly on both its past values and inputs, but the effects of the input and output values are additive. When the SCCNFS was applied to this identification problem, the learning rate $\eta_w = \eta_c = \eta_d = \eta_m = \eta_\sigma = 0.01$ and the prespecified threshold $\bar{F} = 0.1$ were used. The training patterns were generated with $u(k) = \sin(2\pi k/100)$, and training was performed for 100 epochs. Starting at zero, the number of clusters grew dynamically for the incoming training data. Each cluster corresponded to a circle in the input space.

Figure 2 shows the root-mean-square (RMS) errors during learning. Figure 3 gives the distribution of the training patterns and the final assignment of 11 fuzzy logic rules (i.e., the distribution of input membership functions) in the $[u(k), y(k)]$ plane after training. Figure 4 shows the outputs of the plant and the identification model. In this figure, the outputs of the SCCNFS are represented as os while the plant outputs are represented as *s. The simulation results show perfect identification capability of the well-trained SCCNFS.

We now compare the performance of the SCCNFS with other various existing models [4–6]. The performance indices considered included training steps, rule numbers, and rms errors. The comparison results are tabulated in Table 1. The results show that the proposed SCCNFS needs fewer training steps and has fewer rms errors than the FALCON model [4] and the SONFIN model [5]. Although the needed training steps and rms errors of SCCNFS model approximated

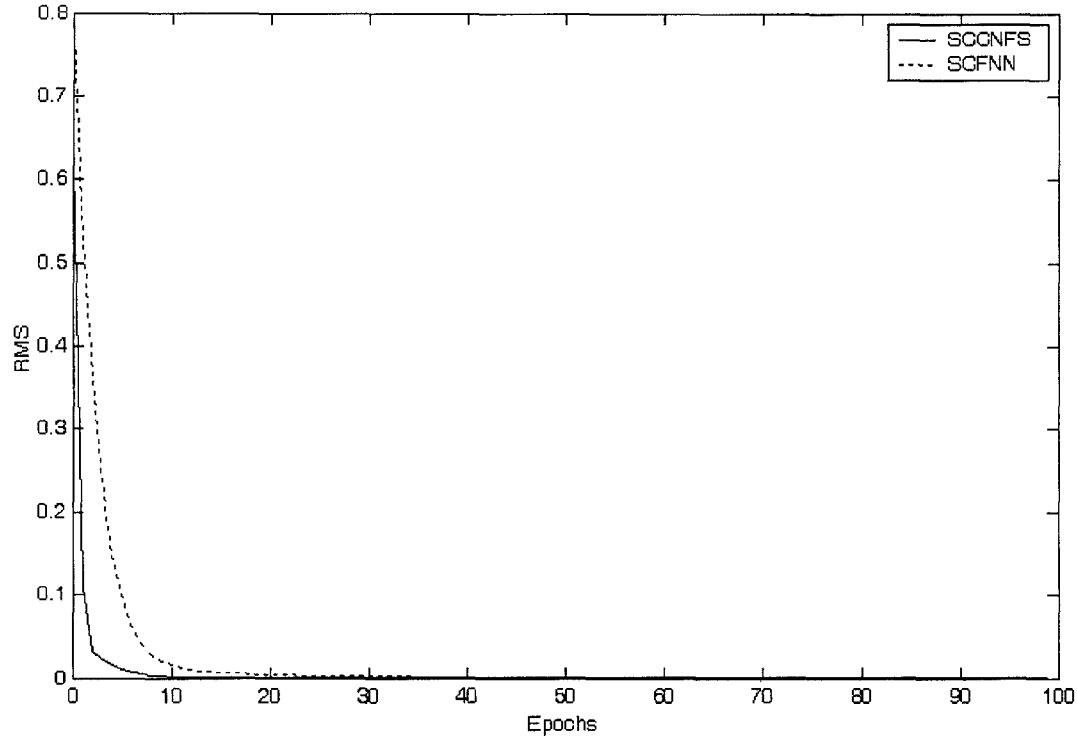

Figure 2. Learning curve of the SCCNFS and SCFNN[6].

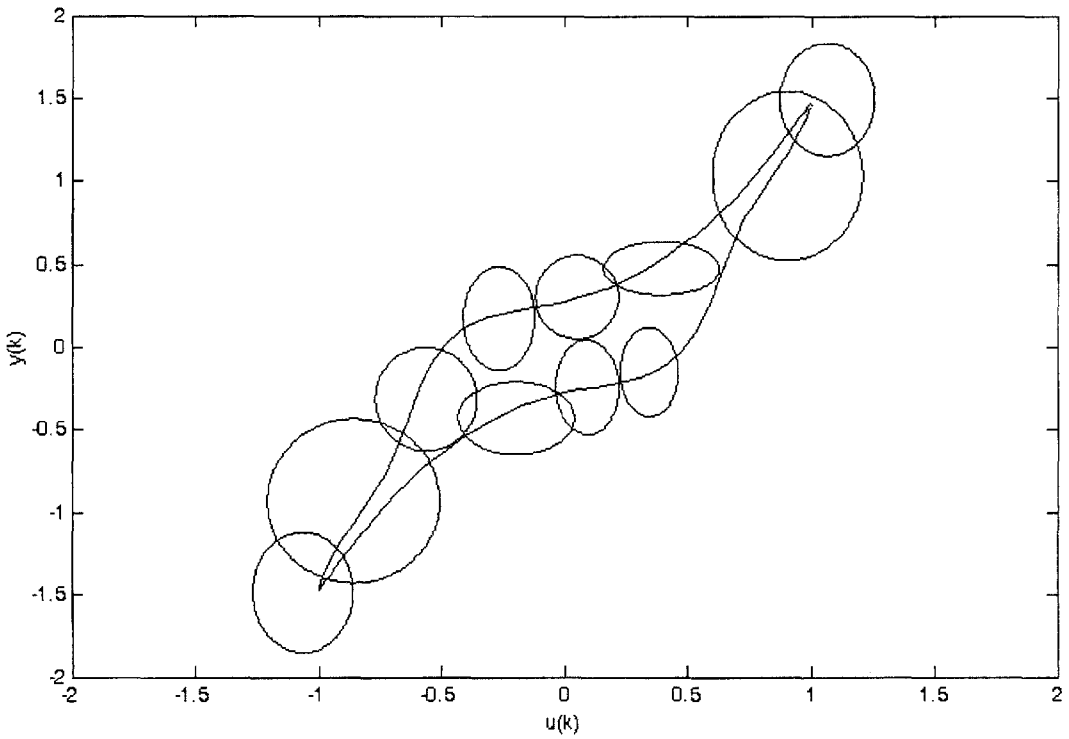

Figure 3. Simulation results of the SCCNFS on the membership functions of each input variable in Example1. The input training patterns and the final assignment of rules.

the SCFNN [6] model, it is obvious that our model has quicker convergent speed and fewer rule numbers than the SCFNN model.

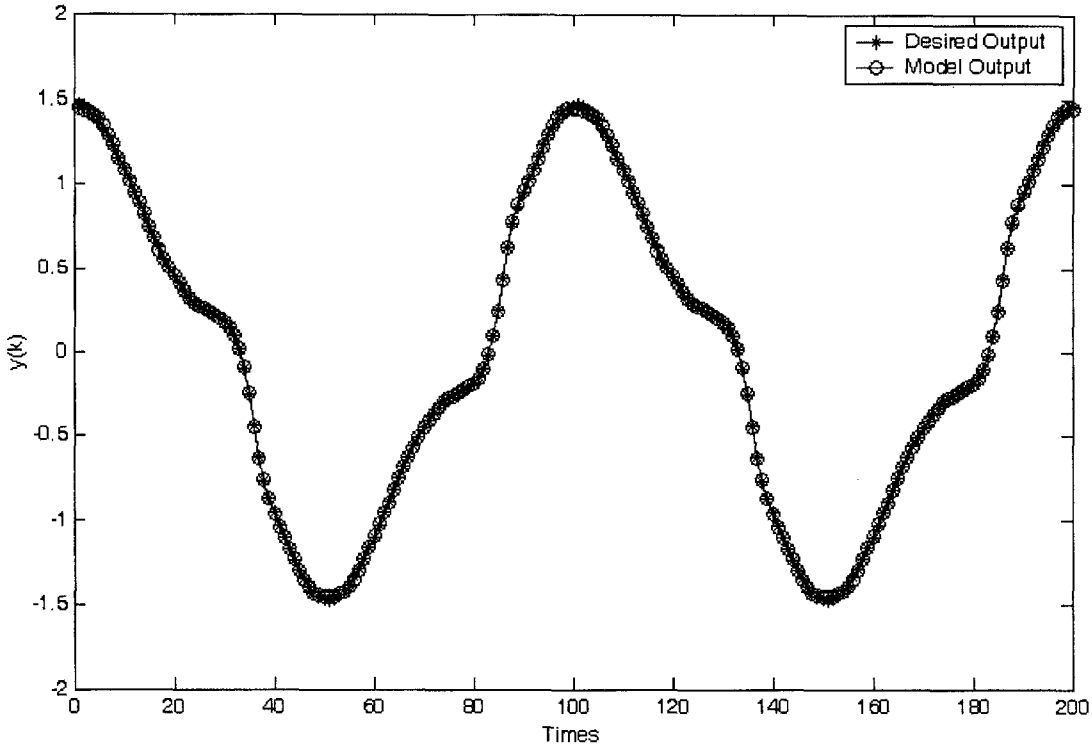

Figure 4. The outputs of the plant and the identification model.

Table 1. Performance comparison of various existing models on identification problem.

	SCCNFS	FALCON [4]	SONFIN [5]	SCFNN [6]
Training steps	100	60000	50000	100
Rule numbers	11	6	10	22
RMS errors	0.0002	0.02	0.013	0.0003

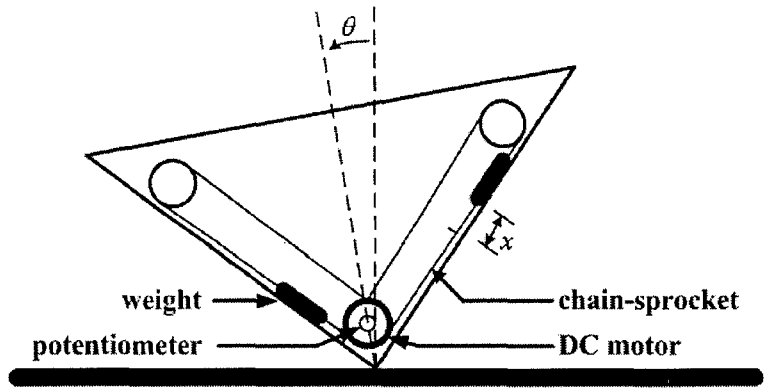

Figure 5. The inverted wedge balancing system.

Example 2: Control of the Inverted Wedge Balancing

The inverted wedge [12] is an inherently unstable nonlinear system, as shown in Figure 5. A DC motor leads the sliding weights via a sprocket-and-chain mechanism. The sliding weights are, in fact, rechargeable batteries that supply power to all the on-board electronics. A linear power amplifier module is used to drive the motor. A multiturn potentiometer is coupled to the motor

shaft to sense the position of the sliding weights. A vertical gyro is used to sense the absolute angle. The vertical gyro is a free gyro with a gravity sensitive erection mechanism that keeps the gyro spin axis in the direction of gravity. A vertical gyro provides virtually unlimited bandwidth.

After the mechanical design and selection of a motor and sensors are finalized, a model of the complete system including all the onboard equipment can be derived. In Figure 5, θ and x are used as generalized coordinates. θ is the wedge angle from the upright position and x is the weight displacement from the wedge center line. The total kinetic energy of the system is shown to be

$$T = \frac{1}{2}J_1\dot{\theta}^2 + \frac{1}{2}m_2\left(\left((1-x)\dot{\theta}\right)^2 + \dot{x}^2\right) + \frac{1}{2}J_2\frac{\dot{x}^2}{k_s^2}, \tag{31}$$

where m_2 is the mass of each weight, J_1 is the wedge inertia, J_2 is the chain-sprocket inertia, and k_s is the sprocket radius. In equation (31), the first term on the right is the kinetic energy of the wedge, the second and the third terms are the kinetic energy of the weights, and the last term is the kinetic energy due to the rotation of the motor rotor, chain, and sprocket. The total potential energy is

$$\begin{aligned} U &= m_1 gh\cos(\theta) + m_2 g(l-x)\cos(\theta+\alpha) + m_2 g(l+x)\cos(\alpha-\theta) \\ &= (m_1 gh + 2m_2 gl\cos(\alpha))\cos(\theta) + 2m_2 g\sin(\alpha)x\sin(\theta), \end{aligned} \tag{32}$$

where h is the wedge center of mass and m_1 is the wedge mass. Substituting Lagrange $L = T - U$ into Lagrange's equation, we have

$$\begin{aligned} &\frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}} - \frac{\partial L}{\partial \theta} = 0, \\ &\frac{d}{dt}\frac{\partial L}{\partial \dot{x}} - \frac{\partial L}{\partial x} = \frac{k_r I}{k_s} - k_{\text{fric}}\,\text{sgn}\,(\dot{x}), \end{aligned} \tag{33}$$

where k_r is the motor resistance, k_{fric} is the sliding mechanic friction, and I is the motor current. Thus, the motor current is given by

$$I = -\frac{k_{\text{bemf}}}{k_s k_r}\dot{x} + \left(\frac{k_{\text{da}}k_{\text{pa}}}{k_r}\right)u, \tag{34}$$

where k_{bemf}, k_{da}, and k_{pa} represent the bemf constant, the D/A converter constant, and the power amplification constant, respectively. Combining equations (33) and (34), and using $q_1 \equiv \theta$, $q_2 \equiv x$, $q_3 \equiv \dot{\theta}$, and $q_4 = \dot{x}$ as the state variables and u as the control signal, we obtain the following open loop system equation:

$$\begin{aligned} \dot{q}_1 &= q_3, \\ \dot{q}_2 &= q_4, \\ \dot{q}_3 &= \frac{-4m_2 q_2 q_3 q_4 + K_v \sin(q_1) - 2m_2 g\sin(\alpha)q_2\cos(q_1)}{J_1 + 2m_2\left(l^2 + q_2^2\right)}, \\ \dot{q}_4 &= \frac{2m_2 q_2 q_3^2 - 2m_2 g\sin(\alpha)\sin(q1) - \text{kfric}\,\text{sgn}(\dot{x})}{(J_2/k_s^2) + 2m_2} - K_v q_4 + K_b u, \end{aligned} \tag{35}$$

where

$$K_v = \frac{k_t k_{bemf}}{k_r k_s^2\left[(J_2/k_s^2) + 2m_2\right]}, \tag{36}$$

$$K_b = \frac{k_{da}k_{pa}k_t}{k_s k_r\left[(J_2/k_s^2) + 2m_2\right]}. \tag{37}$$

We can conclude that both the unbalanced wedge angle θ (the unit is radians) and the offset of the sliding weights x (the unit is meters) contribute to the unbalanced torque of the wedge.

Similarly, both the angular velocity of the wedge, $\dot{\theta}$, and the velocity of the sliding weights, $\dot{x}$, contribute to the rate change of the unbalanced torque. That is, θ and x have similar effects on the wedge dynamics, as do $\dot{\theta}$ and $\dot{x}$.

Therefore, it is reasonable to choose the input variables T and dT

$$T = K_1\theta + K_2 x, \tag{38}$$

$$dT = K_3\dot{\theta} + K_4\dot{x}, \tag{39}$$

where T is an approximation of the unbalanced torque and dT is an approximation of the rate change of the unbalanced torque. A suitable choice for the values of K_1, K_2, K_3, and K_4 is

$$K_1 = K_3 = Mgh, \tag{40}$$

$$K_2 = K_4 = 2mg\sin\alpha, \tag{41}$$

where $M = 3.8$, $h = 0.13$, $m = 0.78$, $\alpha = 45$. In this manner, the number of input variables is reduced to two dimensions, as shown in Figure 6 (G_1, G_2, and G_3 are constants).

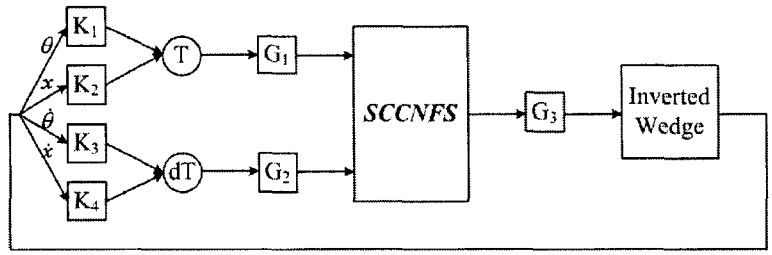

Figure 6. The inverted wedge SCCNFS control system for two dimensions.

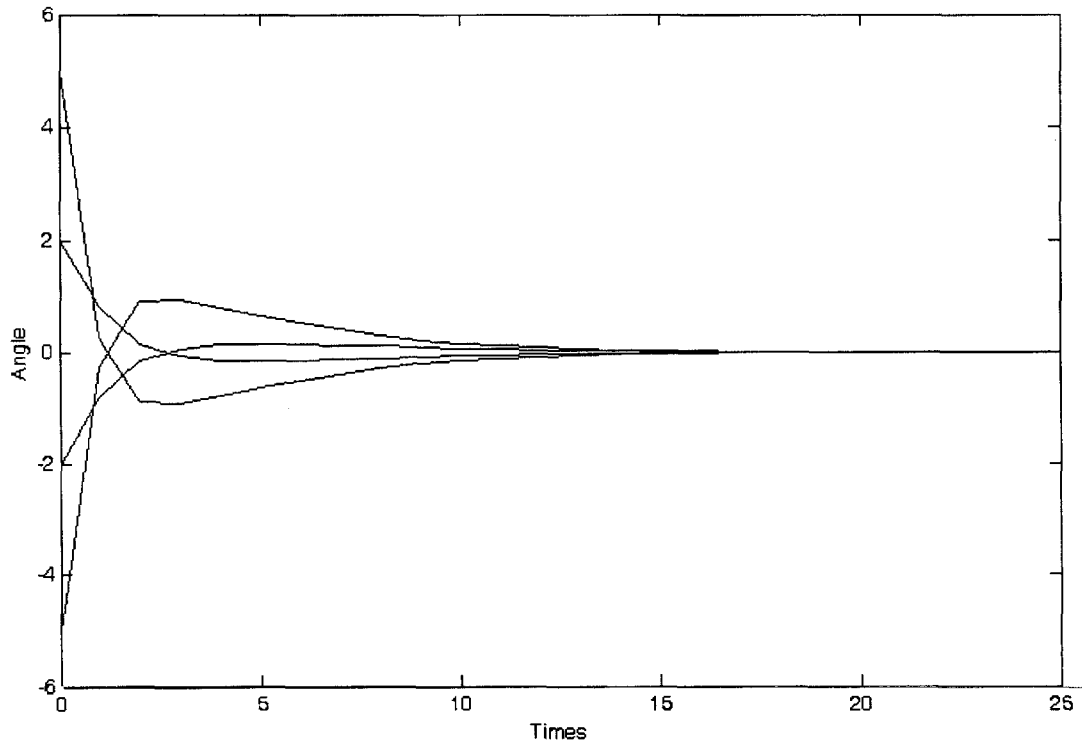

Figure 7. The responses of the closed-loop inverted wedge system controlled by the SCCNFS model for four initial conditions.

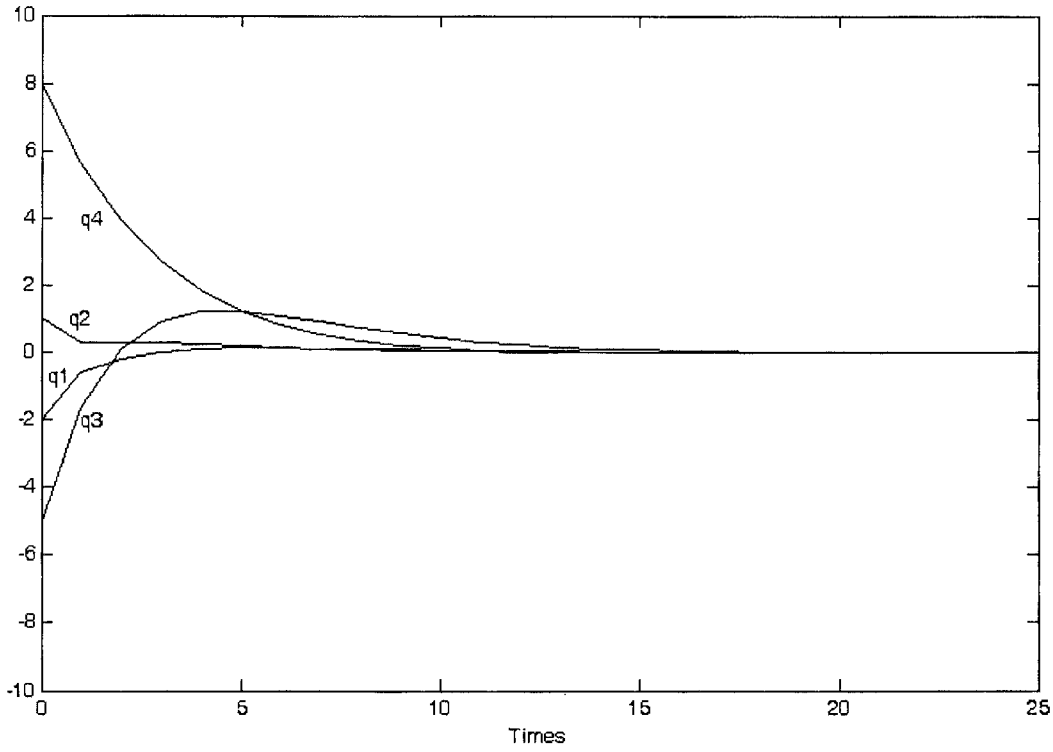

Figure 8. The responses of the four states of the inverted wedge system under the control of the learned SCCNFS controller.

In our simulation, we used the linear quadratic optimal design approach [12] to generate the input/output training pair. The learning rate $\eta_w = \eta_c = \eta_d = \eta_m = \eta_\sigma = 0.05$, and the prespecified threshold $\bar{F} = 10^{-8}$ were used. After the online structure-parameter learning, ten fuzzy logic rules were generated in our simulation. The rms error of the controller approximated 0.1001. We tested the learned controller under four initial conditions: $q(0) = [5, 8, -5, -16]^\top$, $[-2, 1, -5, 8]^\top$, $[2, -1, 5, -8]^\top$, and $[-5, -8, 5, 16]^\top$. Figure 7 shows the output responses of the inverted wedge system controlled by the SCCNFS model. The behavior of the four states of the inverted wedge system starting at the initial condition $[-2, 1, -5, 8]^\top$ are shown in Figure 8. In this figure, the four states of the system gradually decay to zero. The results show the perfect control capability of the trained SCCNFS model. We also compared the performance of SCCNFS with FALCON [4]. There are 33 fuzzy rules generated in the FALCON model. The results show that the proposed SCCNFS needs fewer fuzzy rules than the FALCON model.

5. CONCLUSIONS

In this paper, a self-constructing compensatory neural fuzzy system (SCCNFS) was proposed. The structure and parameter learning phases are done concurrently and online in the SCCNFS. The compensatory neural fuzzy system with adaptive fuzzy reasoning is more effective and adaptive than the conventional neural fuzzy system with nonadaptive fuzzy reasoning. The simulation results show that a trained SCCNFS model has good identification and control capability.

REFERENCES

1. C.T. Lin and C.S.G. Lee, *Neural Fuzzy Systems: A Neuro-Fuzzy Synergism to Intelligent System*, Prentice Hall, New Jersey, (1996).
2. S. Paul and S. Kumar, Subsethood-product fuzzy neural inference system (SuPFuNIS), *IEEE Trans. on Neural Networks* **13** (3), 579–599, (2002).
3. T. Takagi and M. Sugeno, Fuzzy identification of systems and its applications to modeling and control, *IEEE Trans. on Syst., Man, Cybern.* **15**, 116–132, (1985).

4. C.J. Lin and C.T. Lin, An ART-based fuzzy adaptive learning control network, *IEEE Transactions on Fuzzy Systems* **5** (4), 477–496, (1997).
5. C.F. Juang and C.T. Lin, An online self-constructing neural fuzzy inference network and its applications, *IEEE Transactions on Fuzzy Systems* **6** (1), 12–31, (1998).
6. F.J. Lin, C.H. Lin and P.H. Shen, Self-constructing fuzzy neural network speed controller for permanent-magnet synchronous motor drive, *IEEE Transactions on Fuzzy Systems* **6** (5), 751–759, (2001).
7. H. Pomares, I. Rojas, J. Gonzalez and A. Prieto, Structure identification in complete rule-based fuzzy systems, *IEEE Trans. on Fuzzy Systems* **10** (3), 349–359, (2002).
8. C.H. Lee and C.C. Teng, Identification and control of dynamic systems using recurrent fuzzy neural networks, *IEEE Trans. on Fuzzy Systems* **8** (4), 349–366, (2000).
9. P.A. Mastorocostas and J.B. Theocharis, A recurrent fuzzy-neural model for dynamic system identification, *IEEE Trans. on System, Man, and Cybernetics* **32** (2), 176–190, (2002).
10. J.-S.R. Jang, C.T. Sun and E. Miszutani, *Neuro-Fuzzy and Soft Computing*, Prentice Hall, Upper Saddle River, NJ, (1997).
11. Y.Q. Zhang and A. Kandel, Compensatory neurofuzzy systems with fast learning algorithms, *IEEE Transactions on Neural Networks* **9** (1), 83–105, (1998).
12. P. Hsu, Dynamics and control design project offers taste of real world, *IEEE Control System Mag.*, 31–38, (1992).