Available online at www.sciencedirect.com
ScienceDirect
Neurocomputing 70 (2006) 445–461
NEUROCOMPUTING
www.elsevier.com/locate/neucom

A fast modified constructive-covering algorithm for binary multi-layer neural networks

Di Wang[a,*], Narendra. S. Chaudhari[b]

[a]School of Informatics, The University of Manchester, Sackvill Street, M601QH Manchester, UK
[b]School of Computer Engineering, Nanyang Technological University (NTU), Singapore

Received 2 March 2004; received in revised form 13 April 2005; accepted 19 December 2005
Communicated by J. H. Kim
Available online 15 May 2006

Abstract

For binary neural networks (BNNs), constructive covering frameworks have been investigated recently. While these frameworks are fast, they have limitations of generalization and accurate classification for learning from limited number of samples. In this paper, we propose modified constructive-covering algorithm (MCCA), which consists of two processes: generalization process and modification process. Errors introduced in the generalization process are revised in the modification process by adding modification neurons. In our approach, we visualize hidden neurons in terms of hypershperes. The learning process is the geometrical expansion process of these hypershperes. Through our experimental work in Section 5, we conclude that, MCCA is not sensitive to the order in which the input sequence is given. In addition, MCCA results in simple neural network structures by less training time.

Keywords: Binary neural networks; Constructive learning; Geometrical learning; Linearly separablility; Multi-layer neural networks.

1. Introduction

Binary neural networks (BNNs) drew great attention because of their better performance to deal with non-order (categorical) inputs and their easiness to be realized by hardware. Furthermore, BNNs have the property that all input samples (Boolean vectors) are located on the vertices of a unit hypercube (hence the surface of a hypersphere). This property makes it possible to transform hidden neurons with non-linear representation to hidden neurons with linear representation [5].

In the past decade, based on this property, many algorithms for construction of binary neural networks were proposed. They include binary-like training algorithm (BLTA) by Gray and Michel [2], expand-and-truncate learning (ETL) by Kim and Park [5], improved ETL (IETL) by Yamamoto and Saito [16], constructive set covering learning algorithm (CSCLA) by Ma and Yang [8,9], newly expand-and-truncate learning algorithm (NETLA) by Sung and Jung [13], multi-core learning (MCL) and multi-core expand-and-truncate learning (MCETL) by Wang and Chaudhari [14,15].

BLTA is a dynamic technique and derives its original principle form Boolean algebra with extension. BLTA does well in memorization and generalization, but too many hidden neurons are needed. ETL finds a set of required separating hyperplanes and automatically determines a required number of neurons in the hidden layer based on geometrical analysis of the training set. ETL begins with selecting one true vertex as the core vertex for set of included true vertices (SITV) [5]; then SITV expands to include as many vertices as it can; until no more vertex can be added to SITV, the first separating hyperplane is found; if not all true vertices are separated from all false vertices, the second separating hyperplane is expected to be found; to do this, false vertices are converted to true vertices, and true vertices which are not in SITV are converted to false vertices temperately; then the second separating hyperplane is found in the similar way; this process goes on until all true vertices are separated from all false vertices. In ETL, during training, the reason why a hypersphere cannot

*Corresponding author. Tel.: +44 161 3062063; fax: +44 161 3063346.
E-mail address: di.wang@manchester.ac.uk (D. Wang).

0925-2312/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2005.12.124

expand is due to the existence of false vertices around the SITV hypersphere. These false vertices block the expansion of SITV hypersphere. IETL improves ETL by taking part of vertices covered by previous hidden neurons as "do not care". But which part of vertices is considered as "do not care" is not clear. ETL and IETL need search a lot of training pairs for determining each neuron in the hidden layer. MCL and MCETL begin with several cores. However, there is no sound rule to determine the number of core vertices for MCL and MCETL. CSCLA was proposed based on the idea of weighted Hamming distance hypersphere. CSCLA is faster. But CSCLA only considers including vertices with Hamming distance one from the core, not more than one in a hidden neuron. So the representation ability of each hidden neuron is limited. Many hidden neurons are needed by CSCLA to solve the same problems. In general, ETL, IETL, CSCLA, NETLA, MCL and MCETL have no generalization capability. BLTA has generalization capability, but needs more hidden neurons. In addition, ETL, IETL, CSCLA, MCL and MCETL are core-based learning algorithms. The number of hidden neurons depends on the choice of the core vertex and the input order of the training sequence. Hence the network structures constructed by these algorithms vary a lot for different input sequences of the same training set.

All algorithms mentioned above are constructive learning algorithms. Constructive learning has been an attractive approach for neural network construction in recent years. For example, Meng and Yang [10] designed dynamic neural networks based on constructive learning. The idea of constructive learning was also applied in self-organization mapping algorithms [4,12] and RBF neural network training algorithms [7,11] and led better results. Non-constructive learning fixes the neural network structure before training. However, constructive learning begins with a minimal neural network structure, or empty hidden layer, and dynamically increases the network by adding hidden neurons until a satisfying solution is found. Hence constructive learning algorithms help to avoid inappropriate choice of neural network topology. In addition, based on finite number of training samples, constructive learning algorithms guarantee convergence.

In this paper, we propose modified constructive-covering algorithm (MCCA), which consists of two processes: generalization process and modification process. Errors introduced in the generalization process are revised in the modification process by adding modification neurons. MCCA is based on multi-level geometrical expansion. We visualize hidden neurons in terms of hyperspheres. The training process is the geometrical expansion process of these hyperspheres. The main advantages of MCCA over previous learning algorithms for BNNs are:

(i) The MCCA possesses the generalization capability, but all previous learning algorithms (except for BLTA) do not.
(ii) The MCCA does not need a core vertex before training, which makes it more general to be applied.
(iii) The MCCA needs less training time, because it determines whether to add a vertex to a hidden neuron by judging which region it belongs to, not by computing the weighted sum of every vertices in the training set.
(iv) The MCCA is not sensitive to different input sequences of the same training set.

2. Background

Boolean functions have the geometrical property which makes it possible to transform non-linear representation to linear representation for each hidden neuron [5]. We consider a Boolean function with n input and one output,

$$y = f(x_1, x_2, \ldots x_n,), \tag{1}$$

where y $\in$ (0,1) and $x_i \in$(0,1),, $i = (1 \ldots n)$.

These 2^n binary patterns $(0,1)^n$ can be considered as an n-dimensional unit hypercube. Then each pattern is located on one vertex of this hypercube, hence on the surface of the ex-hypersphere of this hypercube. This ex-hypersphere is defined as the reference hypersphere (RHS) [5] as follows:

$$\left(x_1 - \frac{1}{2}\right)^2 + \left(x_2 - \frac{1}{2}\right)^2 + \cdots + \left(x_n - \frac{1}{2}\right)^2 = \frac{n}{4}. \tag{2}$$

Assume a Boolean function of n variables (1) classifies 2^n vertices into two classes: a set of true vertices (with the output y equal to one), and a set of false vertices (with the output y equal to zero) by m (n-1)-dimensional hyperplanes. These m (n-1)-dimensional hyperplanes partition all vertices into $m+1$ groups. Vertices in each group have the same desired outputs: zero or one. The jth hyperplane can be represented by hidden neuron j. Suppose that, only one (n-1)-dimensional hyperplane is needed to separate the true subset from the false subset, i.e. there exists such a hyperplane that all vertices in the true subset lie on one side of the hyperplane, and all vertices in the false subset lie on the other side, then these two subsets are linearly separable. We can separate the true subset from the false subset by a hyperplane if the true (false) subset is linearly separable.

Kim and Park proposed that [5] a Boolean function f is linearly separable if and only if there exists a separating hypersphere (SHP) such that all true vertices lie inside or on the surface of this hypersphere, and all false vertices lie outside. Let the SHP be expressed in the following form:

$$\sum_{i=1}^{n} (x_i - c_i) = r^2, \tag{3}$$

where (c_1, c_2, … c_n,) is the center, and r is the radius of SHP.

Subtract (3) from (2), we gain the intersection of SHP and RHS—the separating hyperplane—which separates all

true vertices from all false vertices as (4)

$$\sum_{i=1}^{n}(1-2c_i)x_i = r^2 - \sum_{i=1}^{n} c_i^2. \quad (4)$$

For all true vertices inside or on the surface of SHP, we have

$$\sum_{i=1}^{n}(x_i - c_i) \leqslant r^2. \quad (5)$$

For all false vertices outside SHP, we have

$$\sum_{i=1}^{n}(x_i - c_i) > r^2. \quad (6)$$

Subtract (3) from (5), we observe that, all vertices inside or on the surface of SHP lie on one side or on the surface of (4)

$$\sum_{i=1}^{n}(1-2c_i)x_i \leqslant r^2 - \sum_{i=1}^{n} c_i^2. \quad (7)$$

Subtract (3) from (6), we observe that, all vertices outside SHP lie on the other side of (4)

$$\sum_{i=1}^{n}(1-2c_i)x_i > r^2 - \sum_{i=1}^{n} c_i^2. \quad (8)$$

So intersection (4) of SHP and RHS is the separating hyperplane, which separates the true subset from the false subset. To restrict our discussion for integer-valued weights, we multiply both sides of (4) by the number of samples separated by that hypersphere: C_0. We get

$$\sum_{i=1}^{n}\left(\left\{2\sum_{k=1}^{v} x_i^k\right\} - C_0\right)x_i = C_0\left[\left\{\sum_{i=1}^{n} c_i^2\right\} - r^2\right]. \quad (9)$$

The coefficients of x_i (on the left-hand side of Eq. (9)) are the connection weights, and the term on the right-hand side of Eq. (9) is the threshold of the corresponding hidden neuron. We represent (9) by using a hard-limiter model for a hidden neuron, which has output zero for values less than the threshold, and has output one for values greater than or equal to the threshold.

If a Boolean function is linearly separable, it can be represented by an n-dimensional hypersphere (and hence hyperplane). Else it can be represented by some m ($m>1$) n-dimensional hyperspheres (and hence hyperplanes). Each hidden neuron in a BNN represents one of these m hyperspheres (and hence linearly separable subsets). So the number of hidden neurons needed is equal to the number of separating hyperplanes. Different approaches for construction of BNNs use different methods for determining these hyperplanes.

Less number of hidden neurons and simple connection weights indicate a simple neural network structure. Many researchers try to simplify the neural network structure when solving problems. A neural network can be simplified by improving the representation capability of each hidden neuron. In this paper, we reduce the number of hidden neurons by expanding the scope of each hidden neuron in the training process based on multi-level geometrical analysis.

3. Modified constructive- covering algorithm

MCCA constructs a three-layered neural network with one input layer, one hidden layer and one output layer. A neural network constructed by MCCA is illustrated in Fig. 1. MCCA begins with an empty hidden layer, and adds hidden neurons to the hidden layer during training.

3.1. Idea of geometrical method

Assume we separate a linearly separable subset with C_0^j vertices by the jth hidden neuron. In our approach, we define the center of hypersphere j: $\boldsymbol{c_j}$ as the gravity center of these C_0^j vertices. Three central radii with center $\boldsymbol{c_j}$ for this hypersphere are defined as r_1^j, r_2^j and r_3^j, where $r_1^j < r_2^j < r_3^j$.

The r_1^j is called the match radius which defines the match region j. The $r_1^j$is the minimal value which meets the condition that all of the C_0^j vertices are exactly in or on hypersphere j. So it is reasonable to define r_1^j as the largest Euclidean distance between the center of hypersphere j and all these C_0^j vertices represented by hidden neuron j. Vertices having in the match region need not be learned again.

The r_2^j is called the claim radius which defines the claim region j. The r_2^j is a little larger than r_1^j. Vertices in the claim region j can be included in hypersphere j by its immediate expansion. Hence vertices in the claim region j cause the expansion of hypersphere j. This vertex is outside this hypersphere before expansion, and just on the surface of it after expansion.

The r_3^j is called the boundary radius which defines the boundary region j. Vertices in the boundary region j cannot be included in hypersphere j by its immediate expansion, but are expected to be included in hypersphere j in the next (or later) training circle. So vertices in the boundary region are backed up to be learned in the next training circle.

Vertices out of the boundary region j are too far from hypersphere j. We do not expect to include these vertices in it.

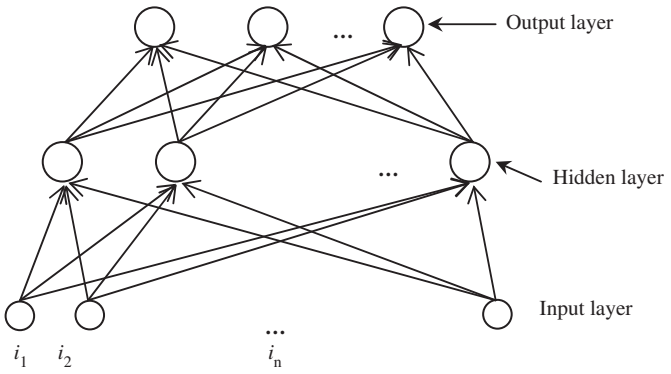


Fig. 1. A three-layered neural network constructed by MCCA.

So a vertex out of the boundary region of any hidden neuron results in the creation of a new hypersphere (a new hidden neuron) to represent it.

Now we illustrate the training process by a visualized example in Fig 2. A new vertex (say $\boldsymbol{x}$) within the "claim region" in Fig. 2(a) (defined using r_2) causes an immediate expansion of the hypersphere, to get a hypersphere with the new vertex on the surface of it after expansion (Fig. 2(b)). A new vertex within the "boundary region" (defined using r_3) cannot be included in the hypersphere by its immediate expansion, but is expected to be included in the next (or later) training circle. A new vertex (say $\boldsymbol{y}$) (Fig. 2(b)) beyond r_3 is too far from this hypersphere. So it causes to the creation of a new hypersphere (a new hidden neuron) to represent it (Fig. 2(c)).

3.2. Parameters for the hidden neurons

3.2.1. Primary parameters: w_{ij} and t_1^j

A typical separating hyperplane in a BNN can be considered as the intersection (separating hyperplane) of a certain separating hypersphere (SHP) and RHS: (4). Each separating hyperplane is represented by a hidden neuron. The process of generating the separating hyperplanes is equal to the process of generating the separating hyperspheres. When the new coming vertex is in the claim region of some hidden neuron, or out of the boundary region of all hidden neurons, the parameters of the neural network need to be modified. Assume we want to separate a linearly separable subset with C_0^j vertices by the jth hidden neuron. For separating hypersphere j we should first determine its center and radius. We define the center of hypersphere j as the gravity center of these C_0^j vertices

$$\boldsymbol{c_j} = (c_1^j, c_2^j, \cdots, c_n^j), \tag{10}$$

$$c_i^j = \sum_{k=1}^{C_{0j}} x_i^k / C_0^j, \tag{11}$$

where $\boldsymbol{c_j}$ is the center of hypersphere j and x_i^k is the ith bit of the kth training vertex of these C_0^j vertices.

After defining the center, we continue to discuss the radii for the separating hypersphere.

The match radius is defined as the largest Euclidean distance between the center $\boldsymbol{c_j}$ and all of these C_0^j vertices

$$(r_1^j)^2 = \max_{k=1}^{C_{0j}} \sum_{i=1}^{n} (x_i^k - c_i^j)^2, \tag{12}$$

where x_i^k is the ith bit of the kth training vertex of these C_0^j vertices.

The corresponding separating hyperplane (hyperplane j) is the intersection of RHS and hypersphere j

$$\sum_{i=1}^{n} (2c_i^j - 1)x_i = \sum_{i=1}^{n} (c_i^j)^2 - (r_1^j)^2. \tag{13}$$

Eq. (13) is the separating hyperplane which can be represented by a linear hidden neuron. In order to restrict our discussion for integer-valued weights, we multiply both sides of (13) by the number of vertices separated by that hypersphere: C_0^j. We get

$$\sum_{i=1}^{n} \left(\left\{ 2 \sum_{k=1}^{C_0^j} x_i^k \right\} - C_0^j \right) x_i = C_0^j \left[\left\{ \sum_{i=1}^{n} (c_i^j)^2 \right\} - (r_1^j)^2 \right]. \tag{14}$$

The coefficients of x_i (on the left-hand side) are the connection weights, and the term on the right-hand side is the threshold of the corresponding hidden neuron. We represent (14) using a hard-limiter model for a neuron, which has an output of zero for values less than the threshold t_1^j, and one for values greater than or equal to the threshold t_1^j.

From (14) we get the integer value of connection weights as follows:

$$w_{ij} = C_0^j (2c_i^j - 1) = 2 \sum_{k=1}^{C_0^j} x_i^k - C_0^j, \tag{15}$$

where w_{ij} is the connection weight between the ith bit of the input and hidden neuron j.

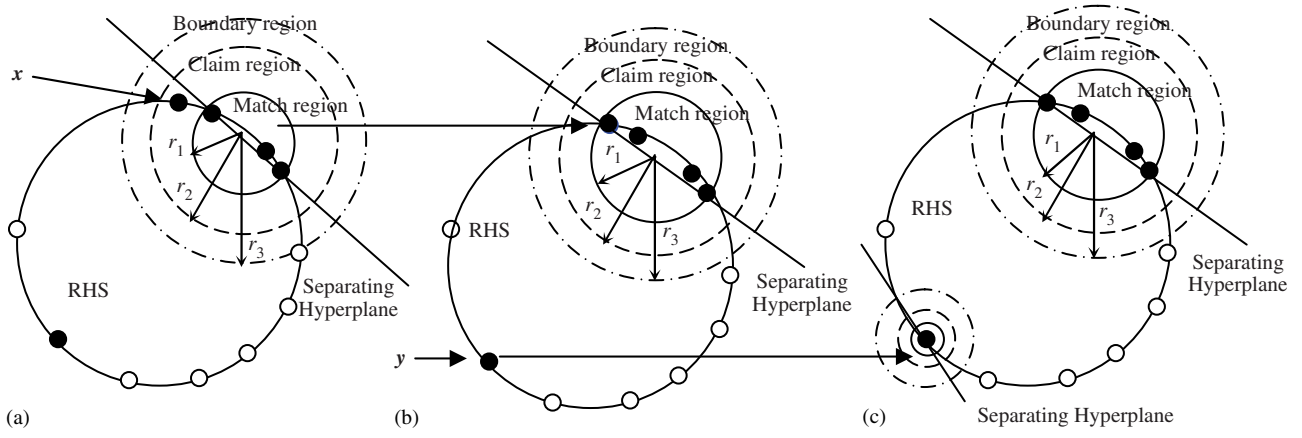


Fig. 2. The process of geometrical expansion.

There are three thresholds for hidden neuron j according to three radii of hypersphere j: t_1^j, t_2^j and t_3^j.

From (12) and (14), we get

$$t_1^j = C_0^j\Big(\sum_{i=1}^{n}(c_i^j)^2 - (r_1^j)^2\Big) = \min_{k=1}^{C_0^j}\sum_{i=1}^{n} C_0^j(2c_i^j x_i^k - (x_i^k)^2), \tag{16}$$

where x_i^k is 0 or 1, so $x_i^k = (x_i^k)^2$.

From (15) and (16) we get

$$t_1^j = \min_{k=1}^{C_0^j}\sum_{i=1}^{n} C_0^j(2c_i^j - 1)x_i^k = \min_{k=1}^{C_0^j}\sum_{i=1}^{n} w_{ij}x_i^k. \tag{17}$$

For all vertices included in hypersphere j

$$\sum_{i=1}^{n} w_{ij}x_i^k \geqslant t_1^j. \tag{18}$$

For all vertices out of hypersphere j

$$\sum_{i=1}^{n} w_{ij}x_i^k < t_1^j. \tag{19}$$

To avoid re-computing, and simplify the computation process, we simplify (15)–(17) to obtain the equivalent form in (20)–(23) in order to compute the connection weights w_{ij} and the threshold t_1^j (for the jth hidden neuron).

For a new coming vertex $\boldsymbol{x^k}$, if $t_3^j > \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x^k}$ (where $\boldsymbol{w}^{\mathrm{T}}$ is the transfer matrix of $\boldsymbol{w}$) for all existing hidden neurons, a new hidden neuron (say, neuron j) is created to represent it. The parameters, w_{ij} and t_1^j, for this neuron are:

$$w_{ij} = \begin{cases} 1 & \text{if} \quad x_i = 1, \\ -1 & \text{if} \quad x_i = 0, \end{cases} \tag{20}$$

$$t_1^j = \sum_{i=1}^{n} w_i x_i^k, \tag{21}$$

where w_{ij} is the connection weight from ith input bit to hidden neuron j, and t_1^j is the match threshold for hidden neuron j.

For a new coming vertex $\boldsymbol{x^k}$, if $t_2^j \leqslant \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^k < t_1^j$, using (15) and (17), we have

$$w_{ij} = \begin{cases} w_{ij} + 1 & \text{if} \quad x_i = 1\,, \\ w_{ij} - 1 & \text{if} \quad x_i = 0, \end{cases} \tag{22}$$

$$t_1^j = \sum_{i=1}^{n} w_i x_i^k, \tag{23}$$

The parameters in (22) and (23) are the same as in (15) and (17).

3.2.2. Secondary parameters: t_2^j and t_3^j

The parameters t_2^j and t_3^j are secondary parameters which are determined by parameters r_2^j and r_3^j to control the geometrical expansion process. We either add a new coming vertex to an existing hidden neuron or create a new hidden neuron to represent it, or put it aside to be reconsidered in the next training circle. To determine the values of thresholds, various approaches are possible. We introduce three approaches to determine the values of thresholds, which are given below. For any hidden neuron, $t_1^j > t_2^j > t_3^j$ due to $r_1^j < r_2^j < r_3^j$.

First approach: Through the analysis above we have $t_1^j > t_2^j > t_3^j$. The simplest way to define t_1^j, t_2^j, and t_2^j is to use ξ_2 and ξ_3 as constant values and we have

$$t_2^j = t_1^j - \xi_1, \tag{24}$$

$$t_3^j = t_2^j - \xi_2. \tag{25}$$

Second approach: If we define the claim region by vertices with Hamming distance σ_1 from the match region, and define the boundary region by vertices with Hamming distance σ_2 from the claim region. We assume every two bits of an input are orthogonal. For hidden neuron j, we define

$$(r_2^j)^2 = (r_1^j)^2 + \sigma_1^2, \tag{26}$$

$$(r_3^j)^2 = (r_2^j)^2 + \sigma_2^2, \tag{27}$$

where σ_1 and σ_2 are the parameters to capture "geometric expansion".

In terms of parameters σ_1 and σ_2, we have Fig. 3 to visualize parameters r_2^j and r_3^j.

According to (14), (26) and (27), we obtain equations to compute t_2^j and t_3^j as

$$t_2^j = \min_{k=1}^{C_0^j}\sum_{i=1}^{n} w_{ij}x_i^k - C_0^j\sigma_1^2, \tag{28}$$

$$t_3^j = \min_{k=1}^{C_0^j}\sum_{i=1}^{n} w_{ij}x_i^k - C_0^j(\sigma_1^2 + \sigma_2^2). \tag{29}$$

Third approach: Using a_1, a_2 and gap, for the jth hidden neuron, we have

$$t_2^j = t_1^j - a_1 \times gap, \tag{30}$$

$$t_3^j = t_2^j - a_2 \times gap, \tag{31}$$

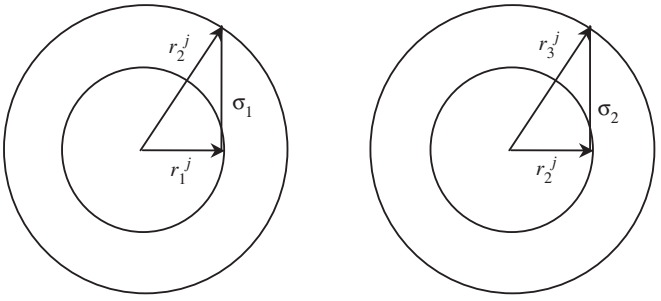


Fig. 3. A choice of r_2^j and r_3^j for MCCA.

```
int gap() // all parameters refer to hidden neuron j
{
//weight[i] is the connection weight w_i;
        Temp1 = Min (Abs (weight[i]-weight[j]));
// which meets the condition Temp1>0; i, j = 1…Input Dimension;
        Temp2 = Min (Abs (weight[i]));
// which meets the condition Temp2>0; i = 1…Input Dimension;
        Temp = Min (Temp1, Temp2);
    Return (Temp);
}
```

Fig. 4. Algorithm to compute the parameter *gap*.

where a_1, a_2 are small integral values, and *gap* is a parameter to update t_2^j and t_3^j. Computing method of parameter *gap* is given in Fig. 4 in forms of peudo-code.

The above three approaches to determine the thresholds t_2^j and t_3^j can be applied under different circumstances. Through our experiments, we observe the following. The first approach helps to balance the number of vertices represented by each hidden neuron, because the expansion process stops after a hidden neuron includes a certain number of vertices. The second method is defined by Hamming distance, and it performs well when solving "nicely clustered" problems (vertices with the same output have small Hamming distance while vertices with different output have large Hamming distance). The third approach aims at a slow expansion and high accuracy at the expense of comparably longer training time. In the third method, generally speaking, a large value of a_1 and a small value of a_2 cause fast learning but more errors, while a small value of a_1 and a large value of a_2 cause precise learning with long training time. For a given problem, choice of these parameters gives our approach freedom to "adapt" it to suit the nature of the given problem.

3.2.3. Parameter discussion for a_1, a_2, and gap in the third method

The third method discussed above has better performance and gives us more freedom in the training process. It is frequently used in our examples. Now we discuss the rationale for the link between parameters a_1, a_2, and *gap*. Firstly, the value of *gap* is computed through the algorithm in Fig. 4 on the fly. In other words, *gap* is the granularity decrease of converting one or two bits of the current hidden neuron. Our algorithm applies binary inputs and integer connection weights. We observe that the decrease of the weighted sum by converting one or two bits is not continuous, but in discrete steps (which we refer to as "granularity"). For example, if the current connection weight for a hidden neuron is {3,3,-3,1,-1,3,3}, and the activation threshold is 12, then the value of *gap* is the minimal nonzero value among the differences of any two weights, here $gap = 2$. This hidden neuron represents vertices $110\begin{pmatrix}00\\10\\11\end{pmatrix}11$. Next we seek the vertices nearest to the corresponding hypersphere. They are all vertices with weighed sum larger than or equal to $t_1^j - a_1 \times gap$. If $a_1 = 1$, then $t_1^j - a_1 \times gap = 10$. The vertices meeting this condition are {1100111, 0101011, 1001011, 1111011, 1101001, 1101010} except the three vertices having been represented by this hidden neuron. Any other vertices are farther than them to this hypersphere. An expansion caused by any of these vertices results in a subset which is linearly separable as well.

We set the initial values for the parameters a_1 and a_2 such that $a_1 > 0$ and $a_2 > 0$. When a_1 is larger than 1, a generation (expansion of a hypersphere) of a vertex may result in a representation involving some vertices not in the training set. During training, if some hypersphere does not expand in a whole training circle, we decrease a_2 (by value '1' in one iteration) for the outer hypersphere and the corresponding middle hypersphere expands by increasing a_1 by one on the fly. It is realized by $t_3^j \leftarrow t_3^j + gap$ and $t_2^j \leftarrow t_2^j - gap$. If some hypersphere does not expand in k continuous training circles, then $t_3^j \leftarrow t_2^j - (a_2 - k) \times gap$ and $t_2^j \leftarrow t_1^j - (a_1 + k) \times gap$. The parameters a_1 and a_2 therefore control the learning speeds and accuracy on the fly.

One conservative way is to set the initial value of a_1 as one and the initial value of a_2 as some large value. Anyway, a_1 increases and a_2 decreases on the fly. To do this, a little longer training time is needed than that with a large initial value of a_1 and a small initial value of a_2.

Given a linearly separable subset, the values of w_{ij}, r_1^j, and $t_1^j$are fixed. The value of t_2^j depends on the required precision and the degree of generalization. The larger the required degree of precision, the larger t_2^j is, and vice versa. The value of t_3^j depends on the requirement of the neural network complexity and the learning speed. A fast and complex system determines a high value of t_3^j, and vice versa. The maximal value of the boundary radius r_3^j is $\sqrt{n}$, the diameter of RHS. And the minimal value of the match radius r_1^j is zero when hypersphere j only includes one vertex.

3.3. Training the hidden layer

3.3.1. Geometrical expansion

Based on the analysis above, we obtain the geometrical expansion process of the hidden neuron.

The function for the jth hidden neuron is defined as

$$f(\boldsymbol{w}^j, \boldsymbol{x}^k) = \sum_{i=1}^{n} w_{ij} x_i^k, \tag{32}$$

where $\boldsymbol{w}^j$ is $(w_{1j}, w_{2j}, \ldots, w_{nj})$, the weight vector of hidden neuron j and $\boldsymbol{x}^k$ is $(x_1^k, x_2^k, \ldots, x_n^k)^{\mathrm{T}}$, the kth vertex.

```
q=1;
for ( p=1,…, maximal interaction times) & if true sample set is not empty do
      for (k=1, …, K) & if sample k is not removed
         for ( j=1, …,q)
            test the j hidden neuron,
               if f(w^j, x^k) ≥ t_1^j
                              ignore this sample, remove x^k from the sample set, and go to test the next
                              sample vector; jump out to check next input;
               if f(w^j, x^k) ≥ t_2^j
                              expand hidden neuron j to include x^k; then remove x^k from the sample and
                              go to test the next sample vector; jump out to check next input;
            next j;
                  if there exists a j, f(w^j, x^k) ≥ t_3^j
                        back up x^k to be dealt with in the next training circle;
                  if for all j,s, f(w^j, x^k) < t_3^j
                        q= q+1 create a new hidden neuron, neuron q, then remove from the sample x^k
                        from the training set;
       next k;
      for ( j=1, …, q)
                  if in this training circle, hidden neuron j is not expanded
                        shirk r_3^j and increase r_2^j ;
                        revise t_2^j and t_3^j ;
            next j
next p
```

Fig. 5. Generalization process of MCCA.

Suppose there are K true samples in the true training set. The geometrical expansion process is shown in Fig. 5.

As we have analyzed, in the geometrical expansion process, new data can be generalized to some hidden neuron through the hypersphere expansion. In other words, the generalization capability of MCCA is realized in the geometrical expansion process. This geometrical expansion process is also called as the generalization process of MCCA by us. The hidden neurons created in this process are called as generalization hidden neurons.

In Fig. 5, q stands for the current number of hidden neurons, $\boldsymbol{w^j}$ stands for the weight vector of the hidden neuron j, K is the total number of vertices in true subset and p stands for the number of iterations.

Select a sample $\boldsymbol{x}^k \in$ true training set, and compute the value of $f(\boldsymbol{w}^j, \boldsymbol{x}^k)$.

If $f(\boldsymbol{w}^j, \boldsymbol{x}^k) \geqslant t_1{}^j$, it means that $\boldsymbol{x}^k$ has been covered by hypersphere j, and need not be learned again.

If $t_2^j \leqslant f(\boldsymbol{w}^j, \boldsymbol{x}^k) < t_1^j$, it means that $\boldsymbol{x}^k$ is not covered by hypersphere j, but can be included in it by an immediate expansion of hidden neuron j. Then we shift the center of hypersphere j to a proper position and expand it as big as to exactly include $\boldsymbol{x}^k$.

If $f(\boldsymbol{w^j}, \boldsymbol{x}^k) < t_3{}^j$, $\boldsymbol{x}^k$ is too far from the cluster represented by hidden neuron j. If we expand hypersphere j to include $\boldsymbol{x}^k$, it may include many more other vertices between $\boldsymbol{x}^k$ and hypersphere j, which will cause over generalization. So if for all existing hidden neurons $f(\boldsymbol{w^j}, \boldsymbol{x^k}) < t_3^j$, a new hidden neuron is created to represent $\boldsymbol{x}^k$.

If $t_3^j \leqslant f(\boldsymbol{w^j}, \boldsymbol{x^k}) < t_2^j$, $\boldsymbol{x}^k$ is not so near as to be included in hidden neuron j immediately. But it is possible to be included after some expansion to include some near vertices. If there exists a hypersphere (hidden neuron) which has $t_3^j \leqslant f(\boldsymbol{w^j}, \boldsymbol{x^k}) < t_2^j$, and is promising to include $\boldsymbol{x}^k$, then $\boldsymbol{x}^k$ will be left to be learned in the next training circle.

But if in one training circle, hidden neuron j does not expand, it is not expected to expand to include vertices within r_3^j in future training circles. In this case, r_3^j shrinks and r_2^jexpands. It avoids some vertices trapped in boundary region j.

When $f(\boldsymbol{w^j}, \boldsymbol{x^k}) < t_3^j$ or $t_2^j \leqslant f(\boldsymbol{w^j}, \boldsymbol{x^k}) < t_1^j$, we need to modify the parameters of $\boldsymbol{w^j}$ and thresholds as discussed below.

When $f(\boldsymbol{w^j}, \boldsymbol{x}^k) < t_3{}^j$, a new hidden neuron is created to represent the new coming vertex $\boldsymbol{x^k}$. The newly created hypersphere is centered at $\boldsymbol{x^k} = (x_1^k, x_2^k, \ldots, x_n{}^k)$, and its radius is zero. This hypersphere (in fact a vertex $\boldsymbol{x^k}$) is large enough to include $\boldsymbol{x^k}$. The corresponding separating hyperplane passes $\boldsymbol{x^k}$. According to (20) the slope of this hyperplane $\boldsymbol{w^j}$ is $(2x_1^k-1, 2x_2^k-1, \ldots, 2x_n{}^k-1)$. And t_1^j is equal to $\sum_{i=1}^n x_i^k$.

If $t_2^j \leqslant f(\boldsymbol{w^j}, \boldsymbol{x^k}) < t_1^j$, we shift the center of hypersphere j to a proper position according to the position of $\boldsymbol{x^k}$ and expand it as much as just to include $\boldsymbol{x^k}$. The center of a hypersphere is always the gravity center of all the vertices included in that hypersphere. The radius is the Euclidean distance between the gravity center and the farthest vertex included in that hypersphere.

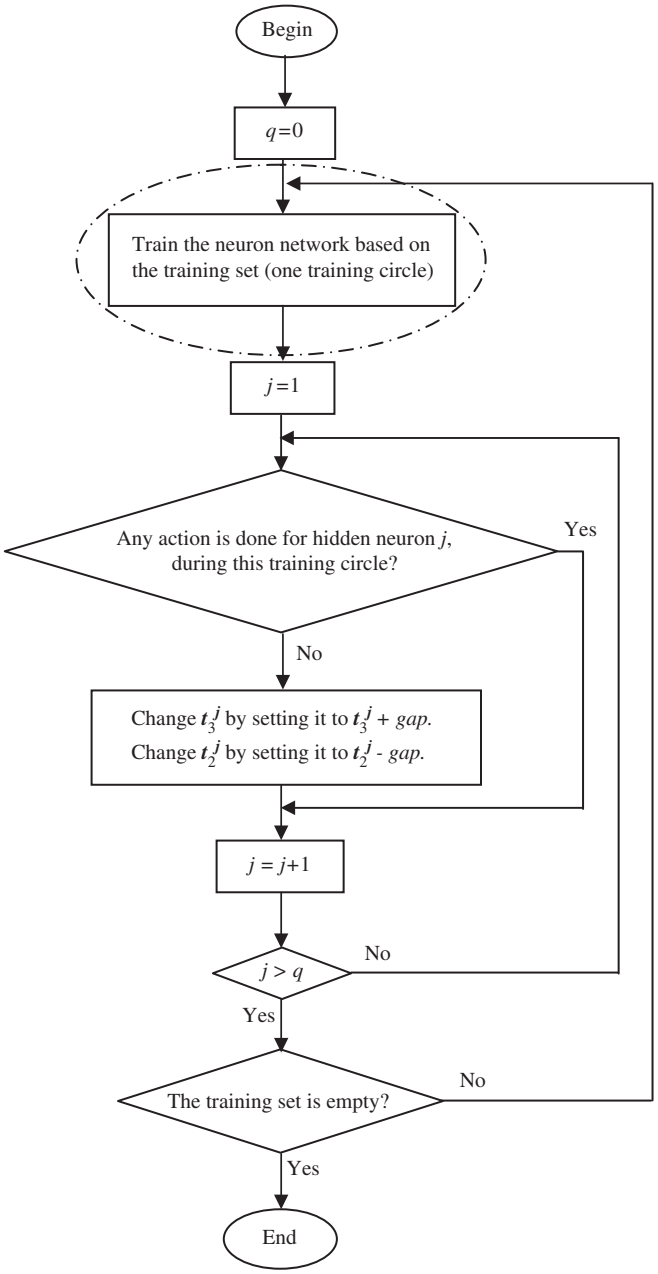


Fig. 6. Flowchart of generalization process for the hidden layer for MCCA.

The flowchart of generalization process for MCCA is shown in Figs. 6 and 7.

3.3.2. Geometrical modification

We call the training process described in Figs. 6 and 7 as the generalization process, and call the hidden neuron created in the generalization process as generalization hidden neurons. In the generalization process, the generalization hidden neurons might generalize data which are not in the training set by geometrical expansion. However this generalization process might introduce some errors. Some generalization hidden neurons might represent some false vertices as well as true vertices. We should modify the hidden layer by adding some modification hidden neurons which represent some false vertices wrongly represented by the generalization hidden neurons. The modification process follows the generalization process. First, in the generalization process, MCCA covers one class (say, the true class) by generalization hidden neurons. Then let all vertices belonging to the other class (say, the false class) go through the learning system. If the output is zero, then omit it. Else if the output is one, this false vertex has been wrongly generalized by some generalization hidden neuron in the generalization process. Then we will represent the wrongly represented vertices by additional hidden neurons. We connect these hidden neurons created in the modification process to the output neuron by negative connection weights to correct the vertices wrongly represent in the generalization process. Fig. 8 visualizes the above modification process. A white vertex wrongly generalized to a generalization hidden neuron representing the black vertices is picked out by an additional hidden neuron (a modification neuron) representing the white vertices. If this modification neuron fires, whether or not the generalization neuron fires, the output for the black class does not fire.

The training algorithm and the flow chart for the modification process of MCCA are shown in Figs. 9 and 10. Here we suppose there are L false samples in the training set.

The equations to train the modification hidden neurons are the same as those in the generalization process

$$w_{ij} = 2\sum_{k=1}^{C_0^j} x_i^k - C_0^j, \tag{33}$$

$$t_1^j = \min_{k=1}^{C_0^j} \sum_{i=1}^{n} w_{ij} x_i^k. \tag{34}$$

t_2^j and t_3^j can be computed through different methods discussed in Section 3.2.

In Fig. 9, o stands for the current number of hidden neurons and $\boldsymbol{w}^j$ stands for the weight vector of hidden neuron j.

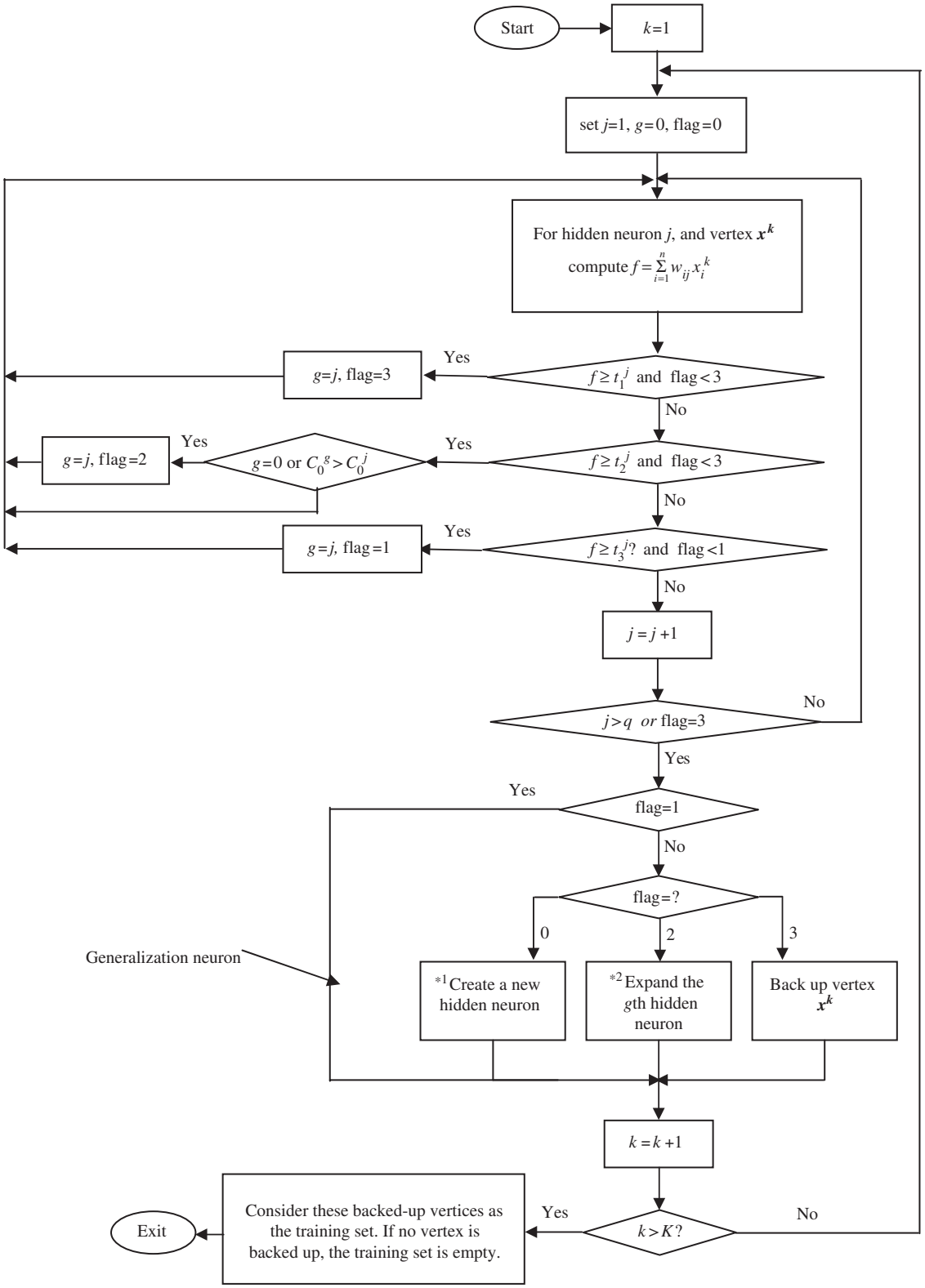
3.4. Train the output layer

The output layer uses one output neuron to collect the outputs from the hidden neurons with connection weights w_j^o from the jth hidden neuron, and threshold t^o for the output neuron

$$w_j^o = \begin{cases} 1 & \text{if} \quad j = 1, \cdots, q, \\ -q & \text{if} \quad j = q+1, \cdots m, \end{cases} \tag{35}$$

$$t^o = 1, \tag{36}$$

where q is the number of modification hidden neurons and m is the number of all hidden neurons.



*1 Create a new hidden neuron: Set $q=q+1$, create hidden neuron q to represent this vertex using (20) and (21)

*2 Expand the gth hidden: Train hidden neuron g using (22) and (23)

Fig. 7. Flow chart of one training circle for MCCA.

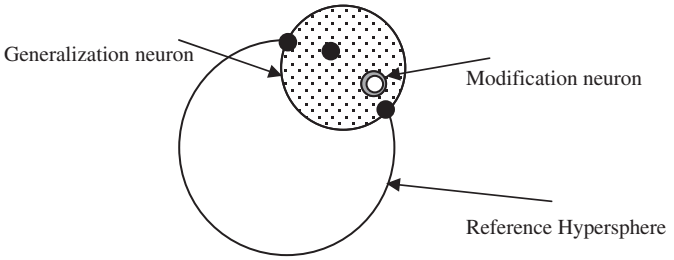


Fig. 8. The modification process for MCCA.

```
o=q+1;
for (l=1, ..., L)
        for ( j=1, ..., q)
    test the j hidden neuron,
     if f (w^j, x^k) ≥ t_1^j
                        for (i=q+1, ..., o)
                          if for all i's f (w^j, x^k) < t_1^j
                           o=o+1, create a new hidden: neuron o;
                        next i;
   next j;
next l.
```

Fig. 9. Algorithm for modification process for MCCA.

3.5. Hard-limiter activation function

The hard-limiter activation function for the jth hidden neuron can be expressed by

$$h_j = \begin{cases} 1 & \text{if} \quad \sum_{i=1}^{n} w_{ij}x_i - t_1^j \geqslant 0, \\ 0 & \text{otherwise.} \end{cases} \tag{37}$$

And the hard-limiter activation function for the output neuron can be expressed by

$$y = \begin{cases} 1 & \text{if} \quad \sum_{j=1}^{m} w_j^o h_j - t^o \geqslant 0, \\ 0 & \text{otherwise.} \end{cases} \tag{38}$$

If $y = 1$, $\sum_{j=1}^{m} w_j^o h_j - t^o \geqslant 0$. That is at least one of the first q hidden neurons fires (output 1), and all the hidden neurons after q do not fire. $\boldsymbol{x}$ is represented by at least one of the generalization hidden neurons, and is not represented by any modification hidden neuron. Then $\boldsymbol{x} \in$ true subset.

If $\boldsymbol{x} \in$ true subset, then $\boldsymbol{x}$ is represented by at least one of the generalization hidden neurons, and is not represented by any modification hidden neuron. Then at least one of the first q hidden neurons fires (output 1), and meanwhile all the hidden neurons after q do not fire. Hence, $\sum_{j=1}^{m} w_j^o h_j - t^o \geqslant 0$, then $y = 1$.

We can verify that, the values of connection weights and thresholds memorized by the neural networks are all integers.

4. Data sets and performance analysis

In order to make comparison, we first consider the example problems considered in previous study.

Example 1. (Approximation of a circular region using 6-bit quantization, Gray and Michel [2], Kim and Park [5], Yamamoto and Saito [16]). This example is to separate a circular region in 2D space which is a square with sides of length 8 with the coordinate origin in the lower left corner. A circle of diameter 4 is placed within the square, locating the center at (4, 4), and then the space is sampled with 64 grad points located at the center of 64 identical squares covering the large square. Forty two of these points fall outside of the circle (desired output 0), and 12 fall within the circle (desired output 1). These 12 points are {010011, 011011, 011010, 101011, 100011, 100010, 011101, 011100, 010100, 101100, 100101, 100100}.

We use different input sequences to show that MCCA is not sensitive to the order of the input sequence. We first consider an arbitrary input sequence: {100100, 011100, 010011, 011010, 100010, 101011, 100011, 011101, 010100, 101100, 100101, 011011}.

We first apply MCCA to this problem by using the second method (using parameters $\sigma_1$and σ_2) to control the geometrical expansion, where $\sigma_1 = \sigma_2 = 1$. The training process is shown in Table 1.

If we apply MCCA with the third method (using parameters a_1 and a_2) to control the geometrical expansion, where $a_1 = a_2 = 1$, the training process is shown in Table 2. The first column of Tables 1 and 2 gives the sequentially input vertices. The second column represents the index of corresponding hidden neuron. The third column is the center of the corresponding separating hypersphere(s) obtained using (11). The connection weights w_i ($i = 1 \ldots 6$) in Tables 1 and 2 are obtained using (20) and (21). The three thresholds in Table 1 are obtained using (21), (28) and (29). The three thresholds in Table 2 are obtained using (21), (30) and (31).

Tables 1 and 2 are similar training processes. The difference is the criteria to expand hidden neurons. For this problem, although the expansion criterion is different, the resultant neural network is the same. In addition, the training process for Table 1 is the same as that for Table 2. Here we only explain the detailed training process through Table 2.

The first twelve rows in Table 2 correspond to 12 true vertices (say, $\boldsymbol{x}^1, \ldots, \boldsymbol{x}^{12}$). The first column in these rows specifies these vertices. The vertex $\boldsymbol{x}^1 = 100100$ causes creation of hidden neuron 1, given in the first row, with weights $\boldsymbol{w}^1 = (1,-1,-1,\ 1,-1,-1)$ obtained using (20) and (22), and thresholds obtained using (21), (30) and (31). For the vertex $\boldsymbol{x}^2 = 011100$, $\boldsymbol{w}^1.\boldsymbol{x}^2 = -1$; so it is outside any region of interest for neuron 1, and causes neuron 2 to be created. Similarly, vertex $\boldsymbol{x}^3 = 010011$ causes neuron 3 to be created. Next, for vertex $\boldsymbol{x}^4 = 011010$, we have $\boldsymbol{w}^1.\boldsymbol{x}^4 = -3$, so it is outside any region of interest for

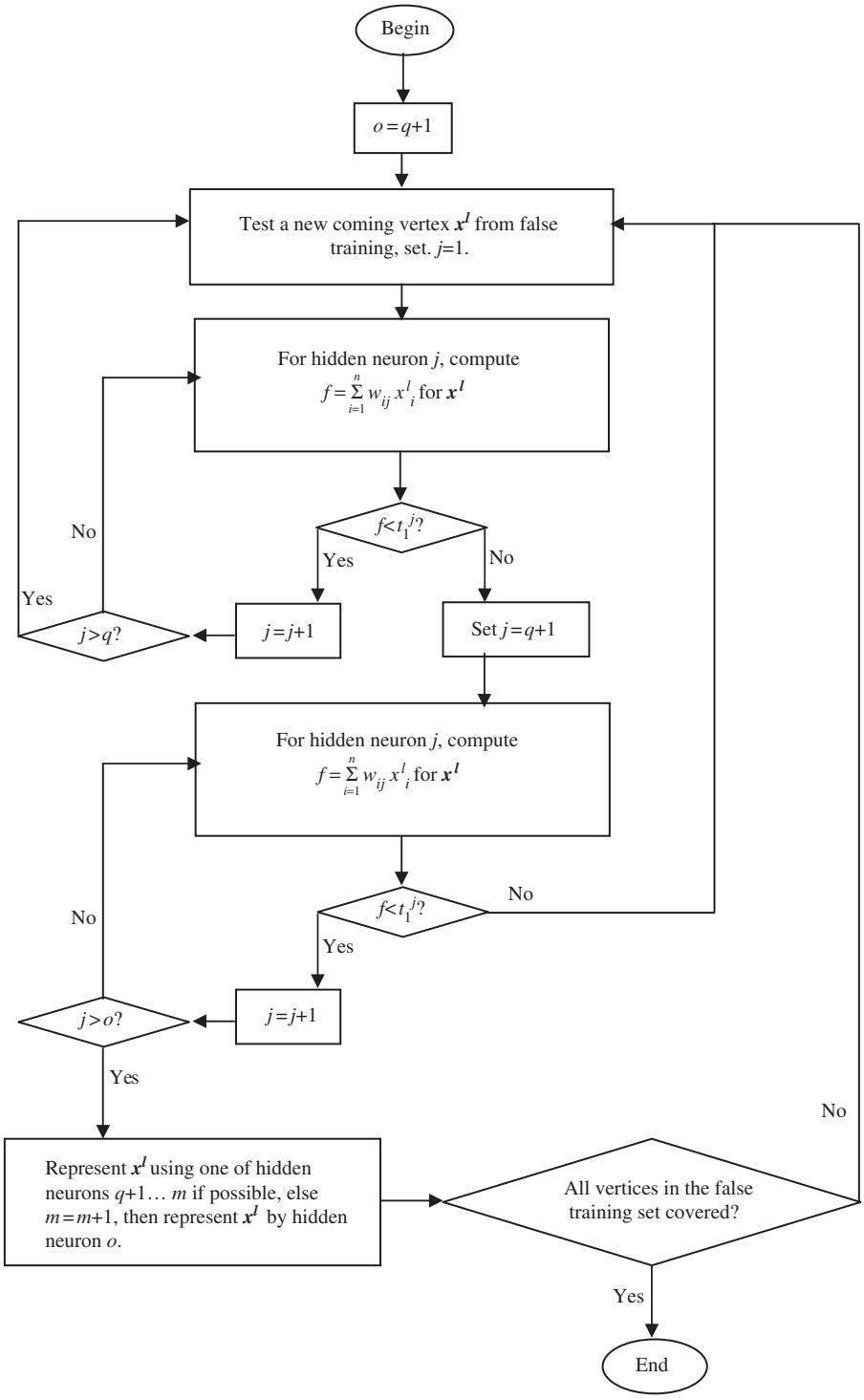


Fig. 10. Flowchart of modification process for MCCA.

neuron 1; however it is in the boundary region for both neurons 2 and 3, ($w^2.x^4 = 1$, $w^3.x^4 = 1$). Since none of the existing neurons can claim x^4, we "put aside" this vertex for re-consideration in the next training circle. This is done by adding it in the shaded portion of Table 1, in row 13. Similarly, $x^5 = 100010$ results in being put aside for re-consideration in the next training circle, thereby resulting in row 14. The vertices x^6, …, x^{12} are in the claim region of one of the existing hidden neurons; so they cause parameter updating for the corresponding hidden neurons. After we finish these 12 vertices, we continue the training process of the two vertices put aside earlier. Neuron 3 and neuron 4 claim these vertices. Modified parameters (weights and threshold values) of these neurons are shown in rows 13 and 14.

After the generalization process, we let all false vertices go through the resultant neural network, and the outputs

Table 1
The learning process of MCCA for 6-bit circle problem by $\sigma_1 = \sigma_2 = 1$

Input	Neuron	Center	r_1	r_2	r_3	w_1	w_2	w_3	w_4	w_5	w_6	t_1	t_2	t_3
100100	1	(1,0,0,1,0,0)	0	1	$\sqrt{2}$	1	−1	−1	1	−1	−1	2	1	0
011100	2	(0,1,1,1,0,0)	0	1	$\sqrt{2}$	−1	1	1	1	−1	−1	3	2	1
010011	3	(0,1,0,0,1,1)	0	1	$\sqrt{2}$	−1	1	−1	−1	1	1	3	2	1
011010	Put aside.													
100010	Put aside.													
101011	4	(1,0,1,0,1,1)	0	1	$\sqrt{2}$	1	−1	1	−1	1	1	4	3	2
100011	4	(1,0,1/2,0,1,1)	1/2	$\sqrt{5}/2$	3/2	2	−2	0	−2	2	2	6	4	2
011101	2	(0,1,1,1,0,1/2)	1/2	$\sqrt{5}/2$	3/2	−2	2	2	2	−2	0	6	4	2
010100	2	(0,1,2/3,1,0,1/3)	$\sqrt{5}/3$	$\sqrt{14}/3$	$\sqrt{23}/3$	−3	3	1	3	−3	−1	6	3	0
101100	1	(1,0,1/2,1,0,0)	1/2	$\sqrt{5}/2$	3/2	2	−2	0	2	−2	−2	4	2	0
100101	1	(1,0,1/3,1,0,1/3)	$\sqrt{5}/3$	$\sqrt{14}/3$	$\sqrt{23}/3$	3	−3	−1	3	−3	−1	5	2	−1
011011	3	(0,1,1/2,0,1,1)	1/2	$\sqrt{5}/2$	3/2	−2	2	0	−2	2	2	6	4	2
011010	3	(0,1,2/3,0,1,2/3)	$\sqrt{5}/3$	$\sqrt{14}/3$	$\sqrt{23}/3$	−3	3	1	−3	3	1	7	4	1
100010	4	(1,0,1/3,0,1,2/3)	$\sqrt{5}/3$	$\sqrt{14}/3$	$\sqrt{23}/3$	3	−3	−1	−3	3	1	6	3	0

Table 2
The learning process of MCCA for 6−bit circle problem by $a_1 = 1$, $a_2 = 1$

Input	Neuron	Center	w_1	w_2	w_3	w_4	w_5	w_6	t_1	t_2	t_3
100100	1	(1,0,0,1,0,0)	1	−1	−1	1	−1	−1	2	1	0
011100	2	(0,1,1,1,0,0)	−1	1	1	1	−1	−1	3	2	1
010011	3	(0,1,0,0,1,1)	−1	1	−1	−1	1	1	3	2	1
011010	Put aside.										
100010	Put aside.										
101011	4	(1,0,1,0,1,1)	1	−1	1	−1	1	1	4	3	2
100011	4	(1,0,1/2,0,1,1)	2	−2	0	−2	2	2	6	4	2
011101	2	(0,1,1,1,0,1/2)	−2	2	2	2	−2	0	6	4	2
010100	2	(0,1,2/3,1,0,1/3)	−3	3	1	3	−3	−1	6	5	4
101100	1	(1,0,1/2,1,0,0)	2	−2	0	2	−2	−2	4	2	0
100101	1	(1,0,1/3,1,0,1/3)	3	−3	−1	3	−3	−1	5	4	3
011011	3	(0,1,1/2,0,1,1)	−2	2	0	−2	2	2	6	4	2
011010	3	(0,1,2/3,0,1,2/3)	−3	3	1	−3	3	1	7	6	5
100010	4	(1,0,1/3,0,1,2/3)	3	−3	−1	−3	3	1	6	5	4

for these false vertices are all zeros. No hidden neuron is created in the modification process.

MCCA applies four hidden neurons by using both methods (as shown in Tables 1 and 2).

Given any input sequence of these 12 true vertices, applying MCCA results in the same neural network structure with four hidden neurons. From Tables 1 and 2 we observe that: given an arbitrary input sequence, FCLA results in a better neural network structure than ETL, which is the same as IETL does. The resultant neural network by ETL and IETL are not always satisfactory even after careful consideration of the core vertex and the input sequence. Given a proper core vertex and input sequence, the simplest neural network obtained by ETL has five hidden neurons. In addition, in IETL which part of vertices is considered as "do not care" is not clear. Constructive set covering learning algorithm (CSCLA) only represented a limited number of vertices (maximally $n+1$, where n is the input dimension) by a hidden neuron. The best solution of MCL and MCETL to this example requires four hidden neurons. But how to determine the number of initial cores should also be considered.

For different order of input sequences, MCCA constructs the similar neural network structure. For example, in solving this problem, MCCA always uses the same neural network structure with four hidden neurons given any input sequence of these twelve true vertices. To illustrate this advantage, we apply two arbitrary input sequences: one is {011011, 011010, 010011, 101011, 011101, 011100, 100011, 100010, 010100, 101100, 100101, 100100} used by Kim and Park [5], and the other is the sequence shown in Tables 1 and 2. If we define the precision as the ratio of the number of vertices correctly represented by the neural network and the total number of vertices in the data set, the comparison to CSCLA and ETL is shown in Figs. 11–14. Figs. 11 and 12 show the result for the first input sequence. We observe that MCCA uses the simplest neural network structure and the least number of hidden neurons (four hidden neurons) to solve the problems of *Circular region representation using 6-bit space quantization*.

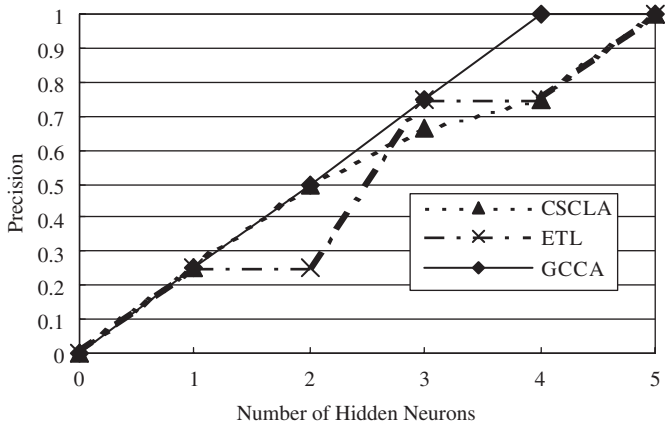


Fig. 11. Hidden neurons for sequence 1 of Example 1.

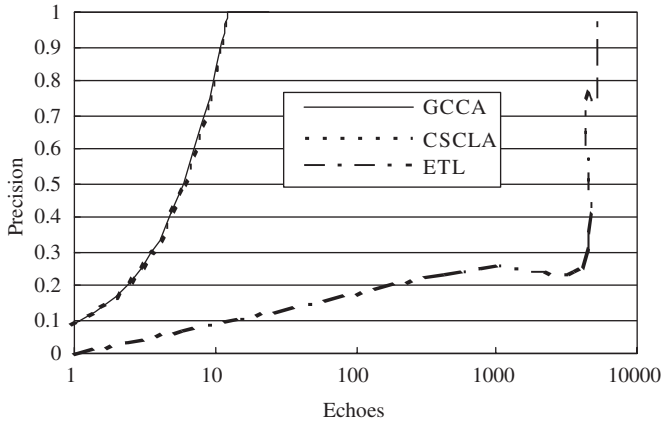


Fig. 12. Precision for sequence 1 of Example 1.

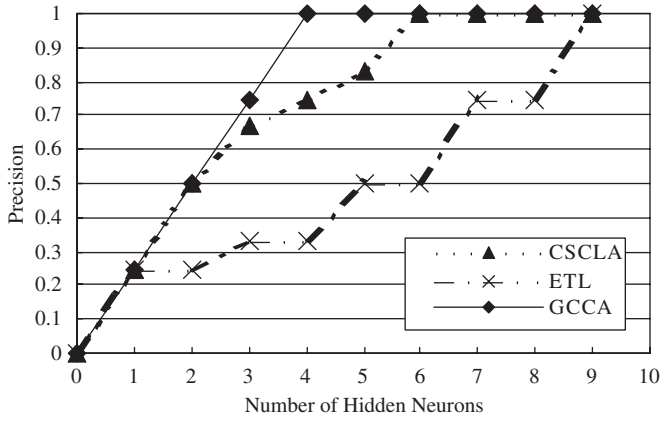


Fig. 13. Hidden neurons of sequence 2 of Example 1.

MCCA and CSCLA are much faster (150 times faster) than ETL. MCCA requires a similar number of echoes with (exactly the same as or slightly more than) CSCLA, while constructs simpler neural network structure (with less hidden neurons) than CSCLA does. Referring to the input sequence in Tables 1 and 2, we observe that MCCA also constructs simpler neural network (with four hidden neurons) than CSCLA (shown in Fig. 13) by using similar training time. MCCA applies fewer computing operations than ETL does in terms of $\sum_{i=1}^{n} w_i x_i$ (shown in Fig. 14).

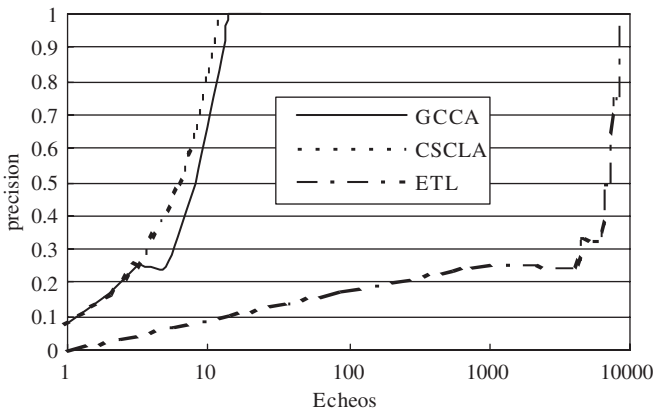


Fig. 14. Precision for sequence 2 of Example 1.

In summary, different input sequences result in quite different neural network structures by ETL and IETL. Given any input sequence, MCCA obtains the same neural network structure with four hidden neurons. In ETL and IETL, the core vertex and the input sequence must be carefully determined before training, but there is no effective rule to determine the core vertex and input sequence yet. So MCCA has the best performance out of all learning algorithms discussed above.

Example 1 does not need the modification process during training. To illustrate the modification process of MCCA, we look into the following example.

Example 2. Considering a 3-bit Boolean function, we want to separate the true subset {000, 010, 011, 111} from the false subset {001, 100, 101, 110}. We can begin with an arbitrary vertex in the true subset, and process an arbitrary sequence of true subsets or false subsets. We use the second method in MCCA by setting $\sigma_1 = \sigma_2 = 1$.

The learning process is shown in Table 3. The upper table shows the generalization process and the lower table shows the modification process.

In the generalization process, the first vertex, 111 causes to create hidden neuron 1; when 000 comes, it is located outside the boundary region 1, hence causes to create hidden neuron 2; when 010 comes, it is located in the claim region 2, hence causes the expansion of hypersphere 2; when 011 comes, it is located in the claim region 1, hence causes the expansion of hypersphere 1. All vertices in the true subset have been covered. In the modification process, the outputs of 001, 100, 101, 110 (false subset) are all zeros. So no modification hidden neuron is needed. Altogether 2 hidden neurons are needed.

If we consider a different sequence of the same training set {011, 000, 010,111}. We can gain the result as shown in Table 4 (analysis is omitted). In this case, 3 hidden neurons are needed.

Table 3
The learning process of sequence 1 for Example 2

Input	Hidden neuron	Center	$r1_j$	$r2_j$	$r3_j$	w_{1j}	w_{2j}	w_{3j}	$t1_j$	$t2_j$	$t3_j$
111	1	(1,1,1)	0	1	$\sqrt{2}$	1	1	1	3	2	1
000	2	(0,0,0)	0	1	$\sqrt{2}$	−1	−1	−1	0	−1	−2
010	2	(0,1/2,0)	1/2	$\sqrt{5}/2$	3/2	−2	0	−2	0	−4	$-4-2\sqrt{2}$
011	1	(1/2,1,1)	1/2	$\sqrt{5}/2$	3/2	0	2	2	4	0	$2\sqrt{2}$

Input	Model output	Hidden neuron	w_{1j}	w_{2j}	w_{3j}	$t1_j$
001	0 (ignore)					
100	0 (ignore)					
101	0 (ignore)					
110	0 (ignore)					

Table 4
The learning process of sequence 2 for Example 2

Input	Hidden neuron	Center	$r1_j$	$R2_j$	$r3_j$	w_{1j}	w_{2j}	w_{3j}	$t1_j$	$t2_j$	$t3_j$
011	1	(0,1,1)	0	1	$\sqrt{2}$	−1	1	1	2	1	0
000	Backup										
010	1	(0,1,1/2)	1/2	$\sqrt{5}/2$	3/2	−2	2	0	2	−2	$-2-2\sqrt{2}$
111	1	(1/3,1,2/3)	$\sqrt{5}/3$	$\sqrt{14}/3$	$\sqrt{23}/3$	−1	3	1	3	$-2\sqrt{5}$	$-3-2\sqrt{10}$
000	1	(1/4,3/4,1/2)	$\sqrt{14}/4$	$\sqrt{30}/4$	$\sqrt{46}/4$	−2	2	0	0	$-4-2\sqrt{14}$	$-8-4\sqrt{7}$

Input	Model output	Hidden neuron	w_{1j}	w_{2j}	w_{3j}	$t1_j$
001	1	2	−1	−1	1	1
100	0 (ignore)					
101	0 (ignore)					
110	1	3	1	1	−1	2

From Tables 3 and 4 we observe that: different input sequences result in similar neural network structures (in terms of similar number of hidden neurons). By using ETL and IETL different input sequences result in quite different neural network structures. The core vertex and the input sequence for ETL and IETL must be carefully determined before training. In ETL algorithm, three hidden neurons are needed for solving this example even after deep consideration, which is the same as the worst result of MCCA. The best solution of MCL and MCETL to this example requires two hidden neurons. But the number of initial cores has to be well determined.

Example 3. A given 7-bit function Now we consider the given 7-bit function [5]. We apply ETL, CSCLA and MCCA to two different input sequences. We take the first sequence $S_1 = \{0000000, 0000001, 0000010, 0000100, 0001000, 0010000, 0100000, 1000000, 0000011, 0000101, 0001001, 0010001, 0100001, 1000001, 0010101, 0100010, 0100100, 0101000, 0110000, 1000101, 1010001, 1100000, 1100101, 1000010, 1000100, 1010100, 1100100, 1100110, 1101010, 1110010, 1010000, 1001000, 1011101, 1110101, 1010101, 1010111\}$, and second sequence $S_2 = \{0100001, 1010101, 0010000, 0100010, 1010000, 1011101, 0000001, 1000000, 1100100, 0001001, 1101010, 1001000, 0000011, 0010101, 1110010, 1010111, 0001000, 1000010, 0000101, 0101000, 1110101, 0100000, 1010001, 0100100, 0010001, 1010100, 0000010, 0000000, 0110000, 0000100, 1000100, 1100110, 1100000, 1000101, 1000001, 1100101\}$. We choose the parameters $a_1 = 1$, and $a_2 = 2$ in our MCCA formulation by using the third method to control the geometrical expansion process. For the first sequence, the methods MCCA, CSCLA and ETL need 5, 22, and 7 hidden neurons respectively (Figs. 15 and 16). For the second sequence, MCCA, CSCLA and ETL use 7, 11 and 13 hidden neurons respectively (Figs. 17 and 18). We gain the same conclusion from this problem that MCCA and CSCLA are much faster than ETL. MCCA uses less hidden neurons than CSCLA and ETL do.

Example 4. Iris data set classification using MCCA [1,6] To illustrate the advantages of MCCA over traditional algorithms based on gradient descent technique, we now apply our algorithm MCCA to a classical data set, Iris data set for classification. There are 150 items in the data set with three target classes: Setosa, Versicolor and Verginica (50 items in each). Each of the items has four real-valued features. They are length of sepal, length of petal, width of sepal and width of petal.

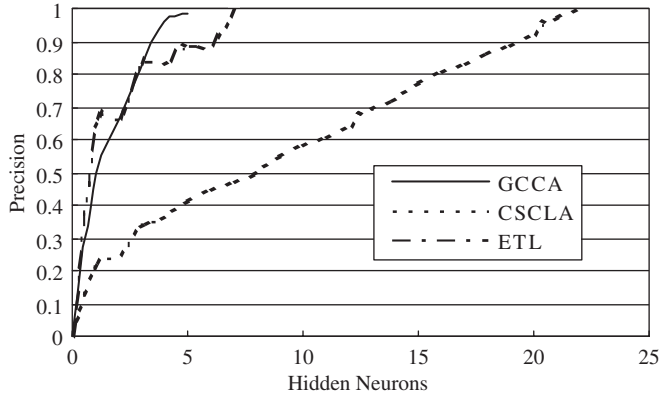


Fig. 15. Hidden neurons for sequence 1 of Example 3.

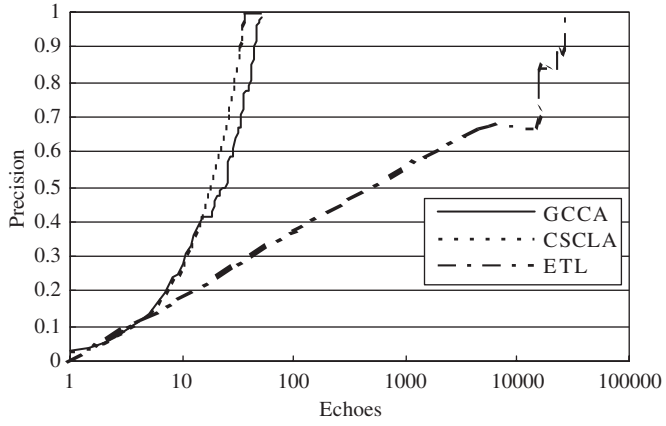


Fig. 16. Precision for sequence 1 of Example 1.

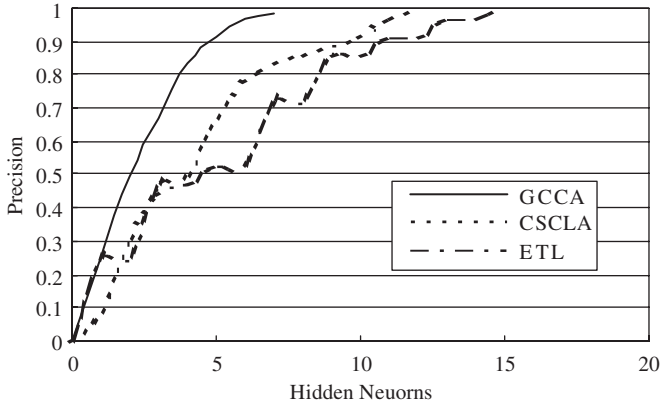


Fig. 17. Hidden neurons for sequence 2 of Example 3.

MCCA is designed for BNNs. We first represent these four features by 55 bits, which is 55 binary inputs. These 55 bits can represent the four numerical features without information losing. In order to retain the ordinal property of continuous variables, we use thermometrical encoding. For example, we use 000, 001, 011, 111 to represent 0, 1, 2, 3, respectively. We construct a neural network with one hidden layer and one output layer. There are two binary output neurons (y_1, y_2) in the output layer. The first output

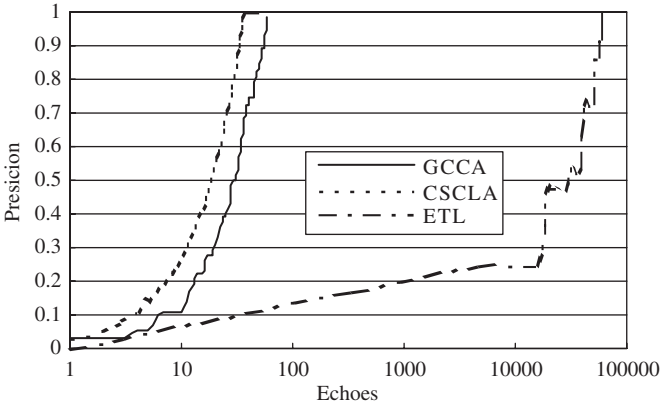


Fig. 18. Precision for sequence 2 of Example 1.

y_1 represents the class Setosa, and the second output y_2 represents the class Versicolor. Items neither belong to Setosa nor belong to Versicolor are considered as Verginica. Given an input, if the first output y_1 fires, then we do not consider the second output y_2 (in this case, $(y_1y_2)\in\{10,11\}$), and this item belongs to Setosa. If the first output y_1 does not fire and the second output y_2 fires (in this case, $(y_1y_2) = (01)$), then this item belongs to Versicolor. Else if both outputs do not fire (in this case, $(y_1y_2) = (00)$), then this item belongs to Verginica.

To demonstrate the features of MCCA, we consider two cases. In the training process of both cases, we apply the third method (a_1 and a_2) to control the geometrical expansion process of the hidden neurons. We apply (21), (30) and (31) for obtaining three threshold values by setting $a_1 = 2$ and $a_2 = 6$.

Case 1: The training data set contains 150 items (the whole Iris data set), and we use the same data set for testing. Table 5 compares the performance of MCCA with other training algorithms based on gradient descent technique. From the results of the second last column in Table 5, we conclude that MCCA uses less number of operations. It needs less training time, and results in zero error. MCCA has the best performance in terms of training speed (fifteen times to several hundred times faster than other algorithms).

Case 2. We randomly choose half of the iris data set (75 items) as the training data set, and let the remaining be the testing data set. From the results of the last column in Table 5, we conclude that MCCA needs a less number of operations. It requires less training time, and results in 2.6% error, which is much less than EBPM and LM do (the second and the third column in Table 5). Furthermore, in this case, half of the data set is used for training and the remaining is used for testing in our method. And the result (reported in the last column) refers to the training data set. But other methods use the whole dataset for training. Hence MCCA has more general interest for real world applications.

In MCCA, the Number of Epochs (NoE), the training time, and the error depend on different values of

parameters a_1 and a_2. Using iris data set and our formulation above, we give such a comparison in Table 6.

From Table 6, generally speaking, a large value of a_1 and a small value of a_2 cause fast learning but more errors, while a small value of a_1 and a large value of a_2 cause accurate learning with a little longer training time. With the increase of the training data set, the precision of both the training set and the testing set increases. We can always gain a 100% precision for the training data set given proper parameters of $a_1 = 2$ and $a_2 = 6$, so the modification process is not applied. We need no modification hidden neurons in solving this problem.

Now we use MCCA to solve a real world Boolean problem and illustrate the function of modification process during training by Example 5.

Example 5. We now apply MCCA to a data set including votes for each of the US House of Representatives Congressmen [URL-Data sets] on the 16 key votes identified by the CQA (Congressional Quarterly Almanac). Sixteen attributes are used for each person, all of which are Boolean values (yes or no). Each person belongs to either democrat or republican. There are 435 (267 democrats, 168 republicans) items in the data set. Many values in the data set are missing. Using these 16 attributes, we classify the items in the data set by party affiliation (democrat or republican). We use data without missing values for training, and the whole data set for testing. We apply generalization process by covering data whose party affiliation are republicans using fifteen hidden neurons. Five items are classified incorrectly because in the process of covering republicans by generalization, five items belonging to democrats are wrongly generalized. While we continue the training process by modifying the wrongly represented instances, four more hidden neurons (modification hidden neurons) are needed. MCCA can gain a 100% precision for the training set. The modification process also reduces errors for the testing set. From Table 7, we observe that MCCA is better than traditional training algorithms. MCCA increases the precision for both the training set and the testing set. The training time used by traditional algorithms was not available from the literatures. However, from previous examples, MCCA is faster (about one hundred times faster) than traditional algorithms. Hence, we do not discuss the training speed for this example.

5. Concluding remarks for MCCA

Modified constructive-covering algorithm (MCCA) for binary neural networks is based on multi-level geometrical expansion. Two processes are involved: the generalization process and the modification process. Three regions are defined by geometrical concept. The learning process is based on the judgment of which region the new coming vertex belongs to. These three regions (match, claim and boundary region) have the same center, but different radii. The match region and the claim region shift and expand in the training process until all vertices of the true (or false) subset are covered. Then to avoid over-generalization, modification process is designed to correct the errors

Table 5
Performance comparison for Iris data classification

	EBPM	LM	EBPT	Lee's (2003)	MCCA (150 training items)	MCCA (75 training items)
Number of Epochs (NoE)	10 776	1278	36 017	1584	2.17 (326 operations)	3.05 (229 operations)
Time (s)	90.8	16.2	287.4	23.6	1.0325	0.898
Mean Square Error (MSE)	0.2086	0.1428	9.84E–6	4.046E–6	0	0.026

Table 6
Parameters analysis for a_1 and a_2 in MCCA

a_1	a_2	Training data/ testing data	NoE	Training operations	Errors for training set	Errors for testing set	Training time
2	6	50/100	4.58	229	0	0.04	0.898
2	4	50/100	4.08	204	0	0.05	0.873
3	4	50/100	3.66	183	0.02	0.08	0.867
2	6	70/150	3.47	243	0	0	0.913

Table 7
Performance comparison for Example 5

	Previous result [3] (%)	MCCA (without modification process) (%)	MCCA (with modification process) (%)
Training Precision	90–95	97.8	100
Testing precision		95.1	96.6

introduced in the generalization process. Our construction process proceeds as follows. A hidden neuron first includes one vertex in it, and then "expands" using one of the three methods we have discussed in Section 3.2. The expansion process is characterized by a suitable choice of parameters such as ξ_1 and ξ_2, σ_1 and σ_2, or a_1, a_2, and "*gap*" depending on the method to control the geometrical expansion process. If we set these parameters to be large values to include vertices with a larger distance, the hidden neuron can expand easily, but a vertex with small distance will be wrongly included. We either introduce more errors (which is the cost we have to pay for having more generalization capability within the construction process of neural networks), or choose small values of parameters such that not many errors are introduced, but long training time is needed.

The main advantages of MCCA over existing algorithms are: MCCA possesses the capability of generalization, but all previous learning algorithms for BNNs (except for BLTA) do not. MCCA generalizes near vertices by expanding hyperspheres to include farther vertices. In MCCA, we do not need assign the core vertex in advance, which makes our algorithm to be applicable more generally. MCCA is faster because we need not search a lot of training pairs for determining each hidden neuron. What MCCA does is to judge which region an input belongs to. MCCA is not based on gradient descent technique either. MCCA constructs simpler neural networks as well as gains satisfactory performance (in terms of both training speed and accuracy). The last and the most important features of MCCA is that it is not sensitive to the change of input sequences of the same training set. In summary, MCCA has a better performance than traditional algorithms based on gradient descent technique in terms of both training speed (about one hundred times faster) and accuracy through computer simulations. MCCA is preferred to previous training algorithms for BNNs (BLTA, ETL, IETL, CSCLA, MCL and MCETL) as well.

References

[1] G. Arulampalam, A. Bouzerdoum, A generalized feedforward neural network classifier. Proceedings of International Joint Conference of Neural Networks (IJCNN~03) 1 (2003) 1429–1434.

[2] D.L. Gray, A.N. Michel, A training algorithm for binary feed forward neural networks, IEEE Trans. Neural Networks 3 (2) (1992) 176–194.

[3] www.ics.uci.edu/~mlearn/MLSummary.html.

[4] K.J. Kim, S.B. Cho, Fusion of structure adaptive self-organizing maps using fuzzy integral. Proceedings of International Joint Conference of Neural Networks (IJCNN~03) 1 (2003) 28–33.

[5] J.H. Kim, S.K. Park, The geometrical learning of binary neural networks, IEEE Trans. Neural Networks 6 (1) (1995) 237–247.

[6] J. Lee, Global optimization for fast multilayer perceptron training, Proceedings of International Joint Conference of Neural Networks (IJCNN~03) 1 (2003) 410–414.

[7] J.Y. Li, S.W. Luo, Y.J. Qi, Y.P. Huang, Numerical solution of elliptic partial differential equation by growing radial basis function neural networks, Proceedings of International Joint Conference of Neural Networks (IJCNN~03) 1 (2003) 85–90.

[8] X.M. Ma, Y.X. Yang, Z. Zhang, Research on the learning algorithm of binary neural networks, Chinese Journal of Computers 22 (9) (1999) 931–935.

[9] X.M. Ma, Y.X. Yang, Z.Z. Zhang, Constructive learning of binary neural networks and its application to nonlinear register synthesis, Proceedings of International Conference on Neural Information Processing—ICONIP'01 1 (2001) 90–95.

[10] J.E. Meng, G. Yang, Online adaptive fuzzy neural identification and control of a class of MIMO nonlinear systems, IEEE Transactions on Fuzzy Systems 11 (4) (2003) 462–477.

[11] K. Okamoto, S. Ozawa, S. Abe, A fast incremental learning algorithm of RBF neural networks with long-tern memory, Proceedings of International Joint Conference of Neural Networks (IJCNN~03) 1 (2003) 102–107.

[12] H. Ressom, D. Wand, P. Nataradan, Adaptive double self-organization map and its application in clustering gene expression data, Proceedings of International Joint Conference of Neural Networks (IJCNN~03 1 (2003) 39–44.

[13] S.K. Sung, J.W. Jung, J.T. Lee, W.J. Choi, Opitonal synthesis method for binary neural network using NETLA, Lecture Notes in Artificial Intelligence (LNAI) 2275 (2002) 236–244.

[14] D. Wang, N.S. Chaudhari, A multi-core learning algorithm for binary neural networks, Proceedings of International Joint Conference on Neural Networks (IJCNN2003) 1 (2003) 450–455.

[15] D. Wang, N.S. Chaudhari, Binary neural network training algorithms based on linear sequential learning, International Journal of Neural Systems 13 (5) (2003) 333–351.

[16] A. Yamamoto, T. Saito, An improved expand-and-truncate learning, Proceedings of the IEEE International Conference on Neural Networks (ICNN) 2 (1997) 1111–1116.

Di Wang received her Bachelor and Ph.D. degree form Tianjin University, China and Nanyang Technological University (NTU), Singapore in 2001 and 2005, respectively. She is currently with School of Informatics, University of Manchester as a postdoctoral research associate. Her research interests include neural networks, fuzzy systems, hierarchical fuzzy neural net systems and their applications in cognitive science.

Narendra S. Chaudhari completed his undergraduate, graduate, and doctoral studies at Indian Institute of Technology (IIT), Mumbai, India, in 1981, 1983, and 1988 respectively. After his Bachelor's (B.Tech.) degree (with distinction) in Electrical Engineering., he was with Electronic Controls Division of Larsen and Toubro Ltd, Mumbai (India). After his masters (M.Tech.) and doctoral (Ph.D.) studies in Computer Science and Engineering, he was involved in graduate level training for the defense scientists (Ministry of Defense, Government of India) in the area of Computer Science till 1998. Later, he was on a visiting assignment in Southern Cross University, NSW (Australia), and on another visiting assignment with Freie Universitat, Berlin. Currently, he is currently with School of Computer Engineering, Nanyang Technological University, Singapore as an associate professor.