Neural Comput & Applic (1997)6:19–41

Neural
Computing
& Applications

Combining Linear Discriminant Functions with Neural Networks for Supervised Learning

Ke Chen[1,2], Xiang Yu[1] and Huisheng Chi[1]

[1]National Laboratory of Machine Perception and Center for Information Science, Peking University, Beijing, China; [2]Department of Computer and Information Science and The Center for Cognitive Science, The Ohio State University, Columbus, OH, USA

A novel supervised learning method is proposed by combining linear discriminant functions with neural networks. The proposed method results in a tree-structured hybrid architecture. Due to constructive learning, the binary tree hierarchical architecture is automatically generated by a controlled growing process for a specific supervised learning task. Unlike the classic decision tree, the linear discriminant functions are merely employed in the intermediate level of the tree for heuristically partitioning a large and complicated task into several smaller and simpler subtasks in the proposed method. These subtasks are dealt with by component neural networks at the leaves of the tree accordingly. For constructive learning, growing *and* credit-assignment *algorithms are developed to serve for the hybrid architecture. The proposed architecture provides an efficient way to apply existing neural networks (e.g. multi-layered perceptron) for solving a large scale problem. We have already applied the proposed method to a universal approximation problem and several benchmark classification problems in order to evaluate its performance. Simulation results have shown that the proposed method yields better results and faster training in comparison with the multi-layered perceptron.*

Keywords: Constructive learning; Divide-and-conquer; Linear discriminant function; Modular and hierarchical architecture; Multi-layered perceptron; Supervised learning

Correspondence and offprint requests to: K. Chen, Department of Computer and Information Science and The Center for Cognitive Science, The Ohio State University, Columbus, OH 43210-1277, USA. Email: kchen@cis.ohio-state.edu.

1. Introduction

Neural networks, particularly Multi-Layered Perceptrons (MLPs), have already been found to be successful for various supervised learning tasks [1–11]. Both theoretical and empirical studies have shown that the neural network is of powerful capabilities for pattern classification and universal approximation which are typical supervised learning tasks. Hornik *et al.* [12] and Irie and Miyate [13] proved that a three-layered MLP with an infinite number of nodes in the hidden layer can also solve arbitrary mapping problems. However, the problem of training an MLP is NP-complete [14], and therefore all existing algorithms are heuristics; which results in that the training of an MLP often suffers from a slow convergence property, though some methods have been suggested to tackle the problem [15,16]. In addition, the exact number of hidden layers and neurons in a hidden layer, as well as connectivity between layers, must be specified before learning can begin. For this problem, however, the aforementioned theoretical results on the MLP are of little practical value, since they cannot be utilised to determine an exact number of hidden layers and neurons in a hidden layer. Although some statistical techniques have been recently borrowed for model selection [1,17,18], most of them are involved in a time-consuming procedure for practical use. Thus, the network architecture must be determined by trial and error. To overcome the difficulty in determining a neural network architecture prior to training, practical approaches for dynamic neural network architecture generation have been sought [19–21]. However, these models do not specify in what exact sequence a neuron should be added to give the maximum effect in classifying training

examples and keep the slow convergence property, since they still suffer from serious catastrophic interference in both spatial and temporal crosstalk during training. As a result, both determination of a neural network architecture and fast training still remain important research topics in the neural network community.

On the other hand, supervised learning has been studied for a long time in the pattern recognition community. The decision tree is one of the most efficient tools for supervised learning in the pattern recognition community [22–29]. In general, decision trees are hierarchical structures which use a sequential decision making strategy to handle a supervised learning task. At each internal node of a decision tree, a test is evaluated to decide which child node the feature vector will be sent to, while the leaves of the decision tree are used to deal with unknown data. In decision tree approaches, linear discriminant functions or hyperplanes are commonly used for test and decision [22,26,27]. The traditional approach to the training of a decision tree has been to first generate a set of possible hyperplanes, and then exhaustively search this set to find the best hyperplanes with respect to some distortion metric [22,26,27]. In most decision tree approaches, the hyperplanes are constrained to be perpendicular to the feature space axes. This is very restrictive, and can result in a large number of hyperplane tests, even in a linearly separable problem [26], if the separating hyperplane is not perpendicular to the feature space axes. Indeed, there are some decision tree approaches which allow for non-perpendicular hyperplanes [22–24,28,29], but these algorithms used are computationally expensive due to the large search space of hyperplanes that needs to be evaluated. In comparison with neural networks, however, the merits of decision trees are that there is no problem of determining an architecture prior to the training in the design of a decision tree, and the performance of a linear decision tree is basically similar to an MLP's for a specific task [30,31]. As a result, many researchers in the neural network community have recently considered hybrid structures between decision trees and neural networks. Although these techniques were developed as neural networks whose structure could be automatically determined, their outcome can be interpreted as decision trees with nonlinear splits [32–39]. In most of these hybrid structures, a small neural network at each node of the tree classifier is used to implement nonlinear and multivariate splits instead of a hyperplane in decision tree approaches. Accordingly, the neural tree is automatically generated by training the neural network at each node instead of searching for the best hyperplane with some distortion metric in decision tree approaches. Basically, such neural trees follow the principle underlying decision trees, i.e. non-overlapping ('hard') split in each nonterminal node, and only one class label associated with each leaf node [34,37–39]. For generalisation, a pruning procedure is also needed in these neural trees, as in decision tree approaches such as the Classification And Regression Tree (CART) [22]. The other hybrid techniques between decision trees and neural networks use elaborate methods for converting a decision tree into a neural network, and then retraining it [32,36]. However, these techniques can be only used in simple pattern classification problems, and their performance for a real world problem remains unknown. Since neural networks at all nodes of the tree need a time-consuming training procedure like MLP, most of these hybrid systems still suffer from slow training though their performance could be improved to some extent. More recently, Jordan and Jacobs [40] presented a tree-structured modular neural network architecture called Hierarchical Mixture of Experts (HME). In the HME architecture, gating networks sit at the nonterminal nodes and expert networks sit at the leaves of the tree. The structures of gating and expert networks could be different. The use of the gating network distinguishes the HME from other tree-based algorithms which make a 'hard' decision in the input space, in contrast to the 'soft' method of the HME, which allows adjacent clusters to overlap. The HME can be used in both regression and pattern classification tasks. However, the problem of determining an architecture prior to training is still encountered in practical use of the HME [41,42].

Unlike the aforementioned work, an alternative tree-structured architecture is proposed in this paper based upon our earlier work [43] for supervised learning. The principle of divide-and-conquer is directly employed in the proposed architecture, which results in a new binary tree-structured hybrid architecture. Unlike their use in decision trees, linear discriminant functions or hyperplanes sitting at nonterminal nodes merely play a 'divide' role to heuristically partition a large and complicated problem into several smaller and simpler problems, while neural networks (e.g. MLPs) sitting at the leaves of the tree play a 'conquer' role to solve those smaller and simpler problems. Motivated by the idea underlying the HME, we adopt a 'soft' method to partition the input space such that there might be an overlapping between two adjacent partitioned input data subsets. As a result, a tree-structured hybrid architecture can be automatically generated during training for a specified task through use of the proposed

constructive learning algorithms. For an unknown pattern during test, the output of the generated architecture is produced by combining outputs of neural networks at the leaves of the tree with 'credits' assigned by the constructive learning algorithms. On the basis of the framework, moreover, some techniques are also introduced to the constructive learning algorithms in order to speed up training and to control the size of a generated architecture for generalisation. To evaluate the performance of the proposed architecture, we have already applied the proposed architecture to several hard supervised learning tasks, including pattern classification and function approximation. Simulations have shown that the proposed architecture yields both satisfactory performance and fast training for supervised learning tasks. In particular, experimental results have also suggested that the performance of the proposed architecture is insensitive to architectures of component neural networks, which implies that the proposed architecture, to a large extent, could avoid the problem of determining an architecture prior to training.

The remainder of this paper is organised as follows. Section 2 reviews two linear discriminant functions employed in the proposed method. Section 3 describes the proposed architecture and constructive learning algorithms. Section 4 reports simulation and comparative results in great detail. Discussions and conclusions are presented in the last section.

2. Linear Discriminant Functions

In this section, we review two linear discriminant functions used in the proposed method. One is Fisher's linear discriminant function [44], and the other is a specific linear discriminant function relevant to the normal density [45].

2.1. Fisher's Linear Discriminant Function

One of the recurring problems encountered in applying statistical techniques to pattern recognition is the so-called *curse of dimensionality*. That is, procedures that are analytically or computationally manageable in low-dimensional spaces can become completely impractical in a space of high-dimensions. To tackle the problem, Fisher's suggestion [44,45] was to look for the linear function which maximises the ratio of the between-class scatter to within-class scatter. The use of this method can find a linear combination of the variables such that the values are as close as possible within classes, and as far apart as possible between classes. Suppose that we have a set of N p-dimensional samples $\mathbf{x}_1,\ldots,\mathbf{x}_N$, N_1 is the subset $\mathcal{X}_1$ labelled ω_1 and N_2 in the subset $\mathcal{X}_2$ labelled ω_2. If we form a linear combination of the components of $\mathbf{x}$, we obtain the scalar

$$y = \mathbf{w}^{\mathrm{T}}\,\mathbf{x} \tag{1}$$

and a corresponding set of N samples $y_1,\ldots,y_N$ divided into the subsets $\mathcal{Y}_1$ and $\mathcal{Y}_2$. Based upon Eq. (1), the sample mean for the projected points is given by

$$\hat{m}_i = \frac{1}{N_i}\sum_{y\in\mathcal{Y}_i} y = \mathbf{w}^{\mathrm{T}}\,\mathbf{m}_i \tag{2}$$

where $\mathbf{m}_i = 1/N_i\Sigma_{\mathbf{x}\in\mathcal{X}_i}\,\mathbf{x}$. If we define the *scatter* for projected samples labelled ω_i by

$$\hat{s}_i^2 = \sum_{y\in\mathcal{Y}_i} (y-\hat{m}_i)^2 \tag{3}$$

Fisher's linear discriminant function is thus defined as the linear function $\mathbf{w}^{\mathrm{T}}\,\mathbf{x}$ for which the *criterion function*

$$J(\mathbf{w}) = \frac{|\hat{m}_1-\hat{m}_2|^2}{\hat{s}_1^2+\hat{s}_2^2} \tag{4}$$

is maximum, where $|\hat{m}_1-\hat{m}_2| = |\mathbf{w}^{\mathrm{T}}(\mathbf{m}_1-\mathbf{m}_2)|$ according to Eq. (2).

To obtain J as an explicit function of $\mathbf{w}$, we define the *scatter matrices* S_i, the *within-class scatter matrix* S_W and the *between-class scatter matrix* S_B, respectively, by

$$S_i = \sum_{\mathbf{x}\in\mathcal{X}_i} (\mathbf{x}-\mathbf{m}_i)(\mathbf{x}-\mathbf{m}_i)^{\mathrm{T}} \tag{5}$$

$$S_W = S_1 + S_2 \tag{6}$$

$$S_B = (\mathbf{m}_1-\mathbf{m}_2)(\mathbf{m}_1-\mathbf{m}_2)^{\mathrm{T}} \tag{7}$$

Using Eqs (2) and (3) and these definitions, we can obtain

$$\hat{s}_1^2 + \hat{s}_2^2 = \mathbf{w}^{\mathrm{T}}\,S_W\mathbf{w} \tag{8}$$

and

$$|\hat{m}_1-\hat{m}_2|^2 = \mathbf{w}^{\mathrm{T}}\,S_B\mathbf{w} \tag{9}$$

Thus, the criterion function J in Eq. (4) can be rewritten as

$$J(\mathbf{w}) = \frac{\mathbf{w}^{\mathrm{T}}S_B\mathbf{w}}{\mathbf{w}^{\mathrm{T}}S_W\mathbf{w}} \tag{10}$$

This expression is well known as the generalised Rayleigh quotient. If S_W is nonsingular, the final solution [45] is

$$\mathbf{w} = S_W^{-1}(\mathbf{m}_1 - \mathbf{m}_2) \tag{11}$$

Thus, we have obtained Fisher's linear discriminant function, the linear function with the maximum ratio of between-class scatter to within-class scatter.

2.2. A Linear Discriminant Function for the Normal Density

In the Bayesian decision, the *minimum-error-rate* classification can be achieved by use of the discriminant functions [45]

$$g_i(\mathbf{x}) = \log p(\mathbf{x}|\omega_i) + \log P(\omega_i) \tag{12}$$

where ω_i is the label of class i, $p(\mathbf{x}|\omega_i)$ is the conditional probability density for $\mathbf{x}$ and $P(\omega_i)$ is *a priori* probability. If the densities $p(\mathbf{x}|\omega_i)$ are multivariate normal, i.e. $p(\mathbf{x}|\omega_i) \sim N(\mathbf{m}_i, \Sigma_i)$, then

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \mathbf{m}_i)^{\mathrm{T}} \Sigma_i^{-1}(\mathbf{x} - \mathbf{m}_i) - \frac{p}{2}\log 2\pi - \frac{1}{2}\log|\Sigma_i| + \log P(\omega_i) \tag{13}$$

where $\mathbf{x}$ is a p-component column vector, $\mathbf{m}_i$ is the p-component mean vector and Σ_i is the p-by-p covariance matrix.

Let us examine this result for the special case: $\Sigma_i = \sigma^2 I$, where I is the identity matrix. After ignoring some unimportant additive constants from Eq. (13), we can obtain the simple discriminant functions

$$g_i(\mathbf{x}) = -\frac{\|\mathbf{x} - \mathbf{m}_i\|^2}{2\sigma^2} + \log P(\omega_i) \tag{14}$$

where $\|\cdot\|$ is the Euclidean norm, with

$$\|\mathbf{x} - \mathbf{m}_i\|^2 = (\mathbf{x} - \mathbf{m}_i)^{\mathrm{T}}(\mathbf{x} - \mathbf{m}_i) \tag{15}$$

Expansion of the quadratic form $(\mathbf{x} - \mathbf{m}_i)^{\mathrm{T}}(\mathbf{x} - \mathbf{m}_i)$ in Eq. (14) yields

$$g_i(\mathbf{x}) = -\frac{1}{2\sigma^2}[\mathbf{x}\mathbf{x}^{\mathrm{T}} - 2\mathbf{m}_i\mathbf{x} + \mathbf{m}_i^{\mathrm{T}}\mathbf{m}_i] + \log P(\omega_i) \tag{16}$$

which appears to be a quadratic function of $\mathbf{x}$. After ignoring an additive constant $\mathbf{x}^{\mathrm{T}}\mathbf{x}$ from Eq. (16), we obtain the equivalent linear discriminant functions

$$g_i(\mathbf{x}) = \mathbf{w}_i^{\mathrm{T}}\mathbf{x} + w_{i0} \tag{17}$$

where $\mathbf{w}_i = 1/\sigma^2$, $\mathbf{m}_i$ and $w_{i0} = -1/2\sigma^2\ \mathbf{m}_i^{\mathrm{T}}\mathbf{m}_i + \log P(\omega_i)$. According to Eq. (17), we can obtain the decision boundary between class i and class j by the linear equations $g_i(\mathbf{x}) = g_j(\mathbf{x})$

$$\mathbf{w}^{\mathrm{T}}(\mathbf{x} - \mathbf{x}_0) = 0 \tag{18}$$

where

$$\mathbf{w} = \mathbf{m}_i - \mathbf{m}_j \tag{19}$$

and

$$\mathbf{x}_0 = \frac{1}{2}(\mathbf{m}_i + \mathbf{m}_j) - \frac{\sigma^2}{\|\mathbf{m}_i - \mathbf{m}_j\|^2}\log\frac{P(\omega_i)}{P(\omega_j)}(\mathbf{m}_i - \mathbf{m}_j) \tag{20}$$

Equation (18) provides another linear discriminant function which defines a hyperplane through the point $\mathbf{x}_0$ and orthogonal to the vector $\mathbf{w}$. Since $\mathbf{w} = \mathbf{m}_i - \mathbf{m}_j$, the hyperplane is orthogonal to the line between the means. If $P(\omega_i) = P(\omega_j)$, then the point $\mathbf{x}_0$ is halfway between the means and the hyperplane is the perpendicular bisector of the line between the means. If $P(\omega_i) \neq P(\omega_j)$, the point $\mathbf{x}_0$ shifts away from the more likely mean. Note, however, that if the variance σ^2 is small relative to the squared distance $\|\mathbf{m}_i - \mathbf{m}_j\|^2$, then the position of the decision boundary is relatively insensitive to the exact values of the *a priori* probabilities [45].

3. Architecture and Constructive Learning Algorithms

In this section, we propose a self-generating hybrid architecture for a supervised learning task by combining linear discriminant functions and neural networks (e.g. MLPs). A heuristic splitting rule is proposed to divide a large and complicated task into two smaller and simpler subtasks in the section. Accordingly, constructive learning algorithms, i.e. *growing* and *credit-assignment*, are also presented to support the proposed architecture. The growing algorithm is developed to automatically generate a structure by recursively using the splitting rule and training subnetworks for a specific task during learning, while the credit-assignment algorithm is designed to produce the final output for unknown data by combining the outputs of neural networks during the test.

3.1. The Hybrid Architecture

The basic idea underlying the proposed architecture is to use hyperplanes defined by linear discriminant

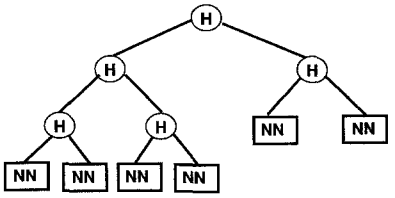

Fig. 1. A typical tree-structured architecture which can be automatically generated by the proposed method during training (H: hyperplane; NN: subnetwork (neural network).

functions for recursively partitioning a large task into several smaller subtasks, and to use neural networks, hereinafter called subnetworks, for finally solving these subtasks. Based upon the idea, a tree-structured architecture is automatically generated by a controlled growing process for a specific supervised learning task. As illustrated in Fig. 1, the architecture generated using the proposed method is a binary tree structure, in which hyperplanes sit at nonterminal nodes and subnetworks sit at the leaves of the tree. During training, each hyperplane at the nonterminal node can be determined according to a heuristic splitting rule. A subnetwork is trained on a data set $\mathcal{X}$. The training of the subnetwork will terminate once a pre-specific condition (failure condition) is satisfied. In this circumstance, the current subnetwork is aborted from the node, and a new hyperplane determined by the splitting rule is created, and sits at the current node instead of the aborted subnetwork. As a result, the data set $\mathcal{X}$ used for training the aborted subnetwork is partitioned by the hyperplane into two adjacent subsets $\mathcal{X}_l$ and $\mathcal{X}_r$, where $\mathcal{X} = \mathcal{X}_l \cup \mathcal{X}_r$ and $\mathcal{X}_l \cap \mathcal{X}_r \neq \phi$ (ϕ denotes the null set). Accordingly, two subnetworks are created and trained on $\mathcal{X}_l$ and $\mathcal{X}_r$, respectively. Such a one-step splitting process is illustrated in Fig. 2. The aforementioned recursive procedure proceeds until all subnetworks created at the leaves of the tree satisfy another condition (success condition). As a result, the proposed method transfers the problem of determining an appropriate architecture of a neural network for a given task to the problem of finding

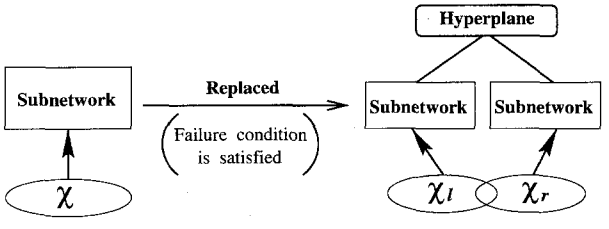

Fig. 2. A one-step splitting process when the failure condition defined for splitting is satisfied. This one-step splitting process proceeds recursively during growing until all subnetworks created satisfy the success condition defined to terminate splitting.

a tree of the right size. During the test, the unknown data is fed to the root node and a series of decisions are made traversing paths down to the leaves of the tree due to the overlapping ($\mathcal{X}_l \cap \mathcal{X}_r \neq \phi$). The final result is produced by combining results produced by subnetworks at the leaves of the tree.

3.2. Splitting Rule

In the proposed method, a splitting rule consists of two parts: determining a hyperplane for partitioning a data set into two adjacent data subsets; and selecting an appropriate size of the overlapping region between two adjacent data subsets. For determining a hyperplane, we propose three heuristic criteria for different learning tasks based upon the linear discriminant functions described in Sect. 2. Given a training data set $\mathcal{X}$ with N p-dimensional samples $\mathbf{x}_1,\ldots,\mathbf{x}_N$, the mean $\mathbf{m}$ of all samples in $\mathcal{X}$ can be computed by

$$\mathbf{m} = \frac{1}{N} \sum_{\mathbf{x} \in \mathcal{X}} \mathbf{x} \tag{21}$$

For pattern classification, moreover, N samples are assumed to be associated with K classes, N_k in the subset $\mathcal{X}_k$ labelled ω_k, $k = 1,\ldots,K$ and $\Sigma_{k=1}^{K} N_k = N$. For all samples labelled by ω_k in the case of pattern classification, its mean $\mathbf{m}_k$ can be achieved by

$$\mathbf{m}_k = \frac{1}{N_k} \sum_{\mathbf{x} \in \mathcal{X}_k} \mathbf{x}, \quad k = 1,\ldots,K \tag{22}$$

For all samples in $\mathcal{X}$, we can find two means of samples labelled by ω_{k_0} and ω_{k_1}, say $\mathbf{m}_{k_0}$ and $\mathbf{m}_{k_1}$, subject to

$$\|\mathbf{m}_{k_0} - \mathbf{m}_{k_1}\| = \max_{1 \le i,j \le K} \|\mathbf{m}_i - \mathbf{m}_j\| \tag{23}$$

where $\|\cdot\|$ is the Euclidean norm in Eq. (15). In the sequel, we shall develop criteria merely based upon samples labelled by ω_{k_0} and ω_{k_1}, except that they are not available.

Criterion 1

For the samples labelled by ω_{k_0} and ω_{k_1}, we look for Fisher's discriminant function based upon the method described in Sect. 2. As a result, we can compute the within-class scatter matrix S_W using Eqs (5) and (6). If the matrix S_W is nonsingular and $\|\mathbf{m}_{k_0} - \mathbf{m}_{k_1}\| \neq 0$, accordingly, we can obtain the Fisher's linear discriminant function by Eq. (11), that is,

$$\mathbf{w} = S_W^{-1}(\mathbf{m}_{k_0} - \mathbf{m}_{k_1}) \tag{24}$$

Therefore, a hyperplane can be determined using $\mathbf{w}$ in Eq. (24) as

$$l(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\left[\mathbf{x} - \left(\frac{N_{k_0}}{N_{k_0}+N_{k_1}}\mathbf{m}_{k_0} + \frac{N_{k_1}}{N_{k_0}+N_{k_1}}\mathbf{m}_{k_1}\right)\right] = 0 \tag{25}$$

Criterion 2

In Criterion 1, Fisher's linear discriminant function will be not available if the within-class scatter matrix S_W is singular in Eq. (24). In this case, we use the decision boundary described in Eqs (18)–(20) to determine a hyperplane for use in the splitting rule instead. Since the hyperplane is merely used to heuristically partition a data set, the hyperplane is determined by the linear discriminant function in Eq. (20), regardless of any information on density in this circumstance. When $\|\mathbf{m}_{k_0} - \mathbf{m}_{k_1}\| \neq 0$, it is

$$l(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\left[\mathbf{x} - \frac{\mathbf{m}_{k_0}+\mathbf{m}_{k_1}}{2}\right] = 0 \tag{26}$$

where $\mathbf{w} = \mathbf{m}_{k_0} - \mathbf{m}_{k_1}$.

Criterion 3

It is obvious that both Criteria 1 and 2 will be invalid when $\|\mathbf{m}_{k_0} - \mathbf{m}_{k_1}\| = 0$. In addition, Criteria 1 and 2 cannot be used for splitting in a regression task, since no class label is available. For this case, we adopt a heuristic way to determine a hyperplane as follows: the training data set should be partitioned into two almost equal size subsets. Therefore, the hyperplane could be

$$l(\mathbf{x}) = \mathbf{m}^{\mathrm{T}}(\mathbf{x} - \mathbf{m}) = 0 \quad \text{if } \mathbf{m} \neq \mathbf{0} \quad \text{or}$$
$$l(\mathbf{x}) = \mathbf{x}_r^{\mathrm{T}}\mathbf{x} = 0 \quad \text{if } \mathbf{m} = \mathbf{0} \tag{27}$$

where $\mathbf{w} = \mathbf{m}$ if $\mathbf{m} \neq \mathbf{0}$ or $\mathbf{w} = \mathbf{x}_r$ if $\mathbf{m} = \mathbf{0}$; $\mathbf{x}_r$ is a sample in $\mathcal{X}$ and randomly chosen.

For pattern classification, criteria are used for determining the hyperplane in the splitting rule in this order of priority: Criterion 1 if the within-class scatter matrix S_W is nonsingular and $\|\mathbf{m}_{k_0} - \mathbf{m}_{k_1}\| \neq 0$; Criterion 2 if $\|\mathbf{m}_{k_0} - \mathbf{m}_{k_1}\| \neq 0$ and Criterion 3. For regression or function approximation, Criterion 3 is merely used for determining the hyperplane in the splitting rule. Since we adopt the 'soft' method to partition data sets, an overlapping factor η is also needed for determining the size of an overlapping region between two adjacent data subsets in the splitting rule. Once both the hyperplane $l(\mathbf{x}) = 0$ and the overlapping factor η are determined, the splitting rule is defined as follows: for the sample $\mathbf{x}_i$ in $\mathcal{X}$,

$$\mathbf{x}_i \in \mathcal{X}_l \quad \text{if } l(\mathbf{x}) \leq D;$$
$$\mathbf{x}_i \in \mathcal{X}_r \quad \text{if } l(\mathbf{x}) \geq -D \tag{28}$$

where $D = \eta D_0$ and $\eta > 0$*. The value of D_0 depends upon the criterion used for determining the hyperplane $l(\mathbf{x}) = 0$ in the splitting rule. In Criteria 1 and 2, D_0 is equal to $\min\{d_{k_0}, d_{k_1}\}$ where d_{k_0} and d_{k_1} denote the distances from means $\mathbf{m}_{k_0}$ and $\mathbf{m}_{k_1}$ to the hyperplane $l(\mathbf{x}) = 0$, respectively. In Criterion 3, D_0 is equal to $d_{\max}$, where $d_{\max}$ denotes the maximal one among distances from all $\mathbf{x}$ in $\mathcal{X}$ to the hyperplane $l(\mathbf{x}) = 0$.

Intuitively, the splitting rule suggests a hyperplane which intends to segregate those data belonging to the two classes, such that the distance between their centroids is the furthest among all classes for pattern classification, or partition a large data set into two smaller data subsets with an almost equal number of samples for regression or function approximation. Moreover, the overlapping defined by the splitting rule is effective to maintain the balance of the number of significant samples used to train two subnetworks for pattern classification, or to keep the smooth property of a function at the boundary of two adjacent data subsets for function approximation. As a result, Fig. 3 illustrates how the splitting rule works through the use of different criteria, and only partial data labelled with ω_{k_0} and ω_{k_1} in $\mathcal{X}$ appears in (a) and (b) of Fig. 3. It is worth noting that the cases depicted in Fig. 3 are merely special examples. In fact, any two data subsets with distinct labels could be overlapping (instead of separation in Fig. 3) for the general case. Anyway, the splitting rule always works as shown in Fig. 3.

3.3. Constructive Learning Algorithms

As mentioned above, the constructive learning algorithms consist of a *growing* algorithm and a *credit-assignment* algorithm in the proposed method.

In general, the growing algorithm provides a procedure to automatically generate a binary tree for a given task. There are at least two issues worth considering in developing the growing algorithm; that is, splitting and stopping rules. The splitting rule has been described in the preceding section. Here, we only consider the stopping rule, and develop a growing algorithm using both the splitting and stopping rules. For controlling the growing process, we define two kinds of conditions, called

* If the number of samples in $\mathcal{X}$ is $N_{\mathcal{X}}$, the value of η must be chosen under the condition that $\max\{N_{\mathcal{X}_l}, N_{\mathcal{X}_r}\} < N$ where $N_{\mathcal{X}_l}$ and $N_{\mathcal{X}_r}$ are numbers of samples in $\mathcal{X}_l$ and $\mathcal{X}_r$, respectively.

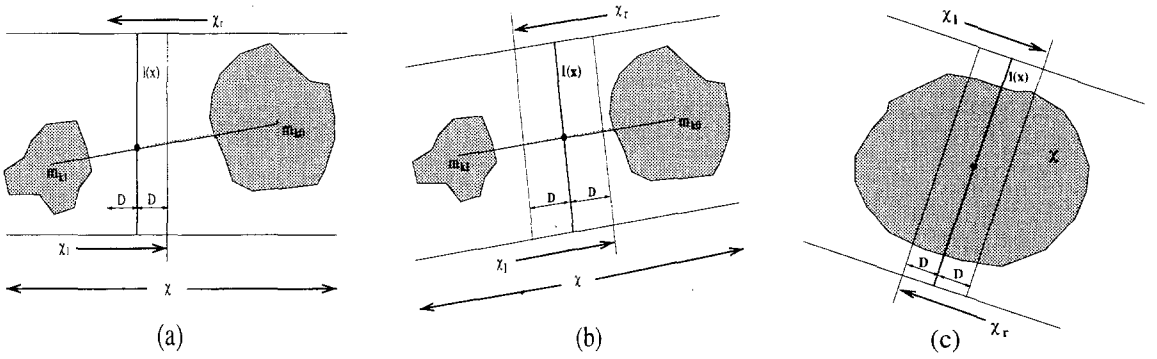

Fig. 3. A diagrammatic procedure demonstrates how the splitting rule works with different criteria. (a) criterion 1; (b) criterion 2; (c) criterion 3.

success and *failure*, respectively. To formally define these two conditions, first we introduce several thresholds† to two kinds of conditions. Let us denote I_T, E_T and $S_{\min}$ as the upper bound of epochs, Mean Square Error (MSE) of a subnetwork during training, and the lower bound of the number of samples in a training set for training the subnetwork, respectively. Thus, the success condition is defined as follows: $E_I \leq E_T$ if and only if $I \leq I_T$ or $N_{\mathcal{X}} \leq S_{\min}$, where I, E_I and $N_{\mathcal{X}}$ denote the epochs of training a subnetwork, the value of MSE of the subnetwork after I epochs and the number of samples in the training sets $\mathcal{X}$, respectively. Intuitively, the success condition means that the training set for a subnetwork is not partitioned into two smaller training subsets, and the node at which the subnetwork is located is one of leaf nodes in the tree generated. Furthermore, we introduced another threshold to the failure condition to speed up training by dividing a large task into several smaller subtasks prior to training any subnetwork. We denote $S_{\max}$ as the threshold of the maximal number of samples used to train a subnetwork. Accordingly, the failure condition is defined as follows: $I > I_T$ and $E_I > E_T$ and $N_{\mathcal{X}} > S_{\min}$ or $N_{\mathcal{X}} \geq S_{\max}$, where $N_{\mathcal{X}}$ is the number of samples in the training set $\mathcal{X}$. Intuitively, the failure condition means that the current training set must be divided into two smaller subsets, and the node with the training set is only a nonterminal node in an intermediate level of the generated tree (or only a hyperplane can be located at the node). Therefore, the stopping rule is defined as follows: when a subnetwork satisfies the success condition, it will reside at the leaf node and the growing process at the leaf node will stop; otherwise, the satisfaction of the failure condition results in the use of a splitting rule to continue the growing process. When the success condition is satisfied, two cases will still need considering further. If $I \leq I_T$ and $E_I \leq E_T$, training of the subnetwork on $\mathcal{X}$ terminates. If $N_{\mathcal{X}} \leq S_{\min}$, however, the training of the subnetwork on $\mathcal{X}$ is prolonged by iterating $K_{PT} \times I_T$ ($K_{PT} > 1$) epochs then stop, where K_{PT} is hereinafter called the *prolong-training factor*. Based upon both the splitting and stopping rules, the growing algorithm is summarised.

Growing Algorithm

1. **Initialisation.** Input the training set $\mathcal{T}$ corresponding to the given task. Set $\mathcal{X} \leftarrow \mathcal{T}$. Select a subnetwork architecture and an existing learning algorithm for training the subnetwork. Initialise the weight matrix of the subnetwork as W_0. Set the overlapping factor η and prolong-training factor K_{PT}. Set thresholds I_T, E_T, $S_{\min}$ and $S_{\max}$.
2. Let $N_{\mathcal{X}}$ denote the number of samples in $\mathcal{X}$. Use the splitting rule in Eq. (28) to partition $\mathcal{X}$ into $\mathcal{X}_l$ and $\mathcal{X}_r$ if $N_{\mathcal{X}} > S_{\max}$.
3. Set $\mathcal{X} \leftarrow \mathcal{X}_l$ and go to step 2 if $N_{\mathcal{X}} > S_{\max}$. Set $\mathcal{X} \leftarrow \mathcal{X}_r$ and go to step 2 if $N_{\mathcal{X}} > S_{\max}$.
4. For a training set $\mathcal{X}$, create a chosen subnetwork and train it on $\mathcal{X}$ using the chosen learning algorithm with the initial weight matrix W_0, until either the success condition or the failure condition is satisfied.
5. If the failure condition is satisfied, use the splitting rule in Eq. (28) to partition the current $\mathcal{X}$ into $\mathcal{X}_l$ and $\mathcal{X}_r$. Randomly perturb the weight matrix of the subnetwork at the current node on $\mathcal{X}$, and put the perturbing version of W_0 into W_l and W_r. Set $\mathcal{X} \leftarrow \mathcal{X}_l$ and $W_0 \leftarrow W_l$, then go to step 4. Set $\mathcal{X} \leftarrow \mathcal{X}_r$ and $W_0 \leftarrow W_r$, then go to step 4.

† One may also use other constraints beyond the thresholds mentioned here to define both success and failure conditions for more efficiently controlling the growing process for a specific problem.

6. If the success condition is satisfied, the subnetwork will reside at the leaf node. Moreover, stop the training of the subnetwork on $\mathcal{X}$ if $I \leq I_T$ and $E_I \leq E_T$; otherwise ($N_{\mathcal{X}} \leq S_{\min}$), continue to train the subnetwork by iterating $K_{PT} \times I_T$ epochs, then stop the training.
7. Repeat from step 4 to step 6 until all created subnetworks at the leaves of the tree satisfy the success condition.

For unknown data during testing, the output produced by the proposed tree-structured architecture could depend upon several subnetworks at the leaves of the tree, since the 'soft' method is adopted in the splitting rule. To draw the final result according to the outputs produced by subnetworks, we develop a credit-assignment algorithm. To serve for development of the algorithm, we first define two functions as

$$C_l(x) = \begin{cases} 1 & x < -D \\ \dfrac{D - x}{2D} & -D \leq x \leq D \\ 0 & x > D \end{cases} \tag{29}$$

and

$$C_r(x) = \begin{cases} 0 & x < -D \\ \dfrac{D + x}{2D} & -D \leq x \leq D \\ 1 & x > D \end{cases} \tag{30}$$

For Eqs (29) and (30), it is easy to show that $C_l(x) + C_r(x) = 1$. Thus, the credit-assignment is summarised.

Credit-Assignment Algorithm

1. **Initialisation.** Let $\mathbf{x}_u$ denote an unknown pattern for test. $\alpha = 1$ and `pointer` $\leftarrow$ `root`. $l(\mathbf{x}) = 0$ is the hyperplane which resides at the current nonterminal node pointed by the `pointer`.
2. If $l(\mathbf{x}_u) \leq D$, do $\alpha \leftarrow \alpha \times C_l[l(\mathbf{x}_u)]$ and `pointer` $\leftarrow$ `pointer` $\rightarrow$ `leftchild`.
3. If $l(\mathbf{x}_u) \geq -D$, do $\alpha \leftarrow \alpha \times C_r[l(\mathbf{x}_u)]$ and `pointer` $\leftarrow$ `pointer` $\rightarrow$ `rightchild`.
4. Repeat steps 2 and 3 until credits are assigned to all the subnetworks which $\mathbf{x}_u$ can reach.

The credit-assignment algorithm provides a way to assign credits to all the subnetworks at the leaves of the tree for unknown data. Suppose that N_u is the set of subnetworks at the leaves of the tree that an unknown pattern $\mathbf{x}_u$ can reach; the output of the tree-structured hybrid system, $O(\mathbf{x}_u)$, is

$$O(\mathbf{x}_u) = \sum_{i \in N_u} \alpha_i(\mathbf{x}_u) \times O_i(\mathbf{x}_u) \tag{31}$$

where $\alpha_i(\mathbf{x}_u)$ is the credit assigned to the ith subnetwork and $O_i(\mathbf{x}_u)$ is the result produced by the ith subnetwork ($i \in N_u$).

4. Simulations

This section presents simulation results on a variety of problems that have appeared in the literature. Most of them have been viewed as benchmarks in machine learning [46], and a function approximation problem has also been used to evaluate the performance of the proposed hybrid architecture. All of these problems were solved on a SUN Sparc II workstation, and programs were written in the C language. In simulations, three-layered MLPs were chosen as the architectures of subnetworks for use in the proposed hybrid architecture. To simplify the presentation, we denote a three-layered MLP with n_i input neurons, n_h hidden neurons and n_o output neurons as the MLP with n_i-n_h-n_o or MLP (n_i-n_h-n_o). For training MLPs in the growing algorithm, the Levenberg–Marquat method, a second-order algorithm [16,47], was employed for parameter estimation of all the subnetworks, instead of the standard Back-Propagation (BP) learning algorithm. For the proposed tree-structured hybrid architecture, we denote a generated tree with N_H nonterminal nodes and N_{MLP} terminal nodes as the tree with (N_H, N_{MLP}). The resulting tree-structured architecture for a specific task is hereinafter called the Modular Tree, denoted as MT (n_i-n_h-n_o) if the architecture of subnetworks are the MLP with n_i-n_h-n_o. In addition, we have conducted more than one simulation for each problem using different architectures of the subnetworks (or MLPs) to see if the performance of the proposed hybrid system is sensitive to the architectures of component neural networks or subnetworks. For the purpose of comparison, we have applied MLPs individually to all problems, and the two-fold *cross-validation* technique was used to select an appropriate architecture from multiple candidates for a specific problem, except that for the problem an appropriate architecture of the individual MLP has been suggested in the literature. For some problems, we have also applied classic methods, such as decision trees, to those problems for the purpose of comparison. In the sequel, we describe all the experimental results in detail.

Table 1. Classification of irises: generalisation ability of modular trees when the number of samples in the training set is 21. 'no. of errors' stands for number of classification errors and is averaged over five trials corresponding to five randomly chosen training sets with 21 samples. During training, thresholds in the growing algorithm are chosen as follows: $I_T = 8$, $E_T = 0{\cdot}03$, $S_{\max} = 20$, $S_{\min} = 9$ and the prolong-training factor $K_{PT} = 2{\cdot}0$. The architecture of subnetworks is the MLP with 4-3-3.

Overlapping factor η	0·3	0·4	0·5	0·6	0·7	0·8	0·9
No. of errors	5·4	5·0	5·0	4·8	4·8	4·6	7·0
architectures (N_H, N_{MLP})	(1,2)	(1,2)	(1,2)	(2,3)	(2,3)	(2,3)	(3,4)

Table 2. Classification of irises: generalisation ability of modular trees when the number of samples in the training set is 30. 'no. of errors' stands for number of classification errors and is averaged over five trials corresponding to five randomly chosen training sets with 30 samples. During training, thresholds in the growing algorithm are chosen as follows: $I_T = 10$, $E_T = 0{\cdot}03$, $S_{\max} = 29$, $S_{\min} = 9$ and the prolong-training factor $K_{PT} = 2{\cdot}0$. The architecture of subnetworks is the MLP with 4-3-3.

Overlapping factor η	0·3	0·4	0·5	0·6	0·7	0·8	0·9
No. of errors	4·0	4·2	4·2	4·2	4·6	5·2	7·0
architectures (N_H, N_{MLP})	(1,2)	(2,3)	(2,3)	(3,4)	(3,4)	(3,4)	(4,5)

Table 3. Classification of irises: generalisation ability of modular trees when the number of samples in the training set is 60. 'no. of errors' stands for number of classification errors and is averaged over five trials corresponding to five randomly chosen training sets with 60 samples. During training, thresholds in the growing algorithm are chosen as follows: $I_T = 10$, $E_T = 0{\cdot}02$, $S_{\max} = 59$, $S_{\min} = 12$ and the prolong-training factor $K_{PT} = 2{\cdot}5$. The architecture of subnetworks is the MLP with 4-3-3.

Overlapping factor η	0·3	0·4	0·5	0·6	0·7	0·8	0·9
No. of errors	4·2	4·8	4·4	4·6	3·6	4·6	5·2
architectures (N_H, N_{MLP})	(1,2)	(2,3)	(3,4)	(3,4)	(4,5)	(4,5)	(5,6)

4.1. Classification of Irises

The classification of irises is a famous benchmark problem in pattern recognition. Fisher used the data set in his classic paper on discriminant analysis [44], and the data set has since become a favourite example in pattern recognition [45]. Irises are classified into three categories: setosa, versicolor and virginica. Each category consists of 50 samples. Each sample possesses four attributes: sepal length, sepal width, petal length and petal width. In experiments, a subset of data was randomly chosen for training, and the remaining data were used for testing. The generalisation ability of a modular tree was evaluated by mean prediction error. For reliability, we randomly selected five subsets of data as training sets for a specific number of samples. An MLP with 4-3-3 was first chosen as the architecture of subnetworks in the proposed tree-structured hybrid architecture, and different overlapping factors in the splitting rule were also investigated in the experiments. Tables 1, 2 and 3 show the generalisation capabilities of the resulting modular trees in terms of different overlapping factors when the number of samples is 21, 30 and 60 in the training sets, respectively. It is obvious that the performance of a resulting modular tree is highly influenced by the overlapping factor in the splitting rule. We also report results produced by the individual MLP with 4-4-3, the method of structural learning with forgetting viewed as a method which can yield better generalisation than the standard BP algorithm [48], as well as the proposed method for comparison, in Table 4. In the same table, we also report the results produced by modular trees in which the architecture of subnetworks was the MLP with 4-4-3 (other

Table 4. Classification of irises: generalisation ability. 'no. of errors' stands for the number of classification errors for test data and is the average over five trials starting from different initial connection weights. SLF (4-4-3) and MLP (4-4-3) respectively denote the method of structural learning with forgetting of the MLP with 4-4-3 and the individual MLP with 4-4-3. The overlapping factors used for generating modular trees are 0·8, 0·3 and 0·7 when numbers of the training sets are 21, 30 and 60, respectively.

No. of samples		No. of errors			
Training	Test	SLF (4-4-3)	MLP (4-4-3)	MT (4-3-3)	MT (4-4-3)
21	129	6·8	16·2	4·6	4·6
30	120	5·0	6·4	4·0	4·2
60	90	5·2	4·8	3·6	3·6

Table 5. Classification of iris: averaging CPU time of training different architectures for all experiments. 'no. of samples' denotes the number of samples in a training set. (unit: second.)

No. of samples	MLP (4-4-3)	SLF (4-4-3)	MT (4-3-3)	MT (4-4-3)
21	54·8	86·4	21·6	22·8
30	61·5	103·8	23·6	24·7
60	106·8	163·4	29·8	31·3

parameters in the growing algorithm remained unchanged for generating modular trees), in order to investigate whether the performance of resulting modular trees is influenced by different architectures of the subnetworks. All averaging training times for the different methods are listed in Table 5. According to Table 4 and 5, modular trees generalise better than the individual MLP, as well as the method of structural learning with forgetting, and yields faster training. From Table 4, in particular, the use of different architectures of the subnetworks in modular trees yields a quite similar generalisation performance. This implies that the performance of the modular tree is insensitive to the architecture of its subnetworks (MLPs) for the classification of iris problem.

4.2. Two Spirals Problem

A well-known benchmark in the neural network community is the so-called two spirals problem, as illustrated in Fig. 4. It consists of 194 two-dimensional vectors lying on two interlocked spirals, which are the classes in this case. The task is to construct a classifier which can distinguish between the two classes. The benchmark is interesting because, due to the low data dimensionality, it is possible to visualise the decision regions of the network during and after training. Moreover, it seems to be a rather difficult task for typical feedforward neural networks (e.g. MLPs with sigmoidal activation functions). Lang and Witbrock [49] could not solve the problem with a standard MLP, and had to use additional connections to achieve convergence. Fahlman and Lebiere [19] used a constructive learning algorithm to solve the problem successfully. The problem has since been popular in the neural network community, and has been extensively used for evaluating both nonlinear separability and generalisation ability of a neural architecture. Here, we use the benchmark to evaluate the performance of the proposed tree-structured architecture.

In simulations, all parameters used in the growing algorithm are as follows: $\eta = 0{\cdot}08$, $K_{PT} = 3$, $I_T = 10$, $E_T = 0{\cdot}01$, $S_{\max} = 90$ and $S_{\min} = 20$. We adopt MLPs with 2-2-1 and 2-3-1 as the architectures of the subnetworks, respectively, for two independent simulations. The resulting decision regions of modular trees in which subnetworks are MLPs with 2-2-1 and 2-3-1 are, respectively, shown in Figs 5 and 6. The resulting decision regions produced by two modular trees with different architectures of subnetworks are slightly different. For the two spirals problem, it implies that the performance of modular trees seems insensitive to the architectures of the subnetworks. Furthermore, it is worth noting that the resulting modular trees with different architectures of

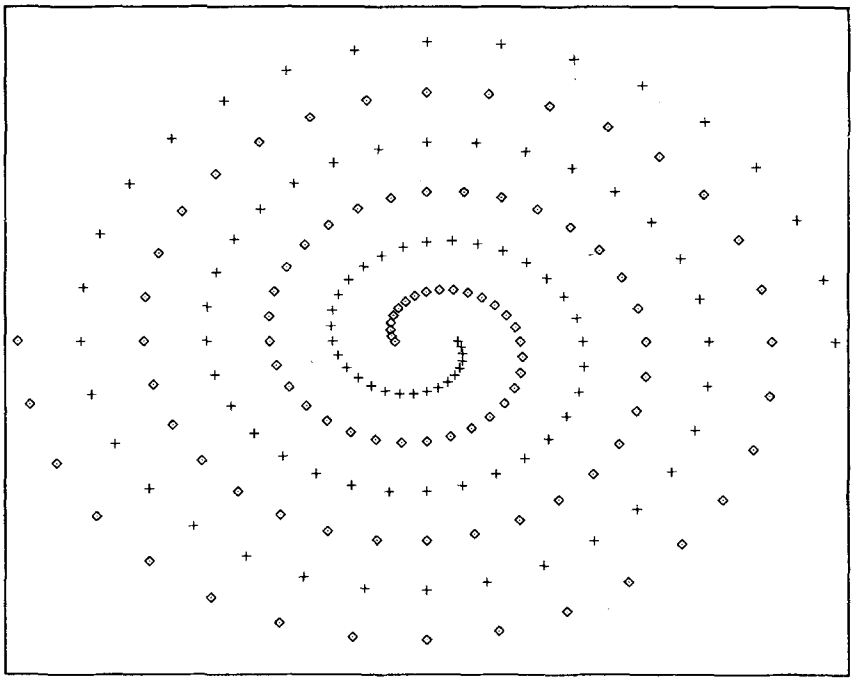

Fig. 4. The two spirals problem (training data).

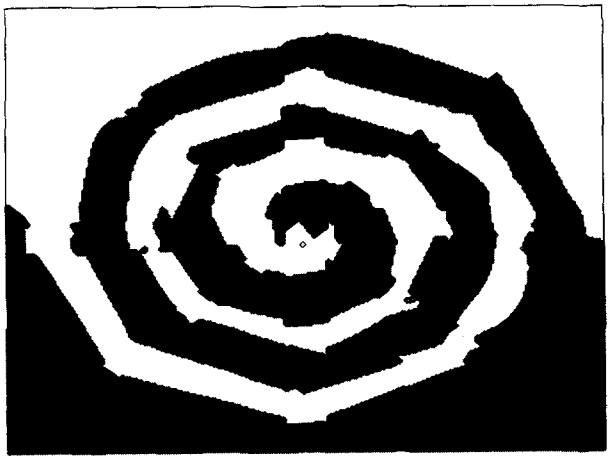

Fig. 5. Decision regions produced by the Modular Tree (2-3-1) for the two spirals problem.

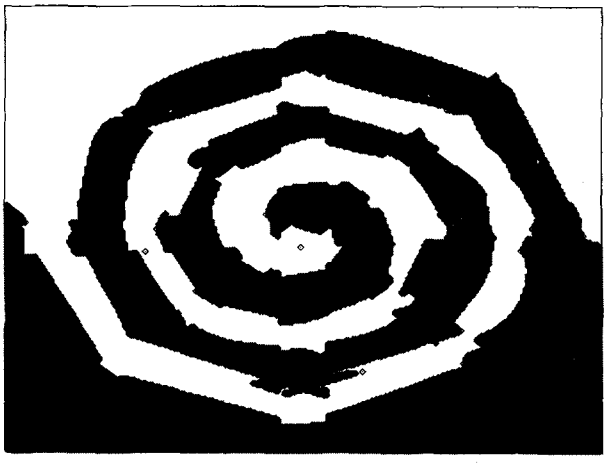

Fig. 6. Decision regions produced by the Modular Tree (2-2-1) for the two spirals problem.

subnetworks share the same architecture, which is depicted in Fig. 7. To visualise the process of growing a modular tree during training, we diagrammatically show how the input space is split up in Fig. 8. In addition, we have also applied the MLP with 2-5-5-5-1 and additional connections suggested by Lang and Witbrock [49] and the cascade correlation achitecture [19] to the two spirals problems on the same workstation. As a result, the CPU time of training modular trees and those architectures are listed in Table 6 for comparison. Although both the special MLP and the cascade correlation can also produce approximately correct resulting decision regions, the modular trees yield faster training.

4.3. Waveform Recognition Problem

The synthetic waveform recognition problem was first introduced [22] to study the behaviour of Classification And Regression Tree (CART). It is a three-class problem based on the waveforms $h_1(t)$, $h_2(t)$ and $h_3(t)$, depicted in Fig. 9. Each class is a random convex combination of two of these waveforms. The

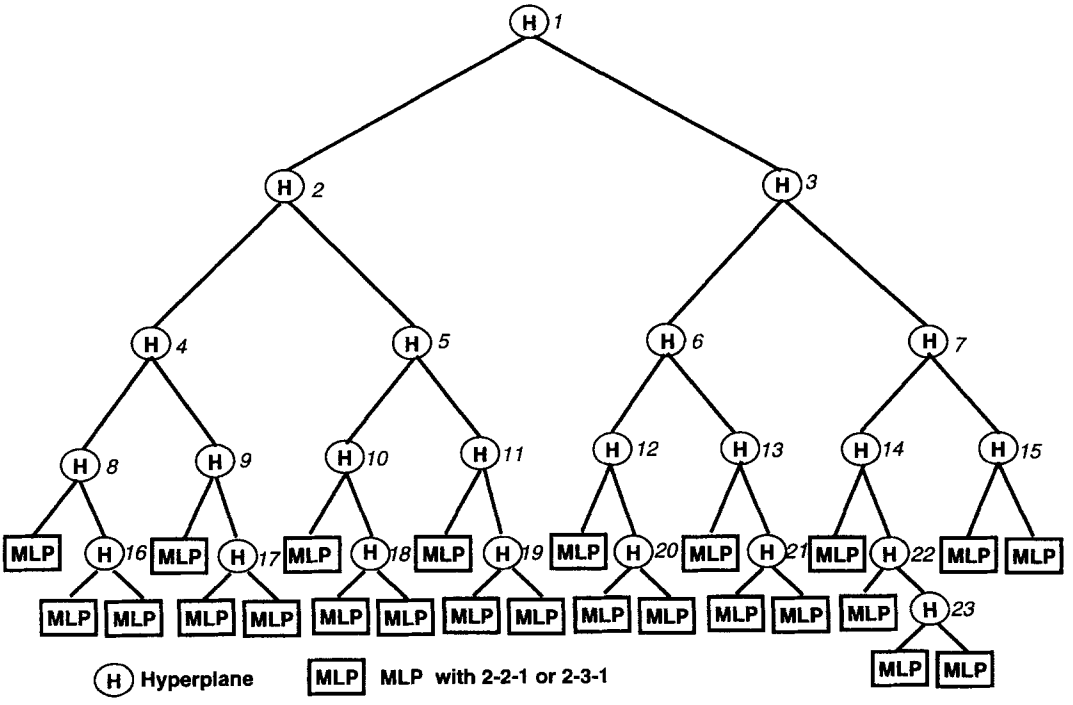

Fig. 7. The modular tree (subnetworks are MLPs with either 2-3-1 or 2-2-1) for the two spirals problem. Each label corresponds to a specific hyperplane used for partitioning input space (see Fig. 8).

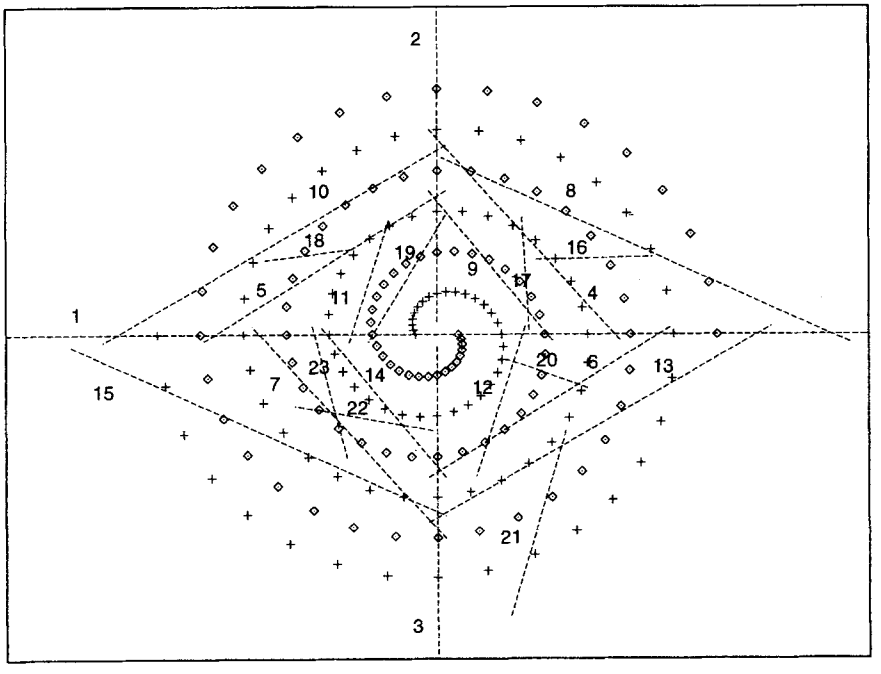

Fig. 8. The process of partitioning input space for the two spirals problem and each labelled line segment corresponds to a hyperplane in the modular tree (the resulting modular tree structure is illustrated in Fig. 7). Note: the overlapping region between any two adjacent clusters is not depicted in the figure.

Table 6. The two spirals problem: CPU time of training different architectures. (unit: second.)

	MLP (2-5-5-5-1)	Cascade-Correlation	MT (2-2-1)	MT (2-3-1)
CPU time	3429	236	112	116

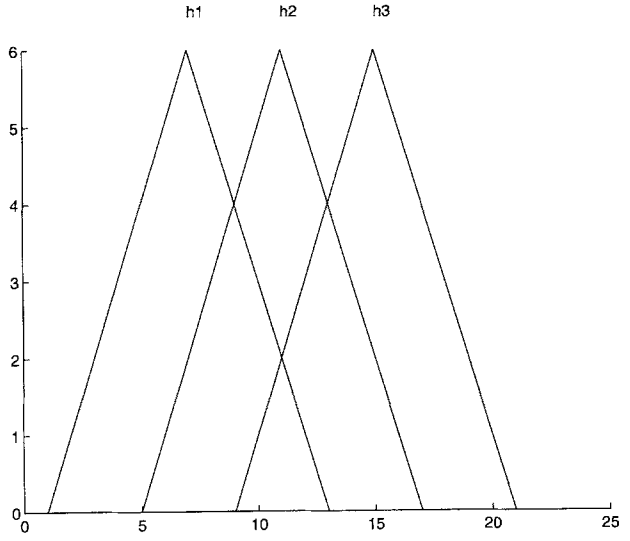

Fig. 9. Three basic waveforms in the waveform recognition problem.

pattern vector is obtained by sampling 21 points and adding noises. Hence, the components of the pattern vector are given as follows:

For class 1,

$$x_i = uh_1(i) + (1-u)h_2(i) + \epsilon_i\,,$$
$$i = 1,\ldots,21$$

For class 2,

$$x_i = uh_1(i) + (1-u)h_3(i) + \epsilon_i\,,$$
$$i = 1,\ldots,21$$

For class 3,

$$x_i = uh_2(i) + (1-u)h_3(i) + \epsilon_i\,,$$
$$i = 1,\ldots,21$$

Here u is a uniform random variable on the interval [0, 1], and $\epsilon_1,\ldots,\epsilon_{21}$ are independent Gaussian random variables with zero mean and unit variance. The three classes have equal *a priori* probabilities. Breiman *et al.* [22] reported that the Bayesian misclassification rate for this problem is approximately 14%.

In simulations, we randomly produced seven independent training sets ranging in size from 500 to 2000 samples. For each training set, a modular tree with the specific architecture of subnetworks was generated. During testing, an additional set of 5000 independent samples was employed to obtain the error rate, so that the performance of modular trees generated on distinct training sets can be respectively evaluated. In experiments, the architecture of the subnetworks was chosen as the MLP with either 21-12-3 or 21-15-3, and other parameters used in the growing algorithm are as shown in Table 7. As a result, 14 architectures of the resulting modular trees are shown in Table 8, and the error rates produced by the resulting modular trees are illustrated in Fig. 10. It is evident from Fig. 10 that modular trees with different architectures of the subnetworks yield a performance similar to the waveform recognition problem. For comparison, we also illustrate testing results produced by the resulting modular trees with MT (21-12-3), CART with two different splitting rules [22] and an individual four-layered MLP consisting of 21 neurons in the input layer, 20 neurons in the first hidden layer, five neurons in the second hidden layer and three neurons in the output layer [34] in Fig. 11. According to Fig. 11, it is shown that modular trees outperform CART in all cases, and the four-layered MLP when the number of samples in the training set is 1000, 1250, 1500 and 1750. In addition, the CPU times of training the individual MLP and modular trees are shown in Table 9 for comparison. It is evident from this table that the proposed method yields significantly faster training than the individual MLP for the problem, though the Levenberg–Marquat learning algorithm was used for training both the individual MLP and subnetworks of modular trees.

4.4. Speaker Independent Vowel Recognition

To explicitly investigate the generalisation capability of the proposed method, we performed experiments with a speaker independent vowel recognition problem. The data used was collected by Deterding [50], who recorded examples of the 11 steady-state vowels of English spoken by 15 speakers for a speaker normalisation study. Eleven words including 11 vowel sounds were recorded, and each word was uttered once by each of the 15 speakers, seven of

Table 7. The waveform recognition problem: parameters used in the growing algorithm and architectures of resulting modular trees. 'no. of samples' stands for the number of samples in a training set.

No. of samples	500	750	1000	1250	1500	1750	2000
Overlapping factor η	0·9	0·6	0·6	0·6	0·4	0·4	0·3
Prolong-training factor K_{PT}	3·0	2·5	2·5	2·5	2·0	2·0	2·0
Threshold I_T	20	20	20	20	20	20	20
Threshold E_T	0·4	0·4	0·4	0·4	0·4	0·4	0·4
Threshold $S_{\max}$	450	700	800	900	800	950	1100
Threshold $S_{\min}$	50	75	100	125	150	175	200
Architectures (N_H, N_{MLP})	(1,2)	(1,2)	(2,3)	(6,7)	(5,6)	(6,7)	(5,6)

Table 8. The waveform recognition problem: resulting structures of modular trees on use of different architecture of subnetworks.

No. of samples	500	750	1000	1250	1500	1750	2000
MT (21-12-3)	(1,2)	(1,2)	(2,3)	(6,7)	(5,6)	(6,7)	(5,6)
MT (21-15-3)	(1,2)	(1,2)	(2,3)	(5,6)	(5,6)	(6,7)	(6,7)

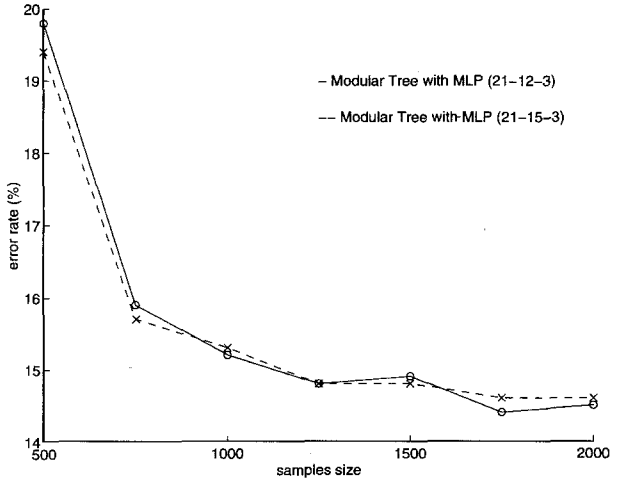

Fig. 10. Error rates of modular trees with different architecture of subnetworks on the waveform recognition problem. Each 'o' or 'x' corresponds to the result produced by a generated modular tree trained on an independent training set.

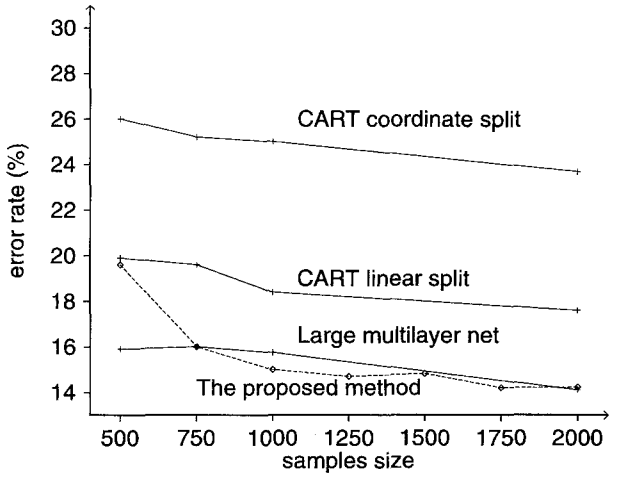

Fig. 11. Error rates versus samples size for CART method, a large individual MLP with 21-20-5-3 and the Modular Tree (21-17-3) on the waveform recognition problem.

whom were female and eight male. The speech signals were low pass filtered at 4.7 kHz and then digitised to 12 bits with a 10 kHz sampling rate. 12-order linear predictive analysis was carried out on six 512-sample Hamming windowed segments from the steady part of the vowel. The reflection coefficients were used to calculate 10 log area parameters, giving a 10-dimensional input space. Each speaker thus yielded six frames of speech from 11 vowels. This gave 990 frames from the 15 speakers.

Robinson [51] used this data to investigate several types of neural network algorithms and classic classifiers. He used 528 frames from four male and four female speakers to train the networks, and the remaining 462 frames from four male and three female speakers to test the performance. The classifiers he examined were single-layer perceptrons, multilayer neural networks with sigmoidal, Gaussian, and quadratic activation functions, a modified Kanerva model, radial basis networks, and also a conventional method, the nearest-neighbour classifier.

In our experiments, the architecture of the subnetworks was chosen as the MLP with either 10-18-11 or 10-20-11, and all the parameters used in the

Table 9. The waveform recognition problem: CPU time of training the individual MLP and modular trees. (unit: second.)

No. of samples	500	750	1000	1250	1500	1750	2000
MLP (21-20-5-3)	793	1123	1494	1981	2087	2385	2698
MT (21-12-3)	380	459	588	673	715	818	869
MT (21-15-3)	414	466	601	659	732	826	893

Table 10. The speaker independent vowel recognition problem: parameters used in the growing algorithm.

Overlapping factor η	0·2	0·3	0·4	0·5	0·55	0·6	0·7
Prolong-training factor K_{PT}	3·0	3·0	3·0	3·0	3·0	3·0	3·0
Threshold I_T	12	12	12	12	12	12	12
Threshold E_T	0·11	0·11	0·11	0·11	0·11	0·11	0·11
Threshold $S_{\max}$	320	320	3200	320	320	320	320
Threshold $S_{\min}$	30	30	30	30	30	30	30

Table 11. The speaker independent vowel recognition problem: architectures of resulting modular trees, (N_H, N_{MLP}), with different architectures of subnetworks.

Overlapping factor η	0·2	0·3	0·4	0·5	0·55	0·6	0·7
MT (10-18-11)	(4,5)	(4,5)	(4,5)	(5,6)	(6,7)	(6,7)	(10,11)
MT (10-20-11)	(4,5)	(4,5)	(4,5)	(5,6)	(5,6)	(6,7)	(9,10)

growing algorithm are listed in Table 10. We generated several modular trees corresponding to different overlapping factors with the same data used by Robinson [51], and thereafter used his test data to evaluate the generalisation ability of modular trees. As a result, all the architectures of the resulting modular trees are shown in Table 11, and the error rates produced by the modular trees corresponding to different overlapping factors are illustrated in Fig. 12. The architecture of the modular tree, MT (10-18-11), producing the best result (corresponding to the overlapping factor $\eta = 0{\cdot}55$) is illustrated in Fig. 13. In addition, the CPU times of generating modular trees with different overlapping factors are listed in Table 12, and both the results in Robinson [51] and ours are shown in Table 13 for comparison. It is evident from the simulation that the proposed method outperforms the classical classifiers. We could conjecture that the proposed method also yields significantly faster training than MLPs, since Robinson reported that the training of an MLP for

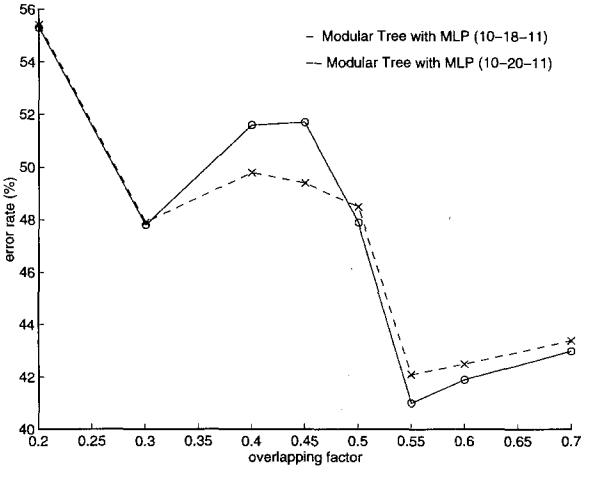

Fig. 12. Error rates of modular trees with different architecture of subnetworks on the vowel recognition problem in terms of different overlapping factors in the splitting rule. Each 'o' or 'x' corresponds to the result produced by a generated modular tree trained using a specific overlapping factor.

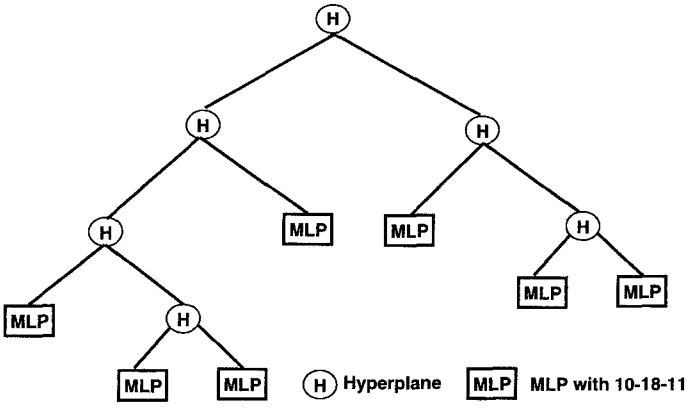

Fig. 13. The generated modular tree corresponding to the recognition rate 59% (the overlapping factor is $\eta = 0{\cdot}55$ in the splitting rule) for vowel recognition problem.

Table 12. The speaker independent vowel recognition problem: CPU time of generating modular trees corresponding to different architectures of subnetworks in terms of different overlapping factors in the splitting rule. (unit: second.)

Overlapping factor η	0·2	0·3	0·4	0·5	0·55	0·6	0·7
MT (10-18-11)	688	723	755	805	911	1022	1491
MT (10-20-11)	704	789	822	879	901	1043	1372

Table 13. The speaker independent vowel recognition problem: test results of different methods for comparison. The table shows the network size, the number of correctly classified test patterns (out of 462), and the corresponding percentages. The modular tree is MT (10-18-11) with overlapping factor $\eta = 0{\cdot}55$ in the splitting rule (the architecture is illustrated in Fig. 13).

Classifier	No. of hidden neurons	Correctly classified	Percent correct
Single-layer perceptron	–	154	33
Multilayer perceptron	88	234	51
Multilayer perceptron	22	206	45
Multilayer perceptron	11	203	44
Modified Kanerva model	528	231	50
Modified Kanerva model	88	197	43
Radial basis function	528	247	53
Radial basis function	88	220	48
Gaussian node network	528	252	55
Gaussian node network	88	247	53
Gaussian node network	22	250	54
Gaussian node network	11	211	47
Square node network	88	253	55
Square node network	22	236	51
Square node network	11	217	50
Nearest neighbour	–	260	56
Modular tree ($\eta = 0{\cdot}55$)	18	271	59

the problem took such a long time that the training had to be terminated once a threshold of iterations had been reached.

4.5. Image Segmentation

The image segmentation data was collected by Brodley at the University of Massachusetts, and has become a benchmark for machine learning in the UCI Repository of machine learning database [46]. The instances were drawn randomly from a database of seven outdoor images. The images were manually segmented to create a classification for every pixel. Each instance is a 3×3 region. The feature vector consists of 19 continuous attributes associated with region, density, contrast and intensity, etc. All the data were classified into seven categories: brickface, sky, foliage, cement, window, path and grass. In the database [46], all data have been explicitly divided into two sets, i.e. a training set and a test set. There are 210 samples (30 instances/class) in the training set, and are 2100 samples (300 instances/class) in the test set. According to the statement on the data [46], no result on the data has been published yet.

We applied the proposed method to the image segmentation problem to evaluate the generalisation ability of the resulting modular trees. In our experiments, the architecture of the subnetworks was chosen as the MLP with either 19-17-7 or 19-21-7. As a result, all parameters used in the growing algorithm are listed in Table 14, and the architectures of the resulting modular trees based upon different overlapping factors in the splitting rule are also shown in Table 15. The testing results of these modular trees corresponding to different overlapping factors are shown in Fig. 14. According to these testing results, it has been demonstrated that modular trees with different architectures of the subnetworks yield a similar performance. We also performed some experiments on the use of individual MLPs with the Levenberg–Marquat learning algorithm to solve the same problem. For comparison, the results of some modular trees and individual MLPs are shown in Table 16, and the training time is accordingly listed in Table 17. It is evident from the simulation results that modular trees yield a better generalisation and faster training than the MLPs.

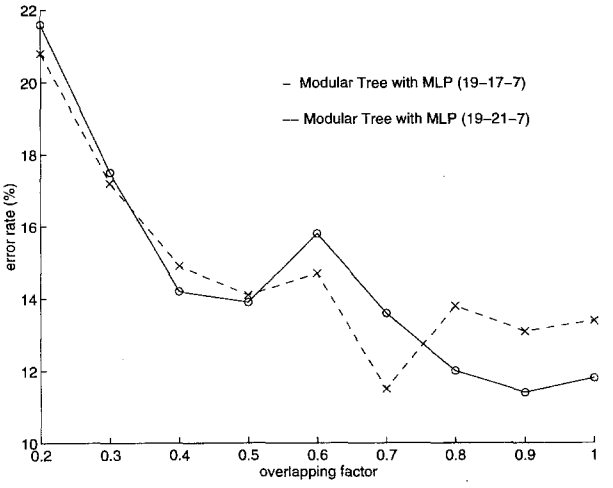

Fig. 14. Error rates of modular trees with different architecture of subnetworks on the image segmentation problem in terms of different overlapping factors in the splitting rule. Each 'o' or 'x' corresponds to the result produced by a generated modular tree trained using a specific overlapping factor.

Table 14. The image segmentation problem: parameters used in the growing algorithm.

η	0·2	0·3	0·4	0·5	0·6	0·7	0·8	0·9	1·0
K_{PT}	3·0	3·0	3·0	3·0	3·0	3·0	3·0	3·0	3·0
I_T	6	6	6	6	6	6	6	6	6
E_T	0·18	0·18	0·18	0·18	0·18	0·18	0·18	0·18	0·18
$S_{\max}$	100	100	100	100	100	100	100	100	100
$S_{\min}$	20	20	20	20	20	20	20	20	20

Table 15. The image segmentation problem: architectures of resulting modular trees, (N_H, N_{MLP}), with different architectures of subnetworks.

η	0·2	0·3	0·4	0·5	0·6	0·7	0·8	0·9	1·0
MT (19-7-7)	(11,12)	(6,7)	(13,14)	(21,22)	(22,23)	(63,64)	(76,77)	(157,158)	(430,431)
MT (19-21-7)	(10,12)	(5,6)	(12,13)	(20,21)	(21,22)	(59,61)	(72,73)	(148,149)	(423,425)

Table 16. The image segmentation problem: test results of MLPs and the resulting modular trees. The table shows the network architecture, the number of correctly classified test patterns (out of 2100), and the corresponding percentages. The architecture of subnetworks in resulting modular trees is the MLP with either 19-17-7 or 19-21-7.

Classifier	Architecture	Correctly classified	Percent correct
Three-layered MLP (1)	19-17-7	1735	82·6
Three-layered MLP (2)	19-28-7	1770	84·3
Three-layered MLP (3)	19-50-7	1712	81·5
Four-layered MLP (4)	19-22-10-7	1787	85·2
MT (19-7-7) ($\eta = 0{\cdot}9$)	(157,158)	1862	88·6
MT (19-21-7) ($\eta = 0{\cdot}7$)	(59,61)	1859	88·5

Table 17. The image segmentation problem: CPU time of training four individual MLPs and some modular trees listed in Table 16. (unit: minute.)

	MLP (1)	MLP (2)	MLP (3)	MLP (4)	MT (19-17-7)	MT (19-21-7)
CPU time	78·5	89·3	99·4	218·2	33·8	26·4

4.6. Function Approximation

It is well known that an MLP with a sigmoidal activation function can perform the universal approximation of any continuous multivariate function to any desired degree of accuracy, provided that sufficiently many hidden neurons are available [12,52,53]. To evaluate the universal approximation ability of the proposed method, we performed an experiment by learning a multivariate function approximation task. To visualise the results, we selected a function as

$$f(x,y) = (x^2 - y^2) \sin \frac{x}{2}, \quad -10 \leq x,y \leq 10 \tag{32}$$

In the experiment, we used a training set with 625 samples to learn the mapping. All parameters used in the growing algorithm, architectures of both subnetworks and resulting modular trees are listed in Table 18. Obviously, the two resulting modular trees with different architectures of subnetworks share the same architecture. To exactly evaluate the generalization ability of the modular tree, we used three data sets with 1600, 2500 and 4489 samples for testing, respectively. Two modular trees produce very similar testing results on all three testing data sets. Due to the limited space, we merely show the results produced by the modular tree, MT (2-2-1). As a result, the data in the training set is shown in Fig. 15(a) for reference, and the testing results produced by the resulting modular tree on different testing data sets are respectively shown in Figs 15(b)–(d). It is evident from the simulation that the modular tree can perform the universal approximation task very well in multiple scales. For comparison, we also employed an individual five-layered MLP (two input neurons, three neurons in the first hidden layer, five neurons in the second hidden layer, three neurons in the third hidden layers and one output neuron) along with the Levenberg–Marquat learning algorithm to deal with the same

Table 18. Function approximation: parameters used in the growing algorithm and architectures of modular trees. The architecture of subnetworks in resulting modular trees is the MLP with either 2-2-1 or 2-3-1.

η	K_{PT}	I_T	E_T	$S_{\max}$	$S_{\min}$	MT (2-2-1)	MT (2-3-1)
0·2	3	10	0·005	200	20	(55,56)	(55,56)

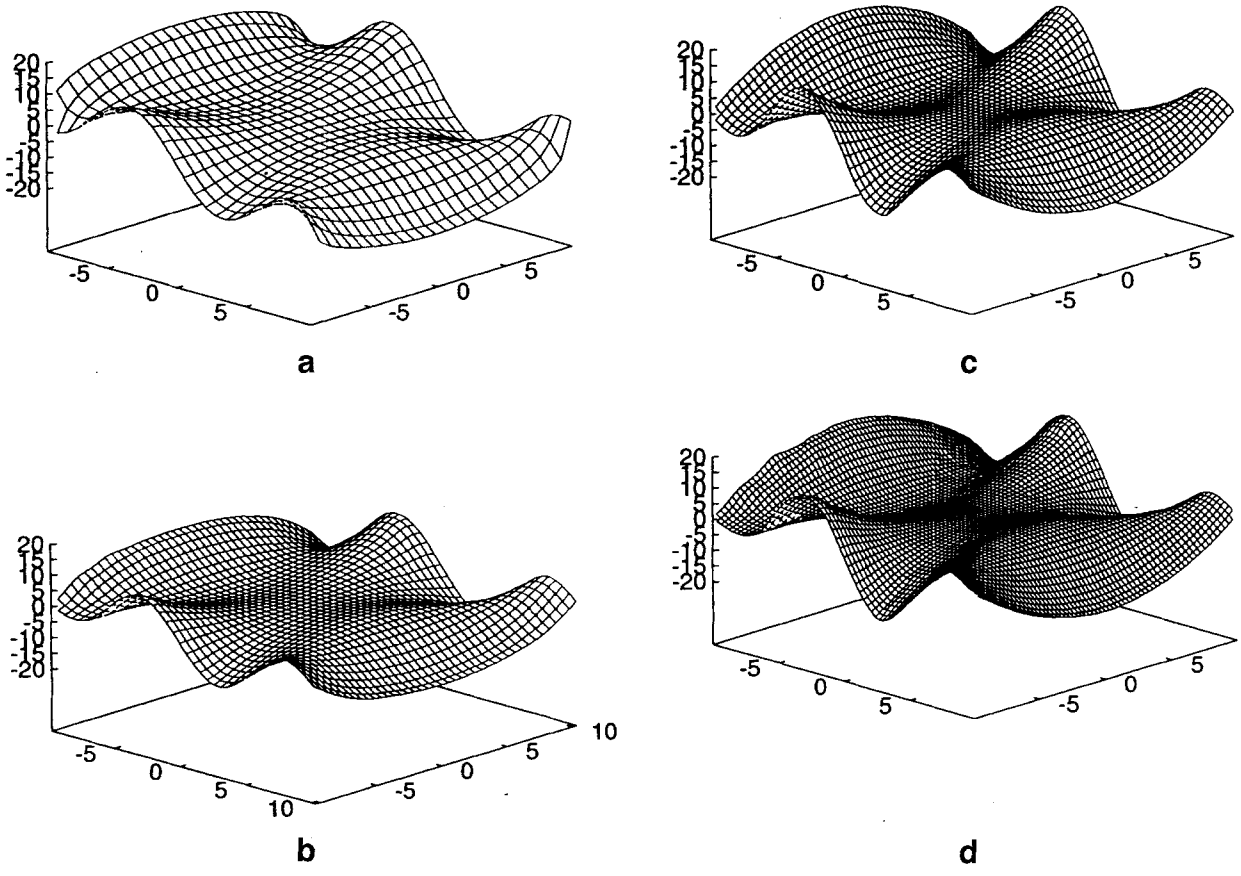

Fig. 15. The results of universal approximation on the function in Eq. (32) ($-10 \leq x,y \leq 10$). (a) the surface produced by the function in Eq. (32) on the training data (625 samples); (b) the resulting surface produced by the modular tree on test data (1600 samples); (c) the resulting surface produced by the modular tree on test data (2500 samples); (d) the resulting surface produced by the modular tree on test data (4489 samples).

problem. The CPU time of the training modular trees and the MLP are listed in Table 19. It is obvious from the table that the proposed method yields significantly faster training than the MLP, though both can yield satisfactory performance for universal approximation.

5. Discussions and Conclusions

We have described a novel method for supervised learning by combining linear discriminant functions and neural networks. The extensive simulation results have shown that the self-generated tree-structured hybrid system outperforms MLPs for several benchmark problems of classification, and a problem of universal approximation. In particular, the proposed method yields significantly faster training. The application of the proposed method in a real world problem called *speaker recognition* has been already investigated [54,55]. It has also shown that the proposed method is a more effective way than other classic methods to solve a large scale problem [54].

The basic idea underlying the proposed method is the use of the principle of divide-and-conquer. The issue is worth discussing here further. As stated previously, linear discriminant functions play a crucial role for dividing a large or complicated problem into simpler or smaller problems, while neural networks serve to solve those simpler and smaller problems. In the splitting rule, three types of linear discriminant functions could be used for the purpose. The Fisher or normal density related linear discrimi-

Table 19. Function approximation: CPU time of training an individual MLP and modular trees. (unit: second.)

	MLP (2-3-4-3-1)	MT (2-2-1)	MT (2-3-1)
CPU time	3892	227	236

nant functions could result in an optimal or sub-optimal partition to distinguish between the data that belong to two different classes, and the Euclidean distance between their centroids is furthest among all data in a given classification task, while the linear discriminant function in Eq. (27) intends to partition all data into two sets with an almost equal size. Obviously, what these linear discriminant functions do is either reduce the complexity of the original problem, or transfer the original problem into two smaller problems. As theoretically and empirically shown previously, the training time of an MLP often increases exponentially with the size of the problem [14,56,57]. Thus, a real world problem (e.g. image processing) will often be intractable when the MLP is used directly. In our growing algorithm, linear discriminant functions first partition a large problem into several smaller problems prior to the training of MLPs or subnetworks, in order to limit the scale of the problems to a solvable extent for those MLPs or subnetworks. As shown previously, each linear discriminant function is analytically determined according to the given training data. Thus, the partition of a large scale problem can be rapidly available, and all MLPs or subnetworks merely need to independently solve multiple small scale problems simultaneously. Moreover, the growing algorithm has taken the size of the self-generated hybrid architecture into consideration by using a threshold such that training samples could be sufficient for all subnetworks for generalisation. On the other hand, it has been well known that an MLP suffers from serious catastrophic interference that later training disrupts the results of previous training. Fortunately, the problem, to a great extent, can be alleviated in the hybrid system during both training and updating by distributing data to multiple subnetworks. It is also worth mentioning that any improved technique for training an MLP could be expected to improve the performance of the hybrid system if such an MLP, along with the improved technique, is used as a subnetwork of the hybrid system. As a result, the proposed method provides a novel way to use MLPs to solve a large scale problem.

It might be interesting to consider the difference between the proposed architecture and other architectures associated with the use of the principle of divide-and-conquer. As described previously, two architectures might be relevant to the proposed architecture. The decision tree and its variants are a kind of typical architectures to use the principle of divide-and-conquer for dealing with a problem [22]. The basic characteristics of the decision tree might be summarised as follows: (1) uniform apparatus are used for both dividing and conquering a problem; (2) 'hard' partition way is adopted; and (3) each leaf node is only associated often with a class label. As for the proposed method, different apparatus (i.e. a linear discriminant function and neural network) could be used in nonterminal and leaf nodes to serve for different purposes in the principle of divide-and-conquer, respectively. Instead of a 'hard' partition, a 'soft' partition is adopted to 'divide' a problem. As a result, data belonging to different classes could reach the same leaf node at which a neural network associated with multiple class labels is located. All of these characteristics might distinguish the proposed method from the decision tree and its variants. It is worth pointing out that the 'soft' partition method plays an important role in the proposed method. From the standpoint of computational geometry, Minsky and Papert [57] have shown that some problems (e.g. connectedness) cannot be computed at all in parallel by a diameter-limited or order-limited perceptron; moreover, an MLP seems to encounter the same difficulty [57]. A salient reason is that such problems are solvable only if the global information is available, while an order-limited perceptron can only capture local information. The 'soft' partition in the proposed method provides a way of combining multiple order-limited neural networks such that the local information can be accumulated to achieve the global information by means of communication among those neural networks. Our credit-assignment algorithm is designed just for the purpose by incorporating the 'soft' partition method. The application of the proposed method in the two spirals problem, which belongs to a problem of computing connectedness, has demonstrated the usefulness of the 'soft' partition method for such a problem. On the other hand, the HME is a modular neural network architecture based upon the principle of divide-and-conquer [40]. In contrast to the stacked generalisation [58], which makes explicit partitions of the input space, the HME preferentially weights the input space by the posterior probabilities that experts generated the output from the input. The outputs of expert networks are combined by gating networks which are simultaneously trained to stochastically select the expert which is performing best at solving the problem. There are at least two points in distinguishing the proposed method from the HME architecture. One is that the HME also suffers from the problem of determining a structure prior to training like an MLP, while the proposed method might automatically generate a structure for a given problem. The other is that the 'soft' partition method in the HME is different from that used in the

proposed method. Different 'soft' partition methods lead to two different training procedures; all samples in the training set must be fed to every expert and gating network in the HME, while only a part of the samples in the training set can reach a subnetwork at the leaf node in the proposed method during training.

There are still some open problems in the proposed method. Simulation results have shown that the performance of a self-generated hybrid system seems insensitive to the architectures of its subnetworks. Instead of an empirical study, a theoretical analysis on the issue will be of significance and should be underway. Simulations have also indicated that the performance of a hybrid system is highly influenced by the size of an overlapping region between two adjacent data sets split by a hyperplane. How to determine the optimal size of an overlapping region will be another important issue to be studied. Since a specific overlapping region could finally determine the training set of a subnetwork at the leaf node, the architecture of a subnetwork should also depend upon the size of an overlapping region. Therefore, there is an intrinsic relation between both issues. We expect that solutions to these problems will provide a way to significantly improve the performance of the self-generated hybrid architecture in the future.

Acknowledgements. We wish to thank Liping Yang for valuable and constructive discussions, as well as for providing a program on the Levenberg–Marquat algorithm for simulation. We also wish to thank DeLiang Wang and the anonymous reviewers, whose extensive comments have significantly improved the presentation of the paper. This work was in part supported by Chinese National Nature Science Foundation Grant 69571002, and Grant 69475007 as well as the NSF Grant IRI-9423312.

References

1. Bishop M. Neural Networks for Pattern Recognition. Oxford University Press, 1995.
2. Cohen M, Franco H, Morgan N, Rumelhart D, Abrash V. Context-dependent multiple distribution phonetic modeling with MLPs. In: SJ Hanson, JD Cowan, CL Giles (eds.), Advances in Neural Information Processing Systems. Morgan Kaufmann, 1993, pp. 649–657.
3. Gyuyon I, Albrecht P, LeCun Y, Denker J, Hubbard W. Applications of neural networks to character recognition. Int J Pattern Recognition and Artificial Intelligence 1991; 5: 353–382.
4. Haykin S, Deng C. Classification of radar clutter using neural networks. IEEE Trans Neural Networks 1991; 2: 589–600.
5. LeCun Y, Boser B, Denker JS, Henderson D, Howard RE, Hubbard W. Handwritten digit recognition with a back-propagation network. In: DS Touretsky, (ed.), Advances in Neural Information Processing Systems. Morgan Kaufmann, 1990, pp. 396–404.
6. Narendra KS, Parthasarathy K. Indentification and control of dynamical systems using neural networks. IEEE Trans Neural Networks 1990; 1: 4–27.
7. Pomerleau DA. Neural network perception for mobile robot guidance. PhD Thesis, School of Computer Science, Carnegie Mellon University, 1992.
8. Rajavelu A, Musavi M, Shivaikar M. A neural network approach to character recognition. Neural Networks 1989: 2(5): 387–394.
9. Rumelhart D, McClelland J. Parallel Distributed Processing. MIT Press, Cambridge, MA, 1986.
10. Sejnowski TJ, Resenberg CR. Parallel networks that learn to pronounce English text. Complex Systems 1987; 1: 145–168.
11. Sejnowski TJ, Yuhas BP, Goldstein MH, Jenkins RE. Combining visual and acoustic speech signals with a neural network improves intelligibility. In: DS Touretsky (ed.), Advances in Neural Information Processing Systems. Morgan Kaufmann, 1990, pp. 232–239.
12. Hornik K, Stinchcombe M, White H. Multilayer feedforward networks are universal approximators. Neural Networks 1989; 2: 359–366.
13. Irie B, Miyake S. Capabilities of three-layered perceptrons. Proc IEEE Int Conf Neural Networks, vol 1, 1988; pp. 641–648.
14. Judd S. Learning in networks is hard. Proc IEEE Int Conf Neural Networks, vol 2, 1987, pp. 685–692.
15. Jacobs RA. Increased rates of convergence through learning rate adaptation. Neural Networks 1988; 1: 295–307.
16. Van Der Smagt PP. Minimization methods for training feedforward neural networks. Neural Networks 1994; 7(1): 1–11.
17. Ripley BD. Pattern Recognition and Neural Networks. Cambridge University Press, New York, 1996.
18. Wahba G. Generalization and regularization in nonlinear learning systems. In: MA Arbib (ed.), The Handbook of Brain Theory and Neural Networks. MIT Press, 1995, pp. 426–430.
19. Fahlman SE, Lebiere C. The cascade-correlation learning architecture. In: DS Touretsky (ed.), Advances in Neural Information Processing Systems. Morgan Kaufmann, 1990, pp. 524–532.
20. Nadal JP. New algorithms for feedforward networks. In: Theumann and Kiberle (eds.), Neural Networks and Spin Glasses. World Scientific, 1989, pp. 80–88.
21. Shadafan RS, Niranjan M. A dynamic neural network architecture by sequential partitioning of the input space. Neural Computation 1994; 6: 1202–1222.
22. Breiman L, Friedman JH, Olshen RA, Stone CJ. Classification and Regression Trees. Wadsworth & Brooks, 1984.
23. Brown DE, Pittard CL. Classification trees with optimal multivariate splits. Proc IEEE Int Conf Systems, Man and Cybernetics, vol 3, Le Touquet, 1993, pp. 475–477.
24. Friedman JH. A recursive partitioning decision rule for nonparametric classification. IEEE Trans Computer 1977; 26: 404–408.
25. Kim B, Landgrebe DA. Hierarchical classifier design

in high-dimensional numerous class cases. IEEE Trans Geosci Remote Sens 1991; 29(4): 518–528.
26. Murthy KVS. On growing better decision trees from data. PhD Thesis, The Johns Hopkins University, 1995.
27. Park Y, Sklansky J. Automated design of linear tree classifiers. Patt Recogn 1990; 23(12): 1393–1412.
28. Shi QY, Fu KS. A method for the design of binary tree classifiers. Patt Recogn 1983; 16: 593–603.
29. Sklansky J, Wassel GN. Pattern Classifiers and Trainable Machines. Springer-Verlag, New York, 1981.
30. Curram SP, Mingers J. Neural networks, decision tree induction and discriminant analysis: An empirical comparison. J Operat Res Soc 1994; 45(4): 440–450.
31. Park Y. A comparison of neural net classifiers and linear tree classifiers: their similarities and differences. Patt Recogn 1994; 27(11): 1493–1503.
32. Cios KJ, Liu N. A machine learning method for generation of a neural network architecture: A continuous ID3 algorithm. IEEE Trans Neural Networks 1992; 3(2): 280–291.
33. Golea M, Marchand M. A growth algorithm for neural network decision trees. EuroPhysics Lett 1990; 12(3): 205–210.
34. Guo H, Gelfand SB. Classification trees with neural network feature extraction. IEEE Trans Neural Networks 1992; 3(6): 923–933.
35. Herman GT, Yeung KTD. On piecewise-linear classification. IEEE Trans Pattern Analysis and Machine Intelligence 1992; 14(7): 782–786.
36. Ishwar K, Sethi K. Entropy nets: from decision trees to neural networks. Proc IEEE 1990; 78(10): 1605–1613.
37. DAlche-Buc F, Zwierski D, Nadal JP. Trio learning: A new strategy for building hybrid neural trees. Int J Neural Systems 1994; 5(4): 259–274.
38. Sankar A, Mammone RJ. Growing and pruning neural tree networks. IEEE Trans Computer 1993; 42(3): 291–299.
39. Sirat JA, Nadal JP. Neural tree: A new tool for classification. Network: Computation in Neural Systems 1990; 1(4): 423–438.
40. Jordan MI, Jacobs RA. Hierarchical mixture of experts and the EM algorithm. Neural Computation 1994; 6: 181–214.
41. Chen K, Xie DH, Chi HS. A modified HME architecture for text-dependent speaker identification. IEEE Trans Neural Networks 1996; 7(5): 1309–1313.
42. Chen K, Xie DH, Chi HS. Speaker identification using time-delay HMEs. Int J Neural Systems 1996; 7(1): 29–43.
43. Chen K, Yang LP, Yu X, Chi HS. A self-generating modular neural network architecture for supervised learning. Neurocomputing 1997; 16(1): 33–48.
44. Fisher RA. The use of multiple measurements in taxonomic problem. Ann Eugenics 1936; 7: 179–188.
45. Duda R, Hart P. Pattern Classification and Scene Analysis. John Wiley & Sons, New York, 1973.
46. Murthy PM, Aha DW. UCI Repository of machine learning database. [http://www.ics.uci.edu/mlearn/MLRepository.html], Department of Information and Computer Science, Irvine, CA: University of California, 1994.
47. Fletcher R. Practical Methods of Optimization. John Wiley & Sons, New York, 1987.
48. Ishikawa M. Structural learning with forgetting. Neural Networks 1996; 9(3): 509–521.
49. Lang KJ, Witbrock MJ. Learning to tell two spirals apart. In: D Touretzky, G Hinton, T Sejnowski (eds.), Proc 1988 Connectionist Models Summer School, Morgan Kaufmann, 1989; 52–59.
50. Deterding DH. Speaker normalization for automatic speech recognition. PhD Thesis, University of Cambridge, 1989.
51. Robinson AJ. Dynamic error propagation networks. PhD Thesis, University of Cambridge, 1989.
52. Cybenko G. Approximation by superpositions of a sigmoidal function. University of Illinois, Urbana, 1988.
53. Funahashi K. On the approximate realization of continuous mappings by neural networks. Neural Networks 1989; 2: 183–192.
54. Chen K, Yu X, Chi HS. Text-dependent speaker identification based on the modular tree. Chinese J Electr 1996; 5(2): 63–69.
55. Chen K, Yu X, Chi HS. Text-dependent speaker identification based on the modular tree: an empirical study. In: S Amari *et al.* (eds.), Progress in Neural Information Processing. 1996, Springer-Verlag, Singapore, pp. 294–299.
56. Blum A, Rivest R. Training a 3-node neural net is NP-complete. In: DS Touretsky (ed.), Advances in Neural Information Processing Systems, Morgan Kaufmann, 1989, pp. 494–501.
57. Minsky M, Papert S. Perceptrons: An Introduction to Computational Geometry. MIT Press, Camridge, 1988.
58. Wolpert DH. Stacked generalization. Technical Report LA-UR-90-3460, The Santa Fe Institute, 1990.

Nomenclature

$\mathbf{x}$	input data (a feature vector in pattern classification)
$\mathscr{X}$	a training data set consisting of input data
ϕ	the null set
$\mathscr{X}_l$	one of two adjacent subsets of $\mathscr{X}$ ($\mathscr{X}_l \cup \mathscr{X}_r = \mathscr{X}$ and $\mathscr{X}_l \cap \mathscr{X}_r \neq \phi$)
$\mathscr{X}_r$	one of two adjacent subsets of $\mathscr{X}$ ($\mathscr{X}_r \cup \mathscr{X}_l = \mathscr{X}$ and $\mathscr{X}_r \cap \mathscr{X}_l \neq \phi$)
$N_{\mathscr{X}}$	the number of samples in $\mathscr{X}$
$\mathbf{m}$	the mean of samples in a data set
ω_k	the label of class k in pattern classification
$\mathscr{X}_k$	the data set labelled by ω_k in pattern classification
$\mathbf{m}_k$	the mean of samples in $\mathscr{X}_k$ in pattern classification
N_k	the number of samples in $\mathscr{X}_k$ in pattern classification
$\mathbf{w}$	weight vector of a linear discriminant function
$l(\mathbf{x})$	linear discriminant function
η	overlapping factor of determining size of an overlapping region in the proposed splitting rule
K_{PT}	prolong-training factor for prolonging the training of a subnetwork under the success condition
I_T	threshold of iteration (epochs) for a subnetwork
E_T	threshold of MSE for a subnetwork
$S_{\max}$	threshold of maximal number of samples used for training a subnetwork

$S_{\min}$	threshold of minimal number of samples used for training a subnetwork	$\alpha_i(\mathbf{x}_u)$	the credit assigned to the ith subnetwork for $\mathbf{x}_u$
I	the number of iterations (epochs) during the training of a subnetwork	$O_i(\mathbf{x}_u)$	the output of the ith subnetwork for $\mathbf{x}_u$
E_I	the MSE value of a subnetwork after I iterations	$O(\mathbf{x}_u)$	the output of modular tree for $\mathbf{x}_u$
$\mathbf{x}_u$	unknown data for test	(N_H,N_{MLP})	the architecture of a modular tree consisting of N_H hyperplanes and N_{MLP} MLPs
		$\|\cdot\|$	the Euclidean norm