Neural Process Lett (2008) 27:163–177
DOI 10.1007/s11063-007-9067-4

Improved MS_CMAC Neural Networks by Integrating a Simplified UFN Model

Jiun-Chi Jan · Shih-Lin Hung

Received: 4 April 2007 / Accepted: 28 November 2007 / Published online: 13 December 2007

Abstract Macro_Structure_CMAC (MS_CMAC) is a variational CMAC neural network that is designed for modeling smooth functional mappings. The MS_CMAC learning strategy involves constructing virtual grid-distributed data points from random-distributed training data points, and then using the virtual data points to train a tree structure network that is composed of one-dimensional CMAC nodes. A disadvantage of the MS_CMAC is that the prediction errors near the boundary area might sometimes be unexpectedly large. Another disadvantage of the MS_CMAC is that generating virtual grid-distributed data points generally takes a long computational time. Therefore, this study develops an improved model by integrating an unsupervised fuzzy neural network (UFN) into the MS_CMAC to initialize systematically the virtual grid-distributed data points. Additionally, a new error feedback ratio function is adopted to speed up the MS_CMAC training. Several numerical problems are considered to test the improved MS_CMAC. The computed results indicate that a simplified UFN model can produce good initial values of the virtual grid-distributed data points to aggrandize MS_CMAC training. The MS_CMAC prediction is also improved by using the initialized virtual grid-distributed data points.

Keywords Virtual grid-distributed data points · A tree structure network · UFN

1 Introduction

The CMAC (cerebellar model articulation controller) [1] is a fast-learning artificial neural network. It is used mainly in the control domain [2,3] and also applied to other fields such as civil engineering and web searching recently [4–6]. The MS_CMAC (macro structure

J.-C. Jan (✉)
Department of Computer Science and Information Engineering, Ching Yun University, 229 Chien-Hsin Road, Jungli, 320 Taiwan, R.O.C.
e-mail: jcjan@cyu.edu.tw

S.-L. Hung
Department of Civil Engineering, National Chiao Tung University, Hsinchu, Taiwan, R.O.C

CMAC) neural network [4] is a variational/modular CMAC model that is designed for modeling smooth functional mappings. The feature of the MS_CMAC is a tree structure network composed of one-dimensional CMAC nodes, and the MS_CMAC implements a dimensional reduction technique to substantially decrease the number of training instances required.

The principle of CMAC is simple. Each input pattern is assigned to a certain amount of weights; the CMAC output is the sum of the assigned weights; and the weights of the neighborhood input patterns mostly overlap. The initial version of CMAC adopts a constant basis function to perform locally weighted approximations of functions. According to the constant basis function, the prediction error of a training instance is evenly distributed to the assigned weights, and the CMAC output is generally a linear approximation in a local zone. Studies of CMAC have focused on developing high-order CMAC learning algorithms to improve CMAC prediction especially for the smooth functional mapping problems. For example, Lane et al. [7] proposed using a B-Spline receptive field function to replace the constant basis function, and Chiang and Lin [8] applied a Gaussian function as the basis function for the CMAC.

Developing a modular CMAC model is another strategy for improving CMAC prediction for smooth functional mapping problems. For example, Lin and Li [9] proposed a two-level tree structure network that was composed of small CMACs, and Hung and Jan [4] developed a tree structure network that was composed of one-dimensional CMAC nodes. Although the two modular CMAC models use the similar network structures, the advantages of the two CMAC models are entirely different. Lin and Li's CMAC model emphasizes the use of a small computer memory to solve high-dimensional problems because the other CMAC models for solving the problems generally require an extremely large computer memory. Hung and Jan's CMAC model, MS_CMAC, focuses on reducing the number of training instances required. The principle of the MS_CMAC is to decompose a multi-variable problem into several simple one-variable sub-problems. A set of constant space grid-distributed data points enable the MS_CMAC to use a quadratic weight transforming scheme [10] to generate smoothed weight functions under a single learning cycle.

The main advantage of the MS_CMAC is that it uses only a few training instances to predict with high accuracy because of to the dimensional reduction technique and the grid-distributed training data points. However, the grid-distributed training data points are not always available in practical applications. To overcome this flaw, Jan et al. [11] developed an inverse training scheme to supplement the MS_CMAC to construct virtual constant space grid-distributed data points from random-distributed training instances. The essentiality of the inverse training scheme is an optimization approach that uses a first-order heuristic method to update the virtual grid-distributed data points for minimizing the prediction errors of the random-distributed training instances. Jan et al. further indicated that a ratio 3:1 of random-distributed training instances to virtual grid-distributed data points is the minimum requirement for the inverse training scheme.

The training algorithm of MS_CMAC has greatly advanced recently [10,11]. However, MS_CMAC applications suffer from several shortcoming. First, the prediction errors near the boundary area might sometimes be unexpectedly large. Second, generating virtual grid-distributed data points generally takes a long computational time. Therefore, this study develops an improved MS_CMAC neural network by integrating a simplified UFN model (unsupervised fuzzy neural network) into the MS_CMAC. Additionally, a new error feedback ratio function is adopted to speed up MS_CMAC training. In the new model, the simplified UFN model mainly produces initial values of virtual grid-distributed data points, and the new error feedback ratio function is used to replace the linear error feedback ratio function in the original MS_CMAC. In the following sections, Sect. 2 gives a brief introduction for

MS_CMAC, Sect. 3 explains the improved MS_CMAC, Sect. 4 adopts several numerical cases to test the learning performance of the improved MS_CMAC, and Sect. 5 draws conclusions.

2 Review MS_CMAC

2.1 Tree Structure Network

This section uses an example to explain the computation of the MS_CMAC. Figure 1 shows a grid system with 3×4 grid-distributed data points. In Fig. 1, the circles denote virtual data points, the rectangles indicate temporary data points, and the triangle represents a target data point. First, three one-dimensional CMACs are utilized to model three x-direction line domains (i.e., $F(x, y_1)$, $F(x, y_2)$ and $F(x, y_3)$) by the virtual data points. Second, the corresponding outputs of the three temporary data points are calculated by the three one-dimensional CMACs. Third, another one-dimensional CMAC is utilized to model a y-direction line domain ($F(x_0, y)$) by the temporary data points. Fourth, the corresponding output of the target data point is calculated by the last one-dimensional CMAC. The aforementioned computation can be accomplished using a two-level tree structure network composed of four one-dimensional CMAC nodes (root and three leaves), and is shown in Fig. 2. The root CMAC is adopted to model $F(x_0, y)$, and leaf CMACs are adopted to model $F(x, y_1)$, $F(x, y_2)$, $F(x, y_3)$. Therefore, y-variable is called the active parameter of the root CMAC, and the x-variable is called the active parameter of the leaf CMAC. The grid-distributed data points are used as training instances for leaf CMACs; the temporary data points are used as training and testing instances for root CMAC and leaf CMACs, respectively; and the target data point is used as testing instance for root CMAC. The modular CMAC begins from leaf CMACs and proceeds to root CMAC. Restated, the MS_CMAC is based on a grid data system. If the numbers of gridlines in n directions are given by p_1, p_2, …, and p_n, respectively, then the MS_CMAC is a n-level tree structure network that is composed of $\left(\sum_{j=1}^{n-1} \prod_{i=1}^{j} p_i\right) + 1$ one dimensional CMAC nodes, and each CMAC node in the ith $(1 \leq i \leq n-1)$ level has p_i children nodes.

2.2 Training

The MS_CMAC has two levels of weights. The first level weights are a set of virtual grid-distributed data points, and the second level weights are spline weight functions in CMAC nodes. An inverse training scheme is employed to generate the first level weights, and a quadratic weight transforming scheme is adopted to calculate the second level weights. Notably, if the MS_CMAC is based on a constant space grid system, then the MS_CMAC predictions are smooth everywhere.

2.2.1 Inverse Training Scheme

The inverse training scheme is employed to generate the constant space virtual grid-distributed data points (first level weights) from the random-distributed training data points. Basically, the grid-system can be arbitrarily determined by users. However, the ratio of random-distributed training data points to virtual data points had better exceed three. The computation of the inverse training is similar to an optimization computing, where a

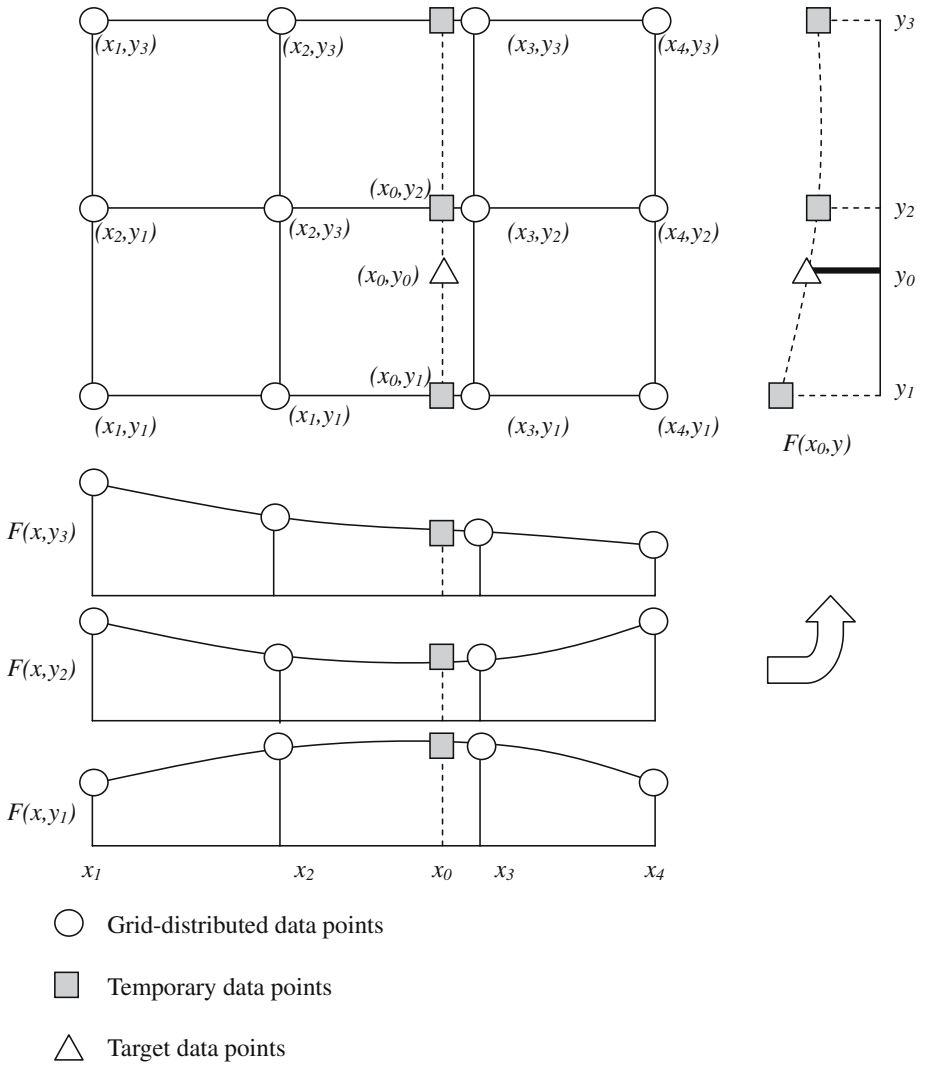

Fig. 1 Illustrate dimensional reduction technique of MS_CMAC

first-order heuristic method is adopted to update the virtual data points for minimizing the prediction errors of random-distributed training data points. The heuristic method assumes that a positive/negative error in the random-distributed training instance indicates a positive/negative updating for near virtual data points, and the error feedback ratios are linearly inverse proportion to the distance between the training instances and the virtual data points. Therefore, the virtual grid-distributed data points are updated by following equations

$$GY_i^{(n+1)} = GY_i^{(n)} + \alpha(\mathbf{GX}_i, \mathbf{TX}_j) \times (TY_j - Y_c) \tag{1}$$

$$\alpha(\mathbf{GX}_i, \mathbf{TX}_j) = \begin{cases} 1 - \frac{Dis(\mathbf{GX}_i, \mathbf{TX}_j)}{r_{\max}} & \text{if } Dis(\mathbf{GX}_i, \mathbf{TX}_j) < r_{\max} \\ 0 & \text{else} \end{cases} \tag{2}$$

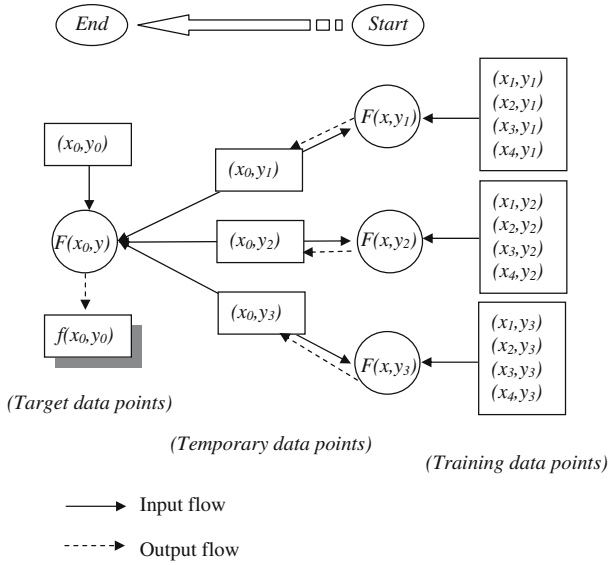

Fig. 2 Two-level tree structure of MS_CMAC for 3 × 4 grid system

$$Dis(\mathbf{GX}_i, \mathbf{TX}_j) = \sqrt{(gx_1 - tx_1)^2 + (gx_2 - tx_2)^2 + \cdots} \tag{3}$$

where $GY_i^{(n)}$ is the value of a virtual grid-state data point at the nth training iteration, $\mathbf{GX}_i$ is the position vector of the ith virtual data point, $\mathbf{TX}_j$ is the position vector of jth training instance, TY_j is the corresponding value of $\mathbf{TX}_j$, Y_c is the corresponding output of MS_CMAC, $\alpha(\mathbf{GX}_i, \mathbf{TX})$ is a first-order error feedback ratio function, gx_1, gx_2, … are elements of $\mathbf{GX}_i$, $tx_1, tx_2, \ldots$ are elements of $\mathbf{TX}_j$, and $r_{\max}$ is a threshold value defined as

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

2.2.2 Quadratic Weight Transforming Scheme

The quadratic weight transforming scheme [10] is a specific learning algorithm for one-dimensional CMAC nodes of the MS_CMAC. The output formula of an original one-dimensional CMAC is the summation of g connective weights, and is defined as follows

$$y(x_i) = \sum_{j=0}^{g-1} w_{i+j} \tag{4}$$

where y is the output function, x_i is the input variable, w_{i+j} is the addressing weight, and g is an integer which is called the generalization size. In each one-dimensional CMAC node, the Hanmin distance between two connective virtual training data points is set to g, which also represents the data point density. If a training data points (x_k, y_k) is given for a one-dimensional CMAC node, then the addressing weights can be calculated as following

$$w_{(k-1)\times g+j} = \frac{y_k}{g} \quad \text{for } j = 0,\ 1,\ 2,\ \ldots, g-1 \tag{5}$$

Equations (4) and (5) produce a step-state weight function and a multi-linear approximation for the one-dimensional CMAC node. To smoothen CMAC outputs, a set of quadratic splines ($f_k(i) = a_k i^2 + b_k i + c_k$) is proposed to replace the step-state weight function by satisfying the following equations

$$\int_{(k-1)\times g}^{k\times g} f_k(i)di = y_k \tag{6}$$

$$f_{k-1}(x_k) = f_k(x_k) \tag{7}$$

$$f'_{k-1}(x_k) = f'_k(x_k) \tag{8}$$

where x_k is the location of a virtual training data point, $f'_k(i)$ is the first-order differential of $f_k(i)$, Eq. (6) ensures that the CMAC prediction and virtual training data points fit well, and Eqs. (7) and (8) ensure that the CMAC prediction is smooth. As a result, the coefficients of the spline functions can be determined using linear algebra.

After the weight transformation is finished, a more complicated output formula that resembled to the numerical integration method of Simpon's rule is proposed to replace the original output formula. The new formula is defined as follows

$$y(x_i) = \frac{1}{3} \sum_{j=0, j=j+2}^{g-2} \left[f(x_{i+j}) + 4f(x_{i+1+j}) + f(x_{i+2+j}) \right] \tag{9}$$

3 Improved MS_CMAC

3.1 Influence of the Initial First Level Weights

In most neural networks, good network weights are generally difficult to define. However, the MS_CMAC does not have the problem. The first level weights of the MS_CMAC are the desired values that corresponded to the location of the grid-distributed data points. The second weights of the MS_CMAC are calculated from the first level weighs. Restated, the MS_CMAC has only a set of optimal weights when it handles a problem. A simple test is performed to elucidate the above opinion. Table 1 shows the learning results of an MS_CMAC that model a three-variable function ($\sin(x)\sin(2y)\sin(3z)$) using three sets (A, B and C) of initial grid-distributed data points, where 648 random-distributed training data points and a 6-6-6 grid system (216 virtual grid-distributed data points) with constant spaces are used. The A set of initial grid-distributed data points is calculated by adding -25% to 25% variances to the desired values, the B set of initial grid-distributed data points is calculated by adding -50% to 50% variances to the desired values, and the C set of initial grid-distributed data points is calculated by adding -100% to 100% variances to the desired values. As indicated from Table 1, the correlation coefficients that link the initial first level weights to the desired values are 0.8821, 0.7181 and 0.404 for the A, B and C sets of data, respectively. After the inverse training, the three correlation coefficients are raised to 0.9869, 0.9785, and 0.9207. The results reveal that the MS_CMAC training is better as the initial values of the virtual grid-distributed data points are closer to the desired values. Moreover, the numbers of learning cycle are 92, 148, and 248 by for the A, B and C sets of data, respectively. The good

Table 1 Learning result of an MS_CMAC to model a function by using three sets of initial grid-distributed data points

Initial values		A set	B set	C set
Correlation coefficients that link the first level weight to the desired values	Before inverse training	0.8821	0.7181	0.4040
	After inverse training	0.9869	0.9785	0.9207
Number of learning cycle		92	148	248
RMSE in learning phase	Before inverse training	0.1127	0.2134	0.4421
	After inverse training	0.0121	0.0125	0.0129

Note: RMSE = Root Mean Square Error

initial virtual grid-distributed data points also reduce the computational time in MS_CMAC training.

3.2 Improved Model

3.2.1 UFN Model

The original MS_CMAC generally randomly initialize its first level weights. The randomly generated values, however, almost certainly cannot be close to the desired values. Therefore, a method for systematically initializing the first level weights of the MS_CMAC is worth developing. Herein, a simplified UFN model (unsupervised neural network) is proposed for the MS_CMAC to initialize systematically virtual grid-distributed data points. The UFN can be considered as a nonlinear interpolation approach which uses local information near the target point to produce a prediction [12]. The prediction phase of the UFN is very simple. Basically, the UFN uses the following three steps to generate predictions: (1) find past cases that resemble the new problem by applying a competing algorithm; (2) establish a fuzzy set to represent the relationship between the similar cases and the new problem by applying a fuzzy membership function; and (3) generate a prediction based on the fuzzy set by applying a defuzzification formula. Additionally, the UFN model has a self-organized learning algorithm that selects systematically its working parameters. The self-organized learning algorithm has two parts. One part is called the weight adjusting process which uses a mathematical optimization algorithm to refine the weights in the competing algorithm. The other part is called the correlation analysis which uses a statistic method to calculate the threshold values of the fuzzy membership function. In this study, the weight adjusting process is not considered because the performance of the weight adjusting process depends on the high density of the training data.

3.2.2 Flow Chart of the Improved MS_CMAC

Figure 3 shows the flow chart of the improved MS_CMAC which is established by integrating a simplified UFN with an original type of MS_CMAC. The gray boxes are the new processes, which are described as follows.

Competing process identifies the degree of difference between the virtual grid-distributed data points and the training instances. The UFN uses a weighted Euclidean distance as index to identify the degree of difference between two data points. However, the improved MS_CMAC simply uses the Euclidean distance as index of the degree of difference. The

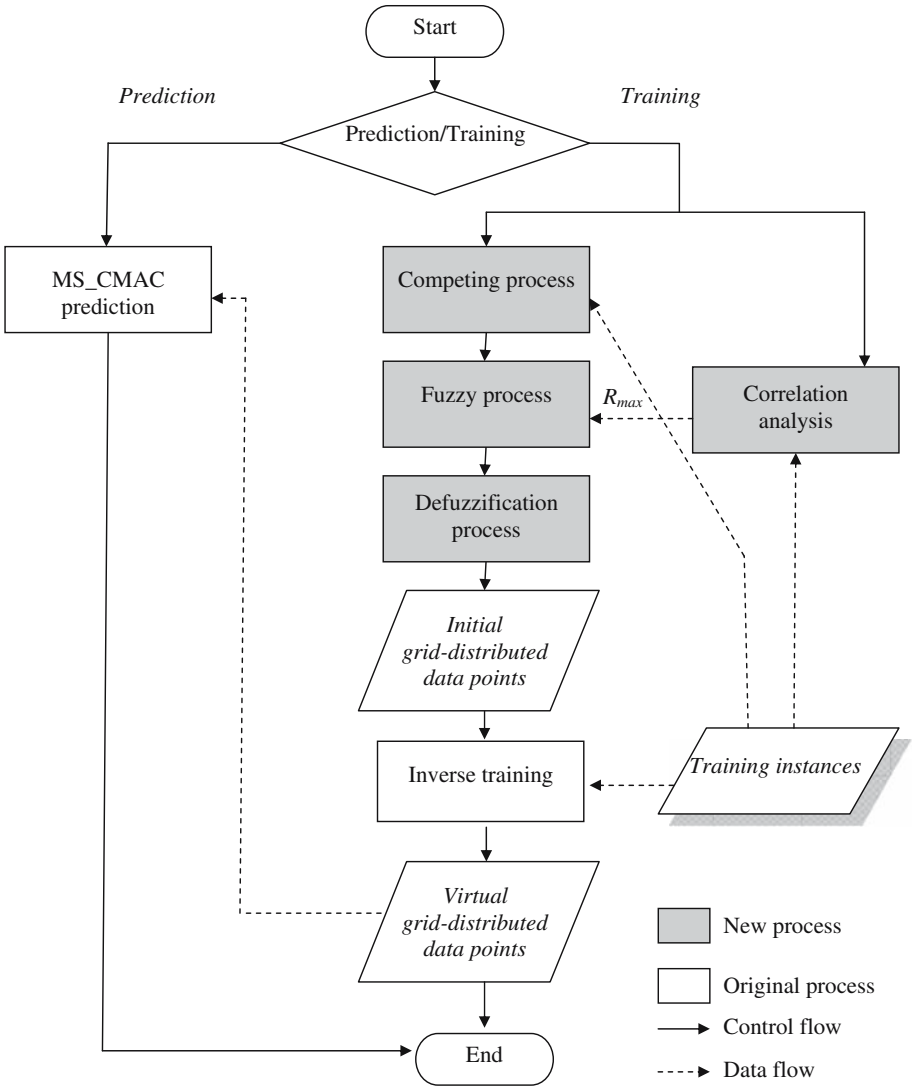

Fig. 3 Flow chart of the improved MS_CMAC

reason is that the weighted Euclidean distance is difficult to define when an MS_CMAC handle the low density of the training data. The formula for the Euclidean distance between a virtual data point and a training data point is defined as follows

$$d_{ij} = \left| GX_i - TX_j \right| = \sum_k (gx_k - tx_k)^2 \tag{10}$$

where d_{ij} denotes the difference degree between a grid data point and a training instance, and other symbols have defined in the previous section. In principle, small d_{ij} indicates the sign of high similarity.

Fuzzy process generates fuzzy sets to represent the relationships between grid-distributed data points and training instances. The UFN uses a quasi-Z type membership function in its fuzzy process. In this study, the improved MS_CMAC still uses the quasi-Z membership function. The quasi-Z membership function is defined as follows

$$\mu(d_{ij}, R_{\max}, R_{\min}) = \begin{cases} 0 & \text{if } d_{ij} > R_{\max} \\ \dfrac{R_{\max}R_{\min} - R_{\min}d_{ij}}{(R_{\max} - R_{\min})d_{ij}} & \text{if } R_{\max} \geq d_{ij} \geq R_{\min} \\ 1 & \text{if } R_{\min} > d_{ij} \end{cases} \tag{11}$$

where $R_{\min}$ and $R_{\max}$ are the bounds of the fuzzy membership function. $R_{\min}$ is generally a small value to avoid computer run-off error, and $R_{\max}$ can be calculated by the self-organized learning.

Defuzzification process initializes grid-distributed data points ($GY_i{}^{(0)}$) by synthesizing the output of the training instances (TY_j) according to the fuzzy sets that are calculated from the fuzzy process. The UFN proposed the center of gravity method, COG, and mean of maxima method, MOM, for different situations. The improved MS_CMAC uses only the COG method because of its simplicity. The COG method is defined as follows

$$GY_i^{(0)} = \frac{\sum_j \mu(d_{ij}) TY_j}{\sum_j \mu(d_{ij})} \tag{12}$$

Correlation analysis an assistant process that is used to determine systematically the value of $R_{\max}$ in Eq. (11). First, a two-column matrix (A) that is composed of all possible combination of output pairs from training instances is constructed. Each row vector of the matrix A should satisfy the following constraint

$$(A[k, 1], A[k, 2]) = (TY_i, TY_j) \quad \text{for } d_{ij} < t \text{ and } i < j$$

where t is an arbitrary real number. Then, the correlation coefficient between column 1 and column 2 of the matrix A is calculated. The computational formula is defined as follows

$$Cor(A(t)) = \frac{\sum_k (A[k, 1] - \overline{A_1})(A[k, 2] - \overline{A_2})}{nS_{A1}S_{A2}} \tag{13}$$

where S_{A1} and S_{A2} are the standard errors of column 1 and column 2 of matrix A, respectively, and $\overline{A_1}$ and $\overline{A_2}$ are the average values of column 1 and column 2 of matrix A, respectively. Finally, a line search algorithm is adopted to find the value of $R_{\max}$ that satisfies the following constraint equation

$$Cor(A(R_{\max})) = 0.8 \tag{14}$$

3.2.3 Second-order Error Feedback Ratio Function

The original type MS_CMAC uses a heuristic method to construct virtual grid-distributed data points, and the heuristic method is based on a linear error feedback ratio function. For further speeding up the inverse training of MS_CMAC, a second-order error feedback ratio function is proposed to replace Eq. (2), and is defined as follows

$$\alpha'(\mathbf{GX}_i, \mathbf{TX}_j) = \begin{cases} 1 - \left(\frac{Dis(\mathbf{GX}_i, \mathbf{TX}_j)}{r_{\max}}\right)^2 & \text{if } Dis(\mathbf{GX}_i, \mathbf{TX}_j) < r_{\max} \\ 0 & \text{else} \end{cases} \tag{15}$$

Basically, it is difficult to demonstrate that the second-order error feedback ratio function has better convergence capability than the linear error feedback ratio function by mathematical method. In the following section, an attempt is made to use the test results to explain the convergence capability of the second-order error feedback ratio function.

4 Numerical Case Studies

This section discusses three new types (I, II and III) of MS_CMAC for modeling smooth functional mapping problems, and uses the original MS_CMAC as a reference model. Type I model uses the randomly initialized first-level weights and a second-order error feedback ratio function in its training phases, the type II model implements the systematically initialized first-level weights by UFN and the original error feedback ratio function in its training phases, and the type III model uses the systematically initialized first-level weights and the second-order error feedback ratio function in its training phases.

4.1 Case I

This case is a three-variable smooth function, which has been discussed in the authors' previous research [10,11]. The three-variable function is defined as follows

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

A grid system with five constant grid spaces in each direction was selected for an MS_CMAC to model the function, and the gridline locations in each direction were defined as follows

$$x_1, x_2, x_3 : \{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$$

Therefore, 216 ($6 \times 6 \times 6$) virtual grid-distributed data points were necessary for the MS_CMAC prediction. According to the grid system, a three-level tree structure network with six branches in each level was adopted for the MS_CMAC to model the function. The *active parameters* for level 1–3 were set as x_1, x_2, and x_3, respectively, and the generalization sizes, g, were set to 40 for all one-dimensional CMAC nodes. As a result, the learning domain was decomposed into 8×10^6 ($40^3 \times 5 \times 5 \times 5$) data points by the MS_CMAC.

About 648 random-distributed data points were selected as training instance for the inverse training of MS_CMAC to construct 216 virtual grid-distributed data points. Another 1,000 random-distributed data points were selected as testing instances for verifying the improved MS_CMACs. Table 2 shows the computing results in case I. As indicated from Table 2, the correlation coefficient that links the initial first level weights (virtual grid-distributed data points) to their desired values is −0.0063 in the original model and type I MS_CMAC. However, the correlation coefficient that links the initial first level weights to their desired values is increased to 0.8467 in type II and III MS_CMAC. Comparing the learning speeds of the four type MS_CMACs, the learning is converged after 210, 161, 105 and 74 learning cycles in the original model, the type I, II, and III, respectively. The results indicate that the second-order error feedback ratio function substantially reduces the learning time of the inverse training of the MS_CMAC, independently of how the first level weights are initialized. Moreover, the initialized first level weights that are systematically generated by the UFN also accelerate the learning convergence of MS_CMAC. Comparing the prediction accuracies of the four types MS_CMAC, the prediction errors are 0.0581, 0.0587, 0.0190 and 0.0178 in original

Table 2 Computing results of four types of MS_CMAC in case I

Types		Original model	I	II	III
Correlation coefficients that link the first level weight to the desired values	Before inverse training	−0.0063	−0.0063	0.8467	0.8467
	After inverse training	0.9053	0.9138	0.9921	0.9932
Number of learning cycle/learning time (time reducing)		210/29 s (–)	161/23 s (23.3%)	105/15 s (50%)	74/10 s (64.8%)
RMSE in learning (improvement)		0.0167 (–)	0.0161 (3.6%)	0.0117 (29.9%)	0.0114 (31.7%)
RMSE in prediction (improvement)		0.0581 (–)	0.0587 (−1.0%)	0.0190 (67.3%)	0.0178 (69.4%)

Note: RMSE = Root Mean Square Error

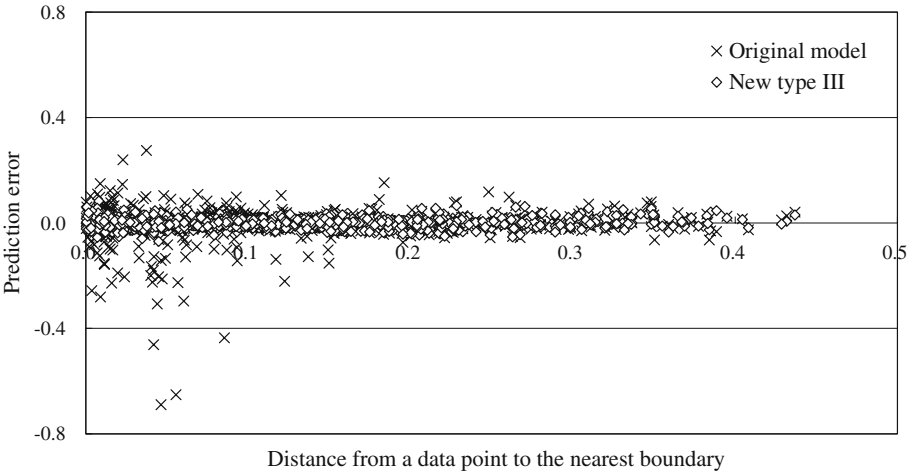

Fig. 4 Comparing the prediction errors of case I predicted by the original model and new type III MS_CMAC

model, type I, II, and III, respectively. The results reveal that the prediction improvement made by the MS_CMAC depends mainly on the systematically initialized first level weights. Furthermore, Fig. 4 shows the prediction errors of the 1,000 testing instances, where the vertical-axis denotes the prediction error and the horizontal-axis represent the distance from a testing instance to its nearest boundary of the learning domain. Clearly, the predictions of the testing instances near the boundary area are greatly improved.

4.2 Case II

This case is a four-variable smooth function, and has been discussed in references [9,11]. The four-variable function is defined as follows

$$F_2(x_1, x_2, x_3, x_4) = x_1 + \sin(x_1\pi)\cos(x_2\pi)\sin(3x_3\pi)\lfloor \sin^2(x_4\pi) - 1 \rfloor$$
$$\text{for } -1 \le x_1, x_2, x_3, x_4 \le 1$$

Table 3 Computing results of improved MS_CMAC in case II

Types		Original model	I	II	III
Correlation coefficients that link the first level weight to the desired values	Before Inverse training	0.0351	0.0351	0.8334	0.8334
	After inverse training	0.9650	0.9670	0.9887	0.9899
Number of learning cycle/learning time(time reducing)		326/ 202 s (–)	261/ 423 s (19.9%)	196/ 318 s (39.9%)	144/ 233 s (55.8%)
RMSE in learning (improvement)		0.0142 (–)	0.0135 (4.9%)	0.0131 (7.7%)	0.0126 (11.3%)
RMSE in prediction (improvement)		0.0474 (–)	0.0429 (9.5%)	0.0309 (34.8%)	0.0280 (40.9%)

Note: RMSE = Root Mean Square Error

A grid system with four constant grid spaces in the x_1 and x_3 directions, five constant grid spaces in the x_2 direction, and eight constant grid spaces in the x_4 direction was selected for the MS_CMAC to model the function. The gridline locations in each direction were defined as follows

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

Therefore, 1,350 ($5 \times 6 \times 5 \times 9$) virtual grid-distributed data points were necessary for the MS_CMAC prediction. According to the grid system, a four-level tree structure network was employed for MS_CMAC to model the function. The *active parameters* were set to 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, and the generalization sizes (g) were set to 40 for all one-dimensional CMAC nodes. As a result, the learning domain is decomposed into $1.6384 \times 10^9 (40^4 \times 4 \times 5 \times 4 \times 8)$ data points by the MS_CMAC.

About 4,050 random-distributed data points were selected as training instance for the inverse training of MS_CMAC to construct 1,350 virtual grid-distributed data points. Another 5,000 random-distributed data points were selected as testing instances for verifying the improved MS_CMACs. Table 3 shows the computed results in the case II. As indicated from Table 3, the correlation coefficient that links the initial first level weights (virtual grid-distributed data points) to their desired values is 0.0351 in the original model and type I MS_CMAC. However, the correlation coefficient that links the initial first level weights to their desired values are increased to 0.8334 in type II and III MS_CMAC. Comparing the learning speeds of the four type MS_CMACs, the learning is converged after 326, 261, 196 and 144 learning cycles in original model, type I, II, and III, respectively. Comparing the prediction accuracies of the four types MS_CMAC, the prediction errors are 0.0474, 0.0429, 0.0309 and 0.028 in original model, type I, II, and III, respectively. Furthermore, Fig. 5 shows the prediction errors of the 5,000 testing instances, where the vertical-axis denotes the prediction error and the horizontal-axis represent the distance from a testing instance to its nearest boundary of the learning domain. Clearly, the predictions of the testing instances near the boundary area are greatly improved.

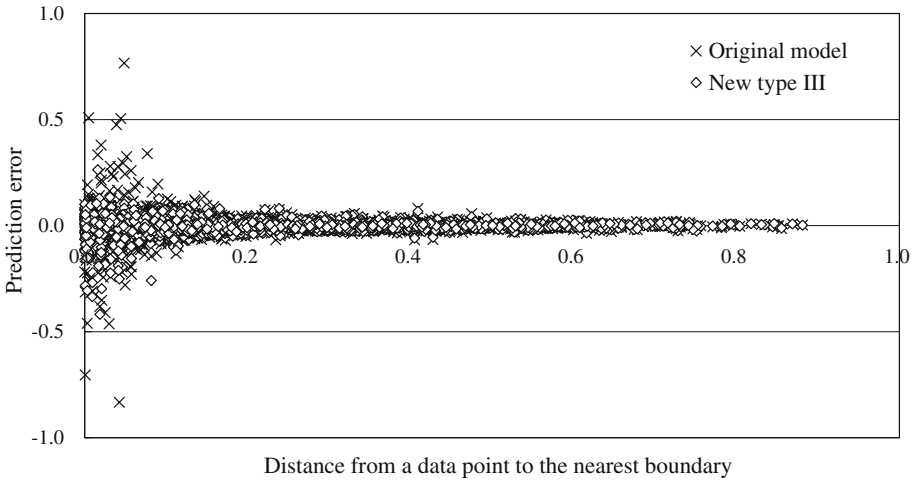

Fig. 5 Comparing the prediction errors of case II predicted by the original model and new type III MS_CMAC

4.3 Case III

This case is a four-variable continuous function, and has been also discussed in references [9,12]. The four-variable function is defined as follows

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

A grid system with four constant grid spaces in each direction was selected for the MS_CMAC to model the function, and 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, 625 ($5 \times 5 \times 5 \times 5$) virtual grid-distributed data points were necessary for the MS_CMAC prediction. According to the grid system, a four-level tree structure network with five branches in each level was employed for MS_CMAC. The *active parameters* for level 1–4 were set to x_1, x_2, x_3 and x_4, respectively, and the generalization sizes, g, were set to 40 for all one-dimensional CMAC nodes. As a result, the learning domain is decomposed into $6.5336 \times 10^8 (40^4 \times 4 \times 4 \times 4 \times 4)$ data points by the MS_CMAC.

About 1,875 random-distributed data points were selected as training instance for the inverse training of MS_CMAC to construct 625 virtual grid-distributed data points. Another 2,000 random-distributed data points were selected as testing instances for verifying the improved MS_CMACs. Table 4 shows the computed results in the case III. As indicated from Table 4, the correlation coefficient that links the initial first level weights (virtual grid-distributed data points) to their desired values is 0.0478 in the original model and type I MS_CMAC. However, the correlation coefficient that links the initial first level weights to their desired values are increased to 0.971 in type II and III MS_CMAC. Comparing the learning speeds of the four type MS_CMACs, the learning is converged after 318, 251, 102 and 77 learning cycles in original model, type I, II, and III, respectively. Comparing the prediction accuracies of the four types MS_CMAC, the prediction errors are 0.1393, 0.1380, 0.1194 and 0.1169 in original model, type I, II, and III, respectively. Furthermore, Fig. 6 shows the prediction errors of the 5,000 testing instances, where vertical-axis denotes prediction error

Table 4 Computing results of improved MS_CMAC in case III

Types		Original model	I	II	III
Correlation coefficient of the first level weight to the desired values	Before inverse training	0.0478	0.0478	0.9710	0.9710
	After inverse training	0.9490	0.9490	0.9743	0.9742
Number of learning cycle/learning time (time reducing)		318/ 1822 s (–)	251/ 1438 s (21.1%)	102/ 584 s (67.9%)	77/ 441 s (75.8%)
RMSE in learning (improvement)		0.1393 (–)	0.1380 (9.3%)	0.1194 (14.3%)	0.1169 (16.1%)
RMSE in prediction (improvement)		0.4461 (–)	0.4111 (7.8%)	0.2849 (36.1%)	0.2759 (38.2%)

Note: RMSE = Root Mean Square Error

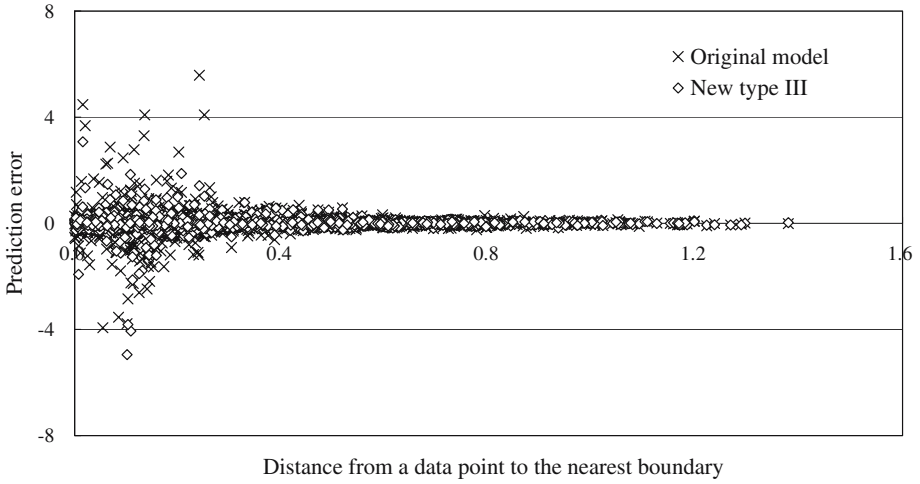

Fig. 6 Comparing the prediction errors of case III predicted by the original model and new type III MS_CMAC

and horizontal-axis represent the distance from a testing instance to its nearest boundary of the learning domain. Clearly, the predictions of testing instances near the boundary area are greatly improved.

5 Conclusion

This study integrates the simplified UFN model into the MS_CMAC to initialize systematically the first level weights of MS_CMAC to improve the prediction accuracy and accelerate the learning convergence. A second-order error feedback ratio function is also proposed for the MS_CMAC to speed up the learning convergence. Three numerical cases were considered to test the improved MS_CMAC. The testing results reveal that the improved MS_CMAC has obviously improvement in the learning phase. The improved MS_CMAC has a 11.3–31.7% lower learning error and a 55.8–75.8% lower learning time than that of the original MS_CMAC in the testing cases. The testing results also show that the improved

MS_CMAC exhibits much greater predictive accuracy than that of the original MS_CMAC. The improved MS_CMAC has a 38.2–69.4% lower prediction error than that of the original MS_CMAC in the testing cases. Moreover, the improvement of MS_CMAC prediction is very clear when the testing instances located in the area near the boundary of the learning domain.

References

1. Albus JS (1975) A new approach to manipulator control: the cerebellar model articulation controller (CMAC). J Dyn Syst Meas Control Trans ASME 97(3):220–227
2. Shelton RO, Peterson JK (1992) Controlling a truck with an adaptive critic CMAC design. Simulation 58(5):319–326
3. Lin CM, Peng YF (2004) Adaptive CMAC-based supervisory control for uncertain nonlinear systems. IEEE Trans Syst Man Cybern B Cybern 34(2):1248–1260
4. Hung SL, Jan JC (1999) MS_CMAC neural network learning model in structural engineering. J Comput Civil Eng ASCE 13(1):1–11
5. Kim DH (2002) Cerebellar model articulation controller (CMAC) for suppression of structural vibration. J Comput Civil Eng ASCE 16(4):291–298
6. Chen CM (2004) Incremental personalized web page mining utilizing self-organizing HCMAC neural network. Web Intell Agent Syst 2(1):21–38
7. Lane SH, Handelman DA, Gelfand JJ (1992) Theory and development of higher-order CMAC neural networks. IEEE Control Syst Mag 12(2):23–30
8. Chiang CT, Lin CS (1996) CMAC with general basis functions. Neural Netw 9(7):1199–1211
9. Lin CS, Li CK (1999) A memory-based self-generated basis function neural network. Int J Neural Syst 9(1):41–59
10. Jan JC, Hung SL (2001) High-order MS_CMAC neural network. IEEE Trans Neural Netw 12(3):598–603
11. Jan JC, Chen CM, Shiao LH (2006) Inverse training scheme for MS_CMAC to handle random data. Neurocomputing 70(1–3):502–512
12. Hung SL, Jan JC (2000) Augmented IFN learning model. J Comput Civil Eng ASCE 14(1):15–22