MS_CMAC NEURAL NETWORK LEARNING MODEL IN STRUCTURAL ENGINEERING

By Shih-Lin Hung[1] and J. C. Jan[2]

ABSTRACT: The present American Institute of Steel Construction specifications use the alignment charts and approximate formulas conveniently to determine some coefficients in design, such as moment gradient coefficient C_b for beams of I-shaped section and effective length factor K of columns. In these methods, the coefficients are unconservative when the boundary conditions are different from the development of specifications. The governing equations, numerical approaches, on the K and C_b coefficients provide more accurate results. The approaches, however, are not readily available for structural engineers to use in design. Applying neural network computing toward structural engineering problems has received increasing interest, with particular emphasis placed on supervised neural networks. The cerebellar model articulation controller (CMAC), one of the supervised neural network learning models, is mostly used in the domain of control. In this work, we use a newly developed Macro Structure CMAC (MS_CMAC) neural network learning model to aid steel structure design. The topology of the novel learning model is constructed by a number of time inversion CMACs as a tree structure. The learning performance of the MS_CMAC is first compared with a stand-alone time inversion CMAC using one structural engineering example. That comparison indicates not only superior prediction but also fast learning propriety for the MS_CMAC neural network learning model. In addition, the MS_CMAC neural network learning model is applied to two steel design problems. It is shown that the MS_CMAC not only can learn structural design problems within a reasonable central processing unit time but also can estimate more accurate coefficients than that estimated through alignment charts and approximate formulas in American Institute of Steel Construction specifications.

Downloaded from ascelibrary.org by National Chiao Tung University on 05/01/14. Copyright ASCE. For personal use only; all rights reserved.

INTRODUCTION

Learning is one of the important features of artificial neural networks. Several neural networks learning algorithms have been developed and explored in a number of different domains. In the application of structural engineering, most research works were based on the supervised back-propagation neural network (BPN) (Rumelhart et al. 1986) due to its simplicity. Vanluchene and Sun (1990) presented a research using the back-propagation learning algorithm in structural design problems. Several other researches, based on BPN learning algorithm, have applied neural network computing models in structural engineering and related engineering problems [Ghaboussi et al. (1991), Hajela and Berke (1991), Elkordy et al. (1994), Goh (1995), Kasperkiewicz et al. (1995), Mukherjee and Desphande (1995), and others].

BPN learning models, however, always take a long time in learning process. Several approaches for improving the learning performance of a BPN learning algorithm have been achieved and reported in recent literature. One approach is to develop more effective learning algorithms with the objective of reducing the learning time. For instance, Adeli and Hung (1994) developed an adaptive conjugate gradient neural network (Ad-CGN) learning algorithm and applied it to structural engineering. Based on a limited memory BFGS quasi-Newton second-order method (Nocedal 1990), a more effective adaptive L-BFGS learning algorithm was developed by Hung and Lin (1994). Another approach is to develop a parallel algorithm on multiprocesser computers with the objective of reducing the overall computing time. For instance, Adeli and Hung (1993) presented a concurrent Ad-CGN learning algorithm to a large-scale pattern recognition problem. Significant improvement for the BPN algorithm in computing time was reported in their work. The third approach is the development of hybrid neural network learning algorithms. Hung and Adeli (1994) presented a parallel hybrid genetic/neural network learning algorithm, integrating a genetic algorithm with the Ad-CGN learning algorithm, in engineering and pattern recognition problems. They reported a superior convergence property of the parallel hybrid neural network learning algorithm as compared with a BPN learning algorithm. Furthermore, Gunaratnam and Gero (1994) discussed the effect of representation of input/output pairs for training instances on the learning performance of the BPN learning algorithm in the problems of structural design.

Unsupervised learning models are the other major neural network learning techniques and have been applied to several domains, mostly in the problems of classification. In structural engineering problems, Adeli and Park (1995) compared the learning performance of counterpropagation neural networks (CPN) and BPN. The CPN learning model is a combination of supervised and unsupervised mapping neural network learning models. Kasperkiewicz et al. (1995) used a fuzzy neural network to predict the strength of high performance concrete. The fuzzy neural network, called fuzzy-ARTMAP, is a mapping neural network of combining Kohonen learning and Grossberg learning algorithms (Hecht-Nielsen 1987). Recently, Hung and Jan (1997) presented an integrated fuzzy neural network learning model by integrating a novel unsupervised fuzzy neural network reasoning model with a supervised learning model in the domain of structural engineering. The unsupervised fuzzy neural network reasoning model was developed on the basis of a single-layer laterally connected neural network with an unsupervised competing algorithm. The integrated fuzzy neural network learning model demonstrated its superior learning performance in complicated structural design problems with a reasonable computational time.

A cerebellar model articulation controller (CMAC) neural network, one of the supervised neural networks learning models, is mostly used in the domain of control. The merits of a CMAC neural network are not only its fast convergence in the learning phase but also its capability of mapping complicated nonlinear functions (Albus 1975a). However, the performance of a well-trained CMAC neural network in verification phases

[1] Assoc. Prof., Dept. of Civ. Engrg., Nat. Chiao Tung Univ., 1001 Ta Hsueh Rd., Hsinchu Taiwan 300, R.O.C.

[2] PhD Candidate, Dept. of Civ. Engrg., Nat. Chiao Tung Univ., 1001 Ta Hsueh Rd., Hsinchu Taiwan 300, R.O.C.

Note. Discussion open until June 1, 1999. To extend the closing date one month, a written request must be filed with the ASCE Manager of Journals. The manuscript for this paper was submitted for review and possible publication on March 6, 1998. This paper is part of the ***Journal of Computing in Civil Engineering***, Vol. 13, No. 1, January, 1999. ©ASCE, ISSN 0877-3801/99/0001-0001–0011/$8.00 + $.50 per page. Paper No. 17857.

Downloaded from ascelibrary.org by National Chiao Tung University on 05/01/14. Copyright ASCE. For personal use only; all rights reserved.

is extremely dependent upon some predefined working parameters, such as the size of association memory and physical memory used. To improve the performance of CMAC neural network learning models, Albus (1975b) proposed a time inversion technique to refine computations in a simple CMAC. Recently, Lin and Chiang (1997) studied the simple CMAC convergence properties with mathematical formulations and concluded that the iterative learning in a simple CMAC neural network always converges.

In this work, we present a newly developed Macro Structure CMAC (MS_CMAC) neural network learning model based on a time inversion CMAC learning algorithm in the domain of structural engineering. The topology of the novel learning model is constructed by a number of time inversion CMACs as a tree structure. The MS_CMAC learning model is first compared with a stand-alone time inversion CMAC using an engineering design example, the concrete beam design problem, from recent literature (Vanluchene and Sun 1990; Gunaratnam and Gero 1994). In addition, the MS_CMAC neural network is applied to two steel structural design problems. The first example is taken from the literature (Adeli and Park 1995). It is the prediction of the moment-gradient coefficient C_b for doubly and singly symmetric steel beams. The second example that is a new problem is the prediction of effective length factor K of columns in unbraced frames. The two examples are also used to train a supervised L-BFGS learning model for the sake of comparison.

SIMPLE AND TIME INVERSION CMAC NEURAL NETWORKS

Simple CMAC Neural Network

A simple CMAC learning algorithm is primarily implemented with three sequential mappings in four multidimensional spaces: (1) Input state space S; (2) association memory space A; (3) physical memory space P; and (4) output space U. The three sequential mappings are $S \rightarrow A$, $A \rightarrow P$, and $P \rightarrow U$, respectively. A simple CMAC neural network learning model with three mappings in four spaces is schematically depicted in Fig. 1. A set of p training instances is given, and each instance consists of n patterns in input and m data in output. The corresponding four spaces are, herein, defined as follows:

- $\mathbf{S}$ is a set of p vectors and each vector contains n components.
- A is a pseudotable and is physically assigned storage space as the elements of the table are indicated. The term α_k, in Fig. 1, is an integer number that denotes the kth element in A.
- $\mathbf{P}$ is a matrix with $m \times r$ elements. The value r denotes the size of physical memory P and it is varying in different learning problems. The term $\mathbf{P}_l$ is the lth row vector in P.
- The output $\mathbf{U}$ is a set of p vectors and each vector contains m components.

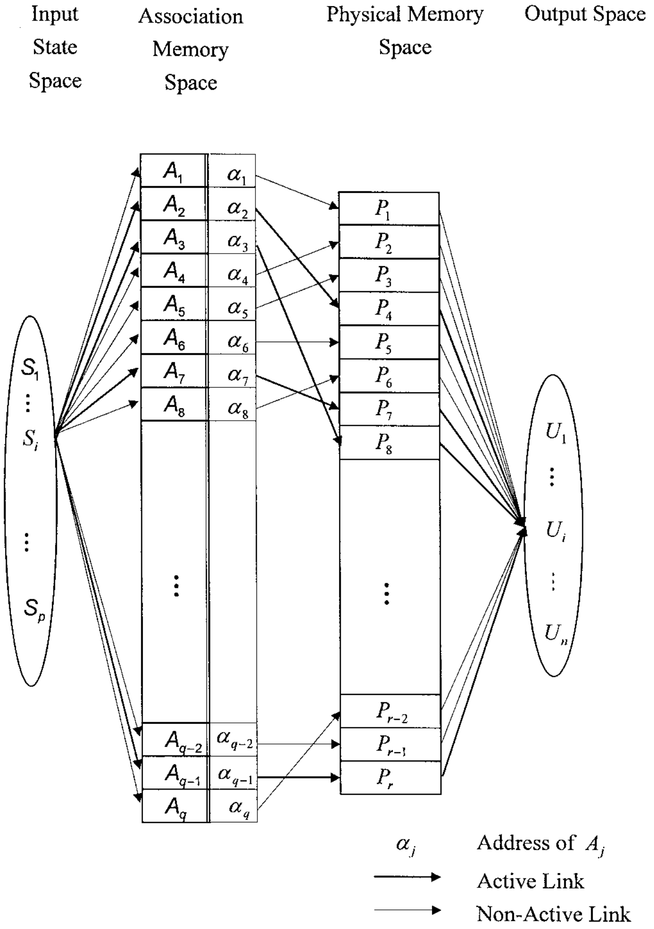

FIG. 1. Simple CMAC Neural Network in Four Spaces with Three Corresponding Mappings

Assume that $\mathbf{S}_i$ is any given training instance, denoted as a vector in S, and $\mathbf{U}_{id}$ is the corresponding output vector. The learning phase in a simple CMAC is performed by means of three sequential mappings in the following steps. The first mapping is between input space S and association memory space A. In this step, the vector $\mathbf{S}_i$ is mapped to an association vector $\mathbf{G}_i$ in space A by means of a predefined function $S\_A(S_i)$. The vector $\mathbf{G}_i$ contains g components. The entity g is a predefined integer number, called generalization size (Albus 1975a). In this work, the function $S\_A(S_i)$ is defined as a function of Hamming distance (Albus 1975a). A situation, in which the Hamming distance between two vectors $\mathbf{S}_i$ and $\mathbf{S}_j$ is less than g, indicates that parts of their association vectors $\mathbf{G}_i$ and $\mathbf{G}_j$ are overlapped. Otherwise, if these two instances markedly differ from each other, the elements in both association vectors $\mathbf{G}_i$ and $\mathbf{G}_j$ are distinct.

Succeeding the first mapping, the next mapping is between association memory space A and physical memory space P. In a simple CMAC neural network, the size of space A is generally much larger than the size of physical memory (hardware of a computer) P. Hence, this mapping is from a large space into a small storage space. Herein, the technique of hash-coding is adopted to perform the mapping. A hash function, $A\_P(G_i)$, is used to map any vector $\mathbf{G}_i$ in A to an active physical memory vector $\boldsymbol{\beta}^i$ with g components in space P. The entity β^i_j is the jth element of vector $\boldsymbol{\beta}^i$.

The third mapping is between 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 vector is calculated by summarizing the physically addressed memory in P as

$$\mathbf{U}_i = \sum_{a=1}^{g} P_{\beta^i_a} \tag{1}$$

Being the same as other supervised learning models, the computed output $\mathbf{U}_i$ is then compared with the desired output $\mathbf{U}_{id}$. If the difference between the computed and desired outputs is larger than a predefined threshold, then the physical memory $P_{\beta^i_a}$ in (1) is updated as

$$P^{(k+1)}_{\beta^i_a} = P^{(k)}_{\beta^i_a} + \frac{\mu}{g}(\mathbf{U}_{id} - \mathbf{U}_i), \quad a = 1 \text{ to } g \tag{2}$$

Downloaded from ascelibrary.org by National Chiao Tung University on 05/01/14. Copyright ASCE. For personal use only; all rights reserved.

where the superscript $(k + 1)$ is used to indicate the $(k + 1)$th learning step; and the quantity μ = predefined learning ratio. In this work, μ is set as a real number in the interval of [0, 1] instead of 1.0 in Albus's work. The learning phase is terminated as the predefined stopping criterion is met.

Time Inversion CMAC Neural Network

A simple CMAC neural network has the property of fast convergence in learning phase (Albus 1975a). However, the prediction performance of a well-trained CMAC neural network in verification phases is extremely dependent upon some predefined working parameters, such as the size of association memory A and physical memory P used. To improve the performance of CMAC neural network learning models, Albus (1975b) proposed a time inversion technique to refine computations in a simple CMAC.

For instance, a set of instances, $S'(x', y', z', u', v', w')$, with six decision variables in input and the corresponding output U' are given. An unsolved instance, $X(x, y, z, u, v, w)$, is used as a verification instance. Then, the problem can be solved by a time inversion CMAC with two connected simple CMACs as shown in Fig. 2. The computations of the time inversion CMAC neural network can be summarized as the following steps:

- Step 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, $S \rightarrow A$, $A \rightarrow P$, and $P \rightarrow U$.
- Step 2. The output $\tilde{U}$ corresponding to $\tilde{S}(x', y', z', u, v, w)$ is computed using the first trained simple CMAC using (1). Herein, the instances $[\tilde{S}(x', y', z', u, v, w)\tilde{U}]$ are called transition instance.
- Step 3. $\tilde{S}(x', y', z', u, v, w)$ and $\tilde{U}$ are used as training instances to train the second simple CMAC of the time inversion CMAC.
- Step 4. Finally, the output U corresponding to $X(x, y, z, u, v, w)$ is computed through the second simple CMAC.

The aforementioned four steps are denoted as Sequences (1) and (2) in the first CMAC and Sequences (3) and (4) in the second CMAC in Fig. 2, respectively.

MS_CMAC NEURAL NETWORK LEARNING MODEL

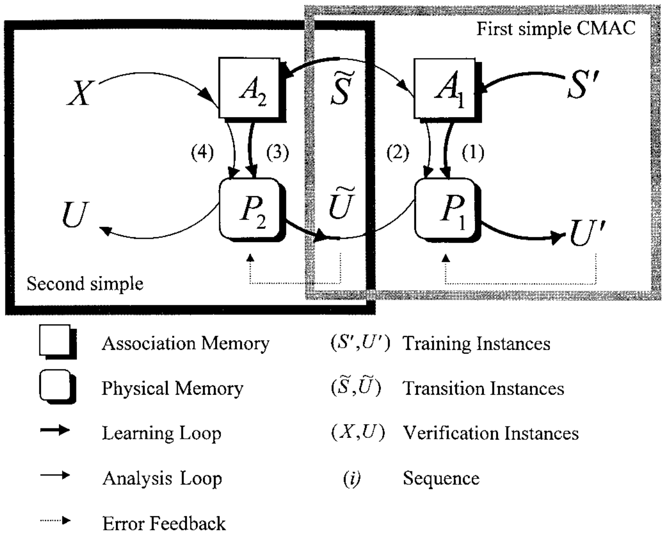

FIG. 2. Time inversion CMAC Neural Network with Two Connected Simple CMACs

This work presents a novel MS_CMAC neural network learning model. The learning algorithm associated with the proposed model is based on the concept of dimensional reduction. Hence, connecting a number of time inversion CMACs as a tree structure allows us to construct the topology of the MS_CMAC. Hereinafter, the learning algorithm of an MS_CMAC neural network is explicated with a simple example with three decision variables x, y, and z. The following combinations of three decision variables are considered as training instances:

$$x = \{x_1, x_2, x_3\}$$

$$y = \{y_1, y_2, y_3\}$$

$$z = \{z_1, z_2, z_3\}$$

Thus, this example contains a total of 27 (3^3) distinct training instances. A vector $\mathbf{X}(x_u, y_u, z_u)$ is the input of any verification instance.

The problem can be solved using an MS_CMAC neural network with a three-level tree structure schematically depicted in Fig. 3. Each node of the tree denotes a simple CMAC neural network. The link between any two connected nodes indicates the data flow. Cumulatively, the tree contains 12 links.

After the topology of the MS_CMAC neural network is derived, the training instances are divided into nine groups and presented to the leaves. Simultaneously, the verification instance is presented to the root. Each node in the level two and the leaves is only fed a transition instance during the verification phase. Restated, this case contains 12 transition instances. After the output of each transition instance is computed through a corresponding node, the transition instances are used in the training phase of the node's parent. In each node, only one input pattern is considered as a key-mapping variable. The input pattern is referred to herein as an active parameter. Notably, the active parameter differs according to each level. Herein, these parameters are x, y, and z in root, level two, and leaf, respectively.

The computations, then, of the MS_CMAC neural network can be implemented in three stages. First, consider the decision variable z as the active parameter. Nine transition instances can be generated. Each transition instance is computed by means of a simple CMAC trained by three of the training instances. Hence, the original three-dimensional (3D) problem is reduced to a two-dimensional (2D) problem.

Next, another variable (i.e., y) is considered as the active parameter in this stage. Three new transition instances can be generated by means of the aforementioned nine transition instances using three simple CMACs. Each new transition instance is computed by a simple CMAC trained by three transition instances generated in the first stage. Therefore, in this work, the 2D problem is altered to a one-dimensional (1D) problem only.

In the final stage, the output of the verification instance x can be computed by a simple CMAC (i.e., trained using the three transition instances generated in the second stage). Instead of using 27 training instances to map a verification instance in a 3D domain, the verification instance is mapped using three transition instances in a 1D space.

Computational performance in an MS_CMAC neural network depends on the number of given training instances. Assume that the decision variables x, y, and z have N distinguishable values; there are N^3 distinguishable possible training instances. To resolve this problem, an amount of N^3 association memory A spaces is required by using a simple CMAC neural network at least. However, only $13N$ $(N + 3N + 9N)$ association memory A spaces are used in an MS_CMAC neural network. Obviously, the amount of association memory spaces used for solving the problem is significantly reduced in the MS_CMAC neural network. Consequently, the corre-

Downloaded from ascelibrary.org by National Chiao Tung University on 05/01/14. For personal use only; all rights reserved.

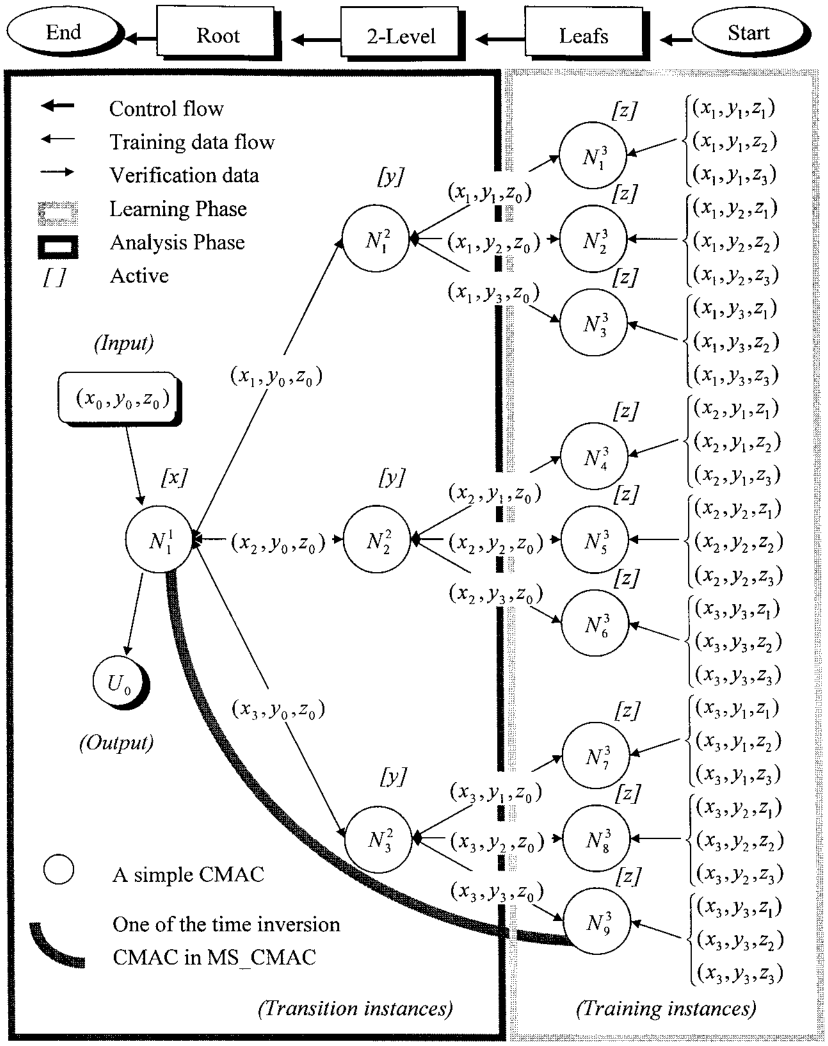
End
Root
2-Level
Leafs
Start
Control flow
Training data flow
Verification data
Learning Phase
Analysis Phase
[ ] Active
(Input)
(x_0, y_0, z_0)
[x]
N_1^1
U_0
(Output)
[y]
N_1^2
N_2^2
N_3^2
(x_1, y_0, z_0)
(x_2, y_0, z_0)
(x_3, y_0, z_0)
(x_1, y_1, z_0)
(x_1, y_2, z_0)
(x_1, y_3, z_0)
(x_2, y_1, z_0)
(x_2, y_2, z_0)
(x_2, y_3, z_0)
(x_3, y_1, z_0)
(x_3, y_2, z_0)
(x_3, y_3, z_0)
[z]
N_1^3
N_2^3
N_3^3
N_4^3
N_5^3
N_6^3
N_7^3
N_8^3
N_9^3
(x_1, y_1, z_1) (x_1, y_1, z_2) (x_1, y_1, z_3)
(x_1, y_2, z_1) (x_1, y_2, z_2) (x_1, y_2, z_3)
(x_1, y_3, z_1) (x_1, y_3, z_2) (x_1, y_3, z_3)
(x_2, y_1, z_1) (x_2, y_1, z_2) (x_2, y_1, z_3)
(x_2, y_2, z_1) (x_2, y_2, z_2) (x_2, y_2, z_3)
(x_3, y_3, z_1) (x_3, y_3, z_2) (x_3, y_3, z_3)
(x_3, y_1, z_1) (x_3, y_1, z_2) (x_3, y_1, z_3)
(x_3, y_2, z_1) (x_3, y_2, z_2) (x_3, y_2, z_3)
(x_3, y_3, z_1) (x_3, y_3, z_2) (x_3, y_3, z_3)
A simple CMAC
One of the time inversion CMAC in MS_CMAC
(Transition instances)
(Training instances)

FIG. 3. Topology of Three-Level MS_CMAC Neural Network

sponding computational time can be substantially reduced in a certain order.

More detailed notations about the MS_CMAC neural network are characterized in the following. In general, the number of levels in an MS_CMAC tree structure is set equal to the number of decision variables in an input. Each training instance is represented as $S'(s'_{1,t_1}, s'_{2,t_2}, \ldots, s'_{n,t_n})$ in an input, where $s'_{i,t_i} = \{s'_{i,t_i} | t_i = 1 \text{ to } p_i\}$, $1 \le i \le n$. The term p_i is a predefined constant. Then, the topology of the tree of MS_CMAC can be defined as follows:

- The hth node in ith level of tree is denoted as N_h^i.
- The number of nodes in ith level can be set as

$$\bar{p}_i = \begin{cases} 1 & \text{if } i = 1 \\ \prod_{j=1}^{i-1} p_j & \text{if } i > 1 \end{cases} \qquad (3)$$

- There are p_i children (nodes) for node N_h^i, where $1 \le i \le n - 1$ and $1 \le h \le \bar{p}_l$.
- Cumulatively, the tree contains $\bar{p}_n$ leaves (nodes) that are denoted as N_c^n, where $1 \le c \le \bar{p}_n$.

If a new instance $X(s_{1,u}, s_{2,u}, \ldots, s_{n,u})$ is given, then the corresponding output U_u can be obtained using the MS_CMAC neural network learning model according to the following steps:

- Step 1: Use instance X as a verification instance in N_1^1 node. The training instances of N_c^n are selected from the given training instance set. Subscript t_i ($i = 1$ to $n - 1$) of $S'(s'_{1,t_1}, s'_{2,t_2}, \ldots, s'_{n,t_n})$ is computed as follows:

$$t_n = \{1, 2, \ldots, p_n\}$$

$$c_n = c$$

for $i = 1$ to $n - 1$

$$t_{n-i} = \begin{cases} p_{n-i} & \text{if } c_{n-i+1} \bmod p_{n-i} = 0 \\ c_{n-i+1} \bmod p_{n-i} & \text{else} \end{cases} \qquad (4a)$$

$$C_{n-i} = (C_{n-i+1} - t_{n-i})/p_{n-i} + 1 \qquad (4b)$$

- Step 2: Generate transition instances in the tree. The input in the transition instance of N_b^l is defined as follows:

$$\tilde{S} = S^*T_l + XU_l \qquad (5a)$$

$$T_l = \begin{bmatrix} I_{l-1} & 0 \\ 0 & 0 \end{bmatrix}_{n \times n} \qquad (5b)$$

$$U_l = \begin{bmatrix} 0 & 0 \\ 0 & I_{n-l+1} \end{bmatrix}_{n \times n} \qquad (5c)$$

where $I_j = j \times j$ identify matrix; S^* = any training instance of node N_b^l; X = verification instance of MS_CMAC neural network; and $\tilde{S}$ = verification instance of node N_b^l.

- Step 3: Perform the learning and verification phases of all simple CMACs from leaves to root. Herein, a binary

Downloaded from ascelibrary.org by National Chiao Tung University on 05/01/14. Copyright ASCE. For personal use only; all rights reserved.

TABLE 1. Binary Valuable Table with g = 4

S_i (1)	α_1 (2)	α_2 (3)	α_3 (4)	α_4 (5)	α_5 (6)	α_6 (7)	α_7 (8)	α_8 (9)	α_9 (10)	α_{10} (11)	α_{11} (12)	α_{12} (13)
1	1	1	1	1	0	0	0	0	0	0	0	0
2	0	1	1	1	1	0	0	0	0	0	0	0
3	0	0	1	1	1	1	0	0	0	0	0	0
4	0	0	0	1	1	1	1	0	0	0	0	0
5	0	0	0	0	1	1	1	1	0	0	0	0
6	0	0	0	0	0	1	1	1	1	0	0	0
7	0	0	0	0	0	0	1	1	1	1	0	0
8	0	0	0	0	0	0	0	1	1	1	1	0
9	0	0	0	0	0	0	0	0	1	1	1	1

variable table proposed by Albus (1975a) is used in the mapping $S \rightarrow A$ of any simple CMAC in the MS_CMAC neural network learning model. For instances, Table 1 lists a binary variable table with g = 4. Hence, the function $S\_A(S^*)$ used in node N_b^l is shown as follows:

$$\alpha^* = S\_A(S^*) = \{\alpha_i^* | \alpha_i^* = s_l^* + i, \quad i = 1 \text{ to } g\} \tag{6}$$

where s_l^* denotes the lth element of S^*. The subsequent mapping of $A \rightarrow P$ for any simple CMAC in the MS_CMAC neural network learning model is set as follows:

$$\beta^* = A\_P(\alpha^*) = \{\beta_j^* | \beta_j^* = \alpha_j^*, \quad j = 1 \text{ to } g\} \tag{7}$$

Herein, the process of training in leaves is called the learning phase of an MS_CMAC neural network. The other processes are then called the verification phase of an MS_CMAC neural network. Notably, although the learning phase can run in off-line, the verification phase should be run in on-line.

In simple and time inversion CMAC neural networks, the size of A is always hundreds or thousands times larger than the size of P. However, the ratio of the size of A over P equals 1 in an MS_CMAC neural network learning model. Furthermore, the jth row vector of physically memory P_j computed by (1) for a simple or time inversion CMAC neural network is a step function of j and listed as follows:

$$P_j = \begin{cases} \mathbf{W}_1 & \text{if } 1 \le j < x_1 \\ \mathbf{W}_2 & \text{if } x_1 \le j < x_2 \\ \vdots & \quad \vdots \\ \mathbf{W}_a & \text{if } x_{a-1} \le j \le r \end{cases}$$

where $\mathbf{W}_i$ represents a scale vector, i = 1 to a. The basis of computation in (1) is then a linear approach. To strengthen the precision in computational ability in an MS_CMAC neural network, a novel quadratic approach is proposed herein. The jth row vector of physically memory P_j is then augmented as:

$$P_j = \begin{cases} W_1 + \left(\dfrac{j}{x_1} - \dfrac{1}{2}\right)(W_2 - W_1) \times \gamma & \text{if } 1 \le j < x_1 \\[2ex] W_i + \left(\dfrac{j - x_{i-1}}{x_i - x_{i-1}} - \dfrac{1}{2}\right)(W_{i+1} - W_{i-1}) \times \gamma & \text{if } x_{i-1} \le j < x_i \\[2ex] W_a + \left(\dfrac{j - x_{a-1}}{r - x_{a-1}} - \dfrac{1}{2}\right)(W_{a-1} - W_a) \times \gamma & \text{if } x_{a-1} \le j \le r \end{cases} \tag{8}$$

where γ = real number in the interval of [0, 1]. For instance, a 1D function $y(x)$ with five given points, denoted as solid circles in Fig. 4, is given. Fig. 4 also displays three mapped functions with three different values of γ based on the five given points by means of an MS_CMAC neural network.

COMPARISON OF TIME INVERSION CMAC AND MS_CMAC NEURAL NETWORK

The example of design of concrete beams was first solved by a back-propagation neural network with five parameters M_u, b/d, ρ, f_y, and f_c' in inputs and one data d in output (Vanluchene and Sun 1990). Where M_u is the ultimate bending moment; b and d are the width and depth of a rectangular section, respectively; ρ is the reinforcement ratio; f_y is the yielding stress of reinforcement; and f_c' is the compress strength of concrete. The relation between input and output can be expressed as follows:

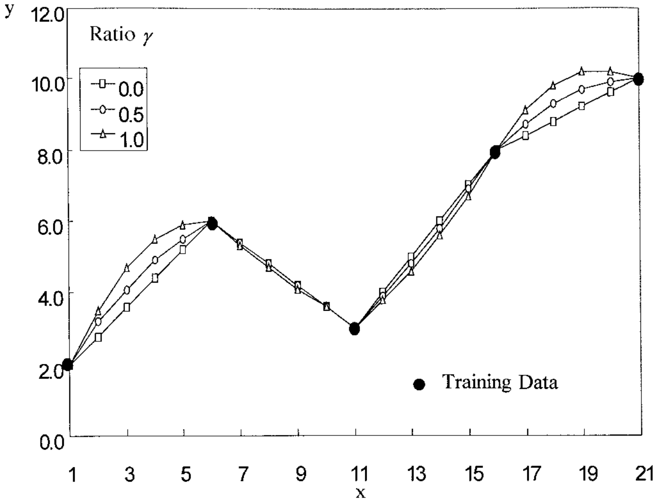

FIG. 4. Learning Results for 1D Example with Different γ Values

$$M_u = bd^2 f_c' \omega(1 - 0.59\omega); \quad \omega = \frac{\rho f_y}{f_c'} \tag{9a,b}$$

Gunaratnam and Gero (1994) solved the example by using the technique of dimensionless analysis to improve the performance of a back-propagation neural network. Adeli and Park (1995) used the same dimensionless data set as training instances of a CPN. For the sake of comparison of the learning performances of a time inversion CMAC and an MS_CMAC neural network learning models, a large example of 7,540 training instances, with the aforementioned five patterns in inputs and one data in outputs, was created. The instances were divided into two sets: 2,940 training instances and 4,620 verification instances. The verification instances are, then, equally divided into two groups, Test 1 and Test 2.

The input data corresponding to the five parameters of training instances are set as follows: M_u is changed from 226 to 474.6 kN·m with increments of 22.6 kN·m; b/d is changed from 0.5 to 0.8 with increments of 0.05; ρ is changed from 0.012 to 0.028 with increments of 0.004; f_y is changed from 2,800 to 4,200 kg/cm^2 with increments of 700 kg/cm^2; and f_c' is changed from 210 to 350 kg/cm^2 with increments of 70 kg/cm^2. For verification instances in Test 1, M_u is changed from 237.3 to 463.3 kN·m with increments of 22.6 kN·m, and the other four variables in input are the same as training instances. For verification instances in Test 2, M_u is changed from 237.3 to 463.3 kN·m with increments of 22.6 kN·m; b/d is changed from 0.525 to 0.775 with increments of 0.05; and the other variables in input are the same as training instances. Note that the inputs of instances are linearly transformed from real numbers into integer numbers of interval [1, 90], as the approach of the binary value table was employed. Consequently, the combinations of the five parameters in inputs for training and verification instances are set as follows:

s_{M_u}: {1, 9, 17, 25, 33, 41, 49, 57, 65, 73, 81, 89} (training)
{5, 13, 21, 29, 37, 45, 53, 61, 69, 76, 84} (Test 1 and Test 2)
$s_{b/d}$: {1, 14, 27, 40, 53, 66, 79} (training and Test 1)
{7, 20, 33, 46, 59, 72} (Test 2)

s_ρ: {1, 21, 41, 61, 81} (training, Test 1, and Test 2)
s_{f_y}: {1, 41, 81} (training, Test 1, and Test 2)
$s_{f'_c}$: {1, 41, 81} (training, Test 1, and Test 2)

First, the example is solved by a time inversion CMAC neural network with two connected simple CMACs. Working parameters were set as follows: The maximum learning iteration is set as 50, generalization size g is set as 50, and the size of physical memory P is set as 14,700. The instances in Test 1 were used as the transition instances in first connected simple CMAC. The instances in Test 2 were, then, used as verification instances in the second connected simple CMAC. It took about 274 s of computing time on a DEC 3000 workstation. The average percentage errors in prediction of the verification instances are 2.19 to Test 1 and 3.13 to Test 2.

Then, the example was solved using a five-level MS_CMAC neural network learning model. The active parameters for Levels 5 to 1 were set as M_u, b/d, ρ, f_y, and f'_c, respectively. The generalization size g was set as 50, 50, 50, 40, and 40 in Levels 5 to 1, respectively. The term γ is set as 0.5 in all five levels. The outputs of instances in Test 1 were calculated from nodes in Level 5, and the outputs of instances in Test 2 were calculated from nodes in Level 2. Accordingly, the average percentage errors in prediction verification instances are 0.64 for Test 1 and 1.28 for Test 2. It took only 24 s of computing time on a DEC 3000 workstation. In sum, the comparing results indicate not only a superprediction performance in verification instances but also a substantial decrease in computing time for the MS_CMAC neural network learning model as compared with a time inversion CMAC neural network.

APPLICATIONS

Two steel structural design examples were used to assess the performance of the MS_CMAC neural network learning model. The two examples are the estimation of moment gradient coefficient in steel beams and the determination of effective length of columns in unbraced frames, respectively. The computations for these two examples are complicated in structural engineering. Moreover, the two examples are generally solved by conventional numerical computing approaches.

Moment Gradient Coefficient C_b for Monosymmetric Steel Beams Subjected to End Moments

In the American Institute of Steel Construction's *Load Resistance Factor Design* (LRFD) specifications (*Manual* 1994), the maximum buckling moment M_{cr} in monosymmetric steel beam subjected to a gradient distributed moment can be approximately calculated by the following equations:

$$M_{cr} = C_b M_{um}; \quad C_b = 1.75 + 1.05\beta + 0.3\beta^2 \le 2.3 \qquad (10a,b)$$

where C_b = real number called moment gradient coefficient; M_{um} = buckling moment in monosymmetric steel beam subjected to a uniformed moment; and $\beta = M_1/M_2$ = ratio of two end moments, in which M_1 is the smaller end moment.

A more conservative method to calculate C_b is suggested by Kitipornchai (1985), and it is expressed as

$$C_b = \frac{\gamma_c}{\pi}\left[\sqrt{1 + 4\rho(1-\rho)\bar{k}^2 + \left(\frac{\beta_x}{h}k\right)^2} + \frac{\beta_x}{h}k\right] \qquad (11)$$

where $\beta_x/h = 0.9(2\rho - 1)[1 - (I_y/I_x)^2]$; $\bar{k} = \sqrt{\pi^2 E I_y h^2/4GJL^2}$ = beam parameter; ρ = coefficient monosymmetric from inverted T-shape ($\rho = 0$) to T-shape ($\rho = 1.0$) shown in Fig. 5; and γ_c = monodimensional elastic critical buckling moment. The γ_c can be determined by the Rayleigh-Ritz method using the nine-term Fourier sine series to express deflection and rotation shape (Kitipornchai 1985). It is a finding that the lowest positive root problem of a power series of order 18 in term of γ_c with β, k, and ρ is a coefficient. In this work, the term γ_c was solved by the Rayleigh-Ritz method using MALAB 4.0 on a personal computer.

A three-level MS_CMAC neural network is used to predict the moment gradient coefficient C_b. The parameters in input are β, k, and ρ. The value of I_y/I_x is set as 0.075 in all instances. The following combinations of the three input parameters are considered as training instances:

β: {−1.0, −0.8, −0.6, −0.4, −0.2, 0, 0.2, 0.4, 0.6, 0.8, 1.0}
k: {0.5, 1.0, 1.5, 2.0}
ρ: {0, 0.07, 0.2, 0.4, 0.6, 0.8, 0.93, 1.0}

These combinations are then transformed to integer numbers as follows:

s_β: {1, 11, 21, 31, 41, 51, 61, 71, 81, 91, 101}
$s_{\bar{k}}$: {1, 21, 41, 61}
s_ρ: {1, 8, 21, 41, 61, 81, 94, 101}

The following combinations of the three input parameters are considered as verification instances:

β: {−1.0, −0.8, −0.6, −0.4, −0.2, 0, 0.2, 0.4, 0.6, 0.8, 1.0}

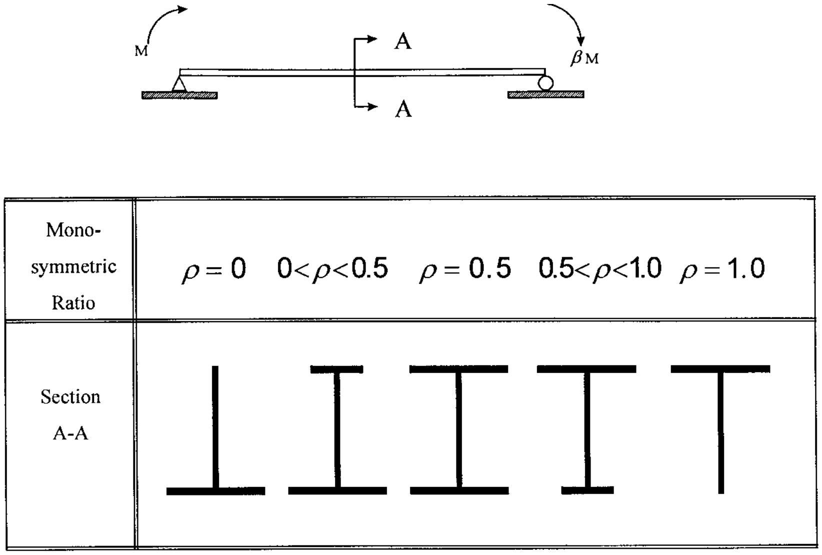

FIG. 5. MS_CMAC Neural Network for Moment Gradient Coefficient Example

Downloaded from ascelibrary.org by National Chiao Tung University on 05/01/14. Copyright ASCE. For personal use only; all rights reserved.

Downloaded from ascelibrary.org by National Chiao Tung University on 05/01/14. Copyright ASCE. For personal use only; all rights reserved.

$\bar{k}$: {0.75, 1.25, 1.75}
ρ: {0.1, 0.3, 0.5, 0.7, 0.9}

These combinations are then transformed as integer numbers as follows:

s_β: {1, 11, 21, 31, 41, 51, 61, 71, 81, 91, 101}
$s_{\bar{k}}$: {11, 31, 51}
s_ρ: {11, 31, 51, 71, 91}

Thus, there are a total of 517 instances created and divided into two sets: 352 training instances and 165 verification instances. The active parameters for Levels 3 to 1 in the three-level MS_CMAC neural network were ρ, $\bar{k}$, and β, respectively. Working parameters were set as follows: γ was set as 1.0, 0.5, and 0 in Levels 3 to 1, respectively, and the generalization size g was set as 100, 60, and 20, respectively.

The computing results are shown in Figs. 6(a–c), respectively. The average percentage error for 165 verification instances is 0.56. The two largest percentage errors in verification instances are 4.94 (as $\bar{k} = 0.75$, $\rho = 0.9$, and $\beta = 1.0$) and 4.92 (as $\bar{k} = 1.75$, $\rho = 0.1$, and $\beta = 0.4$). Notably, the results are within the acceptable limits in an engineering design computation. Note that the MS_CMAC neural network learning model has excellent prediction performance, with the average error of 0.22%, as $0.1 < \rho < 0.9$. Notwithstanding, the computing results for high monosymmetric I-beams ($\rho \geq 0.9$ or $\rho \leq 0.1$) with $\beta > 0$ and $\bar{k} = 0.75$ or $\bar{k} = 1.75$, are poor as comparing with other verification instances. However, the issue can be improved as the number of training instances increased.

Effective Length Factor *K* of Columns in Unbraced Frames

The inelastic buckling strength of an element subjected to axial loading can be calculated by the following equation:

$$F_{cr} = \frac{\pi^2 E_t}{(KL/r)^2} \tag{12}$$

where E_t = tangent modulus of elasticity at critical strength; r is the radius of gyration; and KL is the effective length of columns. In LRFD specifications (*Manual* 1994), the value of K can be evaluated by an alignment chart. Recently, Duan and Chen (1989), as well as Kishi et al. (1997) found that the effective length of columns determined using the alignment chart would be too conservative or not safe with different boundary conditions.

Herein, a subassemblage model of an unbraced frame based on the alignment chart approach is established. The model is composed of Column $c2$ with two ends A and B, two restrained columns ($c1$ and $c3$), and four restrained beams. Column $c1$ and two of the beams are rigidly connected to the end A. Meanwhile, Column $c3$ and the two other beams are rigidly connected to the end B. Herein, the far ends of $c1$ and $c3$ are rigidly or fixed connected. The general governing equations for the K factor of Column $c2$ can be derived as (Kishi et al. 1997)

$$\det \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = 0 \tag{13a}$$

and the entries a_{ij}, i = 1 to 3 and j = 1 to 3 are defined as

$$a_{11} = s_{ii} + \frac{6}{G'_A} - G_{Ac1}\frac{s_{ij}^2}{s_{ii}}; \quad a_{12} = G_{Ac2}s_{ij} \tag{13b,c}$$

$$a_{13} = -(s_{ii} + s_{ij})\left(1 - G_{Ac1}\frac{s_{ij}}{s_{ii}}\right) \tag{13d}$$

$$a_{21} = G_{Bc2}s_{ij}; \quad a_{22} = s_{ii} + \frac{6}{G'_B} - G_{Bc3}\frac{s_{ij}^2}{s_{ii}} \tag{13e,f}$$

$$a_{23} = -(s_{ii} + s_{ij})\left(1 - G_{Bc3}\frac{s_{ij}}{s_{ii}}\right) \tag{13g}$$

$$a_{31} = a_{32} = s_{ii} + s_{ij}; \quad a_{33} = (\pi/K)^2 - 2(s_{ii} + s_{ij}) \tag{13h,i}$$

where

$$G'_A = \frac{\sum_A (EI/L)_c}{\sum_A \alpha_{uf}(EI/L)_b}; \quad G'_B = \frac{\sum_B (EI/L)_c}{\sum_B \alpha_{uf}(EI/L)_b} \tag{14a,b}$$

$$G_{Aci} = \frac{(EI/L)_{ci}}{\sum_A (EI/L)_c}; \quad G_{Bci} = \frac{(EI/L)_{ci}}{\sum_B (EI/L)_c} \tag{15a,b}$$

$$s_{ii} = \frac{\frac{\pi}{K}\sin\frac{\pi}{K} - \left(\frac{\pi}{K}\right)^2\cos\frac{\pi}{K}}{2 - 2\cos\frac{\pi}{K} - \frac{\pi}{K}\sin\frac{\pi}{K}} \tag{16a}$$

$$s_{ij} = \frac{\left(\frac{\pi}{K}\right)^2 - \frac{\pi}{K}\sin\frac{\pi}{K}}{2 - 2\cos\frac{\pi}{K} - \frac{\pi}{K}\sin\frac{\pi}{K}} \tag{16b}$$

In the previous equations, subscripts A and B in G' indicate the columns and beams connected at the Ath and Bth nodal points, respectively. Subscripts b and c denote beam and column, respectively. Subscript ci (i = 1 or 2 or 3) indicates column number.

In this example, two boundary conditions are examined. They are both far ends of $c1$ and $c3$ fixed (K_1) and rigid connected (K_2). Table 2 lists the values of G_{Aci} and G_{Bci}. There are two input parameters (G'_A, G'_B) and two output data (K_1, K_2) in each instance. The following combinations of two input parameters are considered as training instances:

G'_A, G'_B: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

The parameters are then linearly transferred to integers in the interval of [1, 91] as follows:

$s_{G'_A}$, $s_{G'_B}$: {1, 11, 21, 31, 41, 51, 61, 71, 81, 91}

The following combinations of two input parameters are considered as verification instances:

G'_A, G'_B: {1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5, 9.5}

The corresponding combinations of the two parameters in transformed integer numbers are then set as follows:

$s_{G'_A}$, $s_{G'_B}$: {6, 16, 26, 36, 46, 56, 66, 76, 86}

Thus there are a total of 181 instances created using MALAB 4.0 on a personal computer and divided as 100 training instances and 81 verification instances. Note that the K-factor is set as 10 when the value of computed K exceeds 10 in this work. A two-level MS_CMAC neural network was used to solve this example. The active parameters are G'_A in leaf and G'_B in root. The generalization size g was set as 50 in leaf and root, respectively, and the term γ was set as 0.6 in Levels 2 and 1. The learning results are shown in Figs. 7(a and b).

The average percentage errors in K_1 and K_2 are 0.67 and

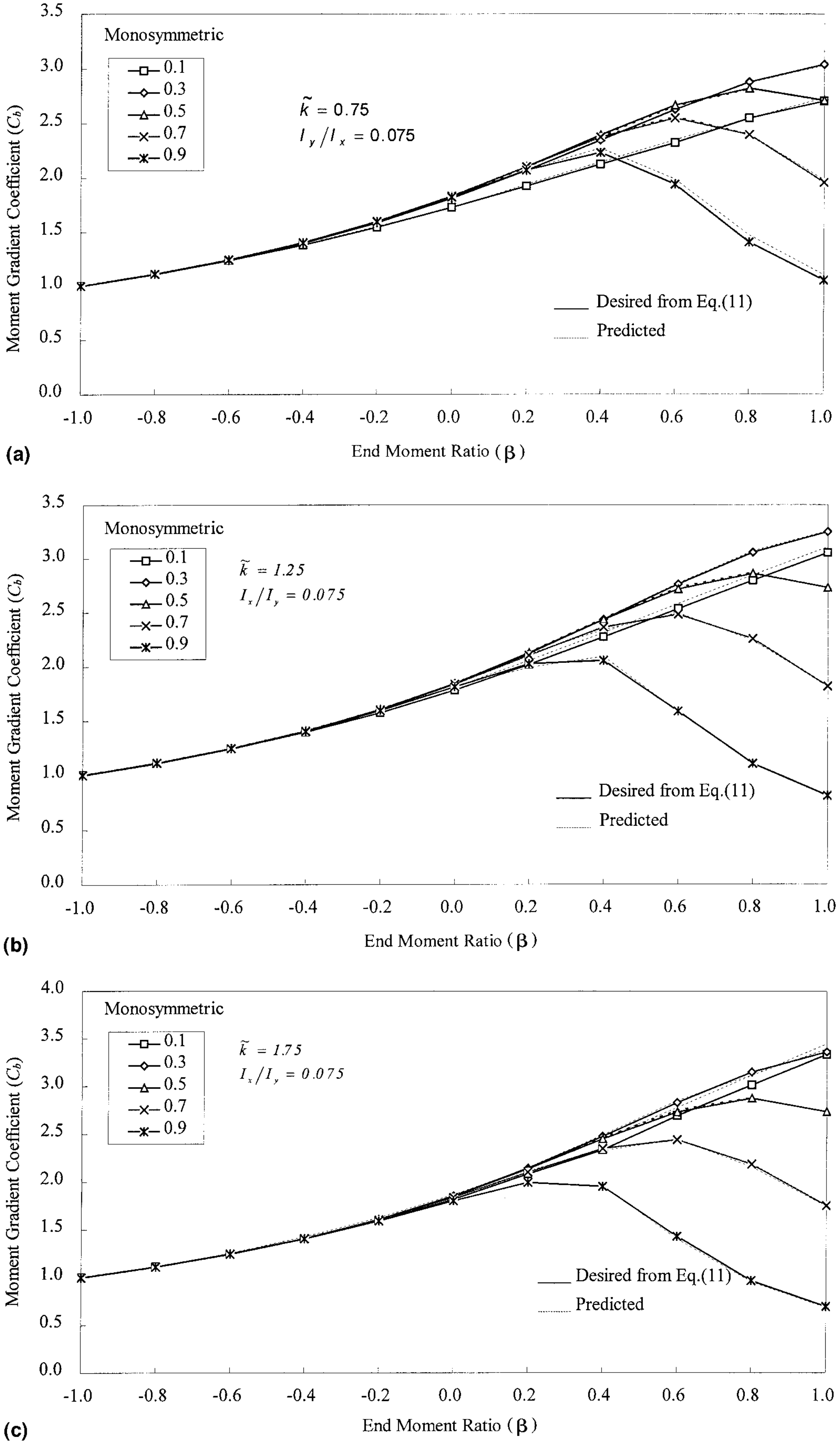

FIG. 6. Prediction of Moment Gradient Coefficient by MS_CMAC Neural Network for: (a) $\bar{k}$ = 0.75; (b) $\bar{k}$ = 1.25; (c) $\bar{k}$ = 1.75

Downloaded from ascelibrary.org by National Chiao Tung University on 05/01/14. Copyright ASCE. For personal use only; all rights reserved.

Downloaded from ascelibrary.org by National Chiao Tung University on 05/01/14. Copyright ASCE. For personal use only; all rights reserved.

0.37, respectively. Kishi et al. (1997) reported that the errors of K-factor determined using the LRFD specification (*Manual* 1994) alignment chart were -2.6 to -56.6% for fixed-fixed boundary, -1.2 to -17.6% for fixed-rigid boundary, and 8.2 to 14.4% for pinned-pinned boundary. Herein, the learning results confirm that the MS_CMAC neural network learning model has excellent prediction performance for K-factors of columns in unbraced frames. In addition, the prediction results are within the acceptable limits in an engineering design computation.

TABLE 2. Values of G_{Aci} and G_{Bci}

Far End Condition of Column $c1$			Far End Condition of Column $c3$		
Pinned (1)	Fixed (2)	Rigid (3)	Pinned (4)	Fixed (5)	Rigid (6)
G_{Ac1} G_{Ac2}	$G_{Ac1} = 0$ G_{Ac2}	$G_{Ac1} = 0$ $G_{Ac2} = 1$	G_{Bc2} G_{Bc3}	G_{Bc2} $G_{Bc3} = 0$	$G_{Bc2} = 1$ $G_{Bc3} = 0$

Comparison of L-BFGS and MS_CMAC Neural Networks

Herein, the two examples were then solved using L-BFGS supervised neural networks. Table 3 summarizes the comparisons of L-BFGS and MS_CMAC neural networks. In the example of moment gradient coefficient, a network with three input nodes, one hidden layer with seven nodes, and one output node, identified as L-BFGS(3-7-1), was used. The average percentage error is 2.83 for verification instances. It took ~1,275-s computing time for the L-BFGS neural network. However, the average system error in MS_CMAC is 0.56%

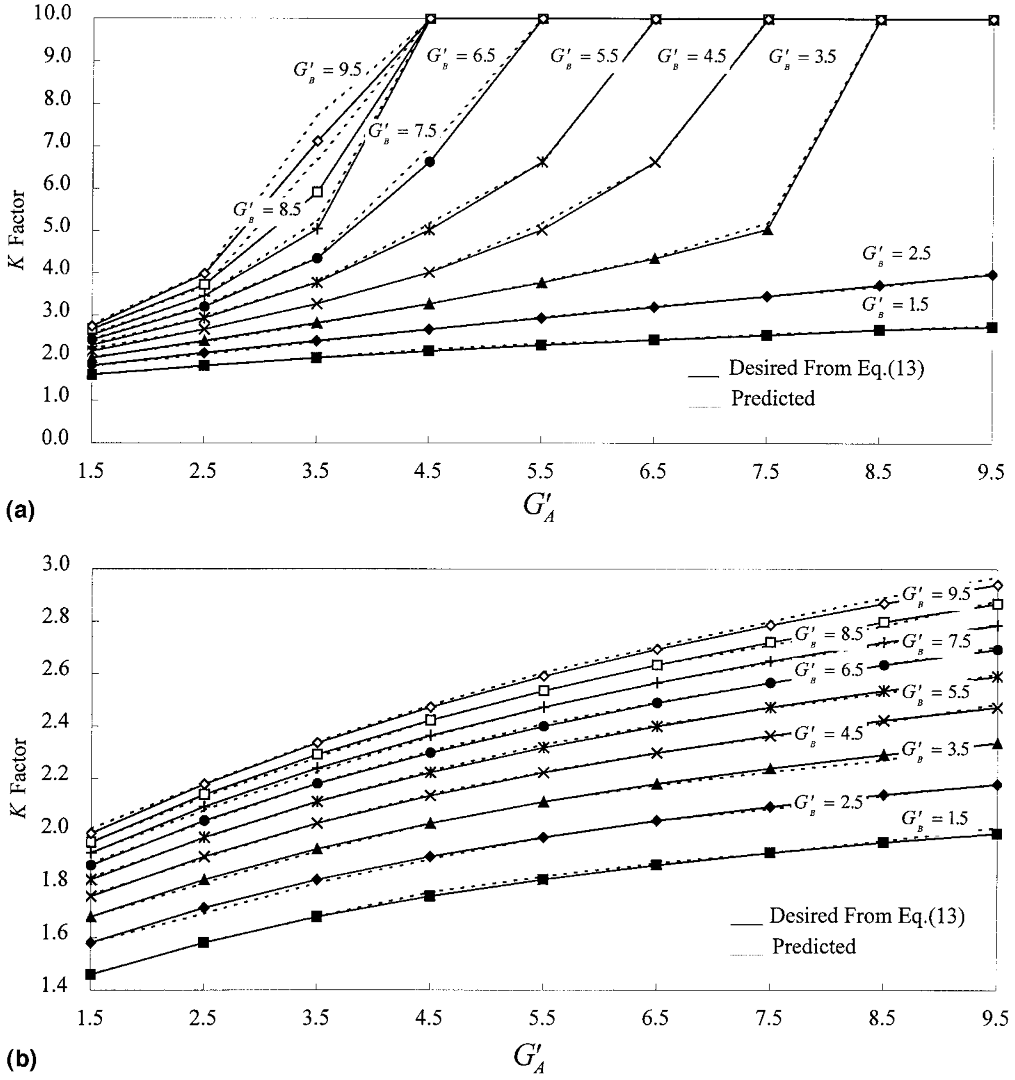

FIG. 7. Prediction of K-Factor of Columns by MS_CMAC Neural Network: (a) Fixed; (b) Rigid

TABLE 3. Comparison of BP-Based and MS_CMAC Neural Networks

Examples (1)	Number of training instances (2)	BP-Based (L-BFGS) Topology (3)	Average error (%) (4)	Computing time (s) (5)	MS_CMAC Number of branch (6)	Average error (%) (7)	Computing time (s) (8)
C_b [Eq. (11)]	352	L-BFGS(3-7-1)	2.83	1,275	R^a-11-5	0.56	21
K-factor (rigid)	100	L-BFGS(2-5-1)	0.52	48	R^a-10	0.37	3
K-factor (fixed)	100	L-BFGS(2-5-3-1)	0.89	102	—	0.67	—

Note: BP = back-propagation.
[a] R = root.

Downloaded from ascelibrary.org by National Chiao Tung University on 05/01/14. Copyright ASCE. For personal use only; all rights reserved.

for verification instances. It took only 21-s computing time for the MS_CMAC neural network. Notably, the problem is taken from the literature (Adeli and Park 1995). The training and verification instances, however, are entirely distinct to their work. Hence, no computational comparison is made between the MS_CMAC and CPN neural networks.

In the example of estimate *K*-factor, a network with two input nodes, one hidden layer with five nodes, and one output node (2-5-1) was used in the cases of rigid-rigid boundary. In addition, a network with two input nodes, two hidden layers with five and three nodes, and one output node (2-5-3-1) was used in the cases of fixed-fixed boundary. The average percentage errors for verification instances with two different boundaries are 0.52 and 0.89, respectively. It took about 48 and 102 s, respectively, for L-BFGS neural networks in two boundary conditions. However, the average percentage errors were 0.37 and 0.67 for verification instances, respectively, and the computing time in MS_CMAC neural network is only 3 s.

CONCLUSIONS

This work presents a novel supervised neural network learning mode, MS_CMAC learning model, by connecting a large number of time inversion CMAC as a topology of tree structure. The MS_CMAC neural network learning model proposed herein applied to engineering design problems. Two structural engineering problems are addressed to assess the learning performance of the MS_CMAC neural network learning model. Based on the results in this work, we can conclude the following:

1. The MS_CMAC neural network learning model can be used to solve structural engineering problems that are generally solved by numerical computing approaches within a reasonable central processing unit time. These numerical approaches, however, are not readily available for structural engineers to use. Moreover, the computing results in the MS_CMAC neural network learning model are more precise than that estimated through approximate charts in LRFD specifications.
2. The prediction performance of the MS_CMAC neural network learning model is superior to that of L-BFGS supervised neural networks for verification instances. Moreover, more additional learning cycles are often required to train the neural network when a new instance is added into the training sets for L-BFGS supervised learning models. However, the issue of additional learning cycles for training is circumvented in the MS_CMAC neural network learning model. Only the associated physical memory corresponding to the new training instance needs to be updated.
3. For a time inversion CMAC neural network, the sizes of *A* and *P* must be large enough for performing good prediction in verification phase. The size of *A* is always hundreds or thousands times larger than the size of *P*. However, in the MS_CMAC neural network learning model, the size of *A* is significantly reduced and the ratio of the size of *A* over *P* is equal to 1. As a result, the MS_CMAC neural network learning model can converge quickly to an accepted solution in a few iterations.
4. A time inversion CMAC neural network uses a linear interpolation approximation to calculate the adjustments for updating the physical memory. Hence, for a nonlinear problem, the performance of verification is poor in a time inversion CMAC neural network. A novel approach for calculating the adjustment in updating physical memory is presented in this work. Instead of the linear interpolation approximation, a quadratic interpolation approximation (trapezium schema) is utilized in the MS_CMAC neural network learning model. Significant improvement in the learning performance for verification instances is achieved in the MS_CMAC neural network learning model.

ACKNOWLEDGMENTS

The writers would like to thank the National Science Council of the Republic of China for financially supporting this research under Contract No. NSC 87-2211-E-009-029.

APPENDIX. REFERENCES

Adeli, H., and Hung, S. L. (1993). ‘‘A concurrent adaptive conjugate gradient learning algorithm on MIMD shared memory machines.’’ *J. Supercomp. Appl.*, 7(2), 155–166.

Adeli, H., and Hung, S. L. (1994). ‘‘An adaptive conjugate gradient learning algorithm for effective training of multilayer neural networks.’’ *Appl. Mathematics and Computation*, 62(1), 81–102.

Adeli, H., and Park, H. S. (1995). ‘‘Counterpropagation neural networks in structural engineering.’’ *J. Struct. Engrg.,* ASCE, 121(8), 1205–1212.

Albus, J. S. (1975a). ‘‘A new approach to manipulator control: The cerebellar model articulation controller (CMAC).’’ *J. Dyn. Sys., Measurement, and Control*, 97(3), 220–227.

Albus, J. S. (1975b). ‘‘Data storage in the cerebellar model articulation controller.’’ *J. Dyn. Sys., Measurement, and Control*, 97(3), 228–233.

Duan, L., and Chen, W. F. (1989). ‘‘Effective length factor for columns in unbraced frames.’’ *J. Struct. Engrg.*, ASCE, 115(1), 149–165.

Elkordy, M. F., Chang, K. C., and Lee, G. C. (1994). ‘‘A structural damage neural network monitoring system.’’ *Microcomp. in Civ. Engrg.*, 9(2), 83–96.

Ghaboussi, J., Garrett, J. H., and Wu, X. (1991). ‘‘Knowledge-based modeling of material behavior and neural networks.’’ *J. Engrg. Mech.*, ASCE, 117(10), 132–153.

Goh, A. T. C. (1995). ‘‘Back-propagation neural networks for modeling complex systems.’’ *Artificial Intelligence in Engrg.*, 9(1), 143–151.

Gunaratnam, D. J., and Gero, J. S. (1994). ‘‘Effect of representation on the performance of neural networks in structural engineering applications.’’ *Microcomp. in Civ. Engrg.*, 9(2), 97–108.

Hajela, P., and Berke, L. (1991). ‘‘Neurobiological computational modes in structural analysis and design.’’ *Comp. and Struct.*, 41(4), 657–667.

Hecht-Nielsen, R. (1987). ‘‘Counterpropagation networks.’’ *Appl. Optics*, 26(3), 4979–4984.

Hung, S. L., and Adeli, H. (1993). ‘‘Parallel back propagation learning algorithm on Cray YMP8/864 supercomputers.’’ *Neurocomputing*, 5(6), 287–302.

Hung, S. L., and Adeli, H. (1994). ‘‘A parallel genetic neural network learning algorithm for MIMD shared memory machines.’’ *IEEE Trans. On Neural Networks*, 5(6), 900–909.

Hung, S. L., and Jan, J. C. (1997). ‘‘Machine learning in engineering design—an unsupervised fuzzy neural network case-based learning model.’’ *Proc., Intelligent Information Sys.*, IEEE Computer Society, Los Alamitos, Calif., 156–160.

Hung, S. L., and Lin, Y. L. (1994). ‘‘Application of an L-BFGS neural network learning algorithm in engineering analysis and design.’’ *Proc., 2nd Nat. Conf. on Struct. Engrg.*, Chinese Soc. of Struct. Engrg., Nantou, Taiwan, R.O.C., 221–230.

Kasperkiewicz, J., Racz, H., and Dubrawski, A. (1995). ‘‘HPC strength prediction using artificial neural network.’’ *J. Computing in Civ. Engrg.*, ASCE, 9(4), 279–284.

Kishi, N., Chen, W. F., and Goto, Y. (1997). ‘‘Effective length factor of columns in semirigid and unbraced frames.’’ *J. Struct. Engrg.*, ASCE, 123(3), 313–320.

Kitipornchai, S., Wang, C. M., and Trahair, N. S. (1986). ‘‘Buckling of monosymmetric I-beams under moment gradient.’’ *J. Struct. Engrg.*, ASCE, 112(4), 781–799.

Lin, C.-S., and Chiang, C. T. (1997). ‘‘Learning convergence of CMAC technique.’’ *IEEE Trans. Neural Networks*, 8(6), 1281–1292.

Manual of steel construction—load and resistance factor design. (1994). American Institute of Steel Construction, Chicago.

Mukherjee, A., and Deshpande, J. M. (1995). ''Modeling initial design process using neural network.'' *J. Computing in Civ. Engrg.*, ASCE, 9(3), 194–200.

Nocedal, J. (1990). ''The performance of several algorithms for largescale unconstrained optimization.'' *Large-scale numerical optimizaion*, T. F. Coleman, and Y. Li, eds., Soc. for Industrial and Applied Math., Philadelphia, 138–151.

Rumelhart, D., Hinton, G., and Williams, R. (1986). ''Learning representations by back-propagation errors.'' *Parallel distributed processing*, D. Rumelhart et al., eds., Vol. 1, MIT Press, Cambridge, Mass., 318–362.

Vanluchene, R. D., and Sun, R. (1990). ''Neural networks in structural engineering.'' *Microcomp. in Civ. Engrg.*, 5(3), 207–215.

Downloaded from ascelibrary.org by National Chiao Tung University on 05/01/14. Copyright ASCE. For personal use only; all rights reserved.