Task Decomposition and Module Combination Based on Class Relations: A Modular Neural Network for Pattern Classification

Bao-Liang Lu and Masami Ito

***Abstract*— In this paper, we propose a new method for decomposing pattern classification problems based on the class relations among training data. By using this method, we can divide a K-class classification problem into a series of $\binom{K}{2}$ two-class problems. These two-class problems are to discriminate class $\mathcal{C}_i$ from class $\mathcal{C}_j$ for $i = 1, \cdots, K$ and $j = i+1$, while the existence of the training data belonging to the other $K-2$ classes is ignored. If the two-class problem of discriminating class $\mathcal{C}_i$ from class $\mathcal{C}_j$ is still hard to be learned, we can further break down it into a set of two-class subproblems as small as we expect. Since each of the two-class problems can be treated as a completely separate classification problem with the proposed learning framework, all of the two-class problems can be learned in parallel. We also propose two module combination principles which give practical guidelines in integrating individual trained network modules. After learning of each of the two-class problems with a network module, we can easily integrate all of the trained modules into a min–max modular (M^3) network according to the module combination principles and obtain a solution to the original problem. Consequently, a large-scale and complex K-class classification problem can be solved effortlessly and efficiently by learning a series of smaller and simpler two-class problems in parallel.**

***Index Terms*—Min–max modular neural network, module combination, parallel learning, pattern classification, task decomposition.**

I. Introduction

ONE OF THE most important difficulties in using artificial neural networks for solving large-scale real-world problems is how to divide a problem into smaller and simpler subproblems; how to assign a network module to learn each of the subproblems; and how to recombine the individual modules into a solution to the original problem. In the last several years, many researchers have studied modular neural-network learning approaches to dealing with this problem, for example see [4], [6], [13], [18], [25], [30]. Up to now, various task decomposition methods have been developed based on the divide-and-conquer technique [7]. These methods can be roughly classified into three classes as follows.

A. Explicit Decomposition

Before learning, a problem is divided into a set of subproblems by the designer who should have some domain knowledge and deep prior knowledge concerning the decomposition of the problem [11]. Several modular neural-network systems have been developed based on this decomposition method, see for instance [15], [29]. The limitation of this method is that sufficient prior knowledge concerning the problems is necessary.

B. Class Decomposition

Before learning, a problem is broken down into a set of subproblems according to the inherent class relations among training data [2], [6], [12]. In contrast to the explicit decomposition, this method requires only some common knowledge concerning the class relations among training data. According to this method, a K-class problem is divided into K two-class problems by using the class relations. The number of training data for each of the two-class problems is the same as the original K-class problem.

C. Automatic Decomposition

A problem is decomposed into a set of subproblems with the progressing of learning. Most of the existing decomposition methods fall into this category: for instance, the mixture of experts architecture [13], [14] and the multisieving neural network [18]–[20]. From computational complexity's point of view, the former two methods are more efficient than this one because a problem has been decomposed into subproblems before learning. Therefore, they are suitable for solving large-scale and complex problems. The advantage of this method is that it is more general than the former ones because it can work when any prior knowledge concerning the problem is absent.

In this paper, we address K-class classification problems, where each input vector belongs to exactly one of K classes represented by $\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_K$, respectively. We propose a new decomposition method for dividing a K-class problem into a set of relatively smaller and simpler two-class problems. The central idea underlying this method is to use the class relations among training data [21], which is similar to the class decomposition method mentioned above. Our method has two important advantages over the class decomposition method as follows.

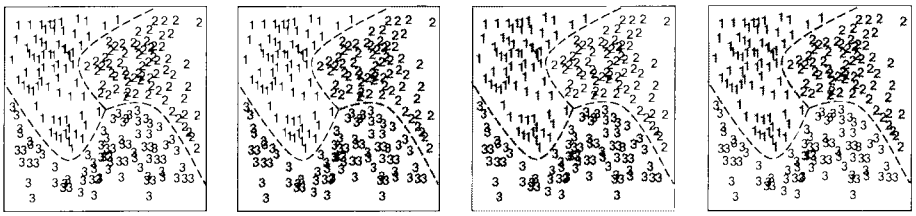
1) The two-class problem obtained by our method is to discriminate class $\mathcal{C}_i$ from class $\mathcal{C}_j$ for $i = 1, \cdots, K$ and $j = i + 1$, while the existence of the training data of the other $K - 2$ classes is ignored. Therefore, the number of training data for each of the two-class problems is $2 \times N$, which is independent of the number of classes K. Here, for simplicity of description, the assumption we made is that each of the classes has the

Manuscript received April 15, 1998; revised March 26, 1999 and May 28, 1999.

B.-L. Lu is with the Laboratory for Brain-Operative Device, RIKEN Brain Science Institute, Wako-shi, Saitama, 351-0198, Japan.

M. Ito, deceased, was with Bio-Mimetic Control Research Center, the Institute of Physical and Chemical Research (RIKEN), Shimoshidami, Moriyama-ku, Nagoya, 463-0003, Japan.

Publisher Item Identifier S 1045-9227(99)07235-5.



(a) (b) (c) (d)

Fig. 1. Illustration of (a) a three-class problem and (b), (c), and (d) the related three two-class problems. In (b), (c), and (d), the desired outputs corresponding to the inputs in the shadow areas should be ϵ. Otherwise, $1-\epsilon$. The three classes are represented as "1," "2," and "3." The dashed lines denote desirable boundaries among the classes. This notation will be also used in Fig. 2.

same number of training data N. However, the two-class problem obtained by the class decomposition method has to discriminate between one class and the rest classes. Therefore, the number of training data for each of the two-class problems is $K \times N$, which is the same as the original K-class problem.

2) If the two-class problem of discriminating class $\mathcal{C}_i$ from class $\mathcal{C}_j$ is still hard to be learned, it can be further divided into a number of two-class subproblems as small as the user expects by using the decomposition method suggested here. However, the class decomposition method can not be applied to decomposing two-class problems.

We also propose two module combination principles which give practical guidelines in integrating the individual trained modules. After learning of each of the two-class problem with a network module, we can easily integrate all of the trained modules into a min–max modular (M^3) network according to the proposed module combination principles and obtain a solution to the original problem. Consequently, a large-scale and complex K-class problem can be solved effortlessly and efficiently by learning a series of relatively smaller and simpler two-class problems in parallel.

The remainder of this paper is organized as follows. In Section II, we present a new decomposition method. In Section III, we give two module combination principles and a new modular neural-network architecture. Section IV presents several examples and simulation results. Section V mentions the related work and compares the proposed modular network with other models. Finally, conclusions are given in Section VI.

II. Task Decomposition

The decomposition of a large-scale and complex problem into smaller and simpler subproblems is the first step to implement modular neural-network learning. In this section, we present a new method for decomposing a K-class classification problem into a series of smaller and simpler two-class problems.

A. Decomposition of K-Class Problems

Suppose that grandmother cells are used as output representation, in which K output units can only represent $K+1$ classes of patterns at most, and one and only one output unit is active at a time [3]. Let $\mathcal{T}$ be the training set for a K-class problem

$$\mathcal{T} = \{(X_l, Y_l)\}_{l=1}^{L} \tag{1}$$

where $X_l \in \mathbf{R}^d$ is the input vector, $Y_l \in \mathbf{R}^K$ is the desired output, and L is the number of training data.

Following the class decomposition method [2], [6], [12], a K-class problem can be divided into K two-class problems. The training set for each of the two-class problems is defined by

$$\mathcal{T}_i = \left\{(X_l, y_l^{(i)})\right\}_{l=1}^{L}, \qquad \text{for} \quad i = 1, \cdots, K \tag{2}$$

where $y_l^{(i)} \in \mathbf{R}^1$ is the desired output which is defined by

$$y_l^{(i)} = \begin{cases} 1-\epsilon, & \text{if } X_l \text{ belongs to class } \mathcal{C}_i \\ \epsilon, & \text{if } X_l \text{ belongs to class } \overline{\mathcal{C}}_i \end{cases} \tag{3}$$

where ϵ is a small positive real number, $\overline{\mathcal{C}}_i$ denotes all the classes except $\mathcal{C}_i$. It should be noted that the number of training data for each of the two-class problems given by (2) is the same as the original K-class problem. Fig. 1 illustrates a three-class classification problem and its three partitions, i.e., three two-class problems obtained by the class decomposition method [2], [6], [12].

If a K-class problem is a large and complex problem (e.g., K is a large number and there are a large number of training data for each of the classes), to learn the two-class problems defined by (2) may be still intractable. One may ask: whether can the two-class problems be further decomposed into relatively smaller and simpler two-class problems? We will give an answer to this question in the remainder of this section.

B. Decomposition of Two-Class Problems

By using the class relations provided by training set $\mathcal{T}$, the input vectors can be easily partitioned into K subsets in the form

$$\mathcal{X}_i = \left\{X_l^{(i)}\right\}_{l=1}^{L_i}, \qquad \text{for} \quad i = 1, 2, \cdots, K \tag{4}$$

where L_i is the number of data of $\mathcal{X}_i$, all of $X_l^{(i)} \in \mathcal{X}_i$ have the same desired outputs, and $\sum_{i=1}^{K} L_i = L$. Note that this partition is unique.

We suggest that each of the two-class problems defined by (2) can be further divided into $K-1$ relatively smaller

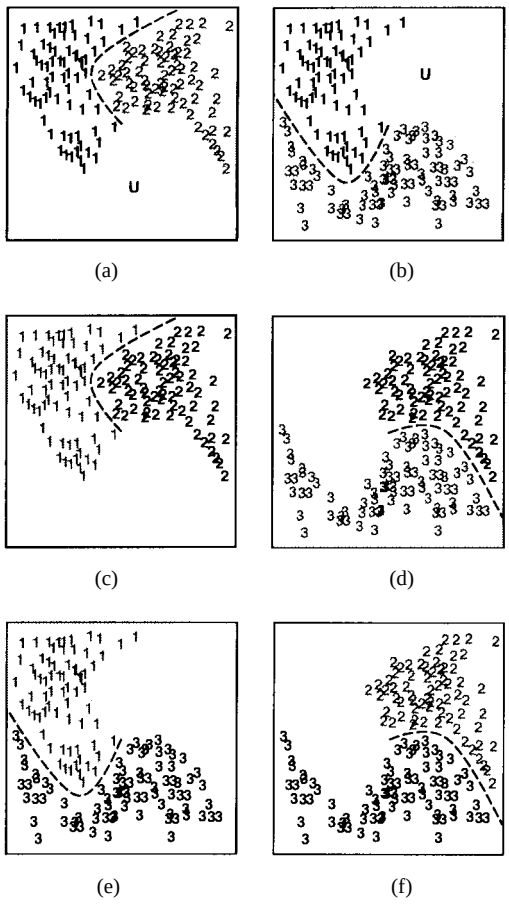


Fig. 2. Illustration of six relatively smaller two-class problems obtained by partitioning the two-class problem shown in Fig. 1(b), (c), and (d) according to the proposed decomposition method.

two-class problems. The training set for each of the smaller two-class problems is given by

$$\mathcal{T}_{ij} = \left\{(X_l^{(i)}, 1-\epsilon)\right\}_{l=1}^{L_i} \cup \left\{(X_l^{(j)}, \epsilon)\right\}_{l=1}^{L_j} \quad \text{for} \quad i, j = 1, \cdots, K \quad \text{and} \quad j \neq i \tag{5}$$

where $X_l^{(i)} \in \mathcal{X}_i$ and $X_l^{(j)} \in \mathcal{X}_j$ are the input vectors belonging to class $\mathcal{C}_i$ and class $\mathcal{C}_j$, respectively. It is important to emphasize that the two-class problem defined by (5) is to discriminate between class $\mathcal{C}_i$ and class $\mathcal{C}_j$ for $i, j = 1, \cdots, K$ and $i \neq j$, and the existence of the training data of the other $K-2$ classes is ignored. Obviously, this two-class problem is much smaller than that defined by (2) if K is large and the number of data for each of the K-classes is roughly equal. Fig. 2 illustrates the partition of the three two-class problems depicted in Fig. 1(b), (c), and (d) into six relatively smaller two-class problems by using the proposed task decomposition method.

From (5), we see that partitioning each of the two-class problems into $K-1$ smaller two-class problem is simple and straightforward, and no domain specialists or a prior knowledge concerning the decomposition of the learning problems are required. Consequently, any user can easily perform this decomposition if he or she knows the number of training data belonging to each of the classes.

C. Fine Decomposition of Two-Class Problems

Even though a K-class problem can be broken down into $K \times (K-1)$ relatively smaller two-class problems, some of the two-class problems may be still hard to be learned. For example, the well-known "two-spirals" problem is a two-class problem, but it is hard to be learned by plain three-layer perceptrons trained by the standard backpropagation algorithm [16]. In order to deal with this problem, we suggest that the two-class problem $\mathcal{T}_{ij}$ defined by (5) can be further decomposed into a number of two-class subproblems as small as the user expects according to the class relations among training data.

Assume that the input set $\mathcal{X}_i$ defined by (4) is partitioned into $N_i\,(1 \leq N_i \leq L_i)$ subsets in the form

$$\mathcal{X}_{ij} = \left\{X_l^{(ij)}\right\}_{l=1}^{L_i^{(j)}}, \qquad \text{for} \quad j = 1, \cdots, N_i \tag{6}$$

where $L_i^{(j)}$ is the number of data of $\mathcal{X}_{ij}$, and $\cup_{j=1}^{N_i} \mathcal{X}_{ij} = \mathcal{X}_i$. This partition is not unique in general. One may give a partition randomly or by using a prior knowledge concerning the learning problems. We will discuss two different methods for implementing this partition and compare their performance in Section IV in detail.

After dividing the training input set $\mathcal{X}_i$ into N_i subsets $\mathcal{X}_{ij}$ (6), the training set for each of the smaller and simpler two-class problems can be given by

$$\mathcal{T}_{ij}^{(u,v)} = \left\{(X_l^{(iu)}, 1-\epsilon)\right\}_{l=1}^{L_i^{(u)}} \cup \left\{(X_l^{(jv)}, \epsilon)\right\}_{l=1}^{L_j^{(v)}} \quad \text{for} \quad u = 1, \cdots, N_i,\ v = 1, \cdots, N_j \quad i, j = 1, \cdots, K, \quad \text{and} \quad j \neq i \tag{7}$$

where $X_l^{(iu)} \in \mathcal{X}_{iu}$ and $X_l^{(jv)} \in \mathcal{X}_{jv}$ are the input vectors belonging to class $\mathcal{C}_i$ and class $\mathcal{C}_j$, respectively. If the training set $\mathcal{T}_{ij}^{(u,v)}$ has only two different elements in the form

$$\mathcal{T}_{ij}^{(u,v)} = \left\{(X_l^{(iu)}, 1-\epsilon) \cup (X_l^{(jv)}, \epsilon)\right\} \quad \text{for} \quad u = 1, \cdots, L_i,\ v = 1, \cdots, L_j \quad i, j = 1, \cdots, K, \quad \text{and} \quad j \neq i \tag{8}$$

this training set is obviously a linearly separable problem because any two different training patterns can always be separated by a hyper-plane. In Section IV, we will demonstrate how to decompose the XOR problem into four linearly separable problems according to (8).

III. Min–Max Modular Network

In order to implement modular learning, we have to deal with two key problems. One is how to decompose a complex learning problem into a number of independent smaller and simpler subproblems. The other is how to recombine individual trained modules into a solution to the original problem. In this section we first introduce three integrating units. Then we present two module combination principles and discuss how to reduce the number of learning problems. Finally, we present a new hierarchical, parallel, and modular neural-network architecture called the *min–max modular* network.

A. Three Integrating Units

We introduce three integrating units, namely MIN, MAX, and INV, respectively, which are the elements of connecting individual trained modules.

The basic function of an MIN unit is to find a minimum value from its multiple inputs. The transfer function of the MIN unit is given by

$$q = \min(p_1, \cdots, p_n) \tag{9}$$

where $p_1, \cdots, p_n$ are the inputs and q is the output which is the smallest one among the inputs, $p_i \in \mathbf{R}^1$ for $i = 1, \cdots, n$, and $q \in \mathbf{R}^1$.

The basic function of an MAX unit is to find a maximum value from its multiple inputs. The transfer function of the MAX unit is given by

$$q = \max(p_1, \cdots, p_n) \tag{10}$$

where $p_1, \cdots, p_n$ are the inputs and q is the output which is the largest one among the inputs.

The basic function of an INV unit is to invert its single input. The transfer function of the INV unit is given by

$$q = b - p \tag{11}$$

where b, p, and q are the upper limit of its input value, input, and output, respectively.

From the definitions of the three integrating units mentioned above, we can see that the MIN, MAX, and INV units are similar to the logical AND, OR, and NOT gates, respectively. The essential difference between the integrating units and the logical gates is that both the inputs and outputs of the integrating units are real continuous values, instead of binary values.

B. The Principles of Module Combination

According to (7), a K-class classification problem can be divided into

$$\sum_{i=1}^{K} \sum_{\substack{j=1 \\ j \neq i}}^{K} N_i \times N_j \tag{12}$$

smaller and simpler two-class problems. Suppose that each of the two-class problems has been learned by a network module completely. One may ask a question: how to recombine the outputs of the individual trained modules into a solution to the original problem? In this section, we will present two module combination principles which give the user a systematic way of integrating the individual trained modules.

1) Minimization Principle: The modules, which were trained on the data sets which have the same training inputs corresponding to desired outputs $1 - \epsilon$, should be integrated by the MIN unit.

Let us give an explanation of the minimization principle through the two-class classification problems depicted in Fig. 2(a) and (b). Suppose that the two-class problems had been learned by two modules $\mathbf{M}_{12}$ and $\mathbf{M}_{13}$, respectively. How can the outputs of the two individual modules be recombined into a solution to the original two-class problem depicted in Fig. 1(b)? A simple way to deal with this problem is to find minimum output values from the two individual modules. The reason for performing this *minimization* operation is that the training inputs corresponding to desired outputs $1 - \epsilon$ are the same in the two problems, while the difference between them is only the training inputs whose desired outputs are ϵ. For example, if the regions represented as "U" in Fig. 2(a) and (b) are classified as class "1" by the two trained modules, i.e., high responses are generated by the two modules when the inputs from the regions are presented, these classification are correct from individual modules' point view because there exists no training data in the regions. But, from the original problem's point of view, these classifications are incorrect since the regions in the original problem should correspond to low response [see Fig. 1(b)]. The combination of the outputs of the two modules through the MIN unit gives low response in both the regions. From this example, we see that the MIN unit makes the proper decision region of each module active and the incorrect decision region of each module prohibitive. In fact, the MIN unit implements a competition among the individual modules. The winner is the module whose output is the lowest. Although this example is simple, it illustrates the essential concepts.

2) Maximization Principle: The modules, which were trained on the data sets which have the same training inputs corresponding to desired outputs ϵ, should be integrated by the MAX unit.

Consider the combination of the individual network modules which were trained on the following $N_i \times N_j$ two-class problems defined by (7):

$$\begin{array}{cccc} \mathcal{T}_{ij}^{(1,1)} & \mathcal{T}_{ij}^{(1,2)} & \cdots & \mathcal{T}_{ij}^{(1,N_j)} \\ \mathcal{T}_{ij}^{(2,1)} & \mathcal{T}_{ij}^{(2,2)} & \cdots & \mathcal{T}_{ij}^{(2,N_j)} \\ \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots \\ \mathcal{T}_{ij}^{(N_i,1)} & \mathcal{T}_{ij}^{(N_i,2)} & \cdots & \mathcal{T}_{ij}^{(N_i,N_j)}. \end{array} \tag{13}$$

From the definition of the above two-class problems, the N_j training sets in each row of (13) have the same training inputs corresponding to the desired outputs $1 - \epsilon$. In contrast, the N_i training sets in each column of (13) have the same training inputs corresponding to the desired outputs ϵ. According to the minimization and maximization principles, the $N_i \times N_j$ modules that were trained on the $N_i \times N_j$ two-class problems can be integrated into an M^3 network as illustrated in Fig. 3.

C. Reduction of the Number of Learning Problems

From (5), we see that a K-class problem can be broken down into $K \times (K-1)$ two-class problems. In fact, among them, only $\binom{K}{2}$ two-class problems are different, and other $\binom{K}{2}$ ones can be replaced by the inverses of the former ones. Therefore, the number of two-class problems that need to be learned can be reduced to $\binom{K}{2}$. For example, the problem $\mathcal{T}_{12}$ depicted in Fig. 2(a) is the same as $\mathcal{T}_{21}$ depicted in Fig. 2(c) from pattern classification's point of view. The difference between them is only their desired outputs. Suppose that $\mathcal{T}_{12}$ has been learned by a network module $\mathbf{M}_{12}$ correctly. If we need to learn $\mathcal{T}_{21}$, we can get the solution by using the inverse

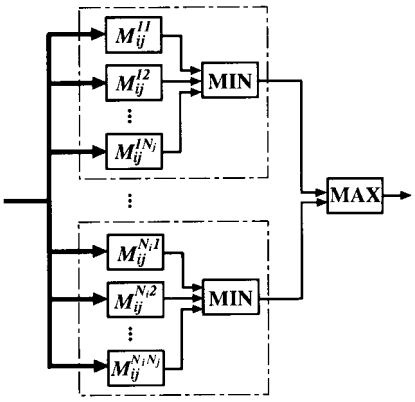


Fig. 3. The M^3 network consisting of $N_i \times N_j$ individual network modules, N_i MIN units, and one MAX unit.

of the output of $\mathbf{M}_{12}$, instead of training a new network module on $\mathcal{T}_{21}$. According to the above discussion, the number of two-class problems as given by (12) can be reduced to

$$\sum_{i=1}^{K} \sum_{j=i+1}^{K} N_i \times N_j. \tag{14}$$

Fig. 13 illustrates the proposed min–max modular network for solving a four-class classification problem, in which $\binom{4}{2}$ INV units are used to invert the outputs of $\mathbf{M}_{12}$, $\mathbf{M}_{13}$, $\mathbf{M}_{14}$, $\mathbf{M}_{23}$, $\mathbf{M}_{24}$, and $\mathbf{M}_{34}$, respectively.

D. Min–Max Modular Network

After training of each of the modules which were assigned to learn associated subproblems, all of the individual trained modules can be easily integrated into an M^3 network by using the MIN, MAX, or/and INV units according to the proposed module combination principles. Let y denote the actual output vector of the whole M^3 network for a K-class classification problem, let $g(x)$ denote the transfer function of the M^3 network. We may then write

$$y = g(x) = [g_i(x), \cdots, g_K(x)]^T \tag{15}$$

where $y \in \mathbf{R}^K$, and $g_i(x)$ is called the *discriminant function*, which discriminates the patterns of class $\mathcal{C}_i$ from those of the rest classes. The M^3 network is said to assign an input x to class $\mathcal{C}_i$ if

$$|g_i(x) - (1-\epsilon)| \le \delta \quad \text{and} \quad |g_j(x) - \epsilon| < \delta \quad \text{for} \quad j \ne i \tag{16}$$

where $1-\epsilon$ and ϵ denote the high and low desired outputs, respectively, δ is a real number, which denotes the error tolerance. For example, ϵ and δ are set to 0.01 and 0.5 in the simulations of this paper.

In the following, we describe the discriminant functions $g_i(x)$ of two kinds of M^3 networks: i) no INV unit is involved in integrating individual trained modules and ii) the INV units are involved in module combination.

1) No INV Unit: The discriminant function $g_i(x)$ of the M^3 network which is constructed to learn the $K \times (K-1)$ two-class problems can be given by

$$g_i(x) = \min_{\substack{j=1 \\ j \ne i}}^{K} h_{ij}(x) \tag{17}$$

where $h_{ij}(x)$ is the activation function of the module $\mathbf{M}_{ij}$ trained on $\mathcal{T}_{ij}$ (5).

In a similar way, the discriminant function $g_i(x)$ of the M^3 network which is constructed to learn

$$\sum_{i=1}^{K} \sum_{\substack{j=1 \\ j \ne i}}^{K} N_i \times N_j$$

two-class problems can be expressed as

$$g_i(x) = \min_{\substack{j=1 \\ j \ne i}}^{K} \left[\max_{k=1}^{N_i} \left[\min_{l=1}^{N_j} h_{ij}^{(k,l)}(x) \right] \right] \tag{18}$$

where $h_{ij}^{(k,l)}$ is the activation function of the module $\mathbf{M}_{ij}^{kl}$ trained on $\mathcal{T}_{ij}^{(k,l)}$ (7). It should be noted that $\max_{k=1}^{N_i}[\min_{l=1}^{N_j} h_{ij}^{(k,l)}(x)]$ is exactly equivalent to $h_{ij}(x)$ if $N_i = N_j = 1$.

2) Involving INV Units: By replacing the module $\mathbf{M}_{st}$ for $s > t$ with the inverse of the module $\mathbf{M}_{ts}$, the discriminant functions $g_i(x)$ defined by (17) and (18) can be restated as

$$g_i(x) = \min \left[\min_{j=i+1}^{K} h_{ij}(x), \min_{r=1}^{i-1} (b - h_{ri}(x)) \right] \tag{19}$$

and

$$g_i(x) = \min \left[\min_{j=i+1}^{K} \left[\max_{k=1}^{N_i} \left[\min_{l=1}^{N_j} h_{ij}^{(k,l)}(x) \right] \right] \min_{r=1}^{i-1} \left(b - \max_{k=1}^{N_r} \left[\min_{l=1}^{N_i} h_{ri}^{(k,l)}(x) \right] \right) \right] \tag{20}$$

respectively, where the terms $b - h_{ri}(x)$ and $b - \max_{k=1}^{N_r}[\min_{l=1}^{N_i} h_{ri}^{(k,l)}(x)]$ denote the inverses of $h_{ri}(x)$ and $\max_{k=1}^{N_r}[\min_{l=1}^{N_i} h_{ri}^{(k,l)}(x)]$, respectively, which can be implemented by the INV units, b denotes the upper limit of the output value of each module. For example, b is set to one in all of the following simulations because the standard sigmoidal activation function is used.

IV. Examples and Simulations

In this section, six examples are presented. The first one is used to illustrate how to decompose a linearly nonseparable problem into a number of linearly separable problems. The second one is used to demonstrate how to divide a complex two-class problem into a number of smaller and simpler two-class problems according to two different partition techniques. The third and fourth ones are simulated to examine the generalization performance of the proposed M^3 network for solving real multiclass classification problems. In the fifth example, the method for randomly dividing a problem into a number of subproblems is examined on its effects on training time and generalization performance. The last one is used to demonstrate the classification power and effectiveness of the

TABLE I
PERFORMANCE COMPARISON OF SINGLE MLQP'S AND THE PROPOSED M^3 NETWORKS. EACH RESULT IS THE AVERAGE OF THREE SIMULATIONS

Task	Network	♯ Modules	CPU time		Success rate (%)	
			Max.	Total	Training	Test
Spiral	Single	1	105447	105447	99.48	72.57
	Modular (ran.)	9	1495	7888	100.00	89.46
	Modular	9	5518	5994	100.00	98.00
	Modular	36	648	1439	100.00	98.11
Vehicle	Single	1	134971	134971	99.76	72.34
	Modular	6	3456	4567	100.00	73.05

TABLE II
PERFORMANCE COMPARISON OF SINGLE MLP'S AND THE PROPOSED M^3 NETWORKS ON THE IMAGE SEGMENTATION PROBLEM. VALUES ARE MEAN (TOP ROW) AND STANDARD DERIVATIONS (BOTTOM ROW) OVER TEN SIMULATIONS

Network	Total CPU	Success rates (%)	
	time (sec.)	Training	Test
Single (h=18)	5647	99.81	89.00
	413	0.23	0.78
Single (h=33)	20538	99.52	89.33
	0	0.0	0.78
M^3 (h=1)	15.02	100	91.29
	0.54	0.0	0.11
M^3 (h=2)	20.60	100	91.22
	2.33	0.0	0.26

TABLE III
PERFORMANCE COMPARISON OF FOUR DIFFERENT RANDOM PARTITIONS. VALUES ARE MEAN (TOP ROW) AND STANDARD DEVIATIONS (BOTTOM ROW) OVER TEN SIMULATIONS

♯ Subtasks	♯ Max.	♯ Total	Total CPU	Successfully rates (%)	
(Modules)	Training data	weights	time (sec.)	Training	Test
20	506	18220	10449	100.0	94.13
			2119	0.0	0.12
51	317	27897	1998	100.0	93.95
			362	0.0	0.19
125	203	45625	772	100.0	93.86
			147	0.0	0.17
489	115	89487	515	100.0	93.51
			151	0.0	0.11

TABLE IV
PERFORMANCE COMPARISON OF PLAIN MLP, CLASS-SENSITIVE NEURAL NETWORK, AND THE PROPOSED M^3 NETWORK ON THE SHUTTLE DATA SET. EACH RESULT IS THE AVERAGE OF THREE SIMULATIONS

	♯ Modules	CPU Time (hour)		Success rate (%)	
		Max.	Total	Training	Test
Single MLP (no converge)	1	1111.1	1111.1	0	0
CSNN	7	161.2	1128.4	95.54	96.90
M^3 network	90	5.8	163.2	99.85	99.86

proposed M^3 network for solving large-scale and complex problems.

In the following simulations, the structures of all the single and modular networks are chosen to be multilayer perceptrons (MLP's) with one hidden layer or multilayer quadratic perceptrons (MLQP's) [17] with one hidden layer. All of the single and modular networks are trained by the standard backpropagation algorithm [27] or the modified backpropagation algorithm [1]. The momentums are set all to 0.9. The learning rates are selected through practical experiments. A summary of the simulation results is shown in Tables I–IV, where "Max" means the maximum CPU time required to train any network modules. All of the simulations were performed on a SUN Ultra2 workstation.

A. XOR Problem

It is known that the XOR problem is a linearly nonseparable problem. The four training inputs for the XOR problem are depicted in Fig. 4(a). According to the proposed decomposition method (8), the XOR problem was divided into four linearly separable problems: $\mathcal{T}^{(1,1)}$, $\mathcal{T}^{(1,2)}$, $\mathcal{T}^{(2,1)}$, and $\mathcal{T}^{(2,2)}$, which are depicted in Fig. 4(b)–(e), respectively. Four perceptrons represented as $\mathbf{P}_{11}$, $\mathbf{P}_{12}$, $\mathbf{P}_{21}$, and $\mathbf{P}_{22}$ were selected to learn $\mathcal{T}^{(1,1)}$, $\mathcal{T}^{(1,2)}$, $\mathcal{T}^{(2,1)}$, and $\mathcal{T}^{(2,2)}$, respectively. Each of the perceptrons was trained by the traditional perceptron learning algorithm [24]. The four trained perceptrons were integrated into an M^3 network as shown in Fig. 5 according to the module combination principles. The responses of the four perceptrons, their combinations, and the entire M^3 network are shown in Fig. 6(a)–(g), respectively. Comparing Fig. 4(a) with Fig. 6(g), we can see that the M^3 network recognizes the XOR problem correctly.

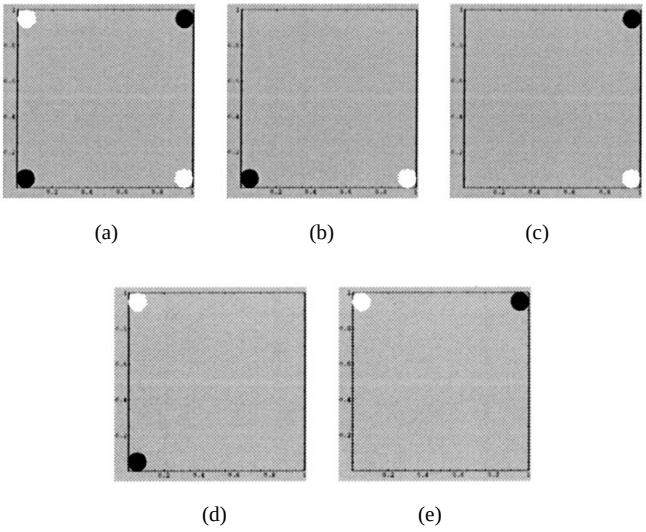


Fig. 4. (a) The four training inputs for the original XOR problem. (b) The training inputs for $\mathcal{T}^{(1,1)}$, (c) $\mathcal{T}^{(1,2)}$, (d) $\mathcal{T}^{(2,1)}$, and (e) $\mathcal{T}^{(2,2)}$, respectively. The black and white points represent the inputs whose desired outputs are "0" and "1," respectively, and grey represents only the background of the figures. This notation will be also used in Figs. 7 and 8.

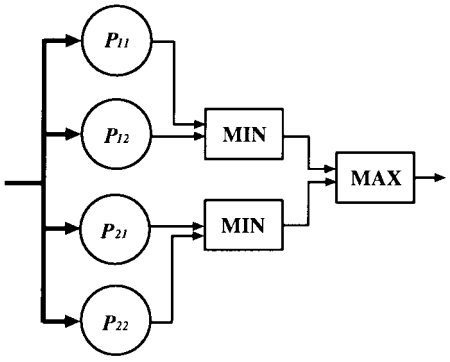


Fig. 5. The M^3 network for the XOR problem, where four perceptron computational units, namely $\mathbf{P}_{11}$, $\mathbf{P}_{12}$, $\mathbf{P}_{21}$, and $\mathbf{P}_{22}$, are regarded as four network modules.

B. Two-Spirals Problem

The "two-spirals" problem [16] is chosen as a benchmark problem for this study because it is an extremely hard two-class problem for plain MLP's, and the input–output mapping formed by each of the individual trained modules is visible. The aim of this example is to demonstrate how to divide a complex two-class problem into a series of smaller and simpler two-class problems by using two different partition techniques, namely randomly partition and space-grid partition, respectively. The 194 training inputs of the original two-spirals problem are shown in Fig. 7(a). In the following four comparative simulations were performed on this problem.

In the first simulation, the original training inputs belonging to class $\mathcal{C}_1$ and class $\mathcal{C}_2$ [see Fig. 7(a)] were divided into six training subsets randomly according to a uniform distribution in the two-dimensional space. These six training subsets are shown in the panels of the top two rows of Fig. 8. The training inputs of the nine subproblems were constructed from the combinations of the above six training subsets. These training inputs are shown in the panels of the bottom three rows of Fig. 8. The nine subproblems are represented as $\mathcal{T}^{(1,1)}$, $\mathcal{T}^{(1,2)}$, $\mathcal{T}^{(1,3)}$, $\mathcal{T}^{(2,1)}$, $\mathcal{T}^{(2,2)}$, $\mathcal{T}^{(2,3)}$, $\mathcal{T}^{(3,1)}$, $\mathcal{T}^{(3,2)}$, and $\mathcal{T}^{(3,3)}$, respectively. Note that the subproblems in the same row of Fig. 8 have the same training inputs corresponding to the desired outputs "1," i.e., the same white points.

Nine MLQP's were selected as the network modules to learn the nine subproblems. Each of the six network modules has 20 hidden units and each of other three network modules has 25 hidden units. After the nine subproblems had been learned by the corresponding network modules, which are represented as $\mathbf{M}^{11}$, $\mathbf{M}^{12}$, $\mathbf{M}^{13}$, $\mathbf{M}^{21}$, $\mathbf{M}^{22}$, $\mathbf{M}^{23}$, $\mathbf{M}^{31}$, $\mathbf{M}^{32}$, and $\mathbf{M}^{33}$, respectively, the individual trained modules were integrated into an M^3 network as illustrated in Fig. 9. The responses of the individual trained modules are shown in Fig. 10(a)–(i), respectively. The combination of the outputs of $\mathbf{M}^{11}$, $\mathbf{M}^{12}$, and $\mathbf{M}^{13}$ by the MIN unit is shown in Fig. 11(a). The combination of the outputs of $\mathbf{M}^{21}$, $\mathbf{M}^{22}$, and $\mathbf{M}^{23}$ by the MIN unit is shown in Fig. 11(b). The combination of the outputs of $\mathbf{M}^{31}$, $\mathbf{M}^{32}$, and $\mathbf{M}^{33}$ by the MIN unit is shown in Fig. 11(c). The response of the entire M^3 network is shown in Fig. 12(a).

In the second simulation, the original training inputs belonging to class $\mathcal{C}_1$ and class $\mathcal{C}_2$ were divided into six training subsets by partitioning the input variable through the axis of abscissas into three slight overlapping intervals [21]. We call this partition method the *space-grid partition*. The training inputs of the nine subproblems were constructed from the combinations of the above six training subsets. Similar to the first simulation, nine MLQP's were selected as the network modules to learn the corresponding nine subproblems. All of the MLQP's were chosen to be five hidden units, except that one module was selected to be 25 hidden units. The corresponding M^3 network has the same structure as shown in Fig. 9. The response of the entire M^3 network is shown in Fig. 12(b).

In the third simulation, the original problem was divided into 36 subproblems by using the space-grid partition method. The aim of this simulation is to show both the maximum and the total CPU times required for training the individual modules can be reduced by dividing the original problem into a large number of smaller and simpler two-class problems, i.e., further decreasing the complexity of each of the subproblems. The response of the corresponding M^3 network is shown in Fig. 12(d).

Although 100% success rates on training data were achieved by all of the three simulations mentioned above, the success rate on test data, i.e., the generalization accuracy, obtained by the first simulation is lower than those obtained by the second and third simulations. This test result can also be observed directly by comparing Fig. 12(a) with (b) and (c). The reason for this result seems that the geometric relations among the original training data [see Fig. 7(a)] was damaged to a large extent by randomly dividing the original problem into several subproblems (see the panels of the bottom three rows of Fig. 8). In contrast to randomly partition, if the interval overlapping is wide enough, the geometric relations among the original training data can be well preserved in each of the subproblems obtained by using the space-grid partition method. We think that analysis of the randomly partition and space-grid partition methods theoretically is an important problem for the future work.

In the fourth simulation, the original two-spirals problem was learned by a plain MLQP with 40 hidden units. Even 200 000 epochs were performed, the sum of squared error was still about 0.57. The network has not yet achieved the desired error (0.01). The response of the network is shown in Fig. 12(d).

The CPU times required to train the single and modular networks in the above four simulations are shown in Table I. The generalization performance of the single and modular networks are examined on 1746 test inputs as shown in Fig. 7(b). The test results are also shown in Table I.

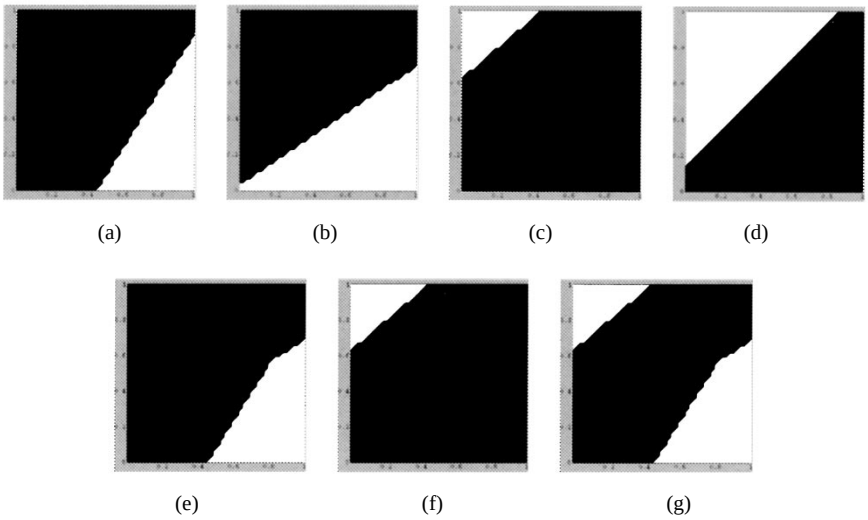


Fig. 6. The responses of (a) $\mathbf{P}_{11}$, (b) $\mathbf{P}_{12}$ (b), (c) $\mathbf{P}_{21}$, (d) $\mathbf{P}_{22}$, (e) the combination of $\mathbf{P}_{11}$ and $\mathbf{P}_{12}$, (f) the combination of $\mathbf{P}_{21}$ and $\mathbf{P}_{22}$, and (g) the entire M^3 network, respectively. Black and white represent the outputs of "0" and "1," respectively, and grey represents intermediate value. This notation will be also used in Figs. 10–12.

C. Vehicle Classification

This real classification problem [22] is to classify a given vehicle silhouette as one of four types of vehicle by using a set of features extracted from the silhouette. The vehicle silhouettes were captured with a spatial resolution of 128 $\times$ 128 pixels quantized to 64 grey levels. These silhouettes were cleaned up, binarized and subsequently processed to produced 18 variables intended to characterize shape. The data set was divided into training and test sets. Each of the two sets consists of 423 data. The number of attributes is 18 and the number of classes is four. The original problem was decomposed into $\binom{4}{2}$ two-class problems. Six MLQP's were selected as the network modules to learn the six subproblems, respectively. All of the MLQP's were chosen to be four hidden units, except that the module used to train on $\mathcal{T}_{23}$ was selected to be eight hidden units. The six individual trained modules were integrated into an M^3 network as illustrated in Fig. 13. The original problem was also learned by a plain MLQP with 24 hidden units. The simulation results are shown in Table I.

D. Image Segmentation

The image segmentation problem [22] is a real problem.[1] The instances in the problem were drawn randomly from a database of seven outdoor color images. The images were hand-segmented to create a classification for every pixel as one of brick-face, sky, foliage, cement, window, path, and grass. The problem consists of 210 training data and 2100 test data. The number of attributes is 18 and the number of classes is seven. The original problem is decomposed into $\binom{7}{2}$ two-class problems according to the proposed decomposition method (5). Each of the two-class problems consists of 60 training data. Each of the 21 two-class problems was learned by an MLP with one and two hidden units, respectively. All of the 21 trained modules were integrated into an M^3 network in a similar way as described in the preceding example. The original problem was also learned by a plain MLP with 18 and 33 hidden units, respectively. The reason of selecting 18 and 33 hidden units for the single MLP's is to make the related M^3 networks have about the same numbers of weights and bias as the single MLP's. Ten simulation runs were performed with both single MLP's and the M^3 networks. The results are shown in Table II. For the single MLP with 33 hidden units, even 200 000 epochs were performed ten times with various initial weights and learning rates, no successfully learning was obtained, i.e., the desired error (0.05) was not achieved. From Table II, we can see that the proposed M^3 network is far superior to single networks in training time, and meanwhile its generalization performance is better than that of single networks.

[1] In the original data set, there are 19 attributes. Since the third attribute is a constant number, we delete it from the data set and use only 18 attributes in the simulation.

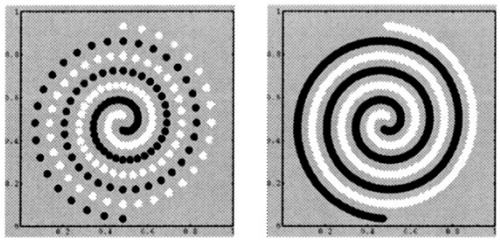


Fig. 7. The training inputs and test inputs for the two-spirals problem: (a) shows the 194 training inputs and (b) shows the 1746 test inputs, which are different from the 197 training inputs.

E. DNA Problem

The DNA problem is a three-class classification problem [22], which is to recognize the following three classes: 1) exon/intron (EI) boundaries; 2) intron/exon (IE) boundaries; and 3) neither (N). The DNA dataset consists of 2000 training data and 1186 test data. The number of attributes is 180 and the number of classes is three. Let $\mathcal{X}_1$, $\mathcal{X}_2$, and $\mathcal{X}_3$ be the training input subsets for class IE, class EI, and class N, respectively.

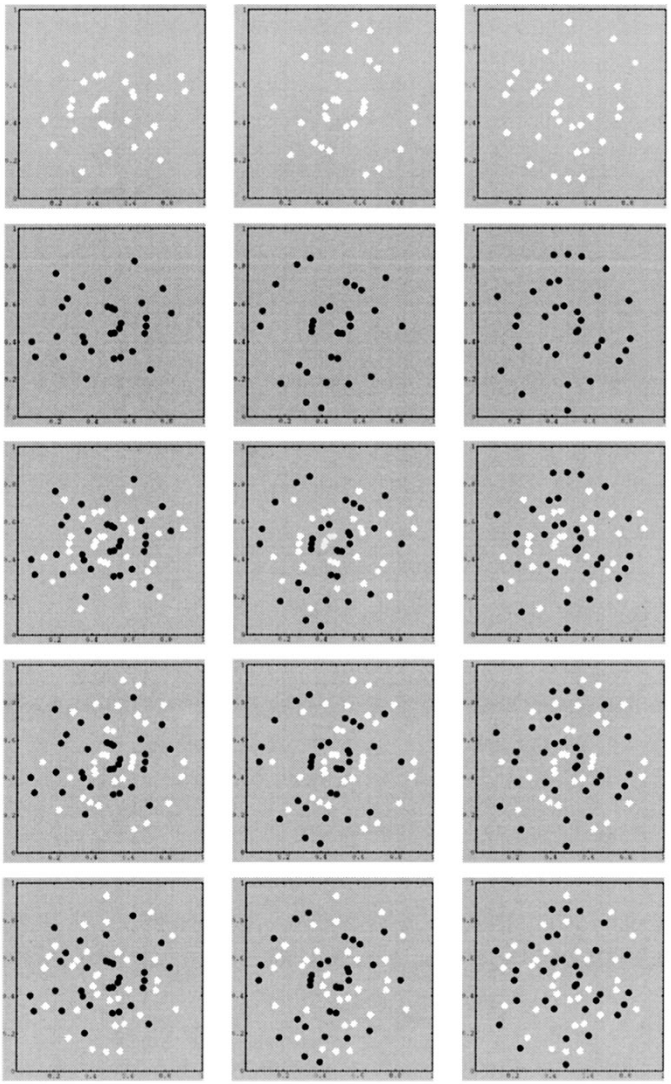


Fig. 8. Randomly partition of the two-spirals problem. The panels in the top two rows show the 97 training inputs corresponding to desired output "1" and 97 training inputs corresponding to desired output "0" are randomly divided into three parts, respectively. The panels in the bottom three rows show the nine subproblems constructed from the combinations of the training inputs in the panels of the top two rows.

The numbers of data in $\mathcal{X}_1$, $\mathcal{X}_2$, and $\mathcal{X}_3$ are 464, 485, and 1051, respectively. In the simulations, $\mathcal{X}_i$ for $i = 1, 2$, and 3 are randomly divided into several roughly equal subsets. The following four kinds of partitions are performed: 1) $N_1 = N_2 = 2$, and $N_3 = 5$; 2) $N_1 = N_2 = 3$, and $N_3 = 7$; 3) $N_1 = N_2 = 5$ and $N_3 = 10$; and 4) $N_1 = 9$, $N_2 = 10$, and $N_3 = 21$. According to (14), the total numbers of subproblems for the above four partitions are 20, 51, 125, and 489, respectively. The maximal numbers of training data for each of the subproblems belonging to the above four partitions are 506, 317, 203, and 115, respectively. In the simulations, for each of the four kinds of partitions, the original dataset was randomly divided into a number of subproblems ten times. Three-layer MLP's with four different numbers of hidden units, i.e., 1) five hidden units; 2) three hidden units; 3) two hidden units; and 4) one hidden unit, were selected as network modules for the above four kinds of partitions, respectively. All the network modules were trained by the conventional backpropagation algorithm [27]. The performance of the corresponding M^3 networks are shown in Table III. From the simulation results, we see that even the DNA problem was randomly divided into 489 subproblems, the corresponding M^3 network can still obtain better generalization performance (93.51%) than single MLP (91.2%), C4.5 (92.4%), k-NN (85.4%), and probabilistic neural network (83.6%) [5], [23], and meanwhile the training time can be reduced greatly.

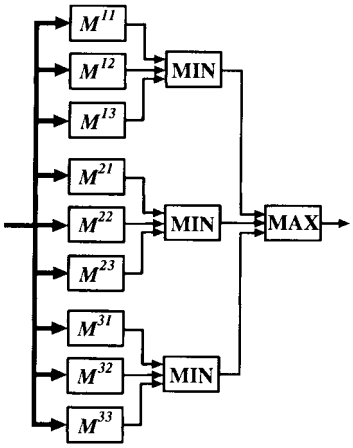


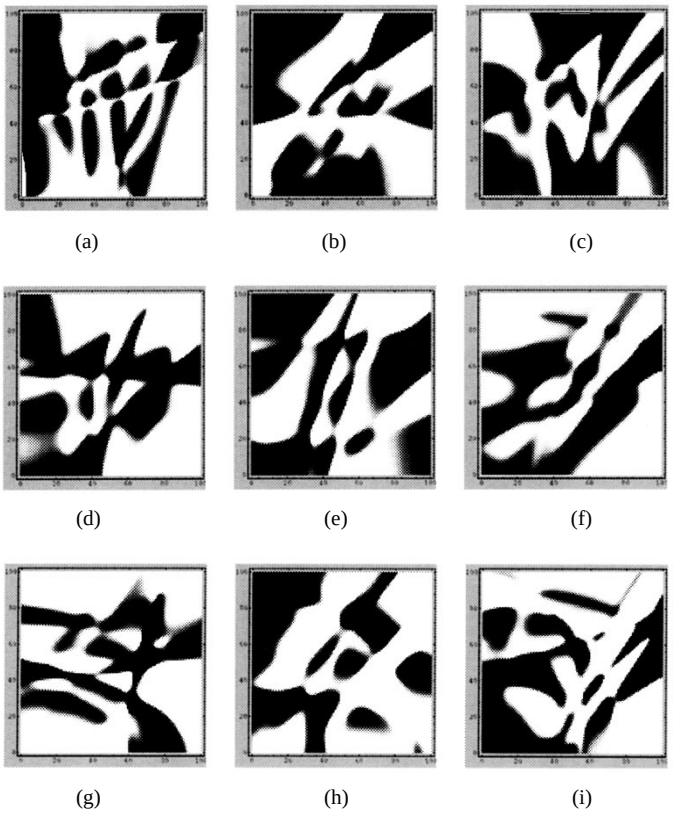
Fig. 9. The M^3 network for the two-spirals problem.

F. Shuttle Problem

The shuttle problem concerns the position of radiators within the Space Shuttle of NASA. The shuttle data set [22] is a seven-class problem and contains nine attributes all of which are numerical. The training set consists of 43 500 patterns and the test set contains 14 500 patterns. In order to investigate the classification power and effectiveness of the M^3 network and to compare it with the conventional MLP and class-sensitive neural network (CSNN) [2], [6], [12], in learning of large-scale and complex pattern classification problems, the shuttle data set is learned by the following three kinds of networks: i) single MLP; ii) CSNN; and iii) the M^3 network. The MLP is trained by the standard backpropagation algorithm [27], while both CSNN and the M^3 network were trained by the modified backpropagation algorithm [1]. In the simulations, training was stopped when no training patterns remained misclassified or the total number of epochs was reached to 20 000.

1) Single MLP: The shuttle problem was first learned by a single MLP with one hidden layer. We investigated the following hidden layer sizes for the MLP's: 30, 60, 90, 120, 150, and 180. Unfortunately, no successfully learning was obtained. Since the smallest sum of squared error is still about 4378, no any training data and test data can be correctly recognized by the trained single MLP's, i.e., the successfully rates on both training data and test data are 0!

2) CSNN: According to class decomposition method [2], [6], [12], the original shuttle problem was divided into seven two-class problems each of which contains 43 500 training data. Seven MLP's with one hidden layer are selected to learn the seven two-class problems. We investigated the following

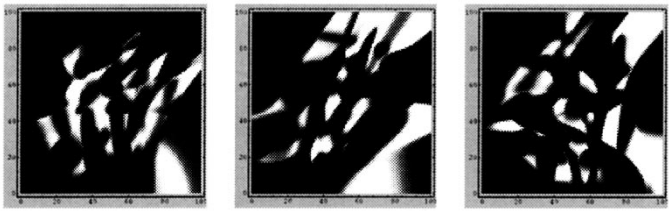


(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Fig. 10. The responses of (a) $\mathbf{M}^{11}$, (b) $\mathbf{M}^{12}$, (c) $\mathbf{M}^{13}$, (d) $\mathbf{M}^{21}$, (e) $\mathbf{M}^{22}$, (f) $\mathbf{M}^{23}$, (g) $\mathbf{M}^{31}$, (h) $\mathbf{M}^{32}$, and (i) $\mathbf{M}^{33}$, respectively.



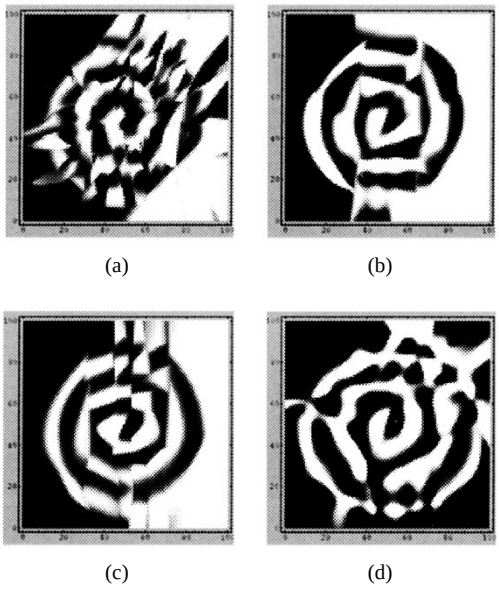
(a) (b) (c)

Fig. 11. The intermediate responses of the M^3 network illustrated in Fig. 9: (a) shows the combination of the outputs of $\mathbf{M}^{11}$, $\mathbf{M}^{12}$, and $\mathbf{M}^{13}$. (b) Shows the combination of the outputs of $\mathbf{M}^{21}$, $\mathbf{M}^{22}$, and $\mathbf{M}^{23}$. (c) Shows the combination of the outputs of $\mathbf{M}^{31}$, $\mathbf{M}^{32}$, and $\mathbf{M}^{33}$.

hidden layer sizes for the MLP's: 30, 60, and 90. The best learning accuracy was obtained with 60 hidden units. All of the seven two-class problems could not be learned completely. The simulation results are summarized in Table IV.

3) The M^3 Network: In this simulation, the training set belonging to class $\mathcal{C}_1$ is randomly divide into ten subsets each of which contains about 3410 patterns, and the training set belonging to class $\mathcal{C}_4$ is randomly divided into two subsets each of which contains 3374 patterns. After performing these partitions, the total number of two-class subproblems becomes 90.

Each of the 90 two-class problems is learned by an MLP with a single hidden layer. The numbers of hidden units for the MLP's are selected within the range from 12 to 36. After training of the 90 modules, the 90 trained modules were integrated into an M^3 network. The simulation results are shown in Table IV. Although there exist 17 hard two-class subproblems which could not be learned completely, success rate on the training data set is near to 100%.



(a) (b)

(c) (d)

Fig. 12. The responses of the M^3 networks and single network: (a) Shows the response of the M^3 network illustrated in Fig. 9, where the original problem is divided into nine subproblems randomly. (b) Shows the response of the M^3 network with nine modules, where the original problem is divided into nine subproblems by partitioning the input variable through the axis of abscissas. (c) Shows the response the M^3 network with 36 modules, where the original problem is also divided into 36 subproblems by partitioning the input variable through the axis of abscissas. (d) Shows the response of a plain MLQP with 40 hidden units.

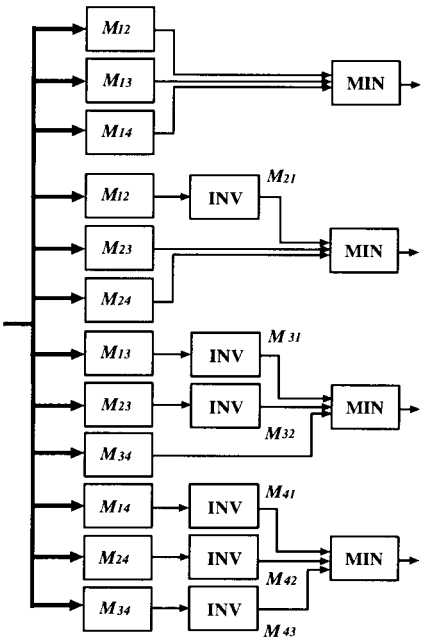


Fig. 13. The M^3 network for the vehicle problem, a four-class problem.

The results of Table IV show that the M^3 network is superior to CSNN's in the aspects of convergence speed, learning accuracy, and generalization performance. The other advantage of the M^3 network over CSNN's is that to get absolute 100% success rate on the training data set or to reduce the CPU time required for training each of the modules can be efficiently achieved with the M^3 network by directly applying the task decomposition method to dividing each of the hard two-class subproblems into a set of smaller and simpler two-class subproblems, while to obtain 100% success rate on training data by CSNN's is quite difficult because it is hard to achieve completely learning of each of the seven large two-class problems by the modified backpropagation algorithm [1] and the large two-class problems can not be further divided into a set of relatively smaller and simpler two-class problems.

V. Related Work

There is a variety of ties that can be made between the M^3 network and related work in machine perception, statistical pattern classification, modular neural networks, and fuzzy neural systems. In this section, we discuss some of these ties and compare the M^3 network with other models.

A. Multiple Discriminant Calculators

In [26], Nilsson presented a general K-class classifier architecture that consists of K individual discriminant calculators. The basic ideas behind this architecture are to divide a K-class classification problem into K individual two-class problems and select the largest output as a solution to the original problem from the K individual discriminant calculators. By applying this architecture to constructing multiclass classifiers, he also proposed the *linear machine* and *piecewise linear machine* for solving a class of multiclass classification problems known as *linearly separable*. In addition, Duda and Hart [8] gave the definition of the *pairwise linearly separable* problem. However, the classification capabilities of the linear machine and piecewise linear machine limit their usefulness because almost all the real classification problems such as the vehicle and shuttle problems mentioned in the preceded section are nonlinearly separable. Nevertheless, both the linear machine and the piecewise linear machine can be considered as special cases of the M^3 network.

B. Pairwise Classifier

The idea of using maximizing operation to make final decision is well-known in statistical pattern recognition literature and has a long theoretical background [8], [10]. Friedman recently proposed an alternative statistical classification method called *pairwise classifier* for solving K-class classification problems [9]. The basic idea behind pairwise classifier is to cast a K-class problem into a series of $\binom{K}{2}$ two-class problems based on statistic theory and use the maximizing operation to select the final decision boundary from these $\binom{K}{2}$ decision boundaries. The common feature between the M^3 network and Friedman's method is that a K-class problem is divided into a series of $\binom{K}{2}$ two-class problems and each of the two-class problems is learned independently, although completely different techniques are used. On the other hand, the combination mechanisms used in the M^3 network and the pairwise classifier are completely distinct. In the pairwise classifier, the final decision boundary is selected from the $\binom{K}{2}$ decision boundaries by performing the maximizing operation, but in the M^3 network, the $\binom{K}{2}$ trained network modules are integrated by K MIN units. A most remarkable difference between the M^3 network and the pairwise classifier is that fine decomposition of two-class problems into a series of smaller and simpler two-class problems can not be carried out with Friedman's method.

C. CSNN

CSNN was first proposed by Chen and You [6], and rediscovered also by Ishihara and Nagano [12], and Anand and his colleagues [2]. The basic idea of CSNN is to split a K-class problem into K two-class problems as defined by (2). A CSNN for a K-class problem consists of K network modules, and the ith network module is used to discriminate the patterns of class $\mathcal{C}_i$ from the patterns of the rest classes. In other words, the ith network module for class $\mathcal{C}_i$ is trained on $\mathcal{T}_i$ defined by (2). Even if the number of patterns for each of K classes is roughly equal, $\mathcal{T}_i$ may contain much more training patterns belonging to $\overline{\mathcal{C}}_i$ than those belonging to $\mathcal{C}_i$. Such a two-class classification problem is called an *imbalanced* classification problem [1]. Anand *et al.* [1] have pointed out that the standard backpropagation algorithm [27] converges slowly for learning these imbalanced two-class problems, and have developed a modified backpropagation algorithm for dealing with the imbalanced two-class problems. They have shown that their modified algorithm is faster than the standard one. However, as we mentioned in Section II, if a K-class problem is a large

and complex problem, to learn each of the related two-class problems defined by (2) is still intractable.

D. Fuzzy Min–Max Neural Networks

Various fuzzy neural networks involved minimizing (intersection) and maximizing (union) operations have been proposed for different purposes. Although the functions of the MIN and MAX units used in the M^3 network are, respectively, the same as the minimizing and maximizing operations involved in the fuzzy neural networks, the essential purposes of the MIN and MAX units in the M^3 network are very different from those of the minimizing and maximizing operations in the fuzzy neural networks. For example, there are two essential differences between the M^3 network and two kinds of *fuzzy min–max* (FMM) neural networks [28], [31].

1) The MIN and MAX units in the M^3 network are not involved in the learning process and are only used to connect each of trained modules after learning, while the minimizing and maximizing operations in the FMM networks are the weighting operations and performed in both learning and recognition processes.
2) The structure of the M^3 network is modular, while that of the FMM networks is nonmodular.

VI. Conclusions

A fundamental modularity design paradigm for a broad variety of problems is the divide-and-conquer technique. The research on applying this technique to neural networks is of extreme importance because it enables a broaden use of neural networks. In this paper, we have proposed a new task decomposition method, two module combination principles, and a new modular neural-network architecture. The central idea underlying the task decomposition method is based on the class relations among training data. For a given K-class classification problem, we can divide the problem into a set of smaller and simpler two-class problems by using the proposed task decomposition method. Several attractive features of this method can be summarized as follows.

1) We can break down a problem into a set of subproblems as small as we expect even though we are not domain specialists or we have no any prior knowledge concerning the decomposition of the problem.
2) Training of each of the two-class problems can be greatly simplified and achieved independently.
3) Different network structures or different learning algorithms can be used to learn each of the problems. The proposed module combination principles give us a systematic method for integrating the individual trained modules into a modular network by use of the three integrating units.

The simulation results (see Tables I–IV) show several significant advantages of the modular network suggested here over single networks such as easily designing network structure, faster training, and high learning accuracy. The generalization performance of the proposed modular network is about the same as the single networks. The simulation results also show that the proposed min–max modular network is superior to the class-sensitive neural network [2], [6], [12] in convergence speed and learning accuracy. The importance of the proposed modular learning framework lies in the fact that it provides us an efficient approach to solving large-scale, real-world pattern classification problems.

Acknowledgment

The authors would like to thank the reviewers of this paper for their thoughtful and valuable suggestions and comments. The authors also would like to thank R. Anand of IBM Thomas J. Watson Research Center for providing the source code of his modified backpropagation algorithm. M. Ito died during the revision of this paper, so under the circumstances B. L. Lu wishes to dedicate this paper to his memory.

References

[1] R. Anand, K. G. Mehrotra, C. K. Mohan, and S. Ranka, "An improved algorithm for neural-network classification of imbalanced training sets," *IEEE Trans. Neural Networks*, vol. 4, pp. 962–963, 1993.
[2] ——, "Efficient classification for multiclass problems using modular neural networks," *IEEE Trans. Neural Networks*, vol. 6, pp. 117–124, 1995.
[3] J. A. Anderson, *An Introduction to Neural Networks.* Cambridge, MA: MIT Press, 1995.
[4] Y. Bennani and P. Gallinari, "Task decomposition through a modular connectionist architecture: A talker identification system," in *Proc. 3rd Int. Conf. Artificial Neural Networks*, I. Aleksander and J. Taylor, Eds., vol. 1. Amsterdam, The Netherlands: North-Holland, Sept. 4–7, 1992, pp. 783–786.
[5] M. R. Berthold and J. Diamond, "Constructive training of probabilistic neural networks," *Neurocomputing*, vol. 19, pp. 167–183, 1998.
[6] C. H. Chen and G. H. You, "Class-sensitive neural network," *Neural Parallel Scie. Comput.*, vol. 1, no. 1, pp. 93–96, 1993.
[7] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms.* Cambridge, MA: MIT Press, 1990.
[8] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis.* New York: Wiley, 1973.
[9] J. H. Friedman, "Another approach to polychotomous classification," Stanford University, Stanford, CA, Tech. Rep., 1996.
[10] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, 2nd ed. San Diego, CA: Academic, 1990.
[11] P. Gallinari, "Modular neural net systems, training of," in *The Handbook of Brain Theory and Neural Networks*, M. A. Arbib, Ed. Cambridge, MA: MIT Press, 1995, pp. 582–585.
[12] S. Ishihara and T. Nagano, "Text-independent speaker recognition utilizing neural-network techniques," Tech. Rep. IEICE, vol. NC93-121, pp. 71–77, 1994, in Japanese.
[13] R. A. Jacobs, M. I. Jordan, M. I. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural Comput.*, vol. 3, pp. 79–87, 1991.
[14] R. A. Jacobs, M. I. Jordan, and A. Barto, "Task decomposition competition in a modular connectionist architecture," *Cognitive Sci.*, vol. 15, pp. 219–250, 1991.
[15] R. Jenkins and B. Yuhas, "A simplified neural-network solution through problem decomposition: The case of the truck backer-upper," *IEEE Trans. Neural Networks*, vol. 4, pp. 718–722, 1993.
[16] K. Lang and M. Witbrock, "Learning to tell two spirals apart," in *Proc. 1988 Connectionist Models Summer School.* San Mateo, CA: Morgan Kaufmann, June 17–26, 1988, pp. 52–59.
[17] B. L. Lu, Y. Bai, H. Kita, and Y. Nishikawa, "An efficient multilayer quadratic perceptron for pattern classification and function approximation," in *Proc. Int. Joint Conf. Neural Networks*, Nagoya, Japan, Oct. 25–29, 1993, pp. 1385–1388.
[18] B. L. Lu, H. Kita, and Y. Nishikawa, "A multisieving neural-network architecture that decomposes learning tasks automatically," in *Proc. IEEE Conf. Neural Networks*, Orlando, FL, June 28–July 2, 1994, pp. 1319–1324.

[19] B. L. Lu, "Architectures, learning and inversion algorithms for multilayer neural networks," Ph.D. dissertation, Dept. Elect. Eng., Kyoto Univ., Japan, 1994.
[20] B. L. Lu, K. Ito, H. Kita, and Y. Nishikawa, "A parallel and modular multisieving neural-network architecture for constructive learning," in *Proc. Inst. Elect. Eng. 4th Int. Conf. Artificial Neural Networks*, Cambridge, U.K., June 26–28, 1995, pp. 92–97.
[21] B. L. Lu and M. Ito, "Task decomposition based on class relations: A modular neural-network architecture for pattern classification," in *Biological and Artificial Computation: From Neuroscience to Technology*, *Lecture Notes in Computer Science*, J. Mira, R. Moreno-Diaz, and J. Cabestany, Eds., vol. 1240. New York: Springer-Verlag, 1997, pp. 330–339.
[22] C. J. Merz and P. M. Murphy, "UCI Repository of machine learning databases," Univ. California, Dept. Inform. Comput. Sci., Irvine, CA, 1996. Available http://www.ics.uci.edu/ mlearn/MLRepository.html
[23] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, Eds., *Machine Learning, Neural and Statistical Classification.* Chicester, U.K.: Ellis Horwood, 1994.
[24] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*, Expanded ed. Cambridge, MA: MIT Press, 1988.
[25] J. M. J. Murre, *Learning and Categorization in Modular Neural Networks.* London, U.K.: Harvester Wheatsheaf, 1992.
[26] N. J. Nilsson, *Learning Machines: Foundations of Trainable Pattern Classifying Systems.* New York: McGraw-Hill, 1965; reissued as *The Mathematical Foundations of Learning Machines*. San Mateo, CA: Morgan Kaufmann, 1990.
[27] D. R. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, D. E. Rumelhart, J. L. McClelland, and PDP Research Group, Eds. Cambridge, MA: MIT Press, 1986.
[28] P. K. Simpson, "Fuzzy min–max neural networks—Part 1: classification," *IEEE Trans. Neural Networks*, vol. 3, pp. 776–786, 1992.
[29] S. Thiria, C. Mejia, F. Badran, and M. Crepon, "Multimodular architecture for remote sensing operations," *Advances in Neural Information Processing Systems 4*, J. E. Moody, S. J. Hanson, and R. P. Lippmann Eds. San Mateo, CA: Morgan Kaufmann, 1992, pp. 675–682.
[30] A. Waibel, H. Sawai, and K. Shkano, "Modularity and scaling in large phonemic neural networks," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, 1989.
[31] X. Zhang, C. C. Hang, S. Tan, and P. Z. Wang, "The min–max function differentiation and training of fuzzy neural networks," *IEEE Trans. Neural Networks*, vol. 7, pp. 1139–1150, 1992.