High-Order MS_CMAC Neural Network

J. C. Jan and Shih-Lin Hung

***Abstract*—A macro structure cerebellar model articulation controller (CMAC) or MS_CMAC was developed by connecting several one-dimensional (1-D) CMACs as a tree structure, which decomposes a multidimensional problem into a set of 1-D subproblems, to reduce the computational complexity in multidimensional CMAC. Additionally, a trapezium scheme is proposed to assist MS_CMAC to model nonlinear systems. However, this trapezium scheme cannot perform a real smooth interpolation, and its working parameters are obtained through cross-validation. A quadratic splines scheme is developed herein to replace the trapezium scheme in MS_CMAC, named high-order MS_CMAC (HMS_CMAC). The quadratic splines scheme systematically transforms the stepwise weight contents of CMACs in MS_CMAC into smooth weight contents to perform the smooth outputs. Test results affirm that the HMS_CMAC has acceptable generalization in continuous function-mapping problems for nonoverlapping association in training instances. Nonoverlapping association in training instances not only significantly reduces the number of training instances needed, but also requires only one learning cycle in the learning stage.**

***Index Terms*—Cerebellar model articulation controller (CMAC), high-order MS_CMAC, macro structure cerebellar model articulation controller (MS_CMAC), quadratic splines.**

I. Introduction

THE CEREBELLAR model articulation controller (CMAC), a supervised neural-network learning model developed by Albus [1], [2], is employed mainly in the control domain [3]–[5]. The original Albus CMAC (or the simple CMAC as it will be subsequently termed) produces a linear interpolation by using the binary basis function as a generalization scheme. Consequently, the output is constant within each quantized state and the derivative information is not preserved. For precisely modeling continuous functions, Moody [6] proposed using the graded neighborhood response functions (linear basis functions) in CMAC to perform continuous interpolation. Meanwhile, Lane *et al.* [7] developed a higher order CMAC neural network which uses the B-spline functions to replace the binary basis functions. In employing these functions, the method of updating weights distributes errors among the assigned weights according to the intensity of the B-spline functions. Additionally, Chiang and Lin [8] employed Gaussian basis functions as a generalization scheme. The Gaussuan basis function CMAC has been affirmed to be capable of learning both functions and function derivatives.

Generally, the CMAC requires extensive memory for mapping multidimensional functions. Lin and Li [9] proposed a CMAC structure comprising numerous small CMACs to learn a multi-dimensional function-mapping problem. In doing so, they intended to solve the fast size-growing problem and the learning difficulty in currently available types of neural networks. In their work, the network structure is always a three-layer tree structure. The root is the sum of the outputs of the nodes in the second level, called the submodules. Submodules are the multiplication of the outputs from the third level, which is a set of combinative small CMACs. Recently, Hung and Jan [10] developed a macro structure CMAC (MS_CMAC) neural network in structural engineering. The MS_CMAC represents a tree-based aggregate of one-dimensional (1-D) simple CMACs, where the ensemble is trained with a simplified time inversion technique borrowed from Albus' work [2]. The topology of the tree structure depends on a set of combinative training instances. Rather than employing a high-order basis function to enhance the prediction accuracy of MS_CMAC in nonlinear systems, a trapezium scheme is proposed in MS_CMAC. The trapezium scheme modifies the stepwise weight contents of CMACs in MS_CMAC into the trapezium-wise weight contents once each CMACs linear optimization process is complete. The trapezium scheme has been demonstrated to be able to provide pseudosmooth outputs in mapping continuous functions [10]. However, their scheme is limited to a cross-validation working parameter.

This work presents a quadratic splines scheme to upgrade the MS_CMAC, and called the high-order MS_CMAC (HMS_CMAC) neural-network model. The proposed scheme systematically alters the stepwise weight contents of CMACs in MS_CMAC into the smooth weight contents. The aim is not only to obtain smooth outputs but also to use a few training instances to achieve acceptable training. An illustrative example, three-variable functional mapping problem, demonstrates the effectiveness of the learning performance of the HMS_CMAC models. The HMS_CMAC model is implemented with MATLAB [11] on a personal computer.

II. Simple CMAC and Time Inversion CMAC

A. Simple CMAC

The simple CMAC learning algorithm is implemented mainly via three sequential mappings in four multidimensional spaces: input state space S, association memory space A, physical memory space P, and output space U. The learning stage in a simple CMAC progresses via three sequential mappings as follows. The first mapping is between input space S and association memory space A. In this step, the vector S_i in space S is mapped to an association vector G_i in space A. The vector G_i contains g elements, where g is termed generalization size. The next mapping is between association memory space A and physical memory space P. Any vector G_i in A is mapped to an active physical memory W_i. The third mapping is between

Manuscript received September 3, 1999; revised May 25, 2000 and December 18, 2000.

The authors are with the Department of Civil Engineering, National Chiao Tung University, Hsinchu, Taiwan 300, R.O.C.

Publisher Item Identifier S 1045-9227(01)03715-8.

physical memory space P and output space U. In a simple CMAC neural network, a function of linear combinations of the physically addressed memory in P is used in this mapping. Therefore, the output is calculated by summing the physically addressed memory in P as follows:

$$U_i = \sum_{j=1}^{g} w_{i,j} \tag{1}$$

where $w_{i,j}$ denotes the jth vector of W_i. Similar to other supervised neural network learning models, the computed output U_i is compared to the desired output U_{id}. If the difference between the computed and desired outputs exceeds a predefined threshold, the $w_{i,j}$ in (1) is updated [1]. The learning phase is terminated when the predefined stopping criterion is satisfied. After the learning stage is completed, the output U_i of any given instance S_i is directly computed via (1).

The simple CMAC produces a nonsmooth response owing to its binary basis function. High-order basis functions, such as the B-spline function [7], have been used in generalization schemes to obtain smoother weight contents. This approach attempts to find a smooth weight content in physical memory which provides a superior generalization in continuous functional mapping problems.

B. Time Inversion CMAC

Albus [2] proposed a time inversion technique to optimize the simple CMAC in multidimensional problems. Herein, the time inversion CMAC is developed using a simplified time inversion technique borrowed from Albus' work [2]. For example, given a set of instances with the six input variables $S'(x', y', z', u', v', w')$ and corresponding output U', plus an unsolved instance (verification instance) with the input $\tilde{S}(x, y, z, u, v, w)$, then the time inversion CMAC, combined by sequentially connecting two simple CMACs, can be briefly described as follows. 1) $S'(x', y', z', u', v', w')$ and U' are used to train the first simple CMAC of the time inversion CMAC according to the aforementioned three sequential mappings. The weight $P1$ of the first simple CMAC is then obtained after the CMAC is successfully trained. 2) Compute the outputs $\hat{U}$ corresponding to some specific instances of input $\hat{S}(x', y', z', u, v, w)$ through the first CMAC with weight $P1$. Herein, the instances $\hat{S}(x', y', z', u, v, w)/\hat{U}$ are termed transition instances. 3) These transition instances, $\hat{S}(x', y', z', u, v, w)/\hat{U}$, are then used for training the second simple CMAC in the time inversion CMAC. Consequently, the weight $P2$ of the second simple CMAC is obtained. Finally, the output $\tilde{U}$ corresponding to $\tilde{S}(x, y, z, u, v, w)$ can be computed through the second CMAC with weight $P2$.

III. HMS_CMAC

A. MS_CMAC

The underlying notion of MS_CMAC is to decompose a multidimensional problem into a set of 1-D subproblems [10]. The MS_CMAC learning algorithm is briefly reviewed below. Assume a set of combinative training instances is given. $S_k(s_1^t, \ldots, s_i^t, \ldots, s_n^t)$ denotes the input, and the corresponding output comprises m-data. Each input element $s_i^t \in \{s_{i,1}, s_{i,2}, \ldots, s_{i,p_i}\}$ where $s_{i,j} \in \{0, 1, \ldots, R\}$ and the subscript p_i represents the number of elements in the combination of s_i^t. According to these training instances, the topology of the MS_CMAC neural network, an n-level tree structure, is then determined [10]. To further depict the tree structure, a three-input combinative example is utilized to illustrate how to construct the MS_CMAC tree structure. In this example, $S(x, y, z)$ represents a training instance. The following combinations of three elements for x, y, and z, respectively, are considered as training instances:

$$\begin{aligned}
&x \in \{x_1, x_2, x_3\}, \quad \text{where } x_i \in \{0, 1, 2, \ldots, R\} \\
&\qquad \text{and } x_1 < x_2 < x_3; \\
&y \in \{y_1, y_2, y_3\}, \quad \text{where } y_i \in \{0, 1, 2, \ldots, R\} \\
&\qquad \text{and } y_1 < y_2 < y_3 \\
&z \in \{z_1, z_2, z_3\}, \quad \text{where } z_i \in \{0, 1, 2, \ldots, R\} \\
&\qquad \text{and } z_1 < z_2 < z_3.
\end{aligned}$$

Now, an MS_CMAC with a three-level tree structure is employed for the case, and is illustrated in Fig. 1. Thee nodes (N_1^2, N_2^2 and N_3^2) exist in the second level and nine nodes (N_1^3, N_2^3, $\ldots$, N_9^3) in the third level. Once the tree structure is set, the corresponding output U_u of any new instance $X(s_{1,u}, s_{2,u}, \ldots, s_{n,u})$ can be solved via the MS_CMAC neural network through the following steps:

Step 1) Instance X is used as a verification instance in the N_1^1 node (root). The training instances are divided into $\prod_{i=1}^{n-1} p_i$ groups and fed to nodes $N_{h_n}^n$ (leaves). Herein, $S_j'(s_{1,j}', \ldots, s_{i,j}', \ldots, s_{n,j}')$ denotes the jth training instance chosen from the training instance base for node $N_{h_n}^n$. The input element $s_{i,j}'$ for $N_{h_n}^n$ is defined as:

$$s_{i,j}' = s_{i,t_i}^t \qquad \text{for } i = 1 \text{ to } n; \quad \text{for } j = 1 \text{ to } p_n.$$

The subscript t_i is determined as follows:

$$\begin{aligned}
&\text{for } i = n \text{ downto } 1 \\
&\quad \text{if } i = n \text{ then } c_n = h_n \\
&\quad \text{else} \begin{cases} t_i = \begin{cases} p_i & \text{if } c_{i+1} \bmod p_i = 0 \\ c_{i+1} \bmod p_i & \text{else} \end{cases} \\ c_i = [(c_{i+1} - t_i)/p_i] + 1 \end{cases} \\
&\quad t_n = \{1, 2, \ldots, p_n\}.
\end{aligned} \tag{2}$$

In Fig. 1, the three training instances $S'(x_1, y_1, z_1)$, $S'(x_1, y_1, z_2)$ and $S'(x_1, y_1, z_3)$ are assigned to N_1^3, the three training instances $S'(x_1, y_2, z_1)$, $S'(x_1, y_2, z_2)$ and $S'(x_1, y_2, z_3)$ are assigned to N_2^3, and so on. Notably, the x and y elements are identical in leaves. Consequently, the leaves are practically to solve a one-dimensional problem (z-element). Herein, the z-element is defined as the *active parameter* in the third level.

Step 2) Generate transition instances for the nodes from levels n to 2. Each node in the tree has a transition instance, except for the root. The transition instance

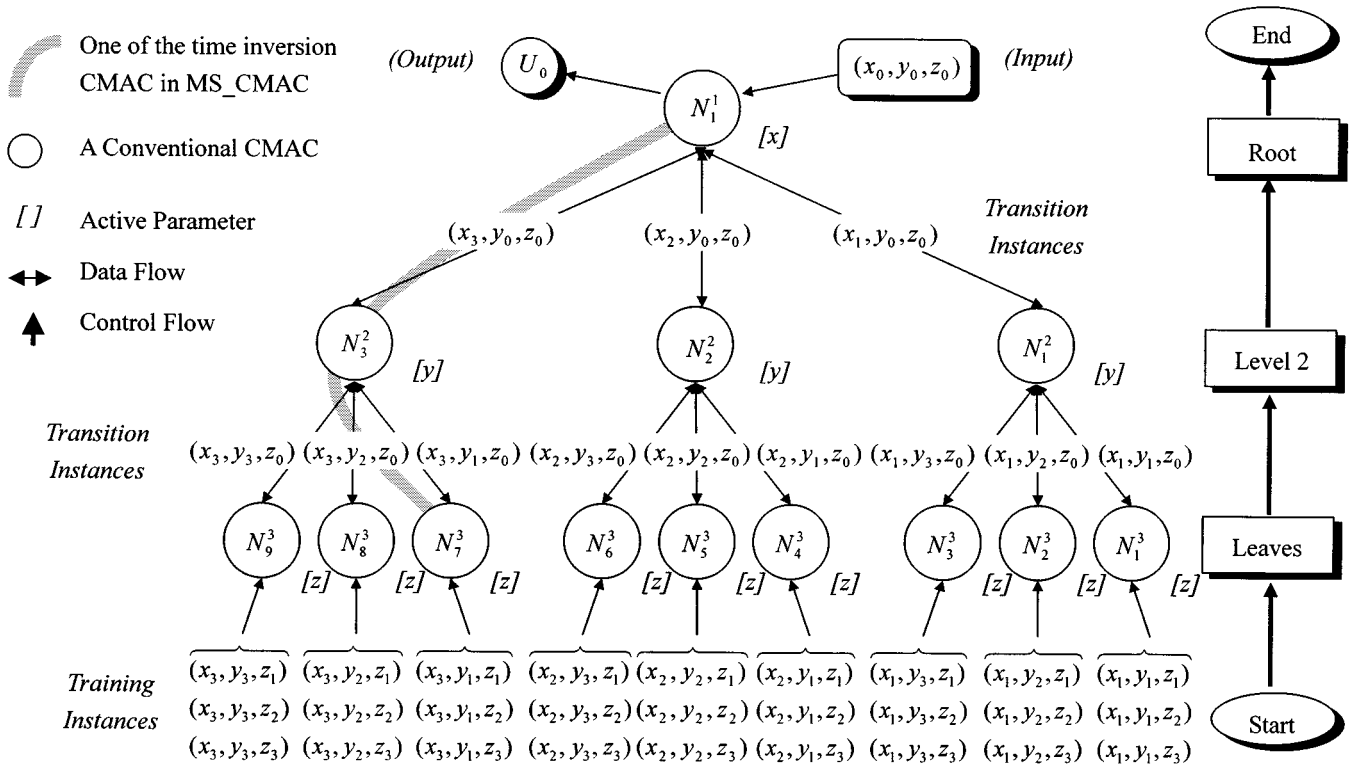

Fig. 1. The topology of a three-level MS_CMAC neural-network model.

is used to verify and train the node and the node's parent, respectively. The transition instance of $N_{h_l}^l$ (for $l = n$ to 2) is defined as follows:

$$\hat{S} = S^* \begin{bmatrix} I_{l-1} & 0 \\ 0 & 0 \end{bmatrix}_{n\times n} + X \begin{bmatrix} 0 & 0 \\ 0 & I_{n-l+1} \end{bmatrix}_{n\times n} \tag{3}$$

where

I_j $j \times j$ identity matrix;
S^* input of any training instance of node $N_{h_l}^l$;
X input of the verification instance of the root;
$\hat{S}$ input of the transition instance of node $N_{h_l}^l$.

The x and y variables of the transition instance corresponding to $N_{h_3}^3$ are the same as those of the training instances in $N_{h_3}^3$, and the z variables of the transition and verification instances are also the same. Similarly, the x variables of the transition instance and the training instances corresponding to $N_{h_2}^2$ are identical, and the y and z variables of the transition instance are the same as those of the verification instance. Consequently, the *active parameters* for the first and second levels are the x-element and y-element, respectively.

Step 3) Perform the learning and verification stages for all CMACs (nodes) from leaves to root, sequentially.

B. Quadratic Splines Scheme for Smoothing Weight Contents

Our previous work [10] adopted a trapezium scheme to assist MS_CMAC to model nonlinear systems instead of using high-order basis functions. The trapezium scheme attempts to modify stepwise weight content of CMACs in MS_CMAC after the linear optimization processes of CMACs are completed. The trapezium scheme has been confirmed to be effective in enhancing the prediction accuracy of MS_CMAC in nonlinear function-mapping problems [10]. However, obtaining proper working parameters for all 1-D CMACs in MS_CMAC is very difficult. Additionally, the trapezium scheme is only an approximate method and does not produce truly smooth outputs.

Accordingly, a quadratic splines scheme is developed to replace the trapezium scheme. Assume that the value of modified physical memory is defined as a quadratic spline $f_i(j) = a_i j^2 + b_i j + c_i$ in the interval x_i and x_{i+1}. The constants a_i, b_i, and c_i in any quadratic spline could be determined via the following conditions.

For $i = 1$ to z, the boundary conditions for each quadratic spline are as follows:

1) The areas between the interval of x_i and x_{i+1} must be equal for both the quadratic spline and the step function: $\int_{x_i}^{x_{i+1}} f_i(j)\,dj = w_i(x_{i+1} - x_i)$.
2) The function must be continuous across each spline boundary; that is, the function values at x_i (for $i = 2$ to $z - 1$) must be equal: $f_{i-1}(x_i) = f_i(x_i)$.
3) The function must be smooth at the boundary of two conjunct splines; that is, the first derivatives at x_i (for $i = 2$ to $z - 1$) must be equal: $f'_{i-1}(x_i) = f'_i(x_i)$.

Hence, $3z - 2$ boundary conditions are derived. The linear system has $3z$ unknowns, but only $3z - 2$ independent equations are available. Two more boundary conditions are required to solve the linear system of equations. Herein, a_1 and a_z are assumed to be equal to zero. Consequently, the physical memory is now modified as a set of quadratic spline functions.

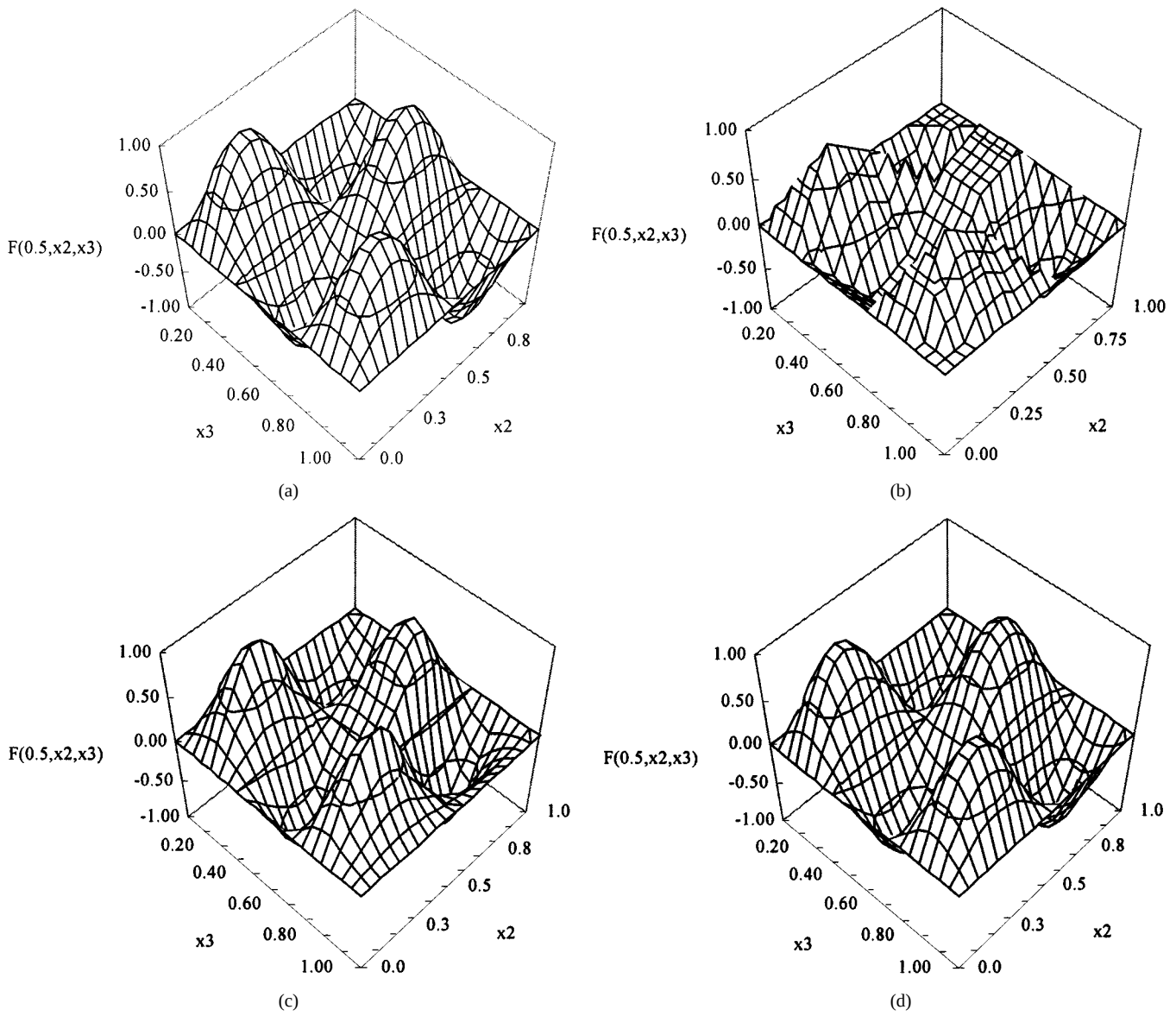

Fig. 2. (a) The plots of $F(0.5,\ x_2,\ x_3)$. (b) The computed output of $F(0.5,\ x_2,\ x_3)$ in MS_CMAC ($\gamma = 0$). (c) The computed output of $F(0.5,\ x_2,\ x_3)$ in MS_CMAC ($\gamma = 0.8$). (d) The computed output of $F(0.5,\ x_2,\ x_3)$ in HMS_CMAC.

Since the weight content is transformed from a discrete-type into a continuous type, the output is computed via the following equation:

$$U = \int_{j=s}^{s+g} f(j)\,dj = \sum_{j=s,\ s=s+2}^{s+g} \frac{f(j) + 4f(j+1) + f(j+2)}{3}. \tag{4}$$

C. Flow of HMS_CMAC

Herein, the HMS_CMAC is constructed via a tree-based structure CMAC with a quadratic splines scheme. Assume that one input set of combinative training data is given and defined as $x \in \{x_0, x_1, x_2, \ldots, x_{p1}\}$, then the set is transformed into the integer space as $s_x \in \{0, g, 2g, \ldots, p1 \times g\}$. Where term g is the generalization size. Notably, the value of g can be arbitrarily defined ($g > 1$) in an HMS_CMAC. Moreover, nonoverlapping association memory exists between training instances.

The computation in HMS_CMAC progresses from leaves to root and includes two stages in each node, the learning stage and the operation stage (1-D simple CMAC). The learning stage involves two steps. First, perform linear optimization. Second, after the linear optimization process is complete, smooth stepwise weight content via the quadratic splines scheme. Meanwhile, the operation stage involves calculating outputs using (4). The flow of HMS_CMAC for mapping an n-variable function is listed as follows.

1) Construct an n-level tree-based structure CMAC according to a set of combinative training instances.
2) Divide training instances into several groups for the leaves using (2).
3) Generate transition instances via (3).
4) Perform learning stages in leaves off-line.
5) Execute the operation stage in leaves on-line.
6) Perform learning and operation stages from $n-1$ level to root on-line.

TABLE I
COMPARISON OF CMAC MODELS

CMAC Models	Generalization Size (g)	Number of Training instances	Learning Convergence	Computational Time Ratio	RMSE (for verification)
HMS_CMAC	20	216	0	1.00	0.0204
MS_CMAC (γ=0.8)	20	216	0	0.55	0.1064
MS_CMAC (γ=0)	20	216	0	0.55	0.1397
Simple CMAC	20	216	0	0.35	0.1453
Simple CMAC	20	416	0.0001	3.41	0.1096

IV. ILLUSTRATIVE EXAMPLE

An illustrative example, mapping a three-variable nonlinear function, is employed to demonstrate the effectiveness of the learning performance of the HMS_CMAC model. The three-variable function is defined as follows:

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

Assume that the values of the three elements x_1, x_2, and x_3 are in the interval 0 to 1, respectively, and are linearly transformed into the inputs s_{x_1}, s_{x_2}, and s_{x_3} in the integer interval 0 to 100. Consequently, a total of 1 030 301 (101^3) distinct data exist in the learning domain. To train the neural network, the following combinations of three inputs are selected as training instances:

$$(s_{x_1}, s_{x2}, s_{x3}) \in \{\{0, 20, 40, 60, 80, 100\}, \{0, 20, 40, 60, 80, 100\} \{0, 20, 40, 60, 80, 100\}\}.$$

The 216 (6^3) training instances are used to train the HMS_CMAC for mapping the function $F(x_1, x_2, x_3)$. Additionally, a set of 200 instances is randomly selected for verification. According to the combinative training instances, the topology of the HMS_CMAC neural network is a three-level tree structure including one node in level 1 (root), six nodes in level 2, and 36 nodes in level 3. The *active parameters* for levels 1 to 3 are set as s_{x_1}, s_{x_2}, and s_{x_3}, respectively.

For comparison, two MS_CMAC with trapezium scheme ($\gamma = 0$ and $\gamma = 0.8$) are also used to map the function. Herein, the learning performance of HMS_CMAC is measured using the root mean square (rms) error and the ratio of computation time. The ratio of time for two CMAC models is defined as the ratio of computing time required by simple CMAC or MS_CMAC with trapezium scheme to that required by HMS_CMAC. Fig. 2(a) presents the plots of the function $F(x_1, x_2, x_3)$ with $x_1 = 0.5$. Fig. 2(b)–(d) show the plots corresponding to the computed output for the MS_CMAC (with $\gamma = 0$), MS_CMAC (with $\gamma = 0.8$), and HMS_CMAC, respectively. The HMS_CMAC clearly achieves the best prediction in this example. The output of HMS_CMAC is completely smooth. In addition, Table I compares the computational time of the various models. The HMS_CMAC requires much more computational time than a simple CMAC in multidimensional function-mappings given an identical number of training instances and generalization size. The ratio of computational time is around 0.35. However, the prediction accuracy of HMS_CMAC is significantly exceeds that of other models. To improve the prediction accuracy, an additional set of 200 training instances are randomly selected to train the simple CMAC. Thus, the number of training instances is increased to 416. Consequently, the learning time is exponentially increased, but the prediction accuracy remains poorer than that of the HMS_CMAC.

V. CONCLUSION

The work presents a novel CMAC model, named the HMS_CMAC, by utilizing a quadratic spline scheme to enhance the prediction accuracy after the weights are updated. The quadratic splines scheme systematically alters the stepwise weight contents of CMACs in MS_CMAC into smooth weight contents to perform smooth outputs. Based on the testing results in this study, we conclude the following:

1) The quadratic splines scheme provides a novel approach to obtain the smooth weight content in a one-dimensional CMAC. The quadratic splines scheme is directly used to modify stepwise weight content, which is yielded after the simple CMAC is successfully trained, into a set of splines.
2) The linear optimization process can be completed in just one-iteration in each node of an HMS_CMAC, due to the situation of no association overlap in the learning stage. Testing results indicate that the HMS_CMAC is confirmed to have a good capacity for generalization in continuous function-mapping problems when the situation of no association memory overlap is set in the learning stage.
3) The proposed scheme reduces the number of instances required for training an HMS_CMAC. Consequently, the total computational time can be significantly reduced, although the tree structure CMAC and splines quadratic scheme may increase computational time.

REFERENCES

[1] J. S. Albus, "A new approach to manipulator control: The cerebellar model articulation controller (CMAC)," *J. Dyn. Syst., Measurement, Contr.*, vol. 97, no. 3, pp. 220–227, 1975.
[2] ——, "Data storage in the cerebellar model articulation controller," *J. Dyn. Syst., Measurement, Contr.*, vol. 97, no. 3, pp. 228–233, 1975.
[3] C. S. Lin and K. Hyongsuk, "CMAC-based adaptive critic self-organized control," *IEEE Trans. Neural Networks*, vol. 2, pp. 530–533, May 1991.
[4] T. W. Miller, R. P. Hewes, F. J. Glanz, and L. G. Kraft, "Real-time dynamic control of an industrial manipulator using a neural-network-based learning controller," *IEEE Trans. Robot. Automat.*, vol. 6, pp. 1–9, 1990.

[5] P. C. Parks and J. Militzer, "A comparison of five algorithm for the training of CMAC memories for learning control systems," *Automat. Remote Contr.*, vol. 50, pp. 254–286, 1989.
[6] J. Moody, "Fast learning in multiresolution hierarchies," *Advances Neural Inform. Processing Syst. I*, pp. 29–39, 1989.
[7] S. H. Lane, D. A. Handelman, and J. J. Gelfand, "Theory and development of higher-order CMAC neural networks," *IEEE Contr. Syst. Mag.*, vol. 12, no. 2, pp. 23–30, 1992.
[8] C. T. Chiang and C. S. Lin, "CMAC with general basis functions," *Neural Networks*, vol. 9, no. 7, pp. 1199–1211, 1996.
[9] C. S. Lin and C. K. Li, "A new neural structure composed of small CMACs," in *Proc. ICNN*, vol. 3, Washington, DC, Jun. 1996, pp. 1777–1783.
[10] S. L. Hung and J. C. Jan, "MS_CMAC neural-network learning model in structural engineering," *J. Computing in Civil Engrg., ASCE*, vol. 13, no. 1, pp. 1–11, 1999.
[11] *MATLAB User's Guide*. Natick, MA: The MATH WORKS, 1996.