Available online at www.sciencedirect.com
ScienceDirect
Neurocomputing 70 (2006) 502–512
NEUROCOMPUTING
www.elsevier.com/locate/neucom

Inverse training scheme for MS_CMAC neural network to handle random training data

Jiun-Chi Jan[a,*], Chih-Ming Chen[b,1], Liang-Hao Hsiao[a]

[a]*Department of Civil Engineering, Ching Yun University, 229 Chien-Hsin Road, Jungli, Taiwan 320, ROC*
[b]*National Hualien University of Education, Graduate Institute of Learning Technology, Hualien, Taiwan, ROC*

Received 21 April 2005; received in revised form 30 November 2005; accepted 30 November 2005
Communicated by T. Heskes
Available online 15 May 2006

Abstract

An MS_CMAC neural network is a modular CMAC model designed to model smooth functional mapping. By using a small number of grid-state training data points, the MS_CMAC can use a tree structure network composed of one-dimensional CMAC nodes to decompose a complex multi-variable problem into several simple one-variable sub-problems. However, the grid-state training data points are not always available because most practical applications only obtain random training data points. To overcome this flaw, this work develops an inverse training scheme to supplement the original MS_CMAC. The inverse training scheme is based on a mathematical optimization approach which can employ random training data points to construct the virtual grid-state training data points. The computational results indicate that the ratio 3:1 of random training data points to virtual grid-state training data points can lead to an acceptable prediction accuracy in MS_CMAC.
© 2006 Elsevier B.V. All rights reserved.

Keywords: MS_CMAC; Grid-state; Inverse training scheme

1. Introduction

The cerebellar model articulation controller (CMAC) neural network was developed by Albus [1,2] in 1975. The CMAC model is applied mainly in the control domain [3], and has fast convergence speed in the learning phase. CMAC has also been successfully applied in signal processing, web page mining and pattern recognition fields [4–6]. Past studies concerning CMAC have mainly focused on developing CMAC learning algorithms [7–9], improving the CMAC topology [10–13], selecting learning parameters [14–16] and applying the CMAC model [3–6,17] in various fields. The CMAC training is based on local learning. Each training data point is assigned to a finite number of weights shared by its neighboring data points. Weight training only interferes with data points near the training data point. In other words, interference from distant training data is minimized.

The generalization error of CMAC is caused mainly by the training rule of the network [18]. Albus' CMAC uses a constant basis function to perform locally weighted approximations of functions. The constant basis function updates weights by distributing errors evenly among the assigned weights. CMAC's output is constant within each quantized state, and does not preserve the derivative information. To smoothen CMAC prediction, several studies proposed using nonlinear basis functions to improve CMAC learning. For example, Lane et al. [10] developed a higher-order CMAC neural network using a B-Spline receptive field function with a general CMAC weight-addressing scheme. Additionally, Lane et al. presented a learning architecture capable of learning both a function's values and its derivative information using a hierarchical and muitilayer CMAC neural network architecture trained by standard error back-propagation learning. Chiang and Lin [19] applied CMAC to a non-constant

*Corresponding author. Tel.: +886 3 4581196 5712; fax: +886 3 4590208.
E-mail address: jcjan@cyu.edu.tw (J.-C. Jan).
[1]Present address: National Chengchi University, Graduate Institute of Library, Information and Archival Studies, Taiwan, ROC.

0925-2312/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2005.11.007

differentiable Gaussian basis function, indicating that CMAC with non-constant differentiable Gaussian basis function converges to the least-square error. Although their method provides an output derivative, its learning is time-consuming and requires more memory than does a traditional CMAC since three learning parameters (weight, mean and variance) must be simultaneously tuned and stored. Therefore, Lin and Chiang [20] further proposed another scheme that integrated the CMAC addressing technique with a local weighted regression to provide an output derivative. Moreover, Chen and Hong [21] proposed a weighted gray CMAC (WGCMAC) which employs a weighted gray prediction model to provide an output derivative and a better learning result.

CMAC's convergence property has also received considerable attention from researchers. Parks and Militzer [9] investigated the learning convergence of CMAC. Their results confirmed that CMAC learning can be converged as the learning ratio is equal to 1. Wong and Sideris [22] indicated that CMAC's learning is equivalent to solving a linear system with a Gauss–Seidel iteration scheme, which always converges to a least-square error if the computer memory is sufficiently large. Additionally, Lin and Chiang's [23] study proved that CMAC learning always converges in a least-square error if the number of iterations approaches infinity and the learning ratio approaches zero. Their study further proved that CMAC's iterative learning approach with or without hash mapping converges to a limit cycle if the learning ratio is between 0 and 2.

In modeling multi-variable functions, CMAC can produce very large numbers of memory units for learning as the number of input vector dimensions increases. Additionally, the required number of training data points should be increased to satisfy CMAC training. The fast memory size requirement limits CMAC's practical utility in real-world applications such as high-dimensional Web page classification. To improve CMAC, several researchers have proposed using modular CMAC to model multi-variable problems. Albus [2] proposed a time-inversion technique to refine CMAC. The time inversion CMAC uses a serially connected low-dimension CMACs to solve high-dimension problems, where the upper low-dimension CMACs specify temporary data points to be used as training data in later low-dimension CMAC. Lin and Li [11] developed a new structure CMAC composed of small CMACs to model a multi-variable functional problem. Actually, the model can be considered as a self-generated basis function neural network with two-level tree structure composed of small two-dimensional CMACs. Lee and Chen et al. [13] proposed a self-organizing HCMAC neural network classifier which contains a self-organizing input space module and a hierarchical CMAC neural network capable of resolving high-dimensional classification problems well. Furthermore, Hung and Jan [17] proposed an MS_CMAC to model multi-variable smooth functions. The MS_CMAC connects many one-dimensional CMACs as a tree structure, where the ensemble is trained by time inversion technique. Hung and Jan [12] further developed a quadratic spline scheme to smoothen outputs of the one-dimensional CMACs by transforming the step-state weights into continuous-smooth weights. The quadratic spline scheme is fairly different to the use of spline basis function as weight updating strategy. The one-dimensional CMAC first uses a constant basis function as the weight updating strategy to obtain weights, which are always step-state and can be represented by several step functions. As the weights are stable, each step function is replaced by a spline function. Therefore, the weights of the one-dimensional CMAC are smooth. Also, the outputs of the one-dimensional CMAC are smooth.

The advantage of MS_CMAC model over other CMAC models is that it needs only a few training data points to obtain accurate prediction rates for function approximation problems when the training data points are distributed over a grid system. However, grid-state training data points are not always available in practical applications. To overcome this flaw, this work develops an inverse training scheme to supplement the original MS_CMAC. The inverse training scheme is based on a mathematical optimization approach which constructs the virtual grid-state training data points from the random training data points. The experimental results show that a ratio 3:1 of random training data points to virtual grid-state training data points can produce an acceptable prediction accuracy for the proposed novel inverse training scheme. In the following sections, Section 2 gives a brief introduction for MS_CMAC model, Section 3 explains the proposed novel training scheme and Section 4 adopts several numerical cases to test the learning performance of the proposed training scheme.

2. MS_CMAC neural network

2.1. Binary CMAC with quadratic spline scheme

Binary CMAC is a one-dimensional Albus' CMAC. The binary CMAC computes output as follows:

$$y(x_i) = \sum_{j=0}^{g-1} w_{i+j}, \tag{1}$$

where y is the output function, x_i is the input variable, w_{i+j} is the addressing weight and g is an integer called the generalization size. The weight updating by a training data point $(x_i,\ y_d(x_i))$ is defined as follows:

$$w'_{i+j} = w_{i+j} + \frac{y_\mathrm{d}(x_i) - y(x_i)}{g} \quad \text{for } j = 0,\ 1,\ 2,\ \ldots,\ g-1, \tag{2}$$

where w'_{i+j} denotes the updated weight, $y_\mathrm{d}(x_i)$ is the desired output value of the binary CMAC for the input varialbe x_i, and $y(x_i)$ is the actual output value of the binary CMAC for the input varialbe x_i.

Typically, Eqs. (1) and (2) lead to a linear approximation in the local zone and step-state weights which can be represented by several step functions. To improve the approximation in modeling nonlinear functions, a quadratic spline scheme was developed to transform the step-state weights into smooth weights. In this scheme, a set of quadratic splines $f_k(i) = a_k i^2 + b_k i + c_k$, respectively replace the step functions in a well-trained binary CMAC, where the well-trained binary CMAC means that it learns all training data points exactly. Two assumptions are defined as follows:

(1) The integral of a spline function equals the summation of the corresponding step weight. This assumption insures that the CMAC prediction and training data points fit well after weight transformation.
(2) The neighboring quadratic weight functions are continuous and smooth in the connect point. This assumption insures that the CMAC prediction is always smooth.

Thus, the coefficient a_k, b_k and c_k should satisfy the following three equations:

$$\int_{X_k}^{X_{k+1}} f_k(i)\,\mathrm{d}i = w_{xk}\,(X_{k+1} - X_k), \tag{3}$$

$$f_{k-1}(X_k) = f_k(X_k), \tag{4}$$

$$f'_{k-1}(X_k) = f'_k(X_k), \tag{5}$$

where X_k and X_{k+1} are the boundaries of $f_k(i)$, and w_{xk} is the corresponding step weight of $f_k(i)$. These coefficients can be solved by linear algebra.

In binary CMAC, Eq. (1) is resembled to a numerical integration method of rectangular rule. After the step-state weights are transformed into the splines weights, a formula resembled to a numerical integration method of Simpon's rule is proposed to replace Eq. (1), and is defined as follows:

$$y(x_i) = \frac{1}{3} \sum_{j=0, j=j+2}^{g-2} \left[f(x_{i+j}) + 4f(x_{i+1+j}) + f(x_{i+2+j}) \right], \tag{6}$$

where the value of g is always an even number.

2.2. Tree structure network and time inversion technique

MS_CMAC neural network uses a dimensional reduction technique to decompose a multi-variable problem into several one-variable sub-problems. Herein, a two-variable problem with 4×4 grid-state data points is adopted to interpret the dimensional reduction technique. Fig. 1 shows a grid system. The circles denote the training data points, the rectangles indicate the temporary data points and the triangle represents a target data point. First, 4 one-dimensional CMACs are utilized to model four y-direction line domains (i.e. N_1^2, N_2^2, N_3^2 and N_4^2). The corresponding outputs of the four temporary data points are calculated by the 4 one-dimensional CMACs. In the next step, another one-dimensional CMAC is utilized to model an x-direction line domain (N_1^1), where the temporary data points and the target data point are located on the x-direction line domain. Finally, the corresponding output of the target data point is calculated by the last one-dimensional CMAC. Consequently, the dimensional reduction technique on the two-variable problem can be accomplished by a two-level tree structure modular CMAC. Fig. 2 shows the modular CMAC. The root and leaf nodes are composed of one-dimensional CMACs. These one-dimensional CMACs model the line domain N_1^1, N_1^2, N_2^2, N_3^2 and N_4^2, respectively. The grid-state data points are assigned to one-dimensional CMAC leaf nodes for training, where training data points of each leaf node have the same value in x-direction. Additionally, four temporary data points are generated by the leaf nodes and are used as training data for the root node. The temporary data points in the y-direction are the same as that of the target data point, and

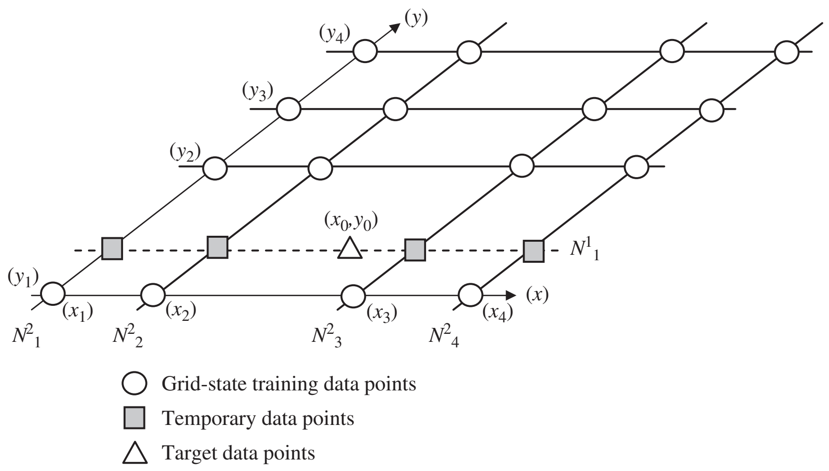


Fig. 1. Illustrate dimensional reduction technique of MS_CMAC.

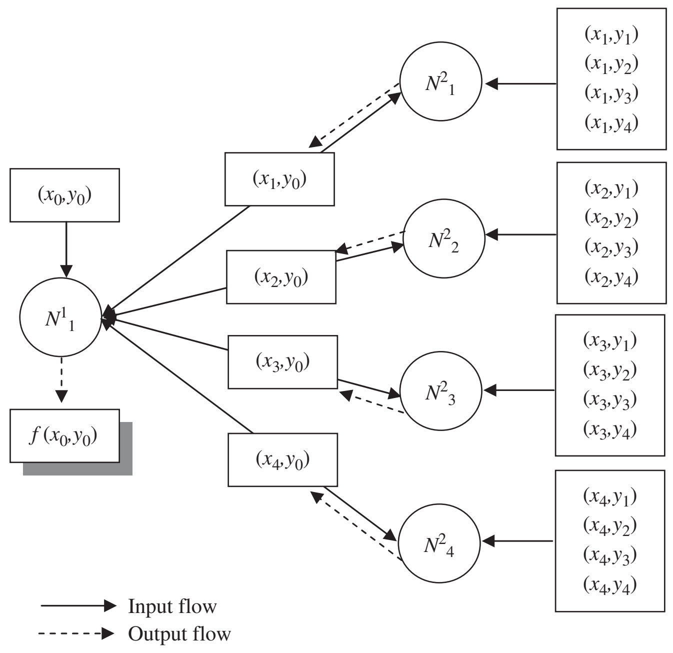

(x_0,y_0)
N^1_1
$f(x_0,y_0)$
(x_1,y_0)
(x_2,y_0)
(x_3,y_0)
(x_4,y_0)
N^2_1
N^2_2
N^2_3
N^2_4
(x_1,y_1) (x_1,y_2) (x_1,y_3) (x_1,y_4)
(x_2,y_1) (x_2,y_2) (x_2,y_3) (x_2,y_4)
(x_3,y_1) (x_3,y_2) (x_3,y_3) (x_3,y_4)
(x_4,y_1) (x_4,y_2) (x_4,y_3) (x_4,y_4)
Input flow
Output flow

Fig. 2. Two-level tree structure of MS_CMAC.

the temporary data points in the x-direction are the same as those of the corresponding leaf nodes. The model learning should find the outputs of the temporary data points first, then use them to generate the target data point. This training scheme is termed as time inversion technique. Notably, the weights of leaf nodes are static, but the weights of other nodes should be dynamically updated in utilizing, since the temporary data points are dependent on the target data point.

2.3. General form of MS_CMAC

(1) The training data points should be grid-state in the learning domain. Assume that a grid system has n directions, and the numbers of gridlines in n directions are given by p_1, p_2, …, and p_n, respectively. The training data points locate at the grid joint. Therefore, the grid system consists of $\prod_{i=1}^{n} p_i$ training data points.
(2) An n-level tree structure composed of one-dimensional CMAC nodes is established to model the learning domain, where each CMAC node in the ith $(1 \leqslant i \leqslant n-1)$ level has p_i children nodes. Therefore, the modular CMAC consists of $\sum_{j=1}^{n-1} \prod_{i=1}^{j} p_i + 1$ one-dimensional CMACs, where leaf nodes are static one-dimensional CMACs, and the other nodes are dynamic one-dimensional CMACs.
(3) The grid-state training data points are assigned to the leaf nodes of the tree, satisfying the time inversion technique.
(4) Two connected CMAC nodes have a temporary data point. The temporary data point is generated by the lower CMAC node and used for training the upper CMAC node.
(5) The MS_CMAC computing procedure starts from leaf nodes to the root node, sequentially. Each level in the MS_CMAC neural network handles only one variable, which is called active parameter in the level.

3. Inverse training scheme

3.1. Constant spacing virtual grid system

Authors' past research [12,17] indicated that MS_CMAC can easily model nonlinear smooth functions without memory overlap in the training phases when a suitable grid system is selected. Generally, no memory overlap in the training phase can reduce computational load of CMAC. Therefore, this work proposes a linear transformation of the real grid system into a constant spacing virtual grid system, where the spacing of the virtual grid system is fixed as g (generalization size) in each direction. For example, a real grid system in the x direction is defined as $x:\{x_1, x_2, \ldots, x_{p1}\}$. The corresponding direction (sx) of the virtual grid system can be calculated as follows:

$$sx(x) = (i-1)g + \text{round}\left(\frac{x - x_i}{x_{i+1} - x_i}\, g\right) \quad \text{for } x_i \leqslant x \leqslant x_{i+1}, \tag{7}$$

where round() is a mathematic operator to round up or down.

3.2. Virtual grid-state training data points

MS_CMAC neural network uses a tree structure modular CMAC to model multi-variable problems by decomposing a complex multi-variable problem into several simple one-variable problems. MS_CMAC training data must be grid-state over the learning domain. However, in practical applications, the grid-state training data points are not always available. This reduces the practicability of MS_CMAC in real-world applications. To overcome this drawback, this section proposes an inverse training scheme to establish virtual grid-state training data points. The basic idea of the inverse training scheme is that the reasonable virtual grid-state training data points can lead to few prediction errors of random training data points in MS_CMAC prediction. Therefore, this study developed a mathematical optimization approach to establish the virtual grid-state training data points. Assume that a positive/negative error of a random training data point indicates a positive/negative updating for near virtual training data points, and the error feedback ratios are inverse proportion to the distance between the random training data point and the virtual grid-state training data points. For a given training data point, the virtual grid-state training data points can be updated as follows:

$$GY_i^{(n+1)} = GY_i^{(n)} + \alpha(\mathbf{GX}_i, \mathbf{TX})\,(TY - Y_c) \quad \text{for } i = 1, 2, \ldots, \prod_{j=1}^{n} p_j, \tag{8}$$

where $GY_i^{(n)}$ is the value of a virtual grid-state training data point at the n-th training iteration, $\mathbf{GX}_i$ is the position vector of the virtual training data point, $\mathbf{TX}$ is the position vector of the training data point, TY is the corresponding value of $\mathbf{TX}$, Y_c is the corresponding output of MS_CMAC, and $\alpha(\mathbf{GX}_i, \mathbf{TX})$ is an error feedback ratio function. The ratio function resembles a triangle fuzzy membership function, and is defined as follows:

$$\alpha(\mathbf{GX}_i, \mathbf{TX}) = \begin{cases} 1 - \frac{\mathrm{Dis}(\mathbf{GX}_i, \mathbf{TX})}{r_{\max}} & \text{if } \mathrm{Dis}(\mathbf{GX}_i, \mathbf{TX}) < r_{\max}, \\ 0 & \text{else,} \end{cases} \tag{9}$$

$$\mathrm{Dis}(\mathbf{GX}_i, \mathbf{TX}) = \sqrt{(gx_1 - tx_1)^2 + (gx_2 - tx_2)^2 + \ldots}, \tag{10}$$

where gx_1, gx_2, etc. are elements of $\mathbf{GX}_i$, tx_1, tx_2, etc. are elements of $\mathbf{TX}$, and $r_{\max}$ is a real number. Basically, the error feedback ratio function provides linear inverse proportion of error feedback ratios to the distances between random training data point and near virtual grid-state training data points.

Herein, $r_{\max}$ represents the maximum influence distance of a training data point, and it can affect the generalization capability and learning convergence of the novel MS_CMAC. A very large value of $r_{\max}$ makes convergence very difficult. On the other hand, a very small value of $r_{\max}$ causes some random training data points to miss the virtual grid-state training data points. To ensure each known data point influences at least one virtual grid-state training data point, this study proposes a formula to calculate the parameter $r_{\max}$, and the formula is shown as follows:

$$r_{\max} = \sqrt{n\left(\frac{g}{2}\right)^2}, \tag{11}$$

where $\sqrt{n(g/2)^2}$ is the maximum distance from any position of a n-dimensional space to the nearest grid-state training data point.

As the same with other neural computing approaches, the training phase is terminated when the sum of square errors of the random training data points satisfies the stop criterion.

4. Numerical case studies

Herein, the root mean square error (RMSE) is utilized to measure errors of the test cases and is defined as follows:

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{i=1}^{t} (TY_i - Y_c)^2}{t}}, \tag{12}$$

where t denotes the number of instances.

4.1. Case I

$$F_1(x_1, x_2, x_3) = \sin(x_1 \pi)\sin(2x_2 \pi)\sin(3x_3 \pi) \quad \text{for } 0 \leqslant x_1, x_2, x_3 \leqslant 1.$$

This case is a three-variable continuous function, which has been discussed in authors' previous research [12]. In Ref. [12], a grid system with five constant grid spaces in all directions and 216 (6^3) grid-state training data points was adopted to approximate the function. Herein, the inverse training scheme adopted the same grid system to approximate the function. The gridline locations in each direction are defined as follows:

$x_1, x_2, x_3 : \{0, 0.2, 0.4, 0.6, 0.8, 1\}$.

An MS_CMAC neural network using a three-level tree structure with six branches in each level was employed to approximate the function. The *active parameters* for levels 1 to 3 were set as x_1, x_2, and x_3, respectively. The generalization sizes, g, were set to 20 for all nodes in the MS_CMAC. The MS_CMAC neural network used 36 static one-dimensional CMACs and seven dynamic one-dimensional CMACs.

Five testing sets of training data points were randomly generated at 216, 432, 648, 864 and 1080 instances to verify the inverse training scheme. Additionally, 441 (21^2) data points with constant grid spaces on $F_1(0.5, x_2, x_3)$ were adopted for verification. Table 1 shows the prediction accuracies of the MS_CMAC under different ratios of random training data points to grid-state training data points. A large ratio of random training data points to virtual grid-state training data points is found to produce a small prediction error. Notably, the iteration of training decreased as the number of training data points increased.

Table 1
Computing results of case I

Number of training data points	Ratio of known data points to the grid system	RMSE			Number of iteration in training	Learning speed (s/100 iterations)
		Training	Verification	Average		
216	1:1	0.02765	0.15530	0.12659	247	14
432	2:1	0.01658	0.08523	0.06169	165	29
648	3:1	0.01590	0.02756	0.02237	129	47
864	4:1	0.01683	0.03043	0.02140	119	60
1080	5:1	0.01628	0.02612	0.01965	78	77

Table 2 shows parts of virtual grid-state training data points under five different ratios of random training data points to virtual grid-state training data points. The first six virtual training data points are located on the boundary of the learning domain, and the others are inner virtual training data points. The experimental results show that the inner virtual training data points are converged well as the ratio of random training data points to virtual grid-state training data points exceeds 3:1. On the other hand, the boundary virtual training data points are also converged well as the ratio of random training data points to virtual grid-state training data points exceeds 4:1.

Based on previous experiments, this study proposes a heuristic rule to indicate that the ratio 3:1 of random training data points to virtual grid-state training data points is a minimum requirement. To prove the heuristic rule, another grid system with six constant grid spaces in each direction was constructed for another MS_CMAC to solve the same function approximation problem. The MS_CMAC used a three-level tree structure with seven branches in each direction, and included 49 static one-dimensional CMACs and eight dynamic one-dimensional CMACs. The generalization sizes were set to 20 as previously. Table 3 shows the computational results. As indicated from the experimental results, only the last set of random training data points satisfies the proposed heuristic rule of the ratio 3:1 of random training data points to virtual grid-state training data points (3.149:1). Additionally, only the set of random training data points corresponds to an acceptable prediction error of MS_CMAC. When comparing the experimental results in Tables 1 and 3, a denser grid-system leads to a better prediction of MS_CMAC when the reasonable ratio of random training data points to virtual grid-state training data points is used, but also increase the computational time of the inverse training scheme.

In addition, this testing case was also used to experimental justify the selection role of $r_{\max}$. Fig. 3 shows three learning curves of the testing case I under different values of $r_{\max}$, where 648 training data points and 216 virtual grid-state training data points are used. As shown in Fig. 3, the value of $r_{\max}$ determined by Eq. (11) obviously leads to the best learning convergence. Although the experimental results do not mean Eq. (11) calculated the optimized $r_{\max}$, Eq. (11) is useful to determine $r_{\max}$. Moreover, three learning curves are sharply down in the first 20 iterations of training. This indicates that the value of $r_{\max}$ just affects slightly on the computational complexity of the proposed inverse training scheme.

Further discussing, a back-propagation (BP) neural network [24] was used as reference model to approximate $F_1(x_1, x_2, x_3)$, and its learning results were compared with

Table 2
Part of virtual grid-state training data points

Location	Desired	Ratio of known data points to the grid system				
		1:1	2:1	3:1	4:1	5:1
F_1(0,0,0)	0	0.450	0.319	0.274	0.074	0.007
F_1(0,0,0.2)	0	0.630	0.628	0.102	0.003	0.001
F_1(0,0,0.4)	0	0.210	0.210	0.210	–0.008	–0.066
F_1(0,0,0.6)	0	0.190	–0.055	–0.002	–0.121	–0.066
F_1(0,0,0.8)	0	0.017	–0.019	–0.013	0.069	0.042
F_1(0,0,1)	0	0.062	0.197	0.119	0.039	0.013
F_1(0.8,0.4,0)	0	0.039	0.045	0.023	0.005	0.011
F_1(0.8,0.4,0.2)	0.329	0.354	0.327	0.338	0.325	0.326
F_1(0.8,0.4,0.4)	–0.203	–0.123	–0.175	–0.227	–0.237	–0.218
F_1(0.8,0.4,0.6)	–0.203	–0.209	–0.216	–0.203	–0.200	–0.203
F_1(0.8,0.4,0.8)	0.329	0.344	0.333	0.335	0.333	0.332
F_1(0.8,0.4,1)	0	–0.004	–0.003	0.002	0.012	0.011

Table 3
Computing results of case I with a dense grid system

Number of training data points	Ratio of known data points to the grid system	RMSE			Number of iteration in training	Learning speed (s/100 iterations)
		Training	Verification	Average		
216	0.629:1	0.01848	0.31012	0.21424	205	25
432	1.259:1	0.01753	0.15575	0.08735	231	50
648	1.889:1	0.01911	0.11643	0.05852	175	73
864	2.519:1	0.01756	0.08159	0.03920	155	98
1080	3.149:1	0.01698	0.02522	0.01937	127	122

that of the novel MS_CMAC. The BP neural network used a single hidden layer feed-forward network. The hidden layer of the network had seven hidden nodes, and the learning ratio was dynamically obtained by a line search algorithm. Fig. 4 plots the desired and computed outputs of $F_1(0.25, x_2, x_3)$, where 648 random data points are used as training instances. Clearly, 648 random training data points are insufficient for BP to model $F_1(x_1, x_2, x_3)$. However, the MS_CMAC with the inverse training scheme has an acceptable prediction error except at the boundary area.

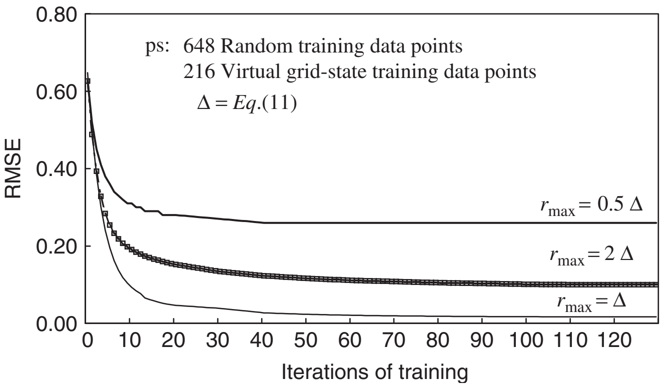


Fig. 3. Learning convergences of case I under different values of r_{max}.

4.2. Case II

$$F_2(x_1, x_2, x_3, x_4) = x_1 + \sin(x_1\pi)\cos(x_2\pi)\sin(3x_3\pi) \times [\sin^2(x_4\pi) - 1] \text{ for } -1 \leqslant x_1, x_2, x_3, x_4 \leqslant 1.$$

This case is a four-variable continuous function, and is discussed in Ref. [11], which approximates the function with 30,000 training data points. Herein, a grid system with four constant grid spaces in x_1 and x_3 directions, five constant grid spaces in x_2 direction and eight constant grid spaces in x_4 direction was selected for the MS_CMAC neural network:

$x_1, x_3 : \{-1, -0.5, 0.5, 1\}$,
$x_2 : \{-1, -0.6, -0.2, 0.2, 0.6, 1\}$,
$x_4 : \{0, -0.75, -0.5, -0.25, 0, 0.25, 0.5, 0.75, 1\}$.

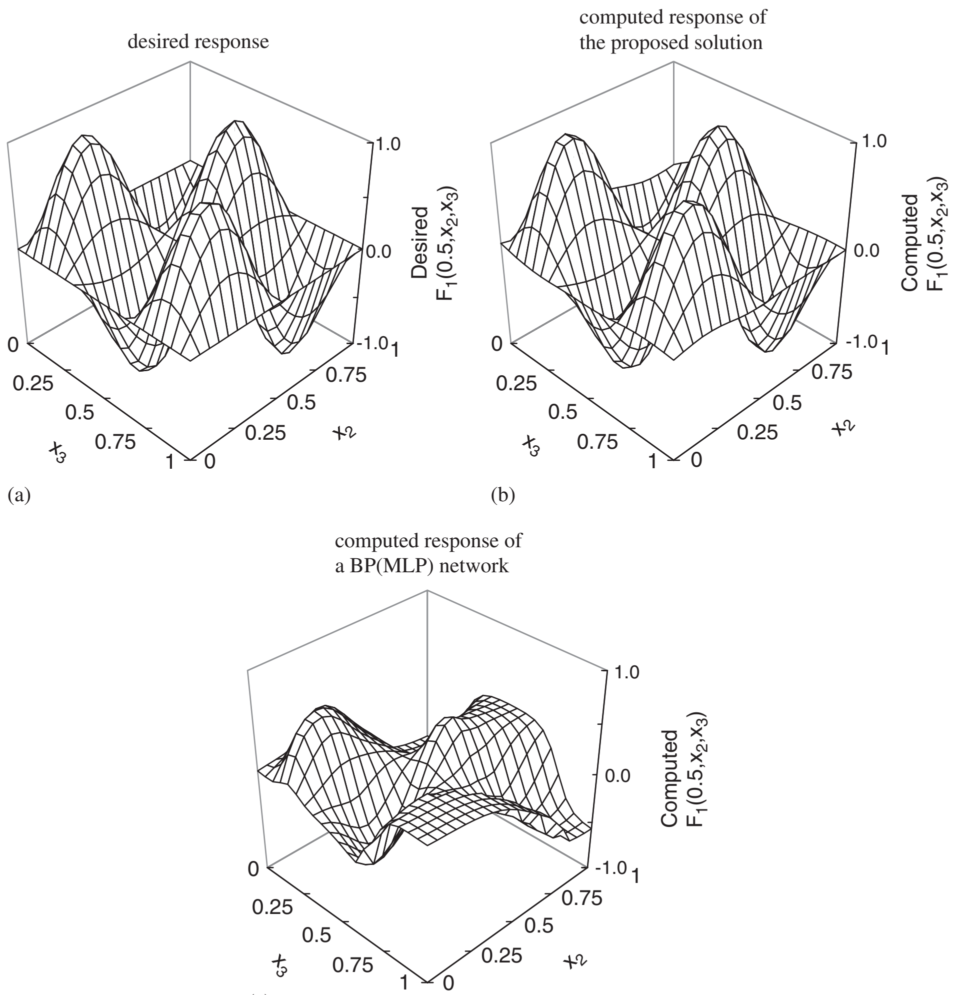


Fig. 4. Desired and computed plots of $F_1(0.25, x_2, x_3)$.

Therefore, the grid system has 1350 ($5 \times 6 \times 5 \times 9$) grid joints. The function was approximated using an MS_CMAC with a four-level tree structure, and the generalization sizes (g) were set to 30 for all one-dimensional CMACs. The active parameters were set as x_1, x_2, x_3 and x_4 for level 1, 2, 3 and 4, respectively. Thus, the numbers of branches from level 1 to 3 were five, six and five, respectively. The MS_CMAC used 150 static one-dimensional CMACs and 36 dynamic one-dimensional CMACs.

Herein, 4050 (1350×3) random training data points were randomly generated for the inverse training scheme. The number of the random training data points is obviously smaller than that in Ref. [11]. Three sets of 441 (21^2) data points with constant grid spaces on $F_2(x_1, 0.25, 0.25, x_4)$, $F_2(0.25, x_2, 0.25, x_4)$ and $F_2(0.25, 0.25, x_3, x_4)$ were selected as verification data points, respectively. The RMSE of the training data points was 0.0144 after 250 iterations of training. The learning speed was 1620 s per 100 iterations. Figs. 5a–c plot the desired and computed outputs of $F_2(x_1, 0.25, 0.25, x_4)$, $F_2(0.25, x_2, 0.25, x_4)$ and $F_2(0.25, 0.25, x_3, x_4)$, respectively. The RMSEs of verification data points on functions of $F_2(x_1, 0.25, 0.25, x_4)$, $F_2(0.25, x_2, 0.25, x_4)$ and $F_2(0.25, 0.25, x_3, x_4)$ were 0.0201, 0.0214 and 0.0269, respectively. Obviously, the prediction is acceptable.

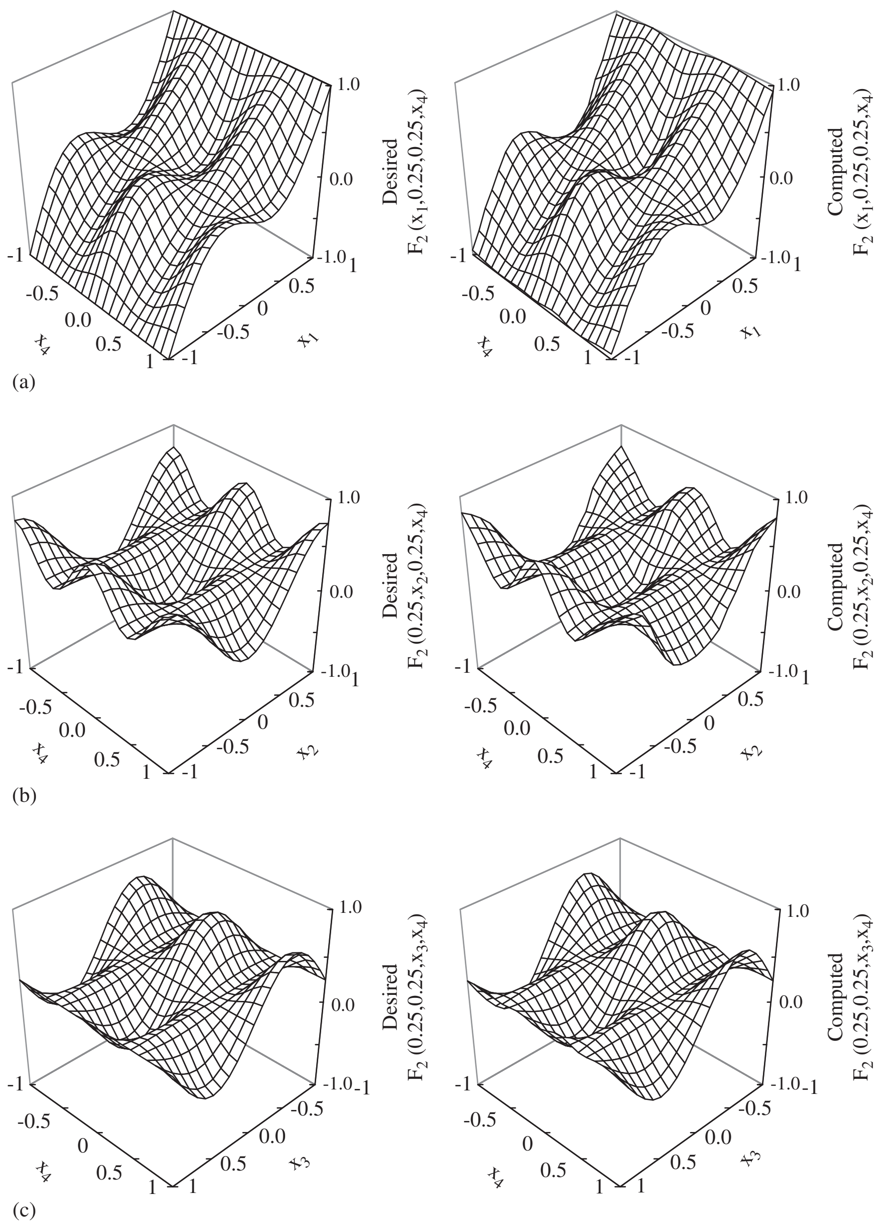


Fig. 5. (a) Desired and computed plots of $F_2(x_1, 0.25, 0.25, x_4)$. (b) Desired and computed plots of $F_2(0.25, x_2, 0.25, x_4)$. (c) Desired and computed plots of $F_2(0.25, 0.25, x_3, x_4)$.

4.3. Case III

$$F_3(x_1,x_2,x_3,x_4) = (\ln(x_1x_2+x_3x_4)+\ln(x_1x_3+x_2x_4))^2 \quad \text{for } 0.1 \leqslant x_1,x_2,x_3,x_4 \leqslant 3.1.$$

This case is a four-variable continuous function, and is also discussed in Ref. [11]. Herein, a grid system with four constant grid spaces in each direction was selected for the MS_CMAC neural network. The locations of gridlines in each direction are defined as follows:

$x_1,x_2,x_3,x_4 : \{0.1, 0.85, 1.6, 2.35, 3.1\}$.

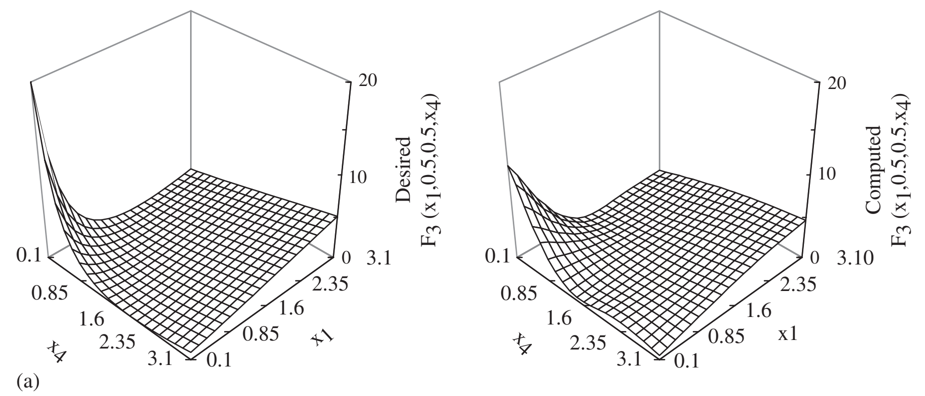
Therefore, the grid system has 625 (5^4) grid joints. An MS_CMAC with a four-level tree structure was exploited to approximate the function. The generalization sizes (g) were set to 30 for all one-dimensional CMACs, and the active parameters were set as x_1, x_2, x_3 and x_4 for level 1, 2, 3 and 4, respectively. Thus, nodes in each level had five branches. The MS_CMAC used 125 static one-dimensional CMACs and 31 dynamic one-dimensional CMACs.

Herein, 1875 (625×3) data points were randomly generated for the inverse training scheme. Two sets of 441 (21^2) data points with constant grid space on $F_3(x_1, 0.5, 0.5, x_4)$ and $F_3(x_1, x_2, 0.2, 1.0)$ were selected as verification data points, respectively. The RMSE of the training data points was 0.0111 after 243 iterations of training. The learning speed was 573 s per 100 iterations. Figs. 6a and b plot the desired and computed outputs of $F_3(x_1, 0.5, 0.5, x_4)$ and $F_3(x_1, x_2, 0.2, 1.0)$, respectively. The RMSE of verification data points on $F_3(x_1, 0.25, 0.25, x_4)$ and $F_3(x_1, x_2, 0.2, 1.0)$ are 0.0212 and 0.0144, respectively. Notably, the errors at $F_3(0.1, 0.5, 0.5, 0.1)$ and $F_3(0.1, 0.1, 0.2, 1.0)$ occur apparent error, as also found in Ref. [11].

4.4. Discussion of errors distribution

Cases I and III indicate that the prediction errors of MS_CMAC mainly occur on the boundary of the training domain under a ratio 3:1 of random training data points to virtual grid-state training data points. These errors occur because finding grid-state training data points at a boundary is an extrapolation problem. This problem can be improved by using a large ratio of random training to grid-state training data points, as indicated from Table 2. To understand the influence of a large boundary error on a border area, this section primarily discusses the errors distribution near the boundary points. For example, the maximum prediction error of case I occurs at $F_1(0.25, 1, 0.2)$, and the maximum prediction error of case III occurs at $F_3(0.1, 0.5, 0.5, 0.1)$. Both these points are boundary points.

Table 4 shows some inner data points near $F_1(0.25, 1, 0.2)$ and $F_3(0.1, 0.5, 0.5, 0.1)$. The experimental results show that the computed errors of the data points



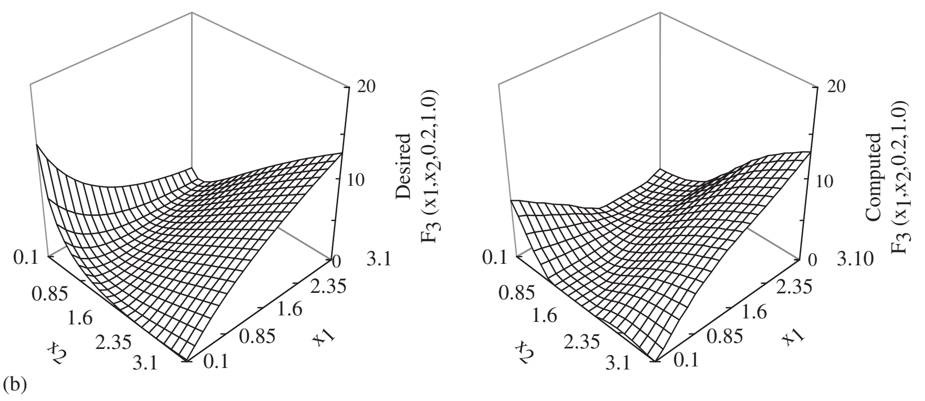


Fig. 6. (a) Desired and computed plots of $F_3(x_1, 0.5, 0.5, x_4)$. (b) Desired and computed plots of $F_3(x_1, x_2, 0.2, 1.0)$.

Table 4
Errors distribution near the location of maximum errors

Data points		Computed	Error	Distance from F_1(0.25,1,0.2) Range = 0–1	Distance from F_3(0.1,0.5,0.5,0.1) Range = 0.1–3.1	Ratio of error to maximum error
Location	Desired					
F_1(0.25,1,0.2)	0	0.121	0.121	0	—	1
F_1(0.25,0.95,0.2)	–0.294	–0.218	0.084	0.05	—	0.69
F_1(0.25,0.9,0.2)	–0.559	–0.527	0.032	0.1	—	0.26
F_1(0.25,0.85,0.2)	–0.769	–0.776	0.007	0.15	—	0.06
F_3(0.1,0.5,0.5,0.1)	21.208	11.036	10.172	—	0	1
F_3(0.25,0.5,0.5,0.1)	12.152	9.469	2.462	—	0.15	0.24
F_3(0.1,0.5,0.5,0.25)	12.152	9.385	2.767	—	0.15	0.27
F_3(0.25,0.5,0.5,0.25)	7.687	8.073	0.386	—	0.212	0.04
F_3(0.4,0.5,0.5,0.1)	7.687	7.557	0.130	—	0.3	0.01
F_3(0.1,0.5,0.5,0.4)	7.687	7.377	0.310	—	0.3	0.03

are apparently decreased as the distances from $F_1(0.25, 1, 0.2)$ or $F_3(0.1, 0.5, 0.5, 0.1)$ are increased. In case I, $F_1(0.25, 1, 0.2)$ large computed error (0.121) occurs. Then move the point along the x_2-direction, the computed error is reduced to 0.084, 0.032 and 0.007 at the points $F_1(0.25, 0.95, 0.2)$, $F_1(0.25, 0.9, 0.2)$ and $F_1(0.25, 0.85, 0.2)$, respectively. In case III, $F_3(0.1, 0.5, 0.5, 0.1)$ very large computed error (10.172) occurs. However, the computed errors at $F_3(0.25, 0.5, 0.5, 0.1)$ and $F_3(0.1, 0.5, 0.5, 0.25)$ are only 2.462 and 2.767, respectively. Moreover, the computed errors at $F_3(0.4, 0.5, 0.5, 0.1)$ and $F_3(0.1, 0.5, 0.5, 0.4)$ are only 0.13 and 0.31, respectively.

5. Conclusion

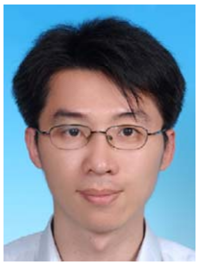
This work mainly developed an inverse training scheme to supplement the original MS_CMAC to handle random training data points. The inverse training scheme uses an optimization approach to establish virtual grid-state training data points which are strongly necessary in MS_CMAC training. The inverse training scheme uses the prediction error of a known data point to update neighboring virtual grid-state training data points, and the corrections are set as an inverse proportion to the distances between the known data point and neighbor virtual grid-state training data points. Through the study, several conclusions are obtained and described in the following:

(1) In the inverse training scheme, the minimum required ratio of the random training data points to the virtual grid-state training data points is 3:1. Under the rule, the MS_CMAC requires a few random training data points to obtain an acceptable prediction accuracy than does a back-propagation neural network.
(2) Large errors might occur on the boundary area of a training domain when training an MS_CMAC using the inverse training scheme. However, the boundary error problem can be solved using a large ratio of random training data points to virtual grid-state training data points. Moreover, the prediction errors are sharply reduced by increasing the distance to the boundary point.
(3) A dense grid system improves the prediction accuracy of an MS_CMAC. However, expanding the network structure of an MS_CMAC substantially increase the computational time. Considering the computational burden and prediction accuracy, the inverse training scheme is not recommend for very high-dimensional problems.

References

[1] J.S. Albus, A new approach to manipulator control: the cerebellar model articulation controller (CMAC), J. Dyn. Syst. Meas. Control Trans. ASME 97 (3) (1975) 220–227.
[2] J.S. Albus, Data storage in the cerebellar model articulation controller, J. Dyn. Syst. Meas. Control Trans. ASME 97 (3) (1975) 228–233.
[3] R.O. Shelton, J.K. Peterson, Controlling a truck with an adaptive critic CMAC design, Simulation 58 (5) (1992) 319–326.
[4] W.T. Miller, F.H. Glanz, CMAC: an associative neural network alternative to backpropagation, Proc. IEEE 78 (10) (1990) 1561–1567.
[5] F.H. Glanz, W.T. Miller, L.G. Kraft, An overview of the CMAC neural network, in: Proceedings of 1991 IEEE Neural Networks for Ocean Engineering, Washington, DC, USA, 1991, p. 301–308.
[6] C.M. Chen, Incremental personalized web page mining utilizing self-organizing HCMAC neural network, Web Intell. Agent Syst. Int. J. 2 (2004) 21–38.
[7] N.E. Cotter, T.J. Guillerm, The CMAC and a theorem of Kolmogorov, Neural Networks 5 (1991) 221–228.
[8] K. Zhang, F. Qian, Fuzzy CMAC and its application, in: Proceedings of the Third World Congress on Intelligent Control and Automation, June/July 2000, Hefei, China, p. 944–947.
[9] P.C. Parks, J. Militzer, A comparison of five algorithm for the training of CMAC memories for learning control systems, Autom. Remote Control 50 (1989) 254–286.
[10] S.H. Lane, D.A. Handelman, J.J. Gelfand, Theory and development of higher-order CMAC neural networks, IEEE Control Syst. 12 (2) (1992) 23–30.
[11] C.S. Lin, C.K. Li, A memory-based self-generated basis function neural network, Int. J. Neural Syst. 9 (1) (1999) 41–59.

[12] J.C. Jan, S.L. Hung, High-order MS_CMAC neural network, IEEE Trans. Neural Networks 12 (3) (2001) 598–603.
[13] 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.
[14] C.S. Lin, H. Kim, Selection of learning parameters for CMAC-based adaptive critic learning, IEEE Trans. Neural Networks 6 (1995) 642–647.
[15] A. Menozzi, M.Y. Chow, On the training of a multi-resolution CMAC neural network, in: Proceedings of the IEEE International Symposium on Industrial Electronics, ISIE '97, vol.3, Guimaraes, Portugal, 1997, p. 1201–1205.
[16] H. Kim, C.S. Lin, Use of adaptive resolution for better CMAC learning, in: Proceedings of International Joint Conference on Neural Networks, vol.1, Baltimore, MD, USA, 1992, p. 517–522.
[17] S.L. Hung, J.C. Jan, MS_CMAC neural network learning model in structural engineering, J. Comput. Civil Eng. ASCE 13 (1) (1999) 1–11.
[18] T. Szab, G. Horvath, Improving the generalization of the binary CMAC, Proc. Int. Joint Conf. Neural Networks, IJCNN'2000, Como, Italy (3) (2000) 85–90.
[19] C.T. Chiang, C.S. Lin, CMAC with general basis functions, Neural Networks 9 (7) (1996) 1199–1211.
[20] C.S. Lin, C.T. Chiang, Integration of CMAC technique and weighted regression for efficient learning and output differentiability, IEEE Trans. Syst. Man Cybernet. Part B Cybernet. 28 (2) (1998) 231–237.
[21] C.M. Chen, C.M. Hong, A weighted grey CMAC neural network with output differentiability, International Fuzzy Systems Association and the North American Fuzzy Information Processing Society Joint Conference (IFSA/NAFIPS), vol.2, 25–28 July 2001, Vancouver, Canada, p. 1009–1014.
[22] Y. Wong, A. Sideris, Learning convergence in the cerebellar model articulation controller, IEEE Trans. Neural Networks 3 (1) (1992) 115–121.
[23] C.S. Lin, C.T. Chiang, Learning convergence of CMAC technique, IEEE Trans. Neural Networks 8 (6) (1997) 1281–1292.
[24] D. Rumelhart, G. Hinton, R. Williams, Learning representations by back-propagation errors, in: D. Rumelhart, et al. (Eds.), Parallel Distributed Proceeding, vol.1, MIT Press, Cambridge, MA, 1986, pp. 318–362.



Jiun-Chi Jan is currently an assistant professor in the Department of Civil Engineering at Ching Yun University, Jungli, Taiwan. He received his Ph.D. degree from the department of Civil Engineering at National Chiao Tung University, Hsinchu, Taiwan, in 2000. His research interests include computer-aided engineering, structural engineering, neural networks and data mining.

Chih-Ming Chen is currently an assistant professor in the Institute of Learning Technology at National Hualien University of Education, Hualien, Taiwan. He received his B.S. and M.S. degree from the department of Industrial Education at National Taiwan Normal University in 1992 and 1997, and received Ph.D. degree from the Department of Electronic Engineering at National Taiwan University of Science and Technology in 2002. His research interests include digital camera, e-health care, e-learning, artificial intelligence, machine learning and intelligent agents on the web.

Liang-Hao Hsiao is currently an instructor in the Department of Civil Engineering at Ching Yun University, Jungli, Taiwan. Also, he is a Ph.D. candidate in the Department of Civil Engineering at National Central University, Jungli, Taiwan. His research interests include materials of civil engineering and multiple criteria decision making.