Neural Processing Letters (2006) 23:201–228
DOI 10.1007/s11063-006-6277-0

Minimal Structure of Self-Organizing HCMAC Neural Network Classifier

CHIH-MING CHEN[1,⋆], YUNG-FENG LU[2] and CHIN-MING HONG[3]
[1]*Graduate Institute of Learning Technology, National Hualien University of Education, Hualien, Taiwan. e-mail: cmchen@mail.nhlue.edu.tw*
[2]*Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan.*
[3]*Institute of Applied Electronic Technology, National Taiwan Normal University, Taipei, Taiwan.*

Abstract. The authors previously proposed a self-organizing Hierarchical Cerebellar Model Articulation Controller (HCMAC) neural network containing a hierarchical GCMAC neural network and a self-organizing input space module to solve high-dimensional pattern classification problems. This novel neural network exhibits fast learning, a low memory requirement, automatic memory parameter determination and highly accurate high-dimensional pattern classification. However, the original architecture needs to be hierarchically expanded using a full binary tree topology to solve pattern classification problems according to the dimension of the input vectors. This approach creates many redundant GCMAC nodes when the dimension of the input vectors in the pattern classification problem does not exactly match that in the self-organizing HCMAC neural network. These redundant GCMAC nodes waste memory units and degrade the learning performance of a self-organizing HCMAC neural network. Therefore, this study presents a minimal structure of self-organizing HCMAC (MHCMAC) neural network with the same dimension of input vectors as the pattern classification problem. Additionally, this study compares the learning performance of this novel learning structure with those of the BP neural network, support vector machine (SVM), and original self-organizing HCMAC neural network in terms of ten benchmark pattern classification data sets from the UCI machine learning repository. In particular, the experimental results reveal that the self-organizing MHCMAC neural network handles high-dimensional pattern classification problems better than the BP, SVM or the original self-organizing HCMAC neural network. Moreover, the proposed self-organizing MHCMAC neural network significantly reduces the memory requirement of the original self-organizing HCMAC neural network, and has a high training speed and higher pattern classification accuracy than the original self-organizing HCMAC neural network in most testing benchmark data sets. The experimental results also show that the MHCMAC neural network learns continuous function well and is suitable for Web page classification.

Key words. Cerebellar Model Articulation Controller (CMAC), minimal structure of self-organzing HCMAC (MHCMAC) neural network, self-organizing hierarchical CMAC (HCMAC) neural network

⋆Corresponding author.

1. Introduction

The Cerebellar Model Articulation Controller (CMAC) neural network was developed by Albus [1, 2] in 1975. The CMAC is trained by local learning approach. Each training data point is assigned to finite weights shared by its neighboring data points. Therefore, weights trained by one training data point only interfere with neighboring data points. Albus' CMAC uses a constant basis function to perform locally weighted approximations of functions. The constant basis function updates weights by distributing errors evenly among the assigned weights. CMAC's output is constant within each quantized state, and does not preserve the derivative information. Currently, CMAC has been successfully applied in many real-world applications such as robotic control, signal processing, web page mining, and pattern recognition [3–6] due to its fast learning, good generalization capability, and ease of implementation by hardware.

Previous CMAC studies have tried to develop enhanced CMAC learning algorithms [7–12], improve the CMAC learning structure [13–20], select learning parameters for improving learning performance [21], implement CMAC model by hardware [22], and apply the CMAC model in various fields [23–25]. In addition, CMAC's convergence properties have also been proved by formal mathematical procedures [26, 27]. However, CMAC model proposed by Albus has two major shortcomings: one is the problem of enormous memory size requirement while modeling multi-variables functions or high-dimensional pattern classification problems, and another is to perform the uniform input space quantization for learning spaces [28, 29]. In modeling multi-variables functions, CMAC can produce very large numbers of memory units for learning as the number of input vector dimensions increases. Additionally, the number of training data points required should be increased. Therefore, the fast memory requirement limits used fields of CMAC model in real-world applications. Meanwhile, to perform the uniform input space quantization for learning spaces neglects the problem of various pattern distributions of training data sets, thus leading to allocating unused memory units as well as reducing learning performance. This is because the memory space utilization for the learned data will not be performed in the most accurate and precise manner while using the uniform input space quantization approach for learning. Hash-coding can often substantially reduce CMAC's memory requirement and work well for some problems [30, 31], but may lead to divergence and the speed and performance of convergence may deteriorate [30].

To reduce the memory requirement, several researchers have proposed using modular CMAC to model multi-variables problems. Albus [2] proposed a time-inversion technique to refine computations in a CMAC. Time inversion CMAC uses a serially-connected low order CMAC to solve high order problems, generating temporary data points from the upper low order CMAC to be used as training data in later low order CMAC. Moreover, Lin and Li presented a unique learning structure composed of small two-dimensional CMACs to solve high-dimensional problems [14, 15, 32]. However, their proposed structure does not work well if its sub-module architecture is not properly considered. Actually, Lin and Li's method suffers from the permutation explosion problem of input variables for some pattern classification problems with large feature dimensions (such as Lung's data set [33], 56 feature dimensions). Recently, Hung and Jan [34] proposed an MS_CMAC to model multi-variables smooth functions. The MS_CMAC connects numerous one-dimensional CMACs as a tree structure trained by time inversion technique. Hung and Jan also developed a quadratic spline scheme to smooth one-dimensional CMAC outputs by transforming the step-state weights into continuous-smooth weights. Like Lin and Li's study, MS_CMAC cannot handle high-dimensional pattern classification problems because its tree structure grows quickly with the input vector dimensions.

Moreover, to perform input space quantization appropriately, some researches to address the input space quantization issue can be found in the literature. Moody's study [35] proposed to use the layers of multi-resolution CMACs to achieve adaptively memory storage allocation. Kim and Lin [36] used iterative CMAC output feedback to adjust the input quantization function to distribute the target system's output signals uniformly. However, iterative output feedback requires the time derivative of the target function to update the quantization function [29], making the training process extremely complicated. Meanwhile, this approach cannot work for pattern classification problems. Additionally, Berger [37] employed the adaptive binary method to divide the input space into appropriate ranges for input space quantization to minimize the training error variance. Moreover, several researchers have proposed clustering to obtain adaptive resolution [28, 29]. In summary, these proposed approaches always involve trade-offs between storage savings and computational complexity of the proposed model.

To solve the above-mentioned two problems with CMAC, our previous study proposed a self-organizing hierarchical CMAC (HCMAC) neural network containing an HCMAC neural network that consists of two-dimensional GCMACs with non-constant differentiable Gaussian basis function [38] and a self-organizing input space scheme [39]. The proposed HCMAC neural network can overcome the original CMAC model's large memory requirement by partitioning a high-dimensional problem into several manageable two-dimensional sub-problems. Furthermore, the self-organizing input space scheme is based on Shannon's entropy measure [40] and the golden section search method [41] to determine the appropriate input space quantization based on the input training data distribution. Compared with the other proposed input space quantization approaches [28, 29], [35–37], the self-organizing input space scheme presented in our previous study [39] does not need the derivative information and is better suited to handle pattern classification problems, but it cannot be applied to perform the input space quantization for the problems without category labels, such as the function approximation problem. After the input space is quantized into discrete regions, the memory allocation of the HCMAC neural network classifier can be constructed automatically. Our previous study also presented a gradient-descent learning rule to train the proposed self-organizing HCMAC classifier [39]. We demonstrated that the hierarchical HCMAC can reduce the memory requirement of the conventional CMAC. Additionally, the proposed self-organizing HCMAC classifier can achieve good high-dimensional pattern classification.

However, the original self-organizing HCMAC neural network's learning structure must be hierarchically expanded by a full binary tree topology according to the dimension of the input vectors to solve pattern classification problems, generating redundant GCMAC nodes when the pattern classification problem's input vector dimensions do not exactly match those of the self-organizing HCMAC neural network. These redundant GCMAC nodes waste various memory units and influence the learning performance of a self-organizing HCMAC neural network. Hence, this study presents a minimal structure of a self-organizing HCMAC (MHCMAC) neural network with the same input vector dimensionality as the pattern classification problem. Additionally, this study infers the proposed self-organizing MHCMAC neural network's complete structural information, automatically-assigned input vector approach and learning rules. The experimental results show that the proposed self-organizing MHCMAC neural network not only significantly reduces the original self-organizing HCMAC neural network's memory requirement, but also maintains a high training speed and classifies patterns more accurately than the original self-organizing HCMAC neural network for most testing benchmark data sets.

2. CMAC Technique

To derive the novel learning structure of the MHCMAC neural network, this section first introduces the CMAC techniques based on different basis functions.

2.1. CMAC WITH CONSTANT BASIS FUNCTION

The Cerebellar Model Arithmetic Computer (CMAC) [1, 2] is a table look-up neurocomputing technique. The constant basis function usually is used as association memory selection vector in the conventional CMAC to model the hypercube structure for learning. Furthermore, the behavior of storing weight information in CMAC is similar to that of the cerebellum of humans, which distributively store information on different cell layers. Before applying this learning model, the input data of each state variable must be quantized into discrete regions according to given learning space. Herein, the number of discrete regions is termed as a resolution. Several quantized discrete regions can be accumulated as a block. Each input data can be mapped to several actual memory units via an association memory selection vector. These mapped actual memory units are called hypercubes. That is, each input data is distributively mapped on several different layered hypercubes. Hence, the actual output can be obtained by summing all mapped hypercube values. CMAC uses the difference of the desired output and actual output to adjust the mapped actual memory contents until the total error converges to a tolerable range.

Assume the actual output for a quantized state is distributively stored in N_e locations of actual memory, and N_h represents the entire actual memory size $(N_h > N_e)$. The actual output for an input state is obtained as the sum of stored contents for hypercubes covering this state. That is, the actual output of a specific input state s can be mathematically expressed as follows:

$$y(s) = a^T(s)w = \sum_{j=1}^{N_h} a_j(s)w_j \tag{1}$$

where $w = [w_1, w_2, \ldots, w_{Nh}]^T$ is the vector of actual memory contents, and $a^T(s) = \left[a_{1(s)}, a_{2(s)}, \ldots, a_{Nh}(s)\right]$ is an association memory selection vector which contains N_e constant 1s', i.e. the conventional CMAC uses the constant basis function to model hypercubes structure.

The conventional CMAC uses a supervised learning approach to adjust the weight values during each learning cycle. Its learning rule can be described as equation (2).

$$w_t = w_{t-1} + \frac{\alpha}{N_e} a(s)(\hat{y}(s) - a^T(s)w_{t-1}) \tag{2}$$

where w_t is the vector of actual memory contents at time t, w_{t-1} is the vector of actual memory contents at previous time $t-1$, α is a learning rate, $\hat{y}(s)$ is the desired output value and $\hat{y}(s) - a^T(s)w_{t-1}$ is the error for the input training state s.

2.2. CMAC WITH GAUSSIAN BASIS FUNCTION

Since the conventional CMAC uses a constant basis function as association memory selection vector to model the hypercube structure, its output is always constant within each quantized state and the derivative information of input and output variables cannot be preserved [38]. The non-differentiable property leads to some limitations when using the conventional CMAC in real-world applications, such as action dependent critical learning [21]. Therefore, Chiang and Lin [38] proposed a differentiable CMAC to solve this problem. Their study includes a non-constant differentiable Gaussian basis function to model the hypercube structure. The mathematical formulation of the one-dimensional Gaussian basis function ϕ can be described as equation (3).

$$\phi(s) = e^{-\left(\frac{s-m}{\sigma}\right)^2} \tag{3}$$

where m is a hypercube center, σ is a hypercube radius, and s is a specific input state.

Consider an N_v dimensional problem. A Gaussian basis function with N_v dimensions is included so that equation (1) is revised to be equation (4) as follows:

$$y(s) = \sum_{j=1}^{Nh} \left[a_j(s) \cdot w_j . \left(\coprod_{i=1}^{Nv} e^{-\left(\frac{s_i - m_{ji}}{\sigma_{ji}}\right)^2} \right) \right] \tag{4}$$

where $a_j(s)$ is the jth element of association memory selection vector for a specific input state s, w_j is the jth memory allocation of actual memory, s_i is the input value of the ith dimension for a specific input state s, m_{ji} is the corresponding hypercube center, and σ_{ji} is the corresponding hypercube radius.

The CMAC with a Gaussian basis function as a non-constant differentiable basis function is termed as GCMAC herein, and both the self-organizing HCMAC neural network presented in our previous work [39] and the proposed minimal structure of self-organizing HCMAC neural network (MHCMAC) are based on the GCMAC technique.

3. The Problems of Self-Organizing HCMAC Neural Network

To derive the self-organizing HCMAC neural network's minimal structure, this study first briefly reviews the self-organizing HCMAC neural network techniques, including the HCMAC architecture and self-organizing input space approach, then proposes several flaws in this learning structure. A hierarchical CMAC (HCMAC) neural network consists of two-dimensional differentiable GCMACs as presented in the previous work [39]. An HCMAC's learning structure can be arbitrarily expanded using the full binary tree topology for the pattern classification problems. Figure 1 illustrates the smallest HCMAC neural network topology, in which each GCMAC includes two features as inputs, and the second layer GCMAC's output values serve as the first layer GCMAC's input values. Thus, a pattern classification problem with three or four input dimensions is solved using the same self-organizing HCMAC neural network learning structure based on a full binary tree topology. This structure generates one redundant GCMAC node ($GCMAC_3$) when solving a pattern classification problem with three dimensions, thus wasting many memory units. The memory wastage may become very serious when solving a high-dimensional pattern classification problem. For example, a pattern classification problem with 513 input dimensions creates a learning structure with 1024 input dimensions, thus generating 511 redundant input dimensions in the leaf GCMAC nodes. In our previous study [39], the redundant input feature s_4 of $GCMAC_3$ shown in Figure 1 is assigned a zero feature value when solving a pattern classification problem with three input features. Although assigning a zero feature value to the redundant inputs in a HCMAC learning structure can overcome non-exact input dimensions, it may degrade the learning performance of the HCMAC neural network. Therefore, this study presents an MHCMAC neural network with the same number of input dimensions as the pattern classification problem, to avoid wasting memory units and enhance the learning performance in the original HCMAC neural network.

4. Minimal Structure of Self-organizing HCMAC (MHCMAC) Neural Network

This section derives the MHCMAC neural network's learning structure and is organized as follows. Section 4.1 presents the learning structure of the MHCMAC

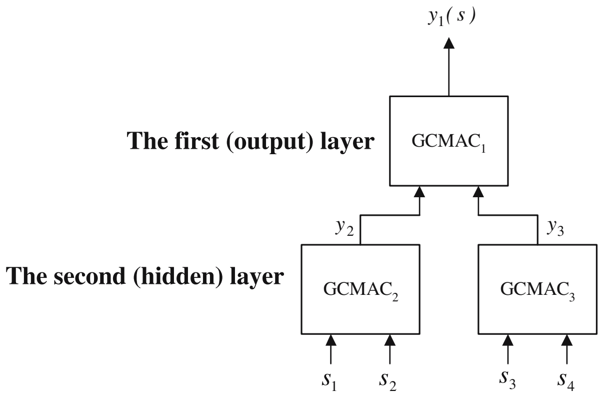

Figure 1. A smallest topology structure of the HCMAC neural network.

neural network. Section 4.2 infers some useful structural information for the MHCMAC neural network. Section 4.3 proposes an automatically assigned feature inputs approach and gives brief descriptions of self-organizing input space for the proposed MHCMAC neural network. Section 4.4 compares the self-organizing HCMAC with the self-organizing MHCMAC neural network. Finally, Section 4.5 details the learning rules of the MHCMAC neural network.

4.1. LEARNING STRUCTURE OF THE MHCMAC NEURAL NETWORK

An MHCMAC neural network is constructed using two-dimensional differentiable GCMACs, but its learning structure is arbitrarily expandable using exact binary tree topology. The exact binary tree topology indicates that an MHCMAC learning structure is constructed using two-dimensional GCMACs, but has the same input vector dimensions as the pattern classification problem. Figure 2 shows several MHCMAC neural networks with various numbers of input vector dimensions for solving the pattern classification problems, where s_i denotes the assigned input feature sequence. The automatically assigned approach of input features for the MHCMAC neural network is explained in Section 4.3 in detail. Clearly, the constructed MHCMAC structure is the same as the original HCMAC structure when the pattern classification problem input vectors have exactly 2^n dimensions, where $n \in Z$, but is different from the constructed HCMAC structure when the input vectors not have exactly 2^n dimensions. In MHCMAC's structure, each two-dimensional GCMAC node from the root node to the last leaf node is assigned a unique serial node number. Each GCMAC's node number can be applied to derive the relationships between the parent and the child GCMAC nodes in an MHCMAC structure. The proposed MHCMAC structure creates the root GCMAC node, labeled as $GCMAC_1$, in the first layer, and the second and third GCMAC nodes in the second layer and marked $GCMAC_2$ and $GCMAC_3$, and so on. That is, the root GCMAC node ($GCMAC_1$)

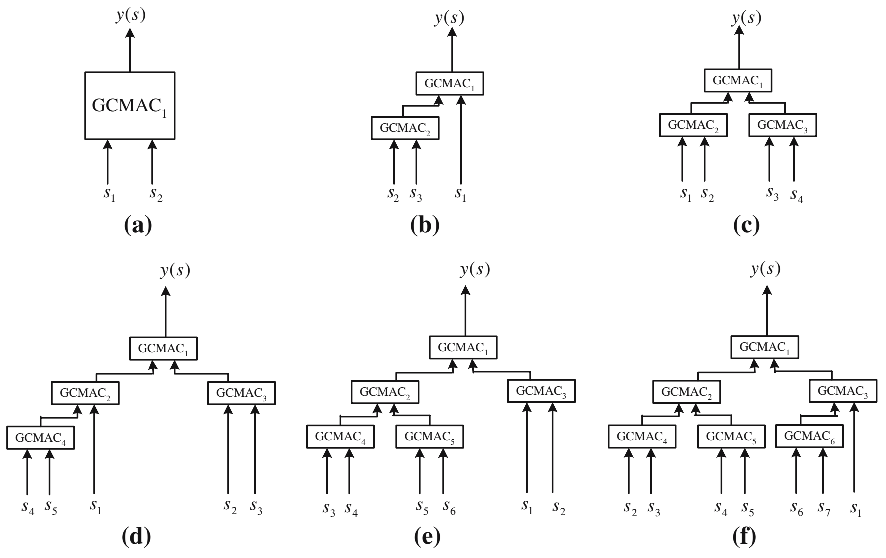

Figure 2. Minimal structure of the HCMAC neural networks with various number of input vector dimensions (where s_i denotes the assigned input feature sequence). (a) Two dimensions, (b) three dimensions, (c) four dimensions, (d) five dimensions, (e) six dimensions and (f) seven dimensions.

is in the output layer, and the other nodes of the GCMACs are in the hidden layers. Compared with the original CMAC, the MHCMAC neural network is more difficult to be implemented by hardware because it has more complicated learning structure than the original CMAC.

4.2. STRUCTURAL INFORMATION OF THE MHCMAC NEURAL NETWORK

The MHCMAC neural network topologies shown in Figure 2 infer a general rule for determining an MHCMAC's structure automatically. First, the number of GCMAC nodes required in a single MHCMAC's structure can be determined from the number of input vector dimensions in the pattern classification problem according to the following formula:

$$N_{\text{MHCMAC\_Node}} = NUM(\text{input dimension}) - 1 \tag{5}$$

where $N_{\text{MHCMAC\_Node}}$ denotes the number of GCMAC nodes required in a single MHCMAC's structure, and NUM (input dimension) stands for the number of the input vector dimensions in the pattern classification problem.

Hence, in solving a pattern classification problem with many output categories, the total number of GCMAC nodes required for the entire MHCMAC structure can be computed by the following formula:

$$\begin{aligned} TN_{\text{MHCMAC\_Node}} = {} & NUM(\text{output category}) \\ & \times [NUM(\text{input dimension}) - 1] \end{aligned} \tag{6}$$

where $TN_{\text{MHCMAC\_Node}}$ represents the total number of GCMAC nodes required to solve a pattern classification problem with many output categories, and NUM (output category) stands for the number of output categories of the pattern classification problem.

Moreover, the parent node of the ith GCMAC node in an MHCMAC structure can be computed as follows:

$$PN_{\text{GCMAC}_i} = \left\lfloor \frac{i}{2} \right\rfloor, \quad \text{for} \quad i = 2, 3, 4, \ldots, N_{\text{MHCMAC\_Node}} \tag{7}$$

where PN_{GCMAC_i} represents the parent node of the ith GCMAC node, and $N_{\text{MHCMAC\_Node}}$ denotes the number of GCMAC nodes required in the structure of a single MHCMAC.

Moreover, the child node of the ith GCMAC can be computed as follows:

$$CN_{\text{GCMAC}_i} = i \bmod 2, \quad \text{for} \quad i = 2, \ldots, N_{\text{MHCMAC\_Node}} \tag{8}$$

where $CN_{\text{GCMAC}_i} = 0$ means that the ith GCMAC node is the left child node of its parent GCMAC node, and $CN_{\text{GCMAC}_i} = 1$ means that the ith GCMAC node is the right child node of its parent GCMAC node.

The ith GCMAC node, constructed in the hidden layer in an MHCMAC's structure can be computed using the following formula;

$$LN_{\text{GCMAC}_i} = \lfloor \log_2(i) \rfloor, \quad \text{for} \quad i = 2, 3, \ldots, N_{\text{MHCMAC\_Node}} \tag{9}$$

where LN_{GCMAC_i} represents the located hidden layer of the layer index of the ith GCMAC node in an MHCMAC topology.

Finally, in the proposed MHCMAC structure, if the input vectors in the pattern classification problem have an odd number of dimensions, then the constructed MHCMAC structure has one GCMAC node whose left and right inputs handle the output of its child GCMAC node and input feature data, respectively. Conversely, if the pattern classification problem has an even number of dimensions, then the left and right inputs of each constructed GCMAC node at the bottom layer handle the input feature data. With the odd-numbered input dimensions, the following formula can be applied to determine which GCMAC node's left and right inputs will be used to respectively handle the output of its child GCMAC node and input feature data;

$$GCMAC_i = GCMAC_{\frac{N_{\text{MHCMAC\_Node}}}{2}}, \quad \text{for } N_{\text{MHCMAC\_Node}} \in \text{even number} \tag{10}$$

where $N_{\text{MHCMAC\_Node}}$ denotes the number of GCMAC nodes required in a single MHCMACs' structure.

In summary, Equations (5–8) and (10) can be applied to construct the MHCMAC structure required for the pattern classification problem. Equations (7) and (9) are helpful for deriving learning rules for the MHCMAC's hidden layers because they commonly determine the error back-propagation frequency in each GCMAC node in a learning cycle. Equations (9) computes the error back-propagation frequency for each GCMAC node in the hidden layers.

4.3. AUTOMATICALLY ASSIGNED APPROACH OF INPUT FEATURES AND SELF-ORGANIZING INPUT SPACE FOR THE MHCMAC NEURAL NETWORK

The assigned input feature sequence, displayed in Figure 2 is helpful in designing a computer program to assign input features to the MHCMAC's learning structure. Restated, a general rule to assign input features automatically to the MHCMAC's learning structure can be induced from the designated input feature sequence. To induce this rule for the MHCMAC, the notation $GCMAC_{l,m}$ determines whether the ith input feature s_i should be designated to the GCMAC node's left or right input. Herein, the values of l and m are respectively determined by the following two mathematical formulae:

$$l = \lfloor (i + N_{\mathrm{MHCMAC\_Node}})/2 \rfloor \tag{11}$$

$$m = (i + N_{\mathrm{MHCMAC\_Node}}) \bmod 2 \tag{12}$$

where $N_{\mathrm{MHCMAC\_Node}}$ denotes the number of GCMAC nodes required in the structure of a single MHCMAC.

When automatically assigning input features, l indicates that the ith input feature s_i is assigned to the left input of the lth GCMAC node in the structure of a single MHCMAC if $m = 0$. Conversely, the ith input feature s_i is assigned to the right input of the lth GCMAC node in the structure of a single MHCMAC if $m = 1$. This approach is a general rule for MHCMAC neural networks because it can automatically assign input features to the MHCMAC structure whether the input features have odd or even dimensions.

Besides, owing to its use of a uniform approach to quantize input space for constructing memory structure, the conventional CMAC approach cannot accurately reflect the actual distribution of training data [28, 29], thus leading to extra cost of memory requirement or poor learning performance. Therefore, a self-organizing input space approach based on Shannon's entropy measure and the golden section search method proposed in our previous work [39] is applied to adaptively determine quantization of each input feature dimension based on the distribution of training data for the proposed MHCMAC neural network. Because each input space of a two-dimensional GCMAC is quantized into some intervals, Shannon's entropy of each interval is respectively computed to obtain a measure of patterns' distribution information. Moreover, since adaptive partition points are unknown, we need an effective strategy to find appropriate partition points for input feature space. Thus, the golden section search method is adopted to obtain adaptive partition points. After the quantization information is obtained, the association memory selection vectors and actual memory structure for the MHCMAC neural network are automatically determined. In sum, the learning spaces of the input variables are quantized first by the self-organizing input space approach, then the learning structure of the MHCMAC will be automatically constructed through a predetermined block parameter. Currently, the self-organizing input space for the proposed MHCMAC is only performed for the leaf GCMAC nodes (i.e. the input layer) based on the pattern distribution of training data. The other GCMAC nodes (i.e. the hidden layers) in the MHCMAC learning structure perform the scheme of uniform input space quantization for input learning spaces. This is because the input features of hidden layer GCMAC nodes in the MHCMAC learning structure are always variable during the learning process until the entire learning process is terminated.

4.4. COMPARISON OF HCMAC NEURAL NETWORK WITH THE MHCMAC NEURAL NETWORK

Table 1 compares the HCMAC neural network with the MHCMAC neural network in terms of memory requirement, topology structure and input feature assignment approach. Table I shows that the memory requirement of the original HCMAC neural network grows with the power 2 of the ceiling logarithm of the input dimensions, but the memory requirement of the MHCMAC neural network grows only linearly with the input feature dimensions. Moreover, the learning structure of the self-organizing HCMAC neural network is expanded based on a full binary tree topology, but the MHCMAC neural network is expanded based on an exact binary tree topology. The input features can be arbitrarily assigned to the leaf GCMAC node inputs for both the original self-organizing HCMAC and MHCMAC neural networks. However, to implement the MHCMAC neural network more easily, this study presents an automatically input feature assignment approach in Section 4.3 to assign each input feature to its corresponding GCMAC node in an MHCMAC learning structure. Finally, the zero value is assigned to the inputs of redundant leaf GCMAC nodes in the original HCMAC neural network, but the MHCMAC neural network can improve this problem because it can construct exact GCMAC nodes to solve the pattern classification problems.

4.5. LEARNING RULE FOR THE MHCMAC NEURAL NETWORK

For simplicity, this study applied the five-input MHCMAC topology structure as shown in Figure 3 to infer the MHCMAC neural network's learning rule. As shown in Figure 3, this structure contains five-input features labeled s_i, where s_i $(i = 1, 2, \ldots, 5)$ represents the ith input feature in the MHCMAC neural network; y_j $(j = 1, 2, 3, 4)$ is the jth output of GCMAC; y_j $(j = 2, \ldots, 4)$ are the hidden layer outputs, and y_1 is the MHCMAC neural network's output for a specific input state s. Moreover, to simplify the inference procedure of learning rules, the five-input

Table 1. Comparison of the HCMAC neural network with the MHCMAC neural network.

	Models	
Items	HCMAC [39]	MHCMAC
Total number of GCMAC nodes required for the entire learning structure	NUM(output category) × $\left(2^{\lceil \log_2 NUM(\text{input dimension})\rceil} - 1\right)$	NUM(output category) × $[NUM(\text{input dimension}) - 1]$
Topology structure	Full binary tree structure	Exact binary tree structure
Input feature assignment approach	Any order to the leaf GCMAC node inputs	Any order to the leaf GCMAC node inputs or using the proposed automatically input feature assignment approach
Handling approach for the inputs of redundant leaf GCMAC nodes	Assign zero values to the inputs of redundant leaf GCMAC nodes	No redundant leaf GCMAC node

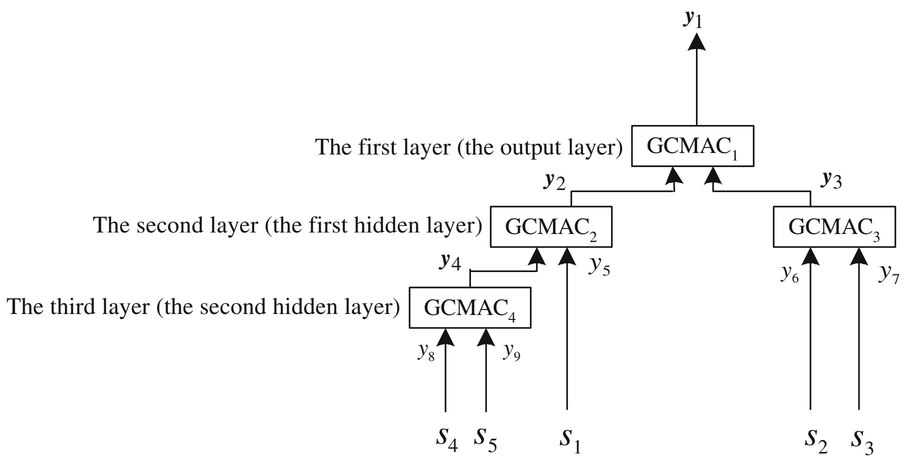
y_1
The first layer (the output layer)
GCMAC$_1$
y_2
y_3
The second layer (the first hidden layer)
GCMAC$_2$
GCMAC$_3$
y_4
y_5
y_6
y_7
The third layer (the second hidden layer)
GCMAC$_4$
y_8
y_9
s_4 s_5 s_1 s_2 s_3

Figure 3. MHCMAC topology structure with five-input dimensions.

features labeled s_i also give another corresponding label y_j $(j = 5, \ldots, 9)$. The learning rule is inferred using differentiable GCMAC and gradient descent. Since the first layer's GCMAC is an output node, the original GCMAC learning rule [38] is used to update the weight parameters. An error cost function is also defined to derive the weights updating formulae of the hidden layer GCMACs, as follows:

$$E = \frac{1}{2}(\hat{y}(s) - y(s))^2 \tag{13}$$

where $\hat{y}(s)$ is the desired output value of the MHCMAC neural network for input state s, and $y(s)$ is the actual output of the MHCMAC neural network for input state s.

The proposed method needs three parameters, i.e. weight w, radius σ and center m, to achieve each GCMAC's learning process. When the actual and desired output values are different, the three GCMAC$_1$ parameters in the output layer are updated first. The error is then back-propagated from the first layer to the second and third layers to update the GCMAC$_2$, GCMAC$_3$, and GCMAC$_4$ parameters.

The MHCMAC neural network architecture (Figure 3) uses the learning rule inference procedures as follows.

4.5.1. *Update the GCMAC$_1$ of the first (output) layer*

$$\begin{aligned}\Delta w_j &= -\frac{\alpha}{Ne}\frac{\partial E}{\partial w_j}\\ &= \frac{\alpha}{Ne}(\hat{y}_1(s) - y_1(s)) \cdot a_j(s) \cdot \prod_{i=2}^{3} e^{-\left(\frac{y_i - m_{jk}}{\sigma_{jk}}\right)^2}\end{aligned} \tag{14}$$

where Δw_j is the update to the weight value of the jth actual memory in the GCMAC$_1$, w_j is the weight value of the jth actual memory in the GCMAC$_1$, m_{jk} is the center of the kth dimension of the Gaussian basis function in the jth mapped hypercube; k is determined by the i, if $i=2$ then $k=1$; if $i=3$ then $k=2$ in the learning rule, σ_{jk} is the radius of the kth dimension of Gaussian basis function in the jth mapped hypercube, α is the learning rate, $a_j(s)$ is the jth element of association memory selection vector for a specific input state s, and Ne is the number of mapped hypercubes for input state s.

$$\begin{aligned}\Delta \sigma_{ji} &= -\frac{\alpha}{Ne}\frac{\partial E}{\partial \sigma_{ji}}\\ &= \frac{\alpha}{Ne}(\hat{y}_1(s) - y_1(s)) \cdot a_j(s) \cdot w_j \cdot \left[\prod_{i=2}^{3} \mathrm{e}^{-\left(\frac{y_i - m_{jk}}{\sigma_{jk}}\right)^2}\right] \cdot \frac{2(y_i - m_{jk})^2}{\sigma_{jk}^3}\end{aligned} \tag{15}$$

In Equation (15) above, $\Delta\sigma_{ji}$ is the update to the ith dimension radius value for the jth mapped hypercube of input state s in the GCMAC$_1$; σ_{ji} is the radius of the ith dimension for the jth mapped hypercube of input state s in the GCMAC$_1$; k is determined by the i, if $i=2$ then $k=1$, if $i=3$ then $k=2$ in the learning rule, and α is the learning rate.

$$\begin{aligned}\Delta m_{ji} &= -\frac{\alpha}{Ne}\frac{\partial E}{\partial m_{ji}}\\ &= \frac{\alpha}{Ne}(\hat{y}_1(s) - y_1(s)) \cdot a_j(s) \cdot w_j \cdot \left[\prod_{i=2}^{3} \mathrm{e}^{-\left(\frac{y_i - m_{jk}}{\sigma_{jk}}\right)^2}\right] \cdot \frac{2(y_i - m_{jk})}{\sigma_{jk}^2}\end{aligned} \tag{16}$$

In Equation (16), Δm_{ji} is the update to the ith dimension center value for the jth mapped hypercube of input state s in the GCMAC$_1$; m_{ji} is the center of the

ith dimension for the jth mapped hypercube of input state s in the GCMAC$_1$; k is determined by the i, if $i=2$ then $k=1$; if $i=3$ then $k=2$ in the learning rule, and α is the learning rate.

4.5.2. *Update the GCMAC$_2$ and GCMAC$_3$ of the second layer*

To derive the GCMAC$_2$ and GCMAC$_3$ learning rules in the first hidden layer, the derivative information of $\partial y_1(s)/\partial y_2$ and $\partial y_1(s)/\partial y_3$ are first computed as Equations (17) and (18).

$$\frac{\partial y_1(s)}{\partial y_2}=\sum_{j=1}^{Nh} a_j(s)\cdot w_j\cdot\left[\prod_{i=2}^{3} \mathrm{e}^{-\left(\frac{y_i-m_{jk}}{\sigma_{jk}}\right)^2}\right]\cdot\frac{-2\left(y_2-m_{j1}\right)}{\sigma_{j1}^2} \tag{17}$$

$$\frac{\partial y_1(s)}{\partial y_3}=\sum_{j=1}^{Nh} a_j(s)\cdot w_j\cdot\left[\prod_{i=2}^{3} \mathrm{e}^{-\left(\frac{y_i-m_{jk}}{\sigma_{jk}}\right)^2}\right]\cdot\frac{-2\left(y_3-m_{j2}\right)}{\sigma_{j2}^2} \tag{18}$$

In the above equations, w_j is the weight of the jth actual memory in the GCMAC$_1$, σ_{jk} is the radius of the kth dimension for the jth mapped hypercube of input state s in the GCMAC$_1$; k is determined by the i, if $i=2$ then $k=1$; if $i=3$ then $k=2$ in the two learning rules, m_{jk} is the center of the kth dimension for the jth mapped hypercube of input state s in the GCMAC$_1$, and N_h is the entire actual memory size in the GCMAC$_1$.

Next, the learning rule of the GCMAC$_2$ in the first hidden layer (Figure 3) can be derived by

$$\begin{aligned}\Delta w_j &=-\frac{\alpha}{Ne}\frac{\partial E}{\partial w_j}\\ &=\frac{\alpha}{Ne}(\hat{y}_1(s)-y_1(s))\left(\frac{\partial y_1(s)}{\partial y_2}\right)\cdot a_j(s)\cdot\left[\prod_{i=4}^{5} \mathrm{e}^{-\left(\frac{y_i-m_{jk}}{\sigma_{jk}}\right)^2}\right]\end{aligned} \tag{19}$$

$$\begin{aligned}\Delta\sigma_{ji} &=-\frac{\alpha}{Ne}\frac{\partial E}{\partial\sigma_{ji}}\\ &=\frac{\alpha}{Ne}(\hat{y}_1(s)-y_1(s))\cdot\frac{\partial y_1(s)}{\partial y_2}a_j(s)\cdot w_j\cdot\left[\prod_{i=4}^{5} e^{-\left(\frac{y_i-m_{jk}}{\sigma_{jk}}\right)^2}\right]\cdot\frac{2\left(y_i-m_{jk}\right)^2}{\sigma_{jk}^3}\end{aligned} \tag{20}$$

$$\begin{aligned}\Delta m_{ji} &=-\frac{\alpha}{Ne}\frac{\partial E}{\partial m_{ji}}\\ &=\frac{\alpha}{Ne}(\hat{y}_1(s)-y_1(s))\cdot\frac{\partial y_1(s)}{\partial y_2}a_j(s)\cdot w_j\cdot\left[\prod_{i=4}^{5} e^{-\left(\frac{y_i-m_{jk}}{\sigma_{jk}}\right)^2}\right]\cdot\frac{2(y_i-m_{jk})}{\sigma_{jk}^2}\end{aligned} \tag{21}$$

In Equations (19), (20) and (21), the symbol definitions can refer to Equations (14), (15) and (16), and k is determined by the i, if $i=4$ then $k=1$; if $i=5$ then $k=2$ in these learning rules. Similarly, the learning rule of the GCMAC$_3$ in the first hidden layer (Figure 3) can be derived as GCMAC$_2$.

4.5.3. *Update the GCMAC$_4$ of the third layer*

To derive the learning rule of the GCMAC$_4$ in the second hidden layer, the derivative information of $\partial y_2(s)/\partial y_4$ is computed as Equation (22):

$$\frac{\partial y_2(s)}{\partial y_4}=\sum_{j=1}^{Nh} a_j(s)\cdot w_j\cdot\left[\prod_{i=4}^{5} e^{-\left(\frac{y_i-m_{jk}}{\sigma_{jk}}\right)^2}\right]\cdot\frac{-2\left(y_4-m_{j1}\right)}{\sigma_{j1}^2} \tag{22}$$

where w_j is the weight value of the jth actual memory in the GCMAC$_2$, σ_{jk} is the radius of the kth dimension for the jth mapped hypercube of input state s in the GCMAC$_2$; k is determined by the i, if $i=4$ then $k=1$; if $i=5$ then $k=2$ in the learning rule, and m_{jk} is the center of the kth dimension for the jth mapped hypercube of input state s in the GCMAC$_2$.

Similarly, the GCMAC$_4$ learning rule in the second hidden layer (Figure 3) can be derived as GCMAC$_2$ using the derivative information as formulae (17) and (22). Restated, to update GCMAC$_4$'s weights, hypercube centers and radiuses, the gradient of the error between the desired and the actual outputs must be back-propagated first from the output layer to the first hidden layer using Equation (17), then from the first to the second hidden layers using Equation (22). Next, the GCMAC$_4$ learning rule in the second hidden layer (Figure 3) can be derived as

$$\begin{aligned}\Delta w_j &= -\frac{\alpha}{Ne}\frac{\partial E}{\partial w_j}\\ &= \frac{\alpha}{Ne}\left(\hat{y}_1(s)-y_1(s)\right)\left(\frac{\partial y_1(s)}{\partial y_2}\right)\left(\frac{\partial y_2}{\partial y_4}\right)\cdot a_j(s)\cdot\left[\prod_{i=8}^{9} e^{-\left(\frac{y_i-m_{jk}}{\sigma_{jk}}\right)^2}\right]\end{aligned} \tag{23}$$

$$\begin{aligned}\Delta\sigma_{ji} &= -\frac{\alpha}{Ne}\frac{\partial E}{\partial\sigma_{ji}}\\ &= \frac{\alpha}{Ne}(\hat{y}_1(s)-y_1(s))\cdot\left(\frac{\partial y_1(s)}{\partial y_2}\right)\left(\frac{\partial y_2}{\partial y_4}\right).a_j(s)\cdot w_j\cdot\left[\prod_{i=8}^{9} e^{-\left(\frac{y_i-m_{jk}}{\sigma_{jk}}\right)^2}\right]\\ &\quad \frac{2(y_i-m_{jk})^2}{\sigma_{jk}^3}.\end{aligned} \tag{24}$$

$$\begin{aligned}\Delta m_{ji} &= -\frac{\alpha}{Ne}\frac{\partial E}{\partial m_{ji}}\\ &= \frac{\alpha}{Ne}(\hat{y}_1(s) - y_1(s)) \cdot \left(\frac{\partial y_1(s)}{\partial y_2}\right)\left(\frac{\partial y_2}{\partial y_4}\right) .a_j(s) \cdot w_j \cdot \left[\prod_{i=8}^{9} e^{-\left(\frac{y_i - m_{jk}}{\sigma_{jk}}\right)^2}\right]\\ &\quad \frac{2\left(y_i - m_{jk}\right)}{\sigma_{jk}^2}.\end{aligned} \tag{25}$$

In Equations (23), (24) and (25), the symbol definitions can also refer to Equations (14), (15) and (16), and k is determined by the i, if $i = 8$ then $k = 1$; if $i = 9$ then $k = 2$ in these learning rules. Equations (19), (20) and (21) show that the error is back-propagated from the GCMAC$_1$ in the output layer to the GCMACs of the first hidden layer by the corresponding derivative information from $\partial y_1(s)/\partial y_2$, $\partial y_1(s)/\partial y_3$. Moreover, Equations (23), (24) and (25) show that the error is back-propagated from the GCMAC$_1$ in the output layer to the GCMACs of the second hidden layer by the corresponding derivative information of $\partial y_1(s)/\partial y_2$, $\partial y_2(s)/\partial y_4$.

5. Experiments

First, a continuous function with three variables was used as a target function to show the function approximation ability of the proposed MHCMAC neural network. Meanwhile, to demonstrate the classification performance of the proposed self-organizing MHCMAC neural network classifier, data sets on ten benchmark pattern classification problems and Syskill & Webert Web page ratings from the UCI machine learning repository [33] were tested. The experimental results are described in the following sections.

5.1. EXPERIMENT ON FUNCTION APPROXIMATION

In this experiment, a three-dimensional function $F(x_1, x_2, x_3) = (1 + \sin(x_1\pi)\sin(2x_2\pi)\sin(3x_3\pi))/2$ for $0 \leqslant x_1, x_2, x_3 \leqslant 1$, is used as a target function to evaluate the ability of function approximation of the MHCMAC neural network. The three-dimensional topology structure of the MHCMAC shown as Figure 2(b) was employed to solve this problem. To show the three-dimensional plots for the desired target function output and the learned output of the MHCMAC neural network, the function $F(0.5, x_2, x_3)$ was sampled with a sampling interval of 41 for each variable. In other words, each sampling slice of variables x_2 and x_3 is $\frac{1}{41}$ in the learning space under the variable x_1 was fixed as 0.5. Therefore, the total number of training patterns was 1681 in this experiment. Besides, the learning rate α of the MHCMAC is set to be 0.025, the resolution parameter is set to be 61 for the input dimensions of the used GCMAC nodes, the block parameter which is equal to the parameter Ne is set to be 3, and the stop criterion of the mean square error is set to be 0.00012 in the experiment of function approximation. Figure 4 shows the learning mean square error convergence curve. Figures 5 and 6 show the 3D plots of the desired and learned outputs of $F(0.5, x_2, x_3)$, respectively. These experimental results demonstrate that the proposed MHCMAC neural network approximates the function well.

5.2. EXPERIMENT ON TEN BENCHMARK PATTERN CLASSIFICATION PROBLEMS FROM UCI MACHINE LEARNING REPOSITORY

Next, in the experiment of ten benchmark pattern classification problems from UCI machine learning repository [33], ten independent runs were performed to yield an average performance; half the original data patterns selected randomly were used as training data and the remaining patterns were used as testing data.

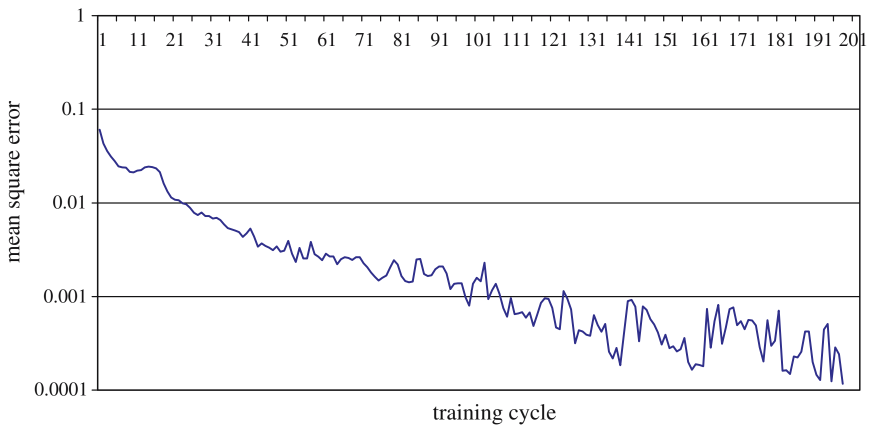

Figure 4. Convergence curve of mean square error.

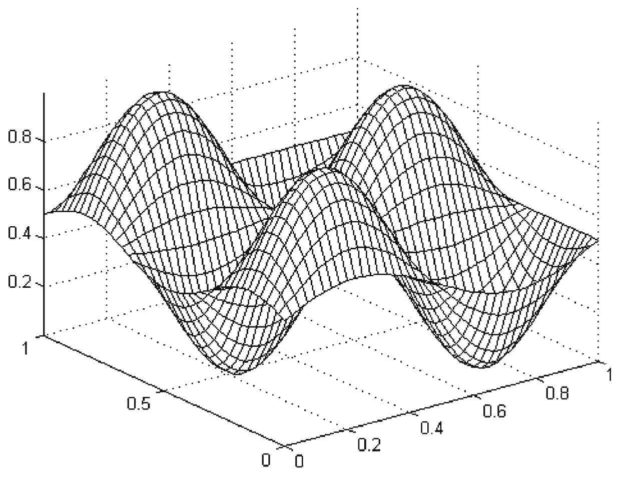

Figure 5. Target function of $F(0.5, x_2, x_3)$.

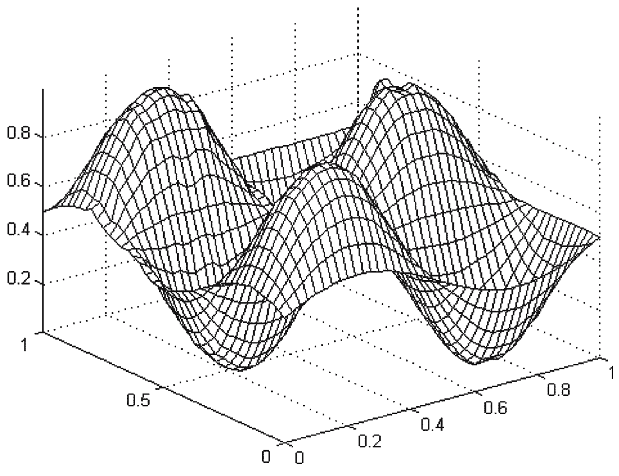

Figure 6. Output result after learning.

Table 2. Several benchmark data sets from UCI machine learning repository [33].

Data set	Number of features	Number of classes	Number of training instances	Number of testing instances	Data type
Iris	4	3	75	75	Continuous
Liver	6	2	173	172	Continuous
Breast	9	2	350	349	Continuous
Echo	11	2	66	66	Continuous
Va-Heart	13	2	100	100	Continuous
Wine	13	3	89	89	Continuous
All-hyper	29	3	486	486	Continuous
Lung	56	3	18	15	Continuous
Soy	208	17	150	139	Binary
Promoter	228	2	53	53	Binary

Table 2 lists the characteristics of these data sets. The ten benchmark data sets include eight continuous data sets and two binary data sets.

In this experiment, ten benchmark data sets were tested using the original BP neural network (i.e. multi-layer perceptron) [42], Lin's support vector machine (LIBSVM) [43], self-organizing HCMAC neural network [39], and the proposed self-organizing MHCMAC neural network. Tables 3 and 4 summarize the training and testing classification results, respectively. To give a fair comparison, the choice of learning parameters for the proposed self-organizing MHCMAC neural network is the same as the self-organizing HCMAC neural network [39]. Lin's LIBSVM can automatically determine the SVM parameters including the used kernel (default as

Table 3. Training results of various learning models.

	Model										
	BP			SVM [43]		Self-organizing HCMAC [39]			Self-organizing MHCMAC		
Data set	Network topology	Averaged training time (sec)	Averaged accuracy	Averaged training time (sec)	Averaged accuracy (%)	Memory require-ment	Averaged training time (sec)	Averaged accuracy (%)	Memory require-ment	Averaged training time (sec)	Averaged accuracy (%)
Iris (MSE = 0.014)	4-4-3	7.812	98.5	0.52	97.3	555	0.115	99.6	555	0.115	99.6
Liver (MSE = 0.04)	6-4-2	249.48	84.9	0.709	75.1	3240	0.849	99.0	3162	2.622	99.4
Breast (MSE = 0.04)	9-5-2	17.99	98.8	0.909	96.9	1932	2.331	99.4	510	0.468	98.2
Echo (MSE = 0.04)	11-7-2	3.37	96.9	0.248	89.4	2670	0.38	99.0	2010	0.534	99.1
Va-Heart (MSE = 0.1)	13-8-2	4.87	98.4	0.549	81.0	1296	4.03	91.0	780	3.24	89.2
Wine (MSE = 0.04)	13-8-3	3.83	98.0	0.566	98.9	15984	1.064	99.1	13650	1.520	99.0
All-hyper (MSE = 0.04)	29-16-3	3.85	98.5	1.754	97.7	7605	5.22	98.3	7083	2.194	97.2
Lung (MSE = 0.04)	56-30-3	5.59	98.0	0.532	61.1	4077	2.245	99.1	3609	2.725	99.8
Soy (MSE = 0.005)	—	—	#	2.351	92.0	504152	14.96	92.6	359516	68.81	99.3
Promoter (MSE = 0.04)	—	—	#	0.647	88.7	30504	14.396	99.2	13584	9.402	99

#: divergence

Table 4. Testing results of various learning models.

	Model							
	BP		SVM [43]		Self-organizing HCMAC [39]		MHCMAC	
Data set	Averaged accuracy rate (%)	Standard deviation	Averaged accuracy rate (%)	Standard deviation	Averaged accuracy rate (%)	Standard deviation	Averaged accuracy rate (%)	Standard deviation
Iris	96.5 %	0.0039	94.7	0.0052	96.8	0.0032	96.8	0.0041
Liver	71.0 %	0.0041	60.5	0.0014	63.5	0.0053	66.8	0.0022
Breast	95.1 %	0.0043	96.6	0.0044	95.7	0.0044	96.8	0.0027
Echo	90.9 %	0.0015	77.3	0.0052	88.4	0.0032	90.0	0.0021
Va-Heart	99.5	0.0044	69.0	0.0056	87.0	0.0041	86.3	0.0052
Wine	96.6	0.0025	95.5	0.0024	83.1	0.0022	85.1	0.0018
All-hyper	97.7	0.0057	97.1	0.0052	97.3	0.0038	97.1	0.0019
Lung	26.6	0.0025	33.3	0.0047	53.3	0.0030	54.6	0.0026
Soy	#	–	54.7	0.0017	84.6	0.0037	91.3	0.0016
Promoter	#	–	66.0	0.0042	77.3	0.0016	83	0.0022

#: divergence

the radial basis function) and kernel parameters by the grid parameter search approach [43]. The experimental results reveal that the training classification of the original self-organizing HCMAC and MHCMAC neural networks is more accurate than those of the BP neural network and support vector machine. The self-organizing MHCMAC neural network exhibits a slightly lower classification accuracy on the training data than the original self-organizing HCMAC neural network, but the self-organizing MHCMAC neural network exhibits a higher classification accuracy on test data than the original self-organizing HCMAC neural network on most testing data sets. Restated, the self-organizing MHCMAC neural network generalizes more effectively than the original self-organizing HCMAC neural network. This is because the zero-valued inputs and allocating redundant GCMAC nodes indeed affect the learning performance of the self-organizing HCMAC neural network, but the proposed self-organizing MHCMAC learning structure with exact input features can obviously improve this flaw that occurred in the self-organizing HCMAC neural network. From the experimental results, we find that the improvement of the learning performance of the self-organizing MHCMAC is highly relevant with the number of the inputs of redundant leaf GCMAC nodes that occurred in the self-organizing HCMAC. That is, the ratio of the number of the input dimensions assigned zero value to the number of the input dimensions assigned normal input feature value will affect the learning performance of the self-organizing HCMAC. In particular, the classification accuracy rate of the self-organizing HCMAC is obviously poorer than the self-organizing MHCMAC when the ratio is raised. This phenomenon could be supported by the experimental results listed in Tables 3 and 4. Based on the observation, we find that the Soy data set has the best improvement after comparing the learning performance of the self-organizing HCMAC with the self-organizing MHCMAC because the ratio is the largest one among the ten testing data sets. Additionally, the classification accuracy rates of the BP neural network on the test data surpass those of the SVM and the original self-organizing HCMAC and MHCMAC neural networks for the Liver, Echo, Va-Heart, Wine and All-hyper data sets. The BP algorithm could not converge to the specified MSE at several assigned learning rates when analyzing the Soy and Promoter binary data, which respectively include 208 and 228 features. However, the experimental results obtained from the original self-organizing HCMAC and MHCMAC neural network classifiers are encouraging because they reveal high accuracy classification rates on the training and testing data for these two data sets with which the BP algorithm and SVM were not very effective. In particular, the classification accuracies of the self-organizing MHCMAC on the training and testing data clearly exceed those of the original self-organizing HCMAC neural network and SVM on these two data sets. In the experiment, the classification results of the self-organizing MHCMAC neural network in the Lung, Soy and Promoter data sets are better than those obtained using the BP neural network, SVM and the original self-organizing HCMAC neural network. Restated, the MHCMAC neural network handles high-dimensional pattern classification problems better than the BP, SVM or the original self-organizing HCMAC neural network.

Moreover, the self-organizing MHCMAC neural network needs much less memory than the original self-organizing HCMAC neural network. An experimental evaluation of the training CPU time for the proposed novel minimal structure implies that the self-organizing MHCMAC neural network takes less CPU time than the original self-organizing HCMAC neural network when applied to Breast, Va-Heart, All-Hyper, and Promoter data sets, but the self-organizing MHCMAC neural network takes much more CPU time than the original self-organizing HCMAC neural network when applied to Liver, Echo, Wine, Lung, and Soy data sets. Furthermore, the structure of the BP neural network is difficult to determine for high-dimensional pattern classification problems and does not appear to converge well.

5.3. EXPERIMENT ON THE DATA SET OF THE SYSKILL & WEBERT'S WEB PAGE RATINGS FROM UCI MACHINE LEARNING REPOSITORY

Syskill & Webert's web page rating system [44, 45] is designed to help users identify Web pages on a particular topic. This system provides a user interface enabling the user to rate a page for three different levels of interest, hot, medium or cold.

The rating results are recorded as a user profile with positive or negative examples. Each user has a set of profiles for each topic. The system's data set contains HTML source code of Web pages and each user's ratings on four separate subjects, Bands-recording artists, Goats, Sheep and BioMedical. Except the HTML source code, each page rating entry includes five records, HTML source file name, rating level (hot, medium or cold), URL, date visited and title of Web page.

In this experiment, the proposed self-organizing MHCMAC neural network classifier was applied to predict user rating via learning user profiles for personalized navigation. Although the rating system can be used by several users, here we only use web page rating data from one user. Table 5 summarizes the data set using rating topics, together with the lists of different levels of interests and the total number of Web pages rated by a single user. Most studies merge the 'medium' and 'cold' interest levels [6, 44, 45] because very few Web pages are marked as 'medium'. To evaluate the effectiveness of the classification algorithms fairly, this study also merged these two interest levels. A random process was used to partition the data set in Table 5 into training and testing data sets, both comprising half the original data set.

To process Web pages into a vector representation with reduced dimensionality for the classification algorithms, three processing components, HTML parser, text processing, and keywords selection, are needed to generate informative keywords. In this experiment, an HTML parser was first designed to remove all tags from the HTML documents, since the tags are not relevant to the topics, and the text contents was then used for further text processing and feature selection. Text processing refers to removing non-textual words (e.g., numeric data, symbols, notation and ASCII drawings) and stop-words from the original Web documents, and transferring the remaining words into stems by a stemming rule [46]. The Brown corpus stop-list [47] was used to preprocess Web documents. To identify informative keywords to characterize Web documents in vector representation, a fair feature subset selection algorithm [48] proposed in our previous study was applied to

Table 5. Topics used in our experiments.

	Interesting levels and corresponding number of pages			
Topic	Number of hot pages	Number of medium pages	Number of cold pages	Total number of pages
Bands-recording artists	15	7	39	61
Goat	32	1	37	70
Sheep	14	0	51	65
BioMedical	32	3	101	136

select the informative keywords from the training data set. After the useful features were identified, the TF $\times$ IDF approach [46] was used to assign an importance, i.e. weight, to each feature in the Web document. This study employed a feature's occurrence frequency to indicate its representativeness in a given Web document. That is, if a keyword appears frequently in a particular Web document, then it is highly representative of the document's subject.

Next, to explore the impact of selecting different numbers of features for learning user profiles, four classifiers, Vector Space Model (VSM) [46], Linear Text Classifier (LTC) [49, 50], BP neural network [42] and self-organizing MHCMAC neural network, were compared using Syskill & Webert's Web page ratings data set, using half the examples for training and the other half for testing. Table 6 lists the training results of the self-organizing MHCMAC neural network, averaging ten independent trials. The proposed method achieves an average training accuracy of almost 100% on four topics of user profiles. Figures 7–10 show the testing accuracy rates from selecting other numbers of features for each of the four classifiers. These experiments show that the proposed self-organizing MHCMAC did not achieve best prediction accuracy rates for all four user profiles with various data distribution on the Syskill & Webert's Web page ratings data set. However, no machine learning method can guarantee to solve all data distribution problems well, especially for highly changeable Web page classification problems. Generally, the average performance is a logical and reasonable benchmark for Web page classification problems. Accordingly, Figure 11 summarizes the previous results, showing that the proposed self-organizing MHCMAC predicted best for all different numbers of informative features on the four user profiles. Moreover, the self-organizing MHCMAC also performs better than the earlier self-organizing HCMAC result in this experiment [6]. The experimental results also show that the self-organizing MHCMAC neural network is better suited to high-dimensional problems than BP neural network.

Table 6. The average training accuracy of varying the number of informative features for the MHCMAC neural network.

	Topics			
Features	BioMedical	Bands	Goats	Sheep
16	94.5	100	100	97.0
32	97.5	100	100	100
64	98.5	100	100	96.8
96	98.5	100	100	100
128	100	100	100	100
200	100	100	100	100
256	98.5	100	100	100
400	98.5	100	100	100

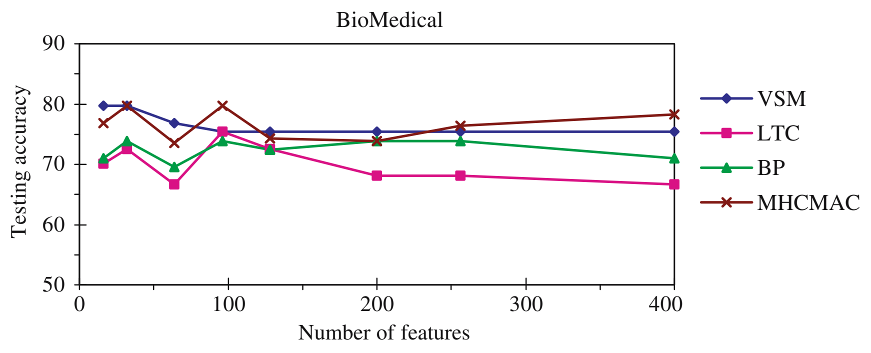

Figure 7. The predicting effect of varying the number of informative features on the user profile of BioMedical.

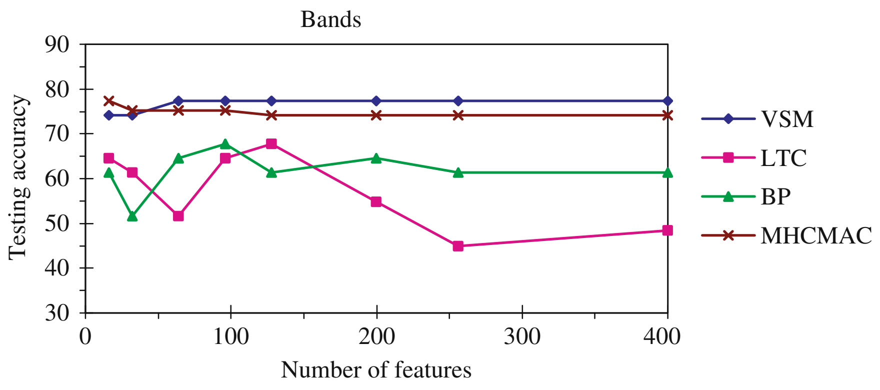

Figure 8. The predicting effect of varying the number of informative features on the user profile of Bands.

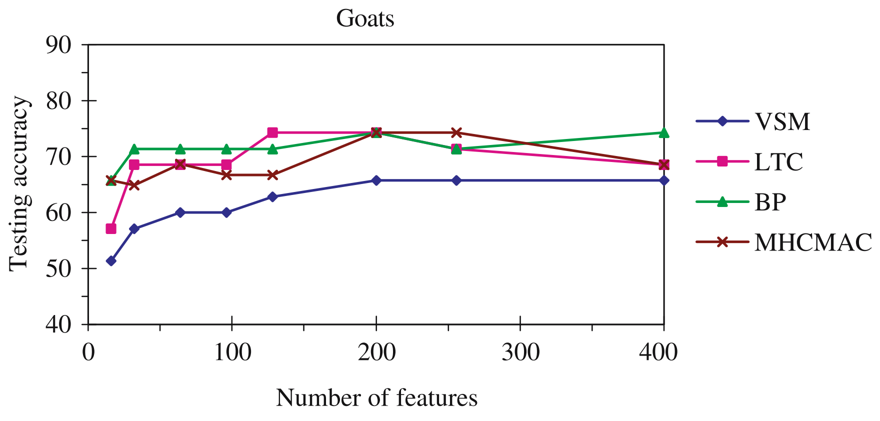

Figure 9. The predicting effect of varying the number of informative features on the user profile of Goats.

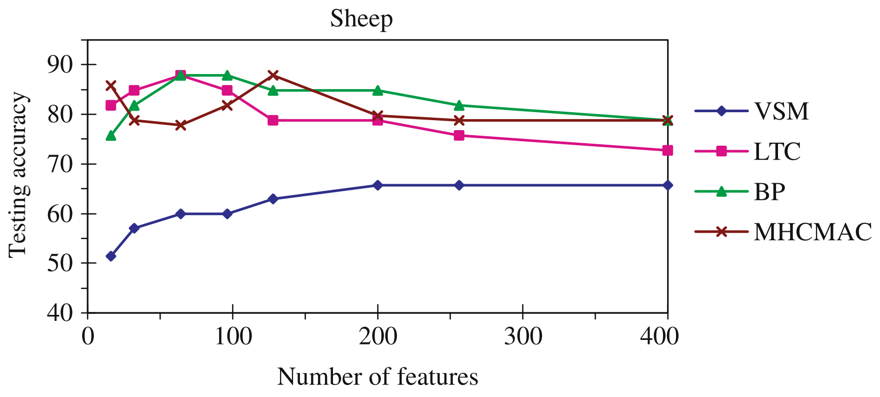

Figure 10. The predicting effect of varying the number of informative features on the user profile of Sheep.

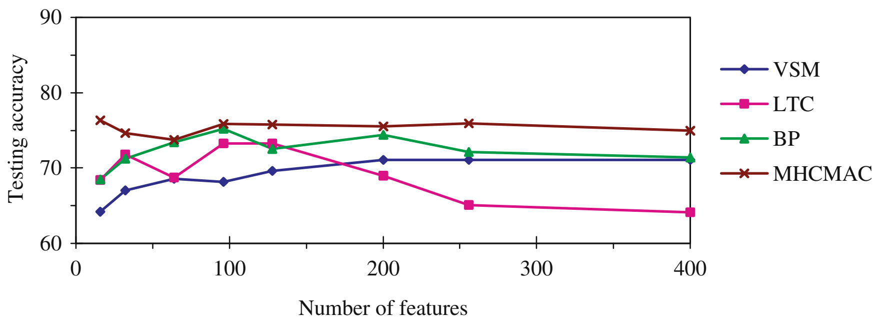

Figure 11. The average predicting effect of varying the number of informative features on the four user profiles.

6. Conclusion

This study presents a minimal structure of a self-organizing HCMAC neural network, called as MHCMAC, to eliminate the wasted extra memory units in the original self-organizing HCMAC neural network. The problem is solved using the structural expansion approach of the exact binary tree topology when solving pattern classification problems. The proposed self-organizing MHCMAC neural network is based on two-dimensional differentiable GCMACs, but this study presents an exact structural expansion approach to replace the full binary tree topology structure herein. The experimental results show that the proposed self-organizing MHCMAC neural network can not only significantly reduce the memory requirement in the original self-organizing HCMAC neural network, but also maintain a high training speed and a higher pattern classification rate than the original self-organizing HCMAC neural network when applied to most testing benchmark data sets. Meanwhile, experiments on the four topics of user profiles also show that the MHCMAC neural network predicts Web pages of interest to users better than other well-known classifiers.

Acknowledgement

The authors would like to thank the National Science Council of the Republic of China, Taiwan for financially supporting this research under Contract No. NSC 94-2520-S-026-002.

References

1. Albus, J. S.: A new approach to manipulator control: The cerebellar model articulation controller (CMAC), *J. Dyn. Syst. Meas. Control Trans. ASME*, (1975), 220–227.
2. Albus, J. S.: Data storage in the cerebellar model articulation controller (CMAC), *J. Dyn. Sys. Meas. Control. Trans. ASME*, (1975), 228–233.
3. Hu, J., Pratt, J. and Pratt, G.: Stable adaptive control of a bipedal walking; Robot with CMAC neural networks, *Proceedings of 1999 IEEE International Conference on Robotics and Automation*, Detroit, MI, USA, vol. 2, pp. 1050–1056, 1999.
4. Miller, W. T. and Glanz, F. H.: CMAC: An associative Neural Network alternative to backpropagation, *Proceedings of the IEEE*, **78**(10), (1990), 1561–1567.
5. Glanz, F. H., Miller, W. T. and Kraft, L. G.: An Overview of the CMAC neural network, *Proceedings of 1991 IEEE Neural Networks for Ocean Engineering*, Washington, DC, USA, 1991, pp. 301–308.
6. Chen, C. M.: Incremental personalized web page mining utilizing self-organizing HCMAC neural network, *Web Intelligence Agent Syst. Int. J.*, vol. 2, pp. 21–38, 2004.
7. Cotter, N. E. and Guillerm, T. J.: The CMAC and a theorem of Kolmogorov, *Neural Networks*, **5** (1991), 221–228
8. Zhang, K. and Qian, F.: Fuzzy CMAC and its application, *Proceedings of the 3th World Congress on Intelligent Control and Automation*, Hefei, China, pp. 944–947, 2000.
9. Sayil, S. and Lee, K. Y.: A hybrid maximum error algorithm with neighborhood training for CMAC, *In Proceedings of IJCNN'02*, vol. 1, pp. 165–170, 2002.
10. Weruaga, L.: Active Training on the CMAC Neural Network, *In Proceedings of IJCNN'2004*, vol. 2, pp. 855–860, 2004.
11. Hu, J. S. and Hu, G. W.: FCMAC based on extreme element algorithm, *In Proceedings of The Third International Conference on Machine Learning and Cybernetics*, vol. 3, pp. 1889–1894, 2004.
12. Zhang, L., Cao, Q., Lee, J. and Zhao, Y.: A modified CMAC algorithm based on credit assignment, *Neural Process. Lett.*, **20** (2004), 1–10.
13. Lane, S. H., Handelman, D. A. and Gelfand J. J.: Theory and development of higher-order CMAC neural networks, *IEEE Control Syst. Mag.* **12**(2) (1992), 23–30.
14. Lin, C. S. and Li, C. K.: A sum-of-product neural network (SOPNN), *Neurocomputing* **30** (2000), 273–291.
15. Lin, C. S. and Li, C. K.: A low-dimensional-CMAC-based neural network, *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, Beijing, China, vol. 2, pp. 1297–1302, 1996.
16. Li, B. and Elliman, D.: Fuzzy Classification by a CMAC network, *Proceedings of IEEE International Conference on Tools with Artificial Intelligence*, Newport Beach, CA, USA, 1997, pp. 392–395.

17. Almeida, P. E. M. and Simoes, M. G.: Parametric CMAC networks: Fundamentals and applications of a fast convergence neural structure, *IEEE Trans. Industry Appl.* vol. 39, no. 5, pp. 1551–1557, 2003.
18. Hwang, K. S. and Hsu, Y. P.: A new approach to improving a CMAC architecture, *IEEE International Conference on Neural Network and Signal Processing*, vol. 1, pp. 320–323, 2003.
19. Lin, C. J., Chen, H. J. and Lee, C. Y.: A self-organizing recurrent fuzzy CMAC model for dynamic system identification, *In Proceedings of IEEE International Conference on Fuzzy Systems*, vol. 2, pp. 697–702, 2004.
20. Horváth, G.: Kernel CMAC with improved capability, *In proceedings of IJCNN'2004*, vol. 1, pp. 681–686, 2004.
21. Lin, C. S. and Kim, H.: Selection of learning parameters for CMAC-based adaptive critic learning, *IEEE Trans. Neural Networks*, **6** (1995), 642–647.
22. Hsu, Y. P., Hwang K. S., Pao, C. Y. and Wang, J. S.: A new CMAC neural network architecture and its ASIC realization, *In Proceedings of the ASP-DAC 2000, Asia and South Pasific Design Automation Conference*, pp. 481–484, 2000.
23. Miller, T. W., Glanz, F. H. and Kraft, L. G.: Application of a general learning algorithm to the control of Robotic manipulators, *Int. J. Robotic Res.*, **6**(2), (1987), 84–98.
24. Shelton, R. O. and Peterson, J. K.: Controlling a truck with an adaptive critic CMAC design, *Simulation*, **58**(5), (1992), 319–326.
25. Miller, W. T. and Aldrich: Rapid learning using CMAC neural networks: Real time control of an unstable system, *Proceedings of the 5th IEEE International Symposium on Intelligent Control 1990*, Philadelphia, PA, pp. 465–470, 1990.
26. Wong, Y. F. and Sideris, A.: Learning convergence in the cerebellar model articulation controller, *IEEE Trans. Neural Networks*, **3**(1), (1992), 115–121.
27. Lin, C.-S. and Chiang, C.-T.: Learning convergence of CMAC technique, *IEEE Trans. Neural Networks*, **8**(6) (1997), 1281–292.
28. Hu, J. and Pratt, F.: Self-organizing CMAC neural networks and adaptive dynamic control, *Proceedings of the 1999 IEEE International Symposium on Intelligent Control/Intelligent Systems and Semiotics*, Cambridge, MA, USA, pp. 259–265, 1999.
29. Menozzi, A. and Chow, M.-Y.: On the training of a multi-resolution CMAC neural network, *Proceedings of the IEEE International Symposium on Industrial Electronics*, ISIE'97, Guimaraes, Portugal, vol. 3, pp. 1201–1205, 1997.
30. Zhong, L., Zhongming, Z. and Chongguang, Z.: The unfavourable effects of hashing coding on CMAC convergence and compensatory measure, *IEEE International Conference on Intelligent Processing Systems*, Beijing, China, October 1997, pp. 419–422.
31. Wang, Z.-Q., Schiano, J. L. and Ginsberg, M.: Hash-coding in CMAC neural networks, *IEEE International Conference on Neural Networks*, Washington, DC, USA, vol. 3, pp. 1698–1703, 1996.
32. Lin, C. S. and Li, C. K.: A Memory-based self-generated basis function neural network, *Inter. J. Neural Syst.*, **9**(1), (1999), 41–59.
33. UCI Machine Learning Repository, web available from http://www.ics.uci.edu/~mlearn/MLRepository.html.
34. Jan, J. C. and Hung, S. L.: High-order MS_CMAC neural network, *IEEE Trans. Neural Networks*, **12**(3), (2001), 598–603.
35. Moody, J.: Fast-learning in Multi-resolution Hierarchies, *Adv. Neural Infor. Processing Syst.*, **14**(1), (1989), 29–38.
36. Kim, H. and Lin, C. S.: Use of adaptive resolution for better CMAC learning, *Proceedings of International Joint Conference on Neural Networks*, Baltimore, MD, USA, vol. 1, pp. 517–522, 1992.

37. Berger, C. S.: Linear splines with adaptive mesh sizes for modelling nonlinear dynamic systems, *IEEE Proceedings of Control Theory Application*, **141**(5), (1994), 277–286.
38. Chiang, C. T. and Lin, C. S.: CMAC with general basis functions, *Neural Networks*, **9**(7), (1996), 1199–1211.
39. Lee, H.-M., Chen, C.-M. and Lu, Y.-F.: A self-organizing HCMAC neural network classifier, *IEEE Trans. Neural Networks*, **14**(1), (2003), 1–13.
40. Shannon, C. E.: A mathematical theory of communication, *Bell Syst. Tech. J.*, **27** (1948), 379–423, 623–656.
41. Bazaraa, M. S., Sherali, H. D. and Shetty, C. M.: *Nonlinear Programming Theory and Algorithms,* 2nd edition, John Wiley & Sons, Inc., pp. 270–271, 1993.
42. Rumelhart, D. E., Hinton, G. E. and Williams, R. J.: Learning internal representation by error propagation, *Parallel Distribut. Process.*, **1** (1986), 318–362.
43. LIBSVM, web available from http://www.csie.ntu.edu.tw/~cjlin/libsvm/
44. Pazzani, M. and Billsus, D.: Learning and revising user profiles: The identification of interesting web sites, *Machine Learning*, **27** (1997), 313–331.
45. Pazzani, M., Muramatsu, J. and Billsus, D.: Syskill & Webert: Identifying interesting web sites, *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pp. 54–59, 1996.
46. Salton, G.: *Automatic Text Processing: The Transformation, Analysis and Retrieval of Information by Computer*, Addison-Wesley, 1989.
47. Francis, W. and Kucera, H.: *Frequency Analysis of English Usage*, New York, 1982.
48. Lee, H.-M., Chen, C.-M. and Tan, C.-C.: An intelligent web-page classifier with fair feature-subset selection, *Proceedings of Joint 9th IFSA World Congress and 20th NAFIP International Conference (IFSA/NAFIPS'01)*, Vancouver, Canada, vol. 1, pp. 395–400, 2001.
49. Lewis, D. D. and Gale, W. A.: A sequential algorithm for training text classifiers, *Proceedings of SIGIR* 1994, pp. 3–12.
50. Lewis, D. D., Schapire, R. E., Callan, J. P. and Papka, R.: Training algorithms for linear text classifier, *In Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'96)*, 1996, pp. 298–306.