Available online at www.sciencedirect.com
ScienceDirect
Information Sciences 177 (2007) 264–280
INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL
www.elsevier.com/locate/ins

A self-constructing fuzzy CMAC model and its applications

Chi-Yung Lee [a], Cheng-Jian Lin [b,*], Huei-Jen Chen [b]

[a] Department of Computer Science and Information Engineering, Nankai Institute of Technology, Nantou 542, Taiwan, ROC
[b] Department of Computer Science and Information Engineering, Chaoyang University of Technology, Taichung 413, Taiwan, ROC

Received 7 April 2004; received in revised form 8 February 2006; accepted 20 March 2006

Abstract

This work presents a self-constructing fuzzy cerebellar model articulation controller (SC-FCMAC) model for various applications. A self-constructing learning algorithm, which consists of the self-clustering method (SCM) and the back-propagation algorithm, is presented. The proposed SCM scheme is a rapid, one-pass algorithm which dynamically estimates the number of hypercube cells in input data space. The clustering method does not require *prior* knowledge, such as the number of clusters in a data set. The back-propagation algorithm is applied to tune the adjustable parameters. Simulation results are obtained to show the performance and applicability of the proposed model.

Keywords: Cerebellar model articulation controller (CMAC); Clustering; Back-propagation; Chaotic; Classification; Control

1. Introduction

The cerebellar model articulation controller (CMAC) [2,3], developed by Albus, is a simple network architecture which provides the advantages of fast learning and a high convergence rate. The CMAC model has been successfully applied to various fields, such as robot control [11,29,30], signal processing [19], pattern recognition [9], and diagnosis [13,36]. However, Albus' CMAC model has three major limitations [2,3]. First, the CMAC model requires a very large amount of memory to solve high-dimensional problems [30]. Therefore, the choice of the clustering approach in the CMAC model is important because *partition-based* clustering methods, including fuzzy *C*-means (FCM) [14,24,28], linear vector quantization (LVQ) [1,12], fuzzy Kohonen partitioning (FKP), and pseudo-FKP [4], are used in cluster analysis. However, the above clustering techniques depend on *prior* knowledge, such as the number of clusters in a data set. Online-based cluster techniques [17,34] have been presented to solve this problem. However, these methods [17,34] have a problem: they consider only the total variations of the mean and deviation in all dimensions per input because the number of clusters increases rapidly.

* Corresponding author. Fax: +886 4 2374 2375.
E-mail address: cjlin@mail.cyut.edu.tw (C.-J. Lin).

0020-0255/$ - see front matter
doi:10.1016/j.ins.2006.03.010

The second limitation of the CMAC model is that it requires a more rigorous theory for function approximation. Thus, several modifications of Albus' CMAC model using B-spline functions [37] or fuzzy concepts [5,7,10,18,23,38] have been made to improve its approximation capability. The basis function in the CMAC model is a binary box function. Input vectors that fall into the same block will produce the same output from the network. Therefore, the CMAC model's output is usually not as smooth as the target function. B-spline basis functions have been developed to enable the different input vectors that fall in the same block to generate different outputs in the basis function layer [37]. Many researchers have integrated the fuzzy concept into the CMAC network [5,10,18,23,38]. They use membership functions rather than basis functions, and the resulting structure is called fuzzy CMAC (FCMAC).

The third limitation is the difficulty with which the CMAC model selects the parameters of the memory structure [6,15,22]. Chow and Menozzi [6] proposed a self-organizing CMAC network controller based on competitive learning. This method uses Kohonen's concept of adapting the CMAC configuration to match the input distribution. Hu and Pratt [15] applied a clustering technique to reduce the memory required by the CMAC network. They successfully demonstrated the self-organizing CMAC network and the Lyapunov function-based unsupervised learning scheme. Lee et al. [22] proposed a self-organizing input space module that employed Shannon's entropy measure and the golden-section search method to quantize the input space according to the various distributions of the training data sets. These methods [6,15,22] all involve off-line learning.

This work presents a new self-constructing fuzzy cerebellar model articulation controller (SC-FCMAC) for identification, classification, and control problems. A Gaussian basis function is used to model a hypercube structure and its fuzzy weight. A self-constructing learning algorithm, which consists of the input space partition scheme and the parameter-learning scheme, is proposed for constructing and training the proposed SC-FCMAC model. The input space partition scheme is based on the self-clustering method (SCM) to determine the appropriate distributions of the input training data. The proposed SCM does not require *prior* knowledge, such as the number of clusters in a data set. A gradient-descent learning algorithm is applied to adjust simultaneously the free parameters in the receptive field functions and the fuzzy weights to minimize the output error function. Moreover, the proposed SC-FCMAC model can effectively solve problems associated with the required memory and the ability of function approximation. Simulation results demonstrate that the proposed SC-FCMAC model outperforms some of the other existing models.

2. The structure of the SC-FCMAC model

2.1. The traditional CMAC model

The traditional CMAC model [2] has fast learning ability and good local generalization capability for approximating nonlinear functions. The basic idea for using the CMAC model is to store learned data in overlapping regions in a way that the data can easily be recalled yet use less storage space. The action of storing weight information in the CMAC model is similar to that of the cerebellum in humans. Take a two-dimensional (2-D) input vector, or the so-called two-dimensional CMAC (2-D CMAC), as an example. The structure of a 2-D CMAC is shown in Fig. 1. The input vector is defined by two input variables, s_1 and s_2, which are quantized into three discrete regions, called blocks. It is noted that the width of the blocks affects the generalization capability of the CMAC. In the first method of quantization, the variable s_1 is divided into blocks A, B, and C, and the variable s_2 is divided into blocks a, b, and c. The areas Aa, Ab, Ac, Ba, Bb, Bc, Ca, Cb, and Cc formed by quantized regions are called hypercubes. When each block is shifted by a small interval, different hypercubes can be obtained. In Fig. 1, there are 27 hypercubes used to distinguish 49 different states in the 2-D CMAC. For example, let the hypercubes Bb, Ee, and Hh be addressed by the state $(s_1, s_2) = (3, 3)$. Only these three hypercubes are set to 1, and the others are set to 0.

2.2. The self-constructing fuzzy CMAC (SC-FCMAC) model

In this paper, we propose a self-constructing fuzzy CMAC (SC-FCMAC) model. The SC-FCMAC model [26], illustrated in Fig. 2, consists of the input space partition, association memory selection, and

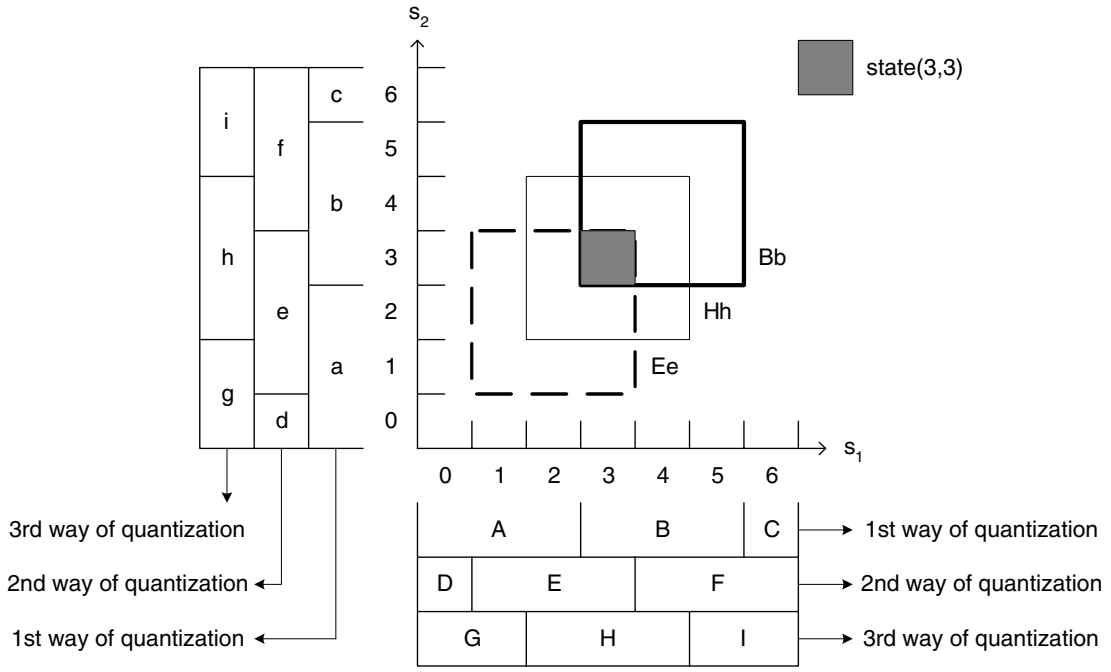


Fig. 1. The structure of a 2-D CMAC.

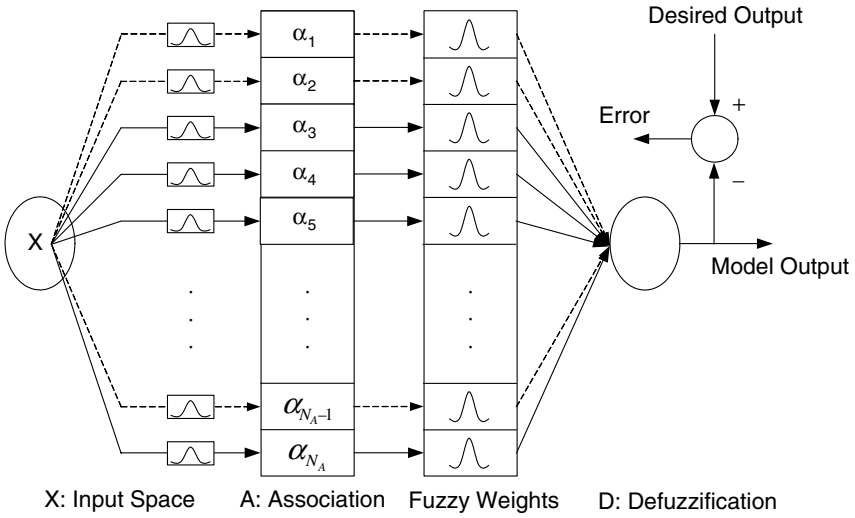


Fig. 2. The structure of the SC-FCMAC model.

defuzzification. The SC-FCMAC model is like the traditional CMAC model that approximates a nonlinear function $y = f(x)$ by using two primary mappings:

$$S : X \Rightarrow A \tag{1}$$

$$P : A \Rightarrow D \tag{2}$$

where X is an s-dimensional input space, A is an N_A-dimensional association space, and D is a 1-D output space. These two mappings are realized by using fuzzy operations. The function $S(x)$ maps each point x in the input space onto an association vector $\alpha = S(x) \in A$ that has N_L nonzero elements ($N_L < N_A$). Here,

$\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_{N_A})$, where $0 \leqslant \alpha \leqslant 1$ for all components in α is derived from the composition of the receptive field functions and sensory inputs. Different from the traditional CMAC model, several hypercubes are addressed by the input state x. The hypercube values are calculated by product operation through the strength of the receptive field functions for each input state.

In the SC-FCMAC model, we use the Gaussian basis function as the receptive field function and the fuzzy weight function for learning. Some learned information is stored in the fuzzy weight vector. The 1-D Gaussian basis function can be given as follows:

$$\mu(x) = \mathrm{e}^{-((x-m)/\sigma)^2} \tag{3}$$

where x represents the specific input state, m represents the corresponding center, and σ represents the corresponding variance.

Let us consider a N_D-dimensional problem. A Gaussian basis function with N_D dimensions is given as follows:

$$\alpha_j = \prod_{i=1}^{N_D} \mathrm{e}^{-((x_i - m_{ij})/\sigma_{ij})^2} \tag{4}$$

where $\prod$ represents the *product* operation, α_j represents the jth element of the association memory selection vector, x_i represents the input value of the ith dimension for a specific input state x, m_{ij} represents the center of the receptive field functions, σ_{ij} represents the variance of the receptive field functions, and N_D represents the number of the receptive field functions for each input state. The function $P(\alpha)$ computes a scalar output y by projecting the association memory selection vector onto a vector of adjustable fuzzy weights. Each fuzzy weight is inferred to produce a partial fuzzy output using the value of its corresponding association memory selection vector as the input matching degree. The fuzzy weight is considered here so that the partial fuzzy output is defuzzified into a scalar output using standard volume-based centroid defuzzification [21,33]. The term volume is used in a general sense to include multi-dimensional functions. For 2-D functions, the volume reduces to the area. If v_j is the volume of the consequent set and ξ_j is the weight of the scale α_j, then the general expression for defuzzification is

$$y = \frac{\sum_{j=1}^{N_L} \alpha_j w_j^m v_j \xi_j}{\sum_{j=1}^{N_L} \alpha_j v_j \xi_j} \tag{5}$$

where w_j^m is the mean value of the fuzzy weights and N_L is the number of hypercube cells. The volume v_j in this case is simply the area of the consequent weights, which are represented by Gaussian fuzzy sets. Therefore, $v_j = w_j^\sigma \sqrt{\pi}$, where w_j^σ represents the variance of the fuzzy weights. If the weight ξ_j is considered to be one, as in this work, then the actual output y is derived as follows:

$$y = \frac{\sum_{j=1}^{N_L} \alpha_j w_j^m w_j^\sigma}{\sum_{j=1}^{N_L} \alpha_j w_j^\sigma} \tag{6}$$

3. A self-constructing learning algorithm for the SC-FCMAC model

In this section, a self-constructing learning algorithm, which consists of an input space partition scheme and a parameter-learning scheme, is presented for constructing the SC-FCMAC model. First, the input space partition scheme is used to determine proper input space partitioning and to find the mean and the width of each receptive field function. This scheme is based on the self-clustering method (SCM) to appropriately determine the various distributions of the input training data. Second, the parameter-learning scheme is based on the gradient descent learning algorithm. To minimize a given cost function, the receptive field functions and the fuzzy weights are adjusted using the back-propagation algorithm. According to the requirements of the system, these parameters will be given proper values to represent the memory information. For the initial system, the values of the tuning parameters w_j^m and w_j^σ of the fuzzy weights are generated randomly, and the m and σ of the receptive field functions are generated by the proposed SCM clustering method.

3.1. The input space partition scheme

The receptive field functions can map input patterns. Hence, the discriminative ability of these new features is determined by the centers of the receptive field functions. To achieve good classification, centers are best selected based on their ability to provide large class separation.

An input space partition scheme, called the self-clustering method (SCM), is proposed to implement scatter partitioning of the input space. Without any optimization, the proposed SCM is a fast, one-pass algorithm for a dynamic estimation of the number of hypercube cells in a set of data, and for finding the current centers of hypercube cells in the input data space. It is a distance-based connectionist-clustering algorithm. In any hypercube cell, the maximum distance between an example point and the hypercube cell center is less than a threshold value, which has been set as a clustering parameter and which would affect the number of hypercube cells to be estimated.

In the clustering process, the data examples come from a data stream, and the process starts with an empty set of hypercube cells. When a new hypercube cell is created, the hypercube cell center, C, is defined, and its hypercube cell distance and hypercube cell width, Dc and Wd, respectively, are initially set to zero. When more samples are presented one after another, some created hypercube cells will be updated by changing the positions of their centers and increasing the hypercube cell distances and hypercube cell width. Which hypercube cell will be updated and how much it will be changed depends on the position of the current example in the input space. A hypercube cell will not be updated any more when its hypercube cell distance, Dc, reaches the value that is equal to the threshold value D_{thr}.

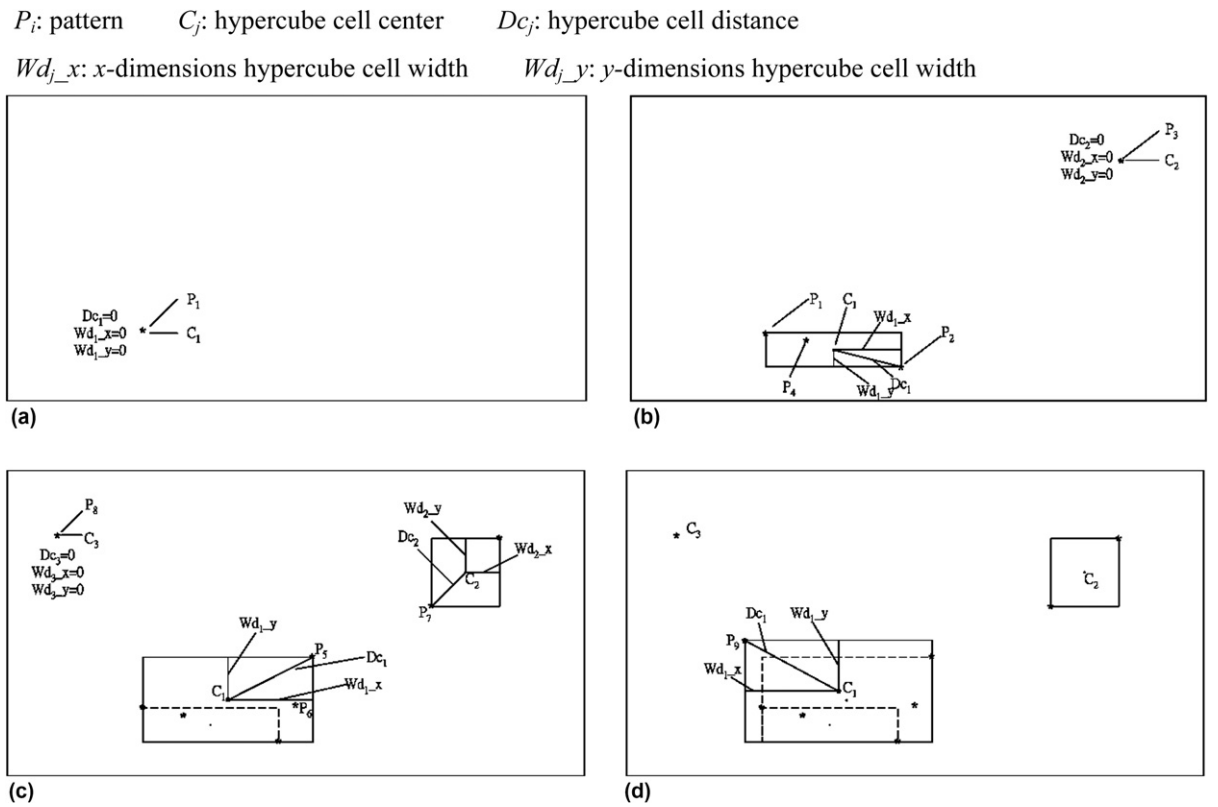


Fig. 3. A brief clustering process using the SCM with samples P_1 to P_9 in a 2-D space. (a) The example P_1 causes the SCM to create a new hypercube cell center C_1. (b) P_2: update hypercube cell center C_1, P_3: create a new hypercube cell center C_2, P_4: do nothing. (c) P_5: update hypercube cell C_1, P_6: do nothing, P_7: update hypercube cell center C_2, P_8: create a new hypercube cell C_3. (d) P_9: update hypercube cell C_1.

Fig. 3 shows a brief clustering process using the SCM in a two-input space. The detailed clustering process is as follows:

Step 1: Create the first hypercube cell by taking the position of the first example from the input stream as the first hypercube cell center C_1 and setting its hypercube cell distance Dc_1 and hypercube cell width $Wd_{1\_}x$ and $Wd_{1\_}y$ to zero, as shown in Fig. 3(a).

Step 2: If all examples of the data stream have been processed, the algorithm is finished. Otherwise, the current input example P_i is taken and the distances between this example and all n already created hypercube cell centers C_j, $Dist_{ij} = \|P_i - C_j\|$, $j = 1,2,\ldots,n$, are calculated.

Step 3: If there is any distance value $Dist_{ij}$ equal to, or less than, at least one of the distance Dc_j, $j = 1,2,\ldots,n$, it means that the current example P_i belongs to a hypercube cell C_m with the minimum distance

$$Dist_{im} = \|P_i - C_m\| = \min(\|P_i - C_j\|), \quad j = 1,2,\ldots,n \tag{7}$$

In this case, neither a new hypercube cell is created, nor any existing hypercube cell is updated, as in the cases of P_4 and P_6 shown in Fig. 3, for example. The algorithm then returns to Step 2. Otherwise, go to the next step.

Step 4: Find a hypercube cell with center C_m and hypercube cell distance Dc_m from all n existing hypercube cell centers by calculating the values $S_{ij} = Wd_{ij} + Dc_j$, $j = 1,2,\ldots,n$, and then choosing the hypercube cell center C_m with the minimum value S_{im}:

$$S_{im} = Wd_{im} + Dc_m = \min(S_{ij}), \quad j = 1,2,\ldots,n \tag{8}$$

In Eq. (7), the maximum distance from any hypercube cell mean to the examples that belong to this hypercube cell is not greater than the threshold D_{thr}, though the algorithm does not keep any information on passed examples. However, we find that the formulation only considers the distance between the input data and the hypercube cell mean in Eq. (8). This special situation indicates that the distances between the given point P_{11} and both hypercube cell means Dc_1 and Dc_2 are the same, as shown in Fig. 4. In the aforementioned techniques [17,34], the hypercube cell C_2, which has small dimension distances D_{2_x}, will be selected for expansion according to Eq. (8). However, this causes the problem of the hypercube cell numbers increasing quickly. To avoid this problem, we give the following rule:

If (the distance between P_{11} and Dc_1 is equal to the distance between P_{11} and Dc_2) and ($D_{1\_x} > D_{2\_x}$), then $D_{im} = Dc_1$.

In the above rule, we find that when the distances between the input data and both hypercube cells are the same, the formulation will choose the hypercube cell that has large dimension distances D_{1_x}.

Step 5: If S_{im} is greater than D_{thr}, the example P_i does not belong to any existing hypercube cells. A new hypercube cell is created in the same way as described in Step 1, as in the cases of P_3 and P_8 shown in Fig. 3, and the algorithm returns to Step 2.

Step 6: If S_{im} is not greater than D_{thr}, the hypercube cell is updated by moving its center C_m and increasing the value of its hypercube cell distance Dc_m and hypercube cell width $Wd_{m\_}x$ and $Wd_{m\_}y$. The parameters are updated using the following equations:

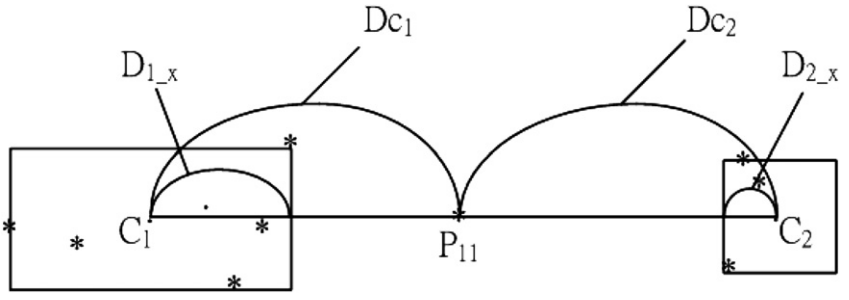


Fig. 4. The special case of the same distances between input data and both hypercube cells.

$$Wd_m\_x^{\text{new}} = (\|C_m\_x - P_i\_x\| + Wd_m\_x)/2 \tag{9}$$

$$Wd_m\_y^{\text{new}} = (\|C_m\_y - P_i\_y\| + Wd_m\_y)/2 \tag{10}$$

$$C_m\_x^{\text{new}} = P_i\_x - D_m\_x^{\text{new}} \tag{11}$$

$$C_m\_y^{\text{new}} = P_i\_y - D_m\_y^{\text{new}} \tag{12}$$

$$Dc_m^{\text{new}} = S_{im}/2 \tag{13}$$

where C_m_x is a value of the x dimension for C_m, C_m_y is a value of the y dimension for C_m, P_i_x is a value of the x dimension for P_i, and P_i_y is a value of the y dimension for P_i, as in the cases of P_2, P_5, P_7, and P_9 shown in Fig. 3. The algorithm returns to Step 2.

In this way, the maximum distance from any hypercube cell center to the examples that belong to this hypercube cell is not greater than the threshold value D_{thr}, though the algorithm does not keep any information on passed examples. The center and the jump positions of the receptive field functions are then defined by the following equation:

$$m_j = C_j, \quad j = 1, 2, \ldots, n \tag{14}$$

$$\theta_j^r = \frac{1}{((n_s+1)/2)} \cdot r \cdot D_j, \quad r = 1, 2, \ldots, n_s, \;\; j = 1, 2, \ldots, n \tag{15}$$

The threshold parameter D_{thr} is an important parameter in the input space partition scheme. A low threshold value leads to the learning of fine clusters (such that many hypercube cells are generated), whereas a high threshold value leads to the learning of coarse clusters (such that fewer hypercube cells are generated). Therefore, the selection of the threshold value D_{thr} critically affects the simulation results, and the threshold value is determined by practical experimentation or trial-and-error tests.

3.2. The parameter-learning scheme

In the parameter-learning scheme, there are four adjustable parameters (m_{ij}, σ_{ij}, w_j^m, and w_j^σ) that need to be tuned. The parameter-learning algorithm of the SC-FCMAC model uses the supervised gradient descent method to modify these parameters. When we consider the single output case for clarity, our goal is to minimize the cost function E, defined as follows:

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

where $y^d(t)$ denotes the desired output at time t and $y(t)$ denotes the actual output at time t. The parameter-learning algorithm, based on back-propagation, is as follows.

The fuzzy weight cells are updated according to the following equations:

$$w_j^m(t+1) = w_j^m(t) + \Delta w_j^m \tag{17}$$

$$w_j^\sigma(t+1) = w_j^\sigma(t) + \Delta w_j^\sigma \tag{18}$$

where j denotes the jth fuzzy weight cell for $j = 1, 2, \ldots, N_L$, w_j^m denotes the mean of the fuzzy weights, and w_j^σ denotes the variance of the fuzzy weights. The elements of the fuzzy weights are updated by the amount

$$\Delta w_j^m = \eta \cdot e \cdot \frac{\partial y}{\partial w_j^m} = \eta \cdot e \cdot \frac{\alpha_j w_j^\sigma}{\sum_{j=1}^{N_L} \alpha_j w_j^\sigma} \tag{19}$$

$$\Delta w_j^\sigma = \eta \cdot e \cdot \frac{\partial y}{\partial w_j^\sigma} = \eta \cdot e \cdot \frac{\alpha_j w_j^m \sum_{j=1}^{N_L} \alpha_j w_j^\sigma - \alpha_j \sum_{j=1}^{N_L} \alpha_j w_j^m w_j^\sigma}{\left(\sum_{j=1}^{N_L} \alpha_j w_j^\sigma\right)^2} \tag{20}$$

where η is the learning rate of the mean and the variance for the fuzzy weight functions between 0 and 1, and e is the error between the desired output and the actual output, $e = y^d - y$.

The receptive field functions are updated according to the following equations:

$$m_{ij}(t+1) = m_{ij}(t) + \Delta m_{ij} \tag{21}$$

$$\sigma_{ij}(t+1) = \sigma_{ij}(t) + \Delta \sigma_{ij} \tag{22}$$

where i denotes the ith input dimension for $i = 1, 2, \ldots, n$, m_{ij} denotes the mean of the receptive field functions, and σ_{ij} denotes the variance of the receptive field functions.

The parameters of the receptive field functions are updated by the amount

$$\begin{aligned}\Delta m_{ij} &= \eta \cdot e \cdot \frac{\partial y}{\partial \alpha_j} \cdot \frac{\partial \alpha_j}{\partial m_{ij}} \\ &= \eta \cdot e \cdot \frac{w_j^m w_j^\sigma \sum_{j=1}^{N_L} \alpha_j w_j^\sigma - w_j^\sigma \sum_{j=1}^{N_L} \alpha_j w_j^m w_j^\sigma}{\left(\sum_{j=1}^{N_L} \alpha_j w_j^\sigma\right)^2} \cdot \alpha_j \cdot \frac{2(x_i - m_{ij})}{\sigma_{ij}^2}\end{aligned} \tag{23}$$

$$\begin{aligned}\Delta \sigma_{ij} &= \eta \cdot e \cdot \frac{\partial y}{\partial \alpha_j} \cdot \frac{\partial \alpha_j}{\partial \sigma_{ij}} \\ &= \eta \cdot e \cdot \frac{w_j^m w_j^\sigma \sum_{j=1}^{N_L} \alpha_j w_j^\sigma - w_j^\sigma \sum_{j=1}^{N_L} \alpha_j w_j^m w_j^\sigma}{\left(\sum_{j=1}^{N_L} \alpha_j w_j^\sigma\right)^2} \cdot \alpha_j \cdot \frac{2(x_i - m_{ij})^2}{\sigma_{ij}^3}\end{aligned} \tag{24}$$

where η is the learning rate of the mean and the variance for the receptive field functions.

4. Simulation results

In this section, we compare the performance of the SC-FCMAC model with other models in three applications: learning chaotic behaviors [39], classifying Iris data sets [8], and controlling the truck backer-upper [31].

4.1. Example 1: Learning chaotic behaviors

A nonlinear system $y(t)$ with chaotic behaviors is given in the equations below:

$$\dot{x}_1(t) = -x_1(t)x_2^2(t) + 0.999 + 0.42\cos(1.75t) \tag{25}$$

$$\dot{x}_2(t) = x_1(t)x_2^2(t) - x_2(t) \tag{26}$$

$$y(t) = \sin(x_1(t) + x_2(t)) \tag{27}$$

We solved the differential equations (25) and (26) with t from $t = 0$ to $t = 20$ and with $x_1(0) = 1.0$ and $x_2(0) = 1.0$. We obtained 107 values of $x_1(t)$ and $x_2(t)$ (the chaotic glycolytic oscillator [35]) and 107 values of $y(t)$. Fig. 5 shows $y(t)$, which is the desired function to be learned by the SC-FCMAC model.

The input data were $x_1^p(t)$ and $x_2^p(t)$, and the output data was $y^p(t)$, for $p = 1, 2, \ldots, 107$. For this chaotic problem, the initial parameters $\eta = 0.1$ and $D_{\text{thr}} = 1.3$ were chosen. First, using the SCA clustering method, we obtained three hypercube cells. The learning scheme then entered parameter learning using the back-propagation algorithm. The parameter training process continued for 200 epochs, and the final trained root mean square (rms) error was 0.000474. The number of training epochs is determined by practical experimentation or trial-and-error tests.

We compared the SC-FCMAC model with other models [26,27]. Fig. 6(a) shows the learning curves of the SC-FCMAC model, the FCMAC model [26], and the SCFNN model [27]. As shown in this figure, the learning curve that resulted from our method has a lower rms error. Trajectories of the desired output $y(t)$ and the SC-FCMAC model's output are shown in Fig. 6(b)–(d). A comparison analysis of the SC-FCMAC model, the FCMAC model [26], and the SCFNN model [27] is presented in Table 1. It can be concluded that the proposed model obtains better results than some of the other existing models [26,27].

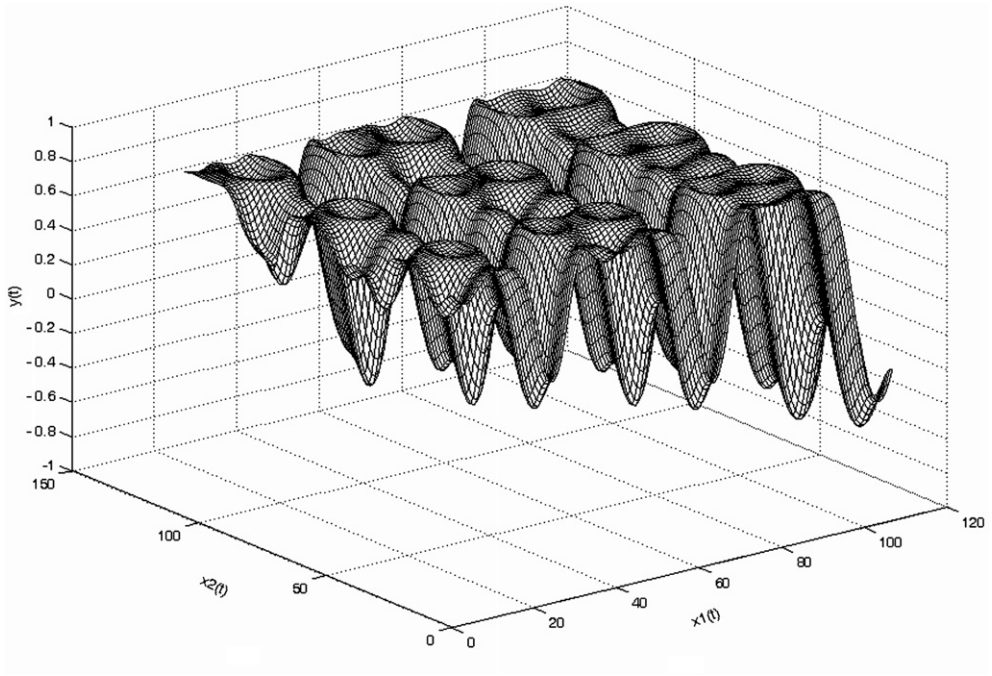


Fig. 5. $y(t) = \sin[x_1(t) + x_2(t)]$.

4.2. Example 2: Classification of Iris data

The Iris data set [8] consisted of 150 patterns associated with three output species – Iris Sestosa, Iris Versiolor, and Iris Virginica. Each species contained 50 cases. Each pattern comprised four input features – sepal length, sepal width, petal length, and petal width. The output y of the SC-FCMAC model was used to apply the following classification rule:

$$\text{Iris} = \begin{cases} \text{Sestosa} & \text{if } y < -0.5 \\ \text{Versiolor} & \text{if } -0.5 \leqslant y \leqslant 0.5 \\ \text{Virginica} & \text{if } y > 0.5 \end{cases} \tag{28}$$

The initial parameters $\eta = 0.01$ and $D_{\text{thr}} = 3.1$ were chosen to solve this classification problem. The experiment was repeated for 10 train-test data sets generated randomly from the original 150 Iris data. In each experiment, 25 instances of each species were randomly selected as the training set (a total of 75 training patterns were used) and the remaining instances were used as the testing set. The average number of hypercube cells was three.

Table 2 shows the results obtained using the 10 data sets in independent runs. The average training and testing accuracy rates obtained using the SC-FCMAC model were 99.7% and 97.04%. Cross validation was performed to check the performance of the proposed classifier. The results demonstrate that the proposed SC-FCMAC model yields similar average testing accuracy rates.

The performance of the proposed model was compared with that of other methods (CMAC [2], self-organizing HCMAC [22], and FCMAC [26]). The performance indices considered were the rms, the average training accuracy, the average testing accuracy, and the required memory. Table 3 presents a comparison of the results. The average testing accuracy rate of the presented method is higher and the mean memory requirement is lower (only about 30).

Several classification methods were compared with the proposed SC-FCMAC model in their application to the Iris data classification problem. Hisao Ishibuchi [16] considered the performance of 11 classification methods (70% training patterns and 30% testing patterns) and nine classification methods (50% training patterns

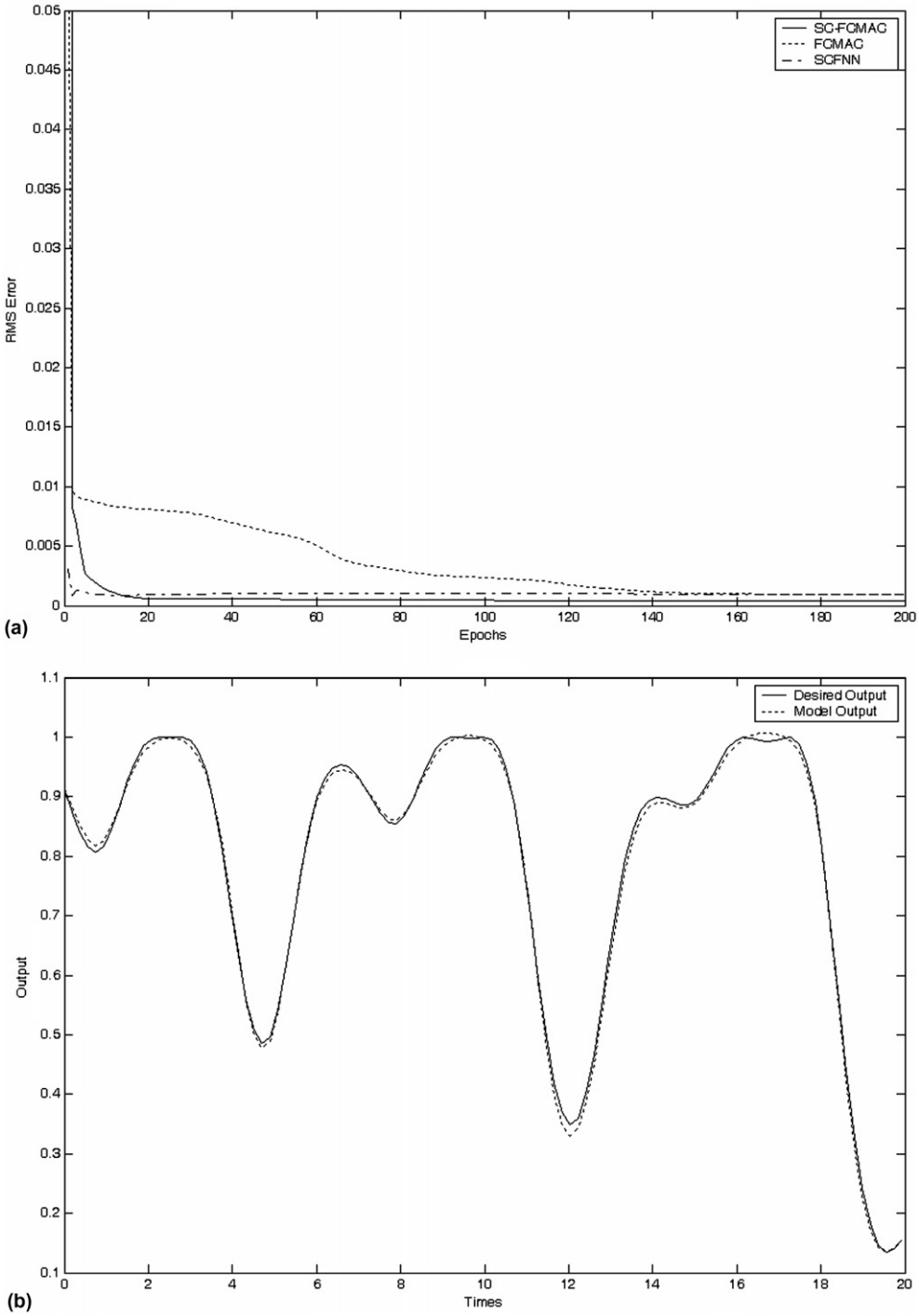
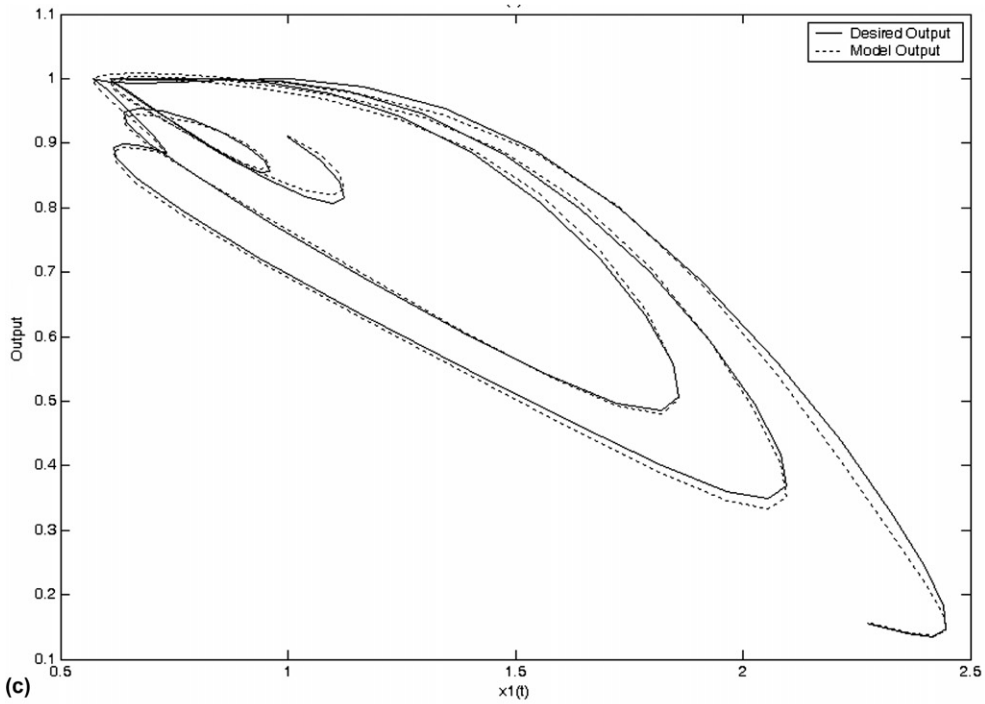


Fig. 6. Simulation results for learning chaotic behaviors. (a) Learning curves of the SC-FCMAC model, the FCMAC model [26], and the SCFNN model [27]. (b) The desired output $y(t)$ and the SC-FCMAC model's output for time t dimension. (c) The desired output $y(t)$ and the SC-FCMAC model's output for $x_1(t)$ dimension. (d) The desired output $y(t)$ and the SC-FCMAC model's output for $x_2(t)$ dimension.

and 50% testing patterns) when they were applied to the Iris data. This work considers a case of only 50% training patterns and 50% testing patterns. Table 4 provides the testing accuracy rates of other models



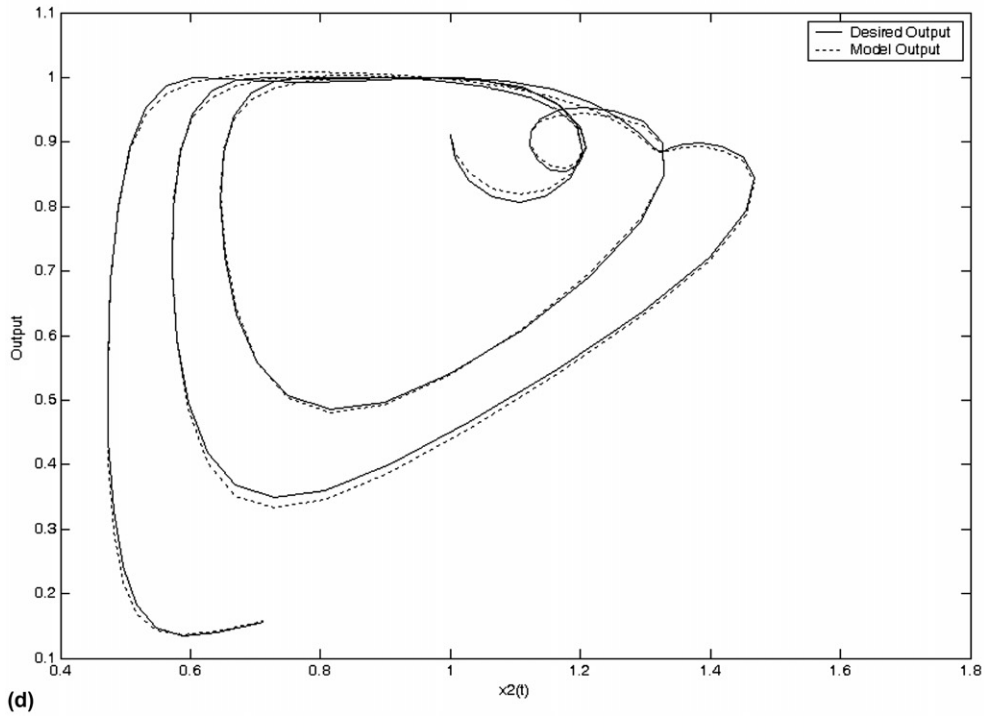


Fig. 6 (*continued*)

reported by Hisao Ishibuchi [16]. The proposed SC-FCMAC model provides the most accurate classification when applied to the Iris data in the random resampling procedure.

Table 1
Comparisons of the SC-FCMAC model with some existing models for dynamic system identification in Example 1

Items	Models		
	SC-FCMAC	FCMAC [26]	SCFNN [27]
Training steps	200	200	200
Parameters	18	18	20
Hypercube cells	3	4	N/A
RMS errors	0.000474	0.000885	0.000908

Table 2
Simulation results for independent runs

	Experiment #										
	1	2	3	4	5	6	7	8	9	10	Average
Testing accuracy (%)	97.3	97.3	97.3	97.3	96	97.3	96	97.3	97.3	97.3	97.04

Table 3
Simulation results on Iris data

Models	Items			
	RMS	Averaged training accuracy (%)	Averaged testing accuracy (%)	Memory requirement
CMAC [2]	0.014	99.7	95.7	9375
Self-organizing HCMAC [22]	0.014	99.6	96.8	555
FCMAC [26]	0.012	99.7	96.91	30
SC-FCMAC	0.012	99.7	97.04	30

Table 4
Classification accuracy of various methods on Iris data [16]

Classification methods	Testing accuracy rates (%)
Fuzzy integral with percetron criterion	95.3
Fuzzy integral with quadratic criterion	96.7
Minimum operator	96
Fast heuristic search with Sugeno integral	92
Simulated annealing with Sugeno integral	91.3
Fuzzy *K*-nearest neighbor	96.7
Fuzzy *C*-means	93.3
Fuzzy *C*-means for histograms	93.3
Hierarchical fuzzy *C*-means	95.3
Neural networks with the BP algorithm	96.7
SC-FCMAC	97.04

4.3. Example 3: Control for backing up the truck

Backing up a truck to a loading dock is difficult. It is a nonlinear control problem for which no traditional control methods exist [20,31]. The objective of this control problem was to use the backward motion of the truck to make the truck arrive at the loading dock at a right angle ($\phi_{\text{desired}} = 90°$) and to position the truck at the desired loading dock ($x_{\text{desired}}, y_{\text{desired}}$). The truck moves backward a fixed distance (d_f) according to how much the steering wheel turns at every step. The loading region was limited to the plane $[0, 100] \times [0, 100]$.

The input and output variables of the SC-FCMAC model must be specified. The model has two inputs, the truck angle ϕ and the cross position x. If enough clearance between the truck and the loading dock is assumed,

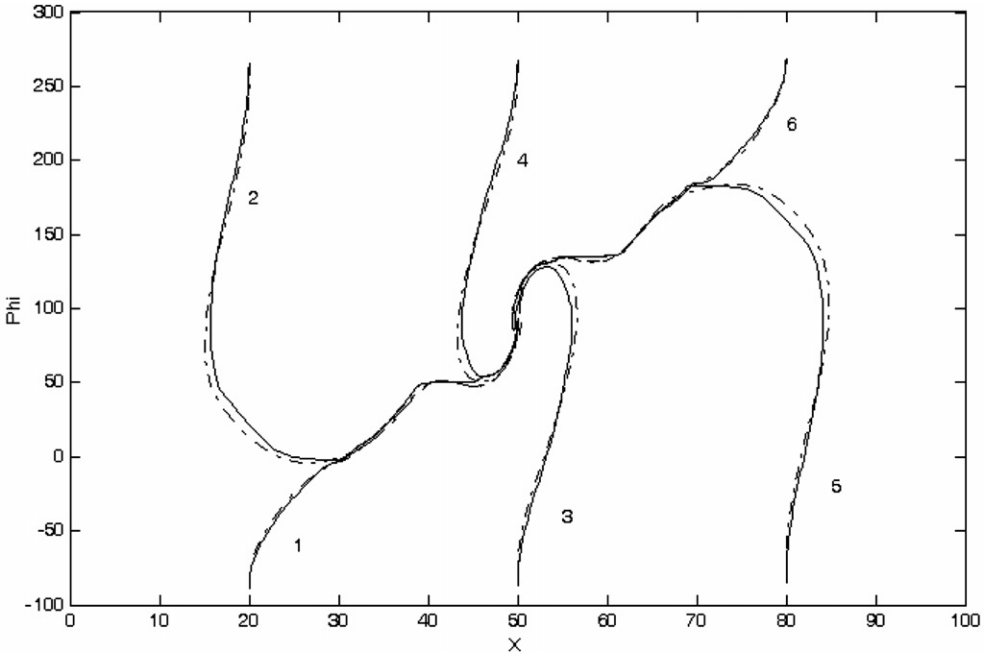


Fig. 7. The moving trajectories of the truck where the solid curves represent the six sets of training trajectories and the dashed curves represent the moving trajectories of the truck under the learned SC-FCMAC controller.

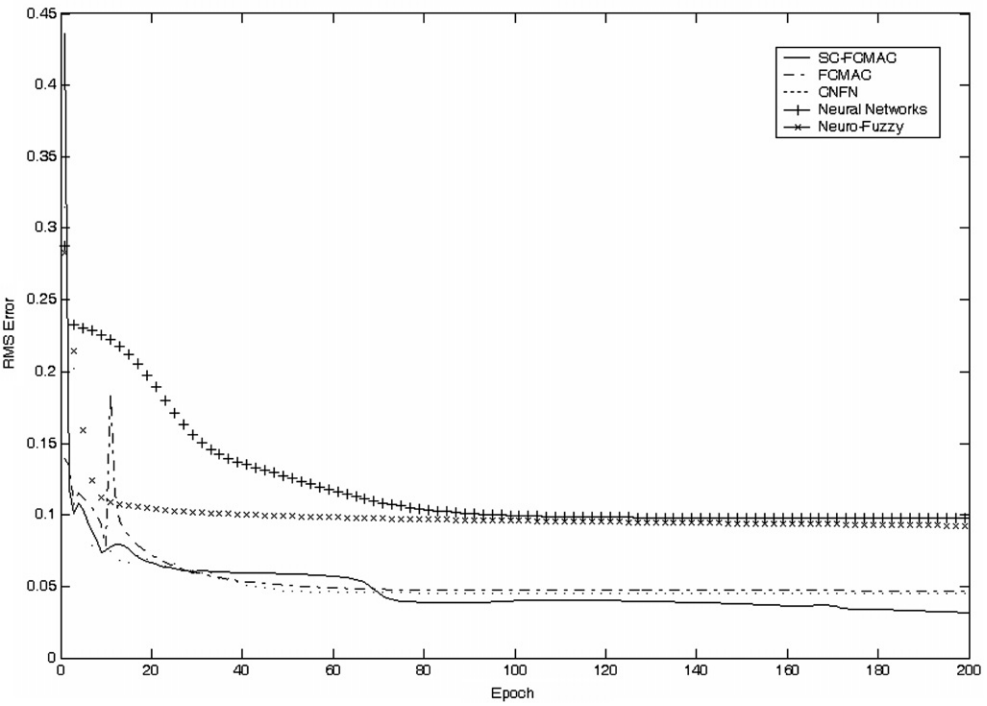


Fig. 8. Learning curves of some existing models in Example 3.

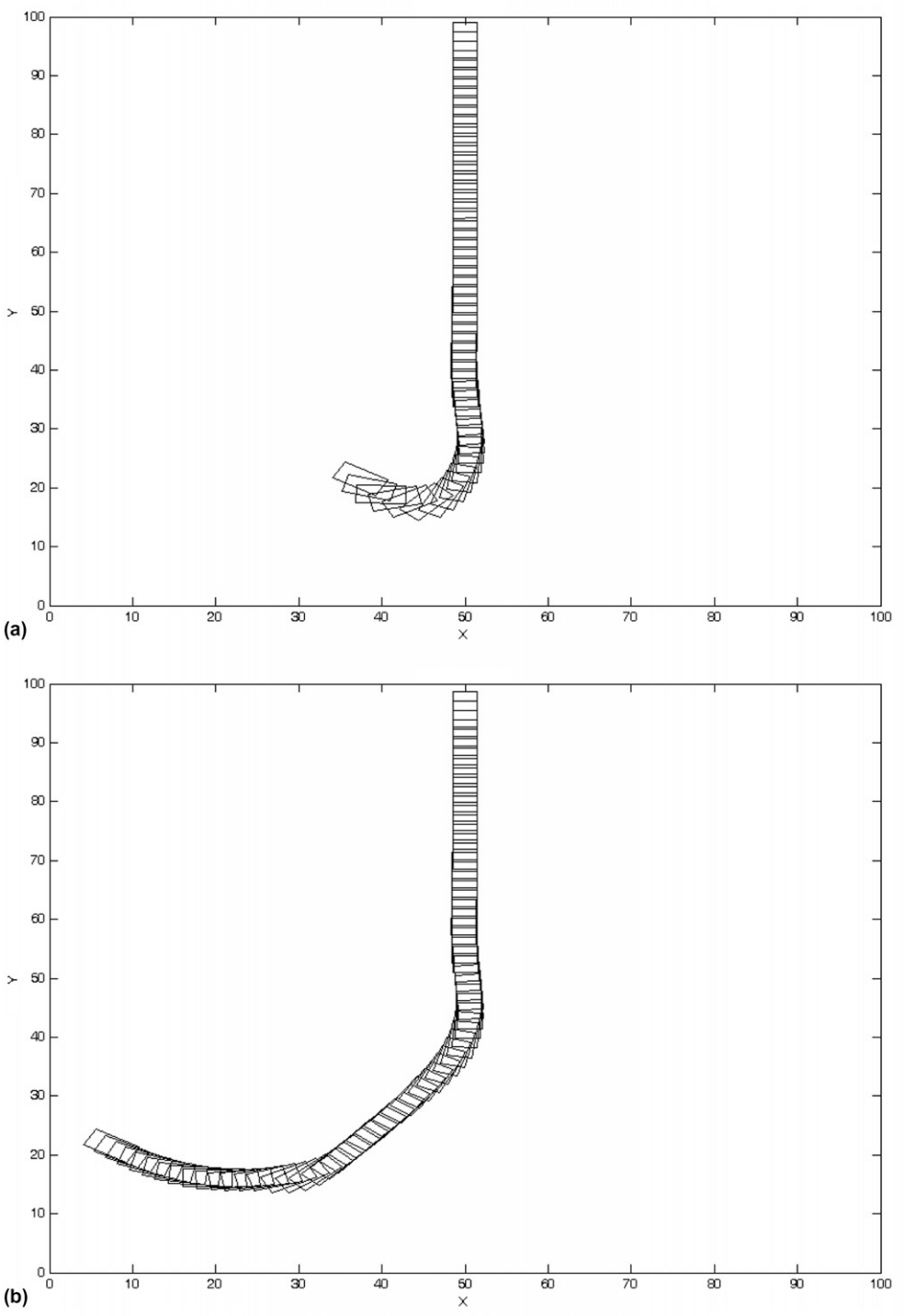


Fig. 9. Truck moving trajectories starting at three different initial positions under the control of the SC-FCMAC model after learning six sets of training trajectories. The initial positions $(x, y, \phi) =$ (a) $(40, 20, -30°)$, (b) $(10, 20, -30°)$, and (c) $(30, 20, -250°)$.

the y coordinate is not considered to be an input variable. The output of the model is the steering angle θ. The ranges of the variables x, ϕ and θ are as follows:

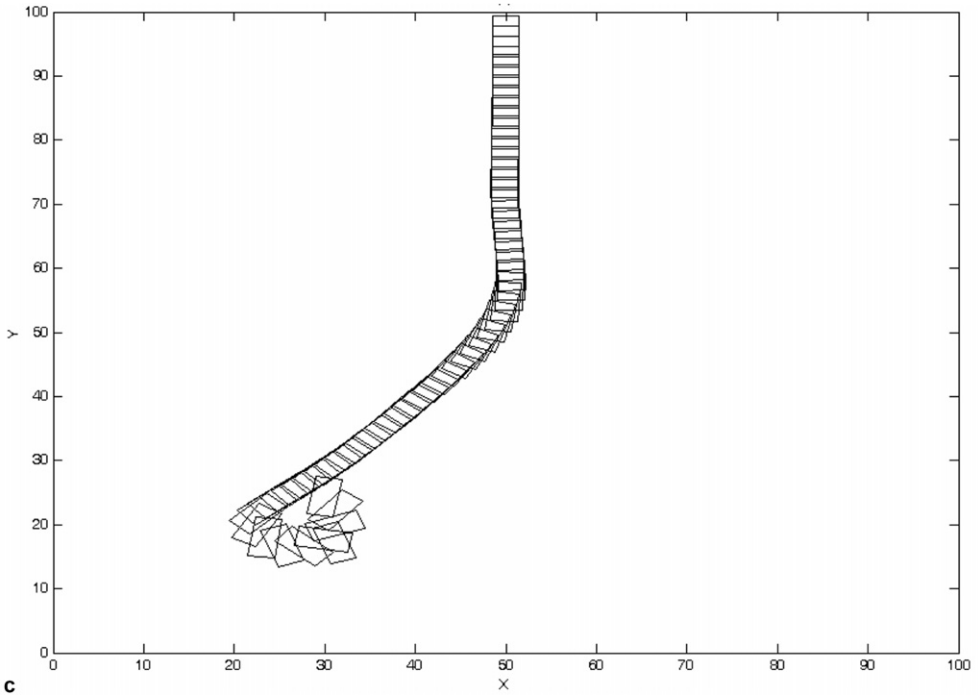


Fig. 9 (*continued*)

$$0 \leqslant x \leqslant 100 \tag{29}$$

$$-90^\circ \leqslant \phi \leqslant 270^\circ \tag{30}$$

$$-30^\circ \leqslant \theta \leqslant 30^\circ \tag{31}$$

The equations of the backward motion of the truck are given by

$$\begin{aligned} x(k+1) &= x(k) + d_f \cos\theta(k) + \cos\phi(k) \\ y(k+1) &= y(k) + d_f \cos\theta(k) + \sin\phi(k) \\ \phi(k+1) &= \tan^{-1}\left\{\frac{l\sin\phi(k) + d_f\cos\phi(k)\sin\theta(k)}{l\cos\phi - d_f\sin\phi(k)\sin\theta(k)}\right\} \end{aligned} \tag{32}$$

where l is the length of the truck. Eq. (32) is used to obtain the next state when the present state is given.

For the purpose of training the SC-FCMAC model, learning takes place during several different tries, each try starting from an initial state and terminating when the desired state is reached. In our simulation, six different initial positions of the truck were chosen. The six training paths are shown in Fig. 7. The truck moved a small fixed distance $d_f = 1.6$ at every step and the length of the truck was set to be $l = 1$. The initial parameters $\eta = 0.1$ and $D_{thr} = 80$ were chosen.

First, using the SCA clustering method, we obtained 12 hypercube cells. The learning scheme then entered parameter learning using the gradient descent method. The parameter training process continued for 200 epochs. In each epoch, all the six sets of the training trajectories were presented once to the SC-FCMAC model in a random order. The final rms error of the SC-FCMAC model approximated 0.0316. In Fig. 7, the solid curves are the training paths and the dotted curves are the paths that the tuck moves under the control of the learned controller. As this figure shown, the SC-FCMAC controller can smooth the training paths.

We also compared the SC-FCMAC model with some of the other existing models [25,26,32]. The learning curves of the other models are shown in Fig. 8. Fig. 9(a)–(c) shows the trajectories of the truck moving and controlled by the SC-FCMAC model, starting from the three different initial positions $(x, y, \phi) =$ (a) $(40, 20, -30^\circ)$, (b) $(10, 20, -30^\circ)$, and (c) $(30, 20, -250^\circ)$.

Table 5
Performance comparison of various models in Example 3

Items	Models				
	SC-FCMAC	FCMAC [26]	CNFN [25]	Neural networks[a]	Neuro-fuzzy [32]
Training steps	200	200	200	200	200
Parameters	72	108	91	75	175
Hypercube cells	12	18	13	N/A	35
RMS errors	0.0316	0.0463	0.0449	0.0978	0.0924

[a] The number of hidden node is 25.

We now compare the performance of the SC-FCMAC model with some of the other existing models [25,26,32]. The performance indices considered include training steps, hypercube cells, and rms error. The comparison results are tabulated in Table 5. The results show that the proposed SC-FCMAC model has smaller rms errors and fewer hypercube cells than some of the other existing methods [25,26,32].

5. Conclusion

In this paper, a self-constructing FCMAC (SC-FCMAC) model was proposed for solving control, identification, and classification problems. A self-constructing learning algorithm was presented for constructing and adjusting the parameters. The proposed algorithm uses the self-clustering method (SCM) to perform input space partitioning and the back-propagation algorithm to perform parameter learning. The advantages of the proposed SC-FCMAC model are summarized as follows: (1) it implements scatter partitioning of the input space dynamically; (2) it results in a smaller rms error; and (3) it has a much lower memory requirement. Simulation results demonstrate that the proposed SC-FCMAC model outperforms some of the other existing models.

Acknowledgement

The authors would like to thank the National Science Council of the Republic of China for financially supporting this research under contract no. NSC94-2213-E-324-004.

References

[1] A. Ahmadi, S. Omatu, T. Fujinaka, T. Kosaka, Improvement of reliability in banknote classification using reject option and local PCA, Inform. Sci. 168 (1–4) (2004) 277–293.
[2] J.S. Albus, A new approach to manipulator control: the cerebellar model articulation controller (CMAC), Trans. ASME J. Dyn. Syst. Meas. Contr. (September) (1975) 220–227.
[3] J.S. Albus, Data storage in the cerebellar model articulation controller (CMAC), Trans. ASME J. Dyn. Syst. Meas. Contr. (September) (1975) 228–233.
[4] K.K. Ang, C. Quek, M. Pasquier, POPFNN-CRI(S): pseudoouter product based fuzzy neural network using the compositional rule of inference and singleton fuzzifier, IEEE Trans. Syst. Man Cybernet. B 33 (6) (2003) 838–849.
[5] J.Y. Chen, A VSS-type FCMAC controller, in: IEEE Int. Conf. on Fuzzy Systems, vol. 1, December 2–5, 2001, pp. 872–875.
[6] M.Y. Chow, A. Menozzi, A self-organized CMAC controller, in: Proc. of the IEEE Int. Conf. on Industrial Technology, December 5–9, 1994, pp. 68–72.
[7] S. Commur, F.L. Lewis, CMAC neural networks for control of nonlinear dynamical systems: structure, stability, and passivity, Automatics 33 (4) (1997) 635–641.
[8] R. Fisher, The use of multiple measurements in taxonomic problems, Ann. Eugen. 7 (pt. 2) (1936) 179–188.
[9] F.H. Glanz, W.T. Miller, L.G. Graft, An overview of the CMAC neural network, Proc. IEEE Neural Networks Ocean Eng. (1991) 301–308.
[10] C. Guo, Z. Ye, Z. Sun, P. Sarkar, M. Jamshidi, A hybrid fuzzy cerebellar model articulation controller based autonomous controller, Comput. Electr. Eng. 28 (2002) 1–16.
[11] F.G. Harmon, A.A. Frank, S.S. Joshi, The control of a parallel hybrid-electric propulsion system for a small unmanned aerial vehicle using a CMAC neural network, Neural Networks 18 (5–6) (2005) 772–780.
[12] J. He, L. Liu, G. Palm, Speaker identification using hybrid LVQ-SLP networks, in: Proc. IEEE Int. Conf. on Neural Networks, vol. 4, November 27–December 1, 1995, pp. 2052–2055.

[13] C.P. Hung, M.H. Wang, Diagnosis of incipient faults in power transformers using CMAC neural network approach, Electric Power Syst. Res. 71 (3) (2004) 235–244.
[14] M.C. Hung, D.L. Yang, The efficient FCM clustering technique, in: Proc. IEEE Int. Conf. on Data Mining, November 29–December 2, 2001, pp. 225–232.
[15] K.S. Hwang, C.S. Lin, Smooth trajectory tracking of three-link robot: a self-organizing CMAC approach, IEEE Trans. Syst. Man Cybernet. B 28 (5) (1998) 680–692.
[16] H. Ishibuchi, T. Nakashima, T. Morisawa, Votingin fuzzy rule-based systems for pattern classification problems, Fuzzy Sets Syst. 103 (1999) 223–238.
[17] N.K. Kasabov, Q. Song, DENFIS: Dynamic evolving neural-fuzzy inference system and its application for time-series prediction, IEEE Trans. Fuzzy Syst. 10 (2) (2002) 144–154.
[18] J.S. Ker, C.C. Hsu, Y.H. Huo, B.D. Liu, A fuzzy CMAC model for color reproduction, Fuzzy Sets Syst. 91 (1997) 53–68.
[19] A. Kolcz, N.M. Allinson, Application of the CMAC input encoding scheme in the N-tuple approximation network, IEE Proc. Comput. Digital Tech. 141 (1994) 177–183.
[20] S.G. Kong, B. Kosko, Comparison of fuzzy and neural truck backer-upper control systems, in: Int. Joint Conf. on Neural Networks, San Diego, CA, vol. 3, June 1990, pp. 349–358.
[21] B. Kosko, Fuzzy Engineering, Prentice-Hall, Englewood Cliffs, NJ, 1997.
[22] H.M. Lee, C.M. Chen, Y.F. Lu, A self-organizing HCMAC neural-network classifier, IEEE Trans. Neural Networks 14 (1) (2003) 15–27.
[23] H.R. Lai, C.C. Wong, A fuzzy CMAC structure and learning method for function approximation, in: IEEE Int. Conf. on Fuzzy Systems, vol. 1, December 2–5, 2001, pp. 436–439.
[24] T.W. Liao, D. Li, Y. Li, Extraction of welds from radiographic images using fuzzy classifiers, Inform. Sci. 126 (1–4) (2000) 21–40.
[25] C.J. Lin, C.H. Chen, Nonlinear system control using compensatory neuro-fuzzy networks, IEICE Trans. Fundamentals E86-A (9) (2003) 2309–2316.
[26] C.J. Lin, H.J. Chen, C.Y. Lee, A self-organizing recurrent fuzzy CMAC model for dynamic system identification, in: IEEE Int. Conf. on Fuzzy Systems, Budapest, July 25–29, 2004, pp. 697–702.
[27] F.J. Lin, C.H. Lin, P.H. Shen, Self-constructing fuzzy neural network speed controller for permanent-magnet synchronous motor drive, IEEE Trans. Fuzzy Syst. 9 (5) (2001) 751–759.
[28] F. Marcelloni, Feature selection based on a modified fuzzy *C*-means algorithm with supervision, Inform. Sci. 151 (May) (2003) 201–226.
[29] E. Mese, A rotor position estimator for switched reluctance motors using CMAC, Energy Convers. Managem. 44 (8) (2003) 1229–1245.
[30] W.T. Miller, R.P. Hewes, F.H. Glanz, L.G. Graft, Real-time dynamic control of an industrial manipulator using a neural-network-based learning controller, IEEE Trans. Robot. Autom. 6 (1990) 1–6.
[31] D. Nguyen, B. Widrow, The truck backer-upper: an example of self-learning in neural network, IEEE Conf. Syst. Mag. 10 (3) (1990) 18–23.
[32] H. Nomura, I. Hayashi, N. Wakami, A learning method of fuzzy inference rules by descent method, in: IEEE Proc. Conf. on Fuzzy Systems, March 1992, pp. 203–210.
[33] S. Paul, S. Kumar, Subsethood-product fuzzy neural inference system (SuPFuNIS), IEEE Trans. Neural Networks 13 (3) (2002) 578–599.
[34] W.L. Tung, C. Quek, GenSoFNN: A generic self-organizing fuzzy neural network, IEEE Trans. Neural Networks 13 (5) (2002) 1075–1086.
[35] L.X. Wang, Adaptive Fuzzy Systems and Control, Prentice-Hall, Englewood Cliffs, NJ, 1994.
[36] S. Wang, Z. Jiang, Valve fault detection and diagnosis based on CMAC neural networks, Energy Build. 36 (6) (2004) 599–610.
[37] J. Wu, F. Pratt, Self-organizing CMAC neural networks and adaptive dynamic control, in: Proc. IEEE Int. Symp. on Intell. Contr./Intell. Systems and Semiotics, Cambridge, MA, September 15–17, 1999, pp. 259–265.
[38] K. Zhang, F. Qian, Fuzzy CMAC and its application, in: Proc. of the 3rd World Congress on Intelligent Control and Automation, Hefei, PR China, June 28–July 2, 2000, pp. 944–947.
[39] Y.Q. Zhang, A. Kandel, Compensatory neurofuzzy systems with fast learning algorithms, IEEE Trans. Neural Networks 9 (1) (1998) 83–105.