Task Decomposition with Pattern Distributor Networks

BAO CHUNYU
(*M.Sc., Peking University, China*)

A THESIS SUBMITTED FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2008

Acknowledgements

Firstly, I would like to devote my deepest thanks and gratitude to my supervisor, Prof. Sheng-Uei, Guan, Steven. Thanks a lot for his valuable guidance and continuous encouragement throughout my research. I have benefited much from his profound experiences and deep insights in many problems. His encouragement has driven me to develop positive attitude towards my research and life. Many thanks to Prof. Guan for his valuable instruction, special concern, encouragement and support!

My thanks to Dr. Wu Chunfeng for giving me her thesis templates and Miss. Mo Wenting for helping to solve latex problem when writing the thesis.

My special thanks also to my wife, Dr. Lan Jinghua. I am thankful for her support to my work. She has persevered with me through this process, and has given me encouragement and support when I was tired.

Finally, I would like to thank the National University of Singapore for the scholarship during my study in NUS.

Contents

Summary

Task decomposition methods modularize a large neural network into several modules. These modules are integrated together to form a modular neural network. Compared with normal neural networks, the classification accuracy can be improved using task decomposition. In the thesis, a new method named Pattern Distributor (PD) is presented as a new task decomposition method. PD method can perform better than ordinary task decomposition networks (for example, Output Parallelism). The thesis is focused on the following aspects:

1. The structure of PD networks is introduced and a theoretical model is presented to compare the performance of PD networks with OP networks. The analysis shows that PD networks can outperform OP networks. A technique called Reduced Pattern Training (RPT) is introduced to the PD network to reduce training time and further decrease classification error.

2. According to the theoretical model, the distributor module's performance in a PD network greatly affects the classification accuracy of the whole network. How to combine classes in the distributor module is a key issue for designing a PD network. Several theorems and corollaries are presented for class combination in the distributor module and for the relations in the non-distributor modules. Based on these theorems and corollaries, three greedy combination algorithms are proposed. We also present another two combination algorithms based on FLD analysis and evolutionary algorithm.

Compared with other typical decomposition methods (for example, Output Parallelism), the PD method can improve the generalization accuracy for classification problem and at the same time, even reduce the training time. The PD method can be easily transplanted to real-world applications, for instance, illness analysis, image and letter processing, molecular biology, sound recognition and so on.

Publication List

1. Sheng-Uei Guan, Chunyu Bao and TseNgee Neo, "Reduced Pattern Training Based on Task Decomposition using Pattern Distributor", IEEE Trans. on Neural Networks, Vol. 18, No. 6, (2007) 1738-1749.

2. Sheng-Uei Guan, Chunyu Bao and Ru-Tian Sun, "Hierarchical Incremental Class Learning with Reduced Pattern Training", Neural Processing Letters, Vol. 24, No. 2, (2006) 163-177.

3. Chunyu Bao and Sheng-Uei Guan and TseNgee Neo, "Reduced Pattern Training in Pattern Distributor Networks", Journal of Research and Practice in Information Technology, Vol. 39, No. 4, (2007) 273-286.

4. Sheng-Uei Guan, Yinan Qi and Chunyu Bao, "An Incremental Approach to MSE-Based Feature Selection", International Journal of Computational Intelligence and Applications (IJCIA), to appear, Vol. 6, No. 4, (2006) 451-471.

5. Sheng-Uei Guan, Tse Ngee Neo and Chunyu Bao, "Task Decomposition Using Pattern Distributor", Journal of Intelligent Systems, Vol. 13, No. 2, (2004) 123-150.

6. Chunyu Bao and Sheng-Uei Guan, "Reduced Training for Hierarchical Incremental Class Learning", 2006 IEEE Conferences on Cybernetics and Intelligent Systems (CIS) Robotics, Automation and Mechatronics (RAM).

List of Tables

List of Figures

List of Abbreviations

CBP	Constructive Backpropagation
CTCS	Cross-talk Based Combination Selection
FLD	Fisher's Linear Discriminant
FPT	Full Pattern Training
GACS	Genetic Algorithm Based Combination Selection
GCS	Greedy Based Combination Selection
MLP	Multiple Layer Perceptron
OP	Output Parallelism
PD	Pattern Distributor
RGCS	Restricted Greedy Based Combination Selection
RPT	Reduced Pattern Training
SGCS	Simplified Greedy Based Combination Selection

Chapter 1

Introduction

1.1 Research Motivation

Multiple layer perceptron (MLP) neural network is widely used as a powerful classifier for nonlinear classification problems (Rumelhart, et al., 1986; Bishop, 1995; Haykin, 1999). However, it still suffers from several drawbacks when applied to complex behavioral problems (Auda et al., 1996). Learning a complex behavior requires bringing together several different kinds of knowledge and processing, which is impossible to be achieved for global neural networks like MLP (Feldman, 1989; Simon, 1981). For the "stability-plasticity dilemma" problem, Carpenter and Grossberg (1988) argued that when two tasks have to be learnt consecutively by a single network, the learning of the second task will interfere with the previous learning. Another common problem for multiple-task neural networks is the "temporal crosstalk"problem (Jacobs and Jordan, 1991), which means that a network tends to introduce high internal interference because of the strong coupling among their hidden-layer weights when several tasks have to be learnt simultaneously.

A widely used approach to overcome these shortcomings is to decompose the original problem into sub-problems (modules) and perform local and encapsulated computation for each sub-problem. Task decomposition methods modularized the single large neural network into several modules. These modules are integrated together to form a modular neural network. Various task decomposition methods have been presented. Compared with normal neural networks, the recognition rate can be improved using task decomposition.

There are three main task decomposition methods, which are ensemble learning, domain decomposition and class decomposition. For ensemble learning and domain decomposition, though the whole problem is divided into several learners or modules and the task for each learner or module is relatively small, the internal interference between classes can not be avoided. Class decomposition algorithms are designed for the problem with several or many classes. The introduction of class decomposition is to reduce the internal interference between classes. However, there are still some shortcomings in existing class decomposition methods. For example, some algorithms decompose a K-class problem into K two-class sub-problems or several sub-problems (Chen and You, 1993; Ishihara and Nagano, 1994; Anand et al., 1995; Guan and Li, 2000, 2002b). For each sub-problem, the dimension is reduced, but the number of training samples is not reduced. Some other methods split a K-class problem into $\binom{K}{2}$ two-class sub-problems (Friedman, 1996; Lu and Ito, 1999) and the size of each sub-problem's training pattern set is reduced. However, if the original K-class problem is complex (K is large), a large number of modules will be needed to learn the sub-problems and thus resulting in excessive computational cost. To overcome these shortcomings, in the thesis, we will continue to explore and refine task decomposition methods.

1.2 Problem Definitions and Overall Solutions

Classification problems and regression problems are two categories of problems widely used in real life. Classification problems generally refer to those problems where one attempts to predict category labels (class, group, etc.) from one or more continuous and/or discrete variables. Regression problems are generally those where one attempts to predict continuous variables from one or more continuous and/or discrete variables. In the thesis, we will design new classifiers for classification problems. Thus, our discussion will be focused on classification problems. As many algorithms originally designed for classification problems can be extended to regression problems, i.e. decision tree classifiers, our algorithms would also be applicable to regression problems after some revisions. Research on regression problems will be one of our future directions.

There are four main components for a classification problem. The first one is the categorical outcome, which is the characteristic we hope to predict. The second component of a classification problem is the continuous and discrete variables (or the predictor variables) which are the characteristics related to the outcome variable of interest. The third component of a classification problem is the learning dataset. This is a dataset which includes values for both the category labels and predictor variables. The fourth component of the classification problem is the test or future dataset, which is used for testing the classification accuracy of the classifiers. This test dataset may or may not exist in practice.

Our research is focused on problems with several or many classes, i.e., the number of classes is greater than three.

Output Parallelism (OP), presented by Guan and Li, is regarded as a typical class decomposition for neural networks (2000 and 2002a). OP method decomposes the original complex problem into a set of smaller sub-problems without any prior knowledge concerning the decomposition of the problem. For example, for an original classification problem with K output classes, the first step is to divide this original problem into R sub-problems each of which has r_i (i = 1, 2, 3, ...,R) output classes where $\sum r_i = K$ output classes. Each sub-problem is composed of the whole input problem space and a fraction of the output problem space. Each sub-problem is then solved by building and training a module (small size neural network).Thus, R modules will be trained independently according to the corresponding sub-problems and the collection of such modules will be the overall solution of the original problem. See Figure 1.1.

The basic idea of Pattern Distributor (PD) is an expansion from the OP method. In the OP network, all the unknown patterns enter each module directly. We may consider to incorporate a special module called a distributor module before the modules of the OP network. Thus, when an unknown pattern enters the network, it is processed by the distributor module first. The distributor module decides which module will continue to classify this pattern. The distributor module has a higher position as compared to the other modules in the network. The overview of the new

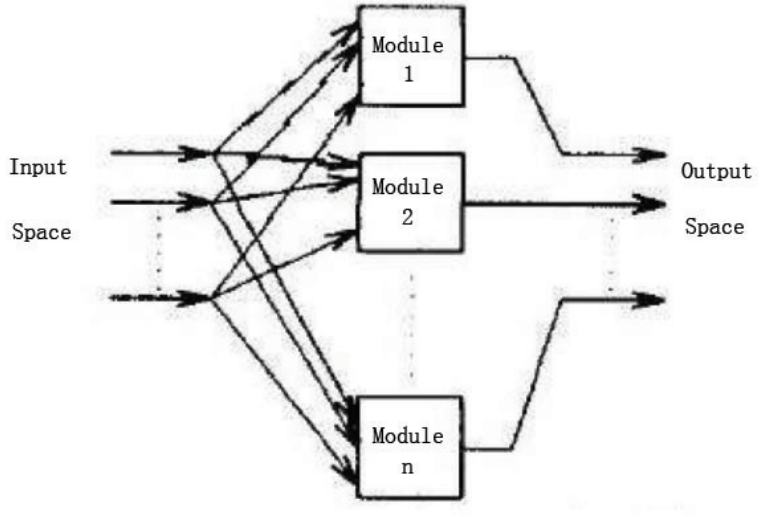

Figure 1.1: Modular networks based on Output Parallelism

architecture is shown in Figure 1.2.

When unseen input patterns enter the network, they are firstly processed by the distributor module. The distributor module assigns these patterns to different modules. Each non-distributor module only classifies a portion of the unseen patterns. While in OP networks, each module will classify all the unseen patterns. Because a non-distributor module in the PD networks only processes a portion of the whole test set, the number of wrongly-classified patterns could be smaller than that of its counterpart in the OP network. Detailed analysis will be presented in Chapter 3. Thus, the classification accuracy could be increased.

A non-distributor module in a PD network only classifies patterns belonging to a few classes. The unseen patterns of other classes will not enter that module. Thus, that module can be learned only using the training patterns and validation patterns which belong to its own classes. The patterns belonging to other classes can be removed. This is the basic idea for Reduced Pattern Training (RPT). The training time could be saved using RPT and the classification accuracy could also be increased.

In the above consideration, we ignored the distributor module's performance and

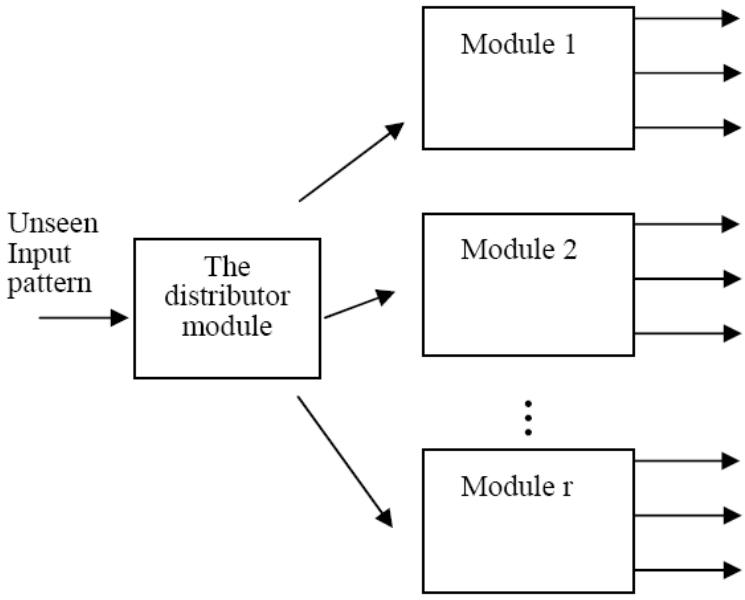

Figure 1.2: Modular networks based on Pattern Distributor method

assume the distributor module classifies all the patterns correctly. In fact, it's nearly impossible. If the distributor module's classification error is large, the PD network can hardly achieve better performance than the OP network. In Chapter 3, we deduce the condition in which the PD network achieves better classification accuracy than the corresponding OP network.

Sometimes, in a PD network, some non-distributor modules are large (it means the module needs to classify a large number of classes). Since the PD method could improve classification accuracy of the network, we may continue to apply the PD method to these non-distributor modules. We expect it will further improve the performance of the whole network. For example, in Figure 1.2, we may substitute Module 1 with a sub-PD network. Thus, a multi-level PD network is formed. The details of multi-level PD networks are discussed in Chapter 4.

It was mentioned that the distributor module's performance greatly affects the whole PD network. Thus, we hope to decrease the classification error of the distributor module. Each output of the distributor module is a combination of several classes. The combinations for all the outputs of the distributor module are grouped

into a combination set. For the distributor modules, different combination sets lead to different classification accuracies. Now we have a question: Could we find combination sets which ensure that the PD module achieve high performance? How to find them?

To answer the above question, an algorithm, called Greedy Based Combination Selection (GCS), is proposed to find a good combination set for the distributor module. The algorithm starts from the combination set that has K elements (K is the number of classes of the problem), i.e. {{1},{2}, ..., {K}}. Here {1}, {2}, ..., {K}are the combinations in the combination set. In each epoch, the combination with the largest classification error is selected, e.g. combination {2}. Then we temporarily combine the combination with other combinations and find a suitable one based on classification error test, e.g. {3}, and combine them together, i.e. {2,3} and proceed to the next epoch. Thus, step by step, the elements in the combination set are reduced. If some stopping criteria are satisfied, stop the algorithm. This way we can find a suitable combination set. For the details of this algorithm, please refer to Chapter 5.

The above algorithm can find a combination set with near optimal (minimum) classification error for the distributor module. However, it needs relatively large computation effort. In order to reduce computation, another algorithm, namely Simplified Greedy Based Combination Selection (SGCS), is proposed. In this algorithm, we still need to do the classification error test for temporarily combinations, but the number of tests is reduced. Thus, the computation effort is saved. And we still could find a combination set for the distributor module with small classification error.

GCS and SGCS work well for a distributor module. However, they usually will lead to an imbalanced combination set. An imbalanced combination set means some combinations have more classes than other ones. In other words, some outputs of the PD module have more classes than the other outputs. An imbalanced combination set may bring harmful effect to the whole PD network. It will result in non-PD modules with imbalanced workload. And the modules with heavy workload

are hard to make a satisfactory classification. There are two approaches to solve the problem. One is to continue to apply the PD method to the non-distributor modules with many classes. This approach will lead to a multi-layer PD network. The other approach is to add restriction to the maximum number of classes in a combination.

Based on the idea of the second approach, the work load between the distributor modules and non-distributor modules is considered and a new rule, call the $\sqrt{K}$ Rule-of-thumb (K is the number of classes in the data set) is deduced. According to this rule, the maximum number of classes in a combination should not exceed $\sqrt{K}$. By adding this constraint to the GCS algorithm, Restricted Greedy Based Combination Selection (RGCS) designed for single-layer PD networks is proposed.

In Chapter 6, another two combination selection algorithms are presented, namely Cross-talk based Combination Selection (CTCS) and Genetic Algorithm based Combination Selection (GACS). These two algorithms are designed for single-layer PD networks. CTCS generates a cross-talk table based on the Fisher's linear discriminant (FLD). Then a combination set is produced based on the cross-talk analysis using some regulations. GACS uses the evolutionary method to find a suitable combination set. The $\sqrt{K}$ rule is used in both algorithms.

1.3 Research Contribution

The algorithms presented in this thesis work towards developing task decomposition based algorithms with the following characteristics:

1. **Can lead to improved performance in terms of better generalization.** All task decomposition algorithms result in reduced generalization accuracy compared with normal MLP neural networks. In our research, we try to further improve the classification rate by introducing the concept of Pattern Distributor. The introduction of distributor module greatly reduces the workload of other modules and the whole network's performance could be improved. However, the distributor module's performance greatly affects the classification accuracy of the whole network. In order to boost the distributor module's

classification accuracy, several combination algorithms are proposed. These algorithms can find good combinations for the distributor module. Thus, the PD network's performance could be ensured.

2. **Improves the overall training time.**
   The class decomposition algorithms are designed to improve the classification rate of the problem. Normally, if using series training (training the modules one after another), the overall training time will increase. To save training times, some methods use parallel training to substitute series training (Guan and Li, 2000 and 2002a). In our PD networks, the improvement of generalization accuracy does not sacrifice the training time. The training time of non-distributor modules could be greatly reduced by removing the patterns of unrelated classes. Thus, even using series training, the overall training time for the PD network can be even smaller than other class decomposition methods.

3. **Find near-optimal combination sets for Pattern Distributor modules automatically.**
   It was mentioned before that the distributor modules' classification accuracy will greatly affect the performance of the whole network. Several combination selection algorithms are presented to find good combination sets for the distributor modules. Thus, we design our combination selection algorithms by which we decompose the patterns automatically in a fashion that is independent of human judgment.

4. **Can be easily modified and extended.**
   We expect that the proposed algorithm can be applied, with minor adjustments, to other training algorithms that involve learning based on training patterns. The PD method should be easily combined with other task decomposition methods. For instance, the PD method can be combined with Mixture-of-expert systems (Jacobs et al., 1991) and Recursive Percentage-based Hybrid Pattern training algorithm (Guan and Ramanathan; 2004). We can apply the PD method to the modules of these systems to boost the performance of the whole network. The PD method can also be easily transplanted to real-world applications, for instance, medical analysis, image processing, molecular biology, letter recognition and so on.

Our research is focused on multi-class classification problems. The larger the number of classes in the problem, the more likely the combination selection algorithms can find a satisfactory combination set. Thereby, the PD network can perform better than other class decomposition methods. For researchers or users in the area of speech recognition and image analysis, the PD network may enlighten them to set up more powerful classifiers which can improve the classification accuracy.

Our PD methods still have some constraints. Firstly, the PD method is not suitable for problems with just a few classes, i.e., the problems with three or less classes. There is another issue in our research. The whole PD network's performance is not only on the distributor modules but also on the non-distributor modules. Several combination selection algorithms are proposed to reduce the classification error of a distributor module, so the distributor module can have good performance. However, we only have preliminary analysis for non-distributor modules (see Chapter 5). Further analysis of the non-distributor modules remains a future research task.

1.4 Thesis Outline

The rest of the thesis is organized as follows. Chapter 2 reviews related literature and presents them in the context of neural networks, decision tree systems and task decomposition. Chapter 3 presents the structure of single-layer PD networks, a theoretical analysis is offered to evaluate the performance of a PD network. Chapter 4 presents the idea of multi-layer PD networks. In Chapter 5, three greedy based combination selection algorithms are presented to find a near-optimal combination set for a distributor module. Chapter 6 presents two other combination algorithms for single-layer PD network and compares all the combination selection algorithms. Chapter 7 presents an overall discussion on PD networks and concludes the thesis.

Chapter 2

Related Work

2.1 Introduction

In this chapter, we will give some background information on neural networks and constructive backpropagation algorithm as they are applied in our research. Thereafter, a review on decision tree algorithms will be given. Our PD network - a task decomposition approach, appears to have some similarity with the decision tree algorithms. Next, we will cover various task decomposition methods, focusing on those based on neural networks. These task decomposition methods fall into three categories, Ensemble learning, domain decomposition and class decomposition. Our PD method belongs to the class decomposition category.

2.2 Background

2.2.1 Neural Networks

A neural network, also known as an artificial neural network, simulates cortical structures of the human brain in a fundamental manner. It is composed of a large number of interconnected neurons or nodes which work together to solve problems. There are various of neural network models such as Hopfield nets, the Boltzmann machine, Kohonen self-organizing feature maps and Multi-layer perceptrons (MLPs) (Rumelhart, et al., 1986; Bishop, 1995; Haykin, 1999). In these neural networks, MLPs are the most popular ones, which are feedforward neural networks with multi-layered structures. Among these multi-layer structures, three-layered structure is the most widely used. Typically, a three-layer MLP has an input layer, a hidden layer and

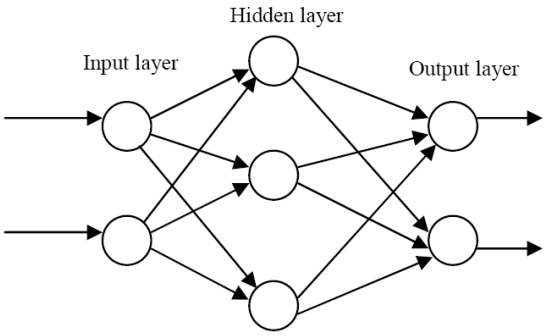

Figure 2.1: Architecture of a typical three-layer MLP neural network

an output layer. The signal propagates through the network from the input layer to the output layer (Figure 2.1).

MLPs have been successfully applied to solve various problems. Generally, the training of MLPs is carried out with a standard backpropagation type of training algorithm. This training algorithm performs gradient descent only in the weight space of a network with a fixed topology, and it is useful when the network architecture is selected properly. A problem can not be learnt well with a too small network, but a size too large will lead to overfitting and poor generalization performance (Geman, etc., 1992). There are three major approaches for solving it. Firstly, a large number of networks with different sizes are trained and then the "best" structure is chosen using some criterion based on information theory (Akaike, 1974; Rissanen, 1975; Schwartz, 1978). The second one is that we train a relatively large network for the problem and then use pruning methods to reduce the size of the network (Reed, 1993; Poggio and Girosi, 1990). The last one, also called constructive algorithm, starts from a small network and then grows hidden nodes over it until a satisfactory solution is reached (Lehtokangas, 1999; Kwok and Yeung, 1997). Compared with the former two approaches, the construction algorithm sets up a relatively smaller network and is more effective in resources. Constructive Backpropagation (CBP) by Lehtokangas (1999) may be the most noted one. We will give a brief introduction to CBP algorithm.

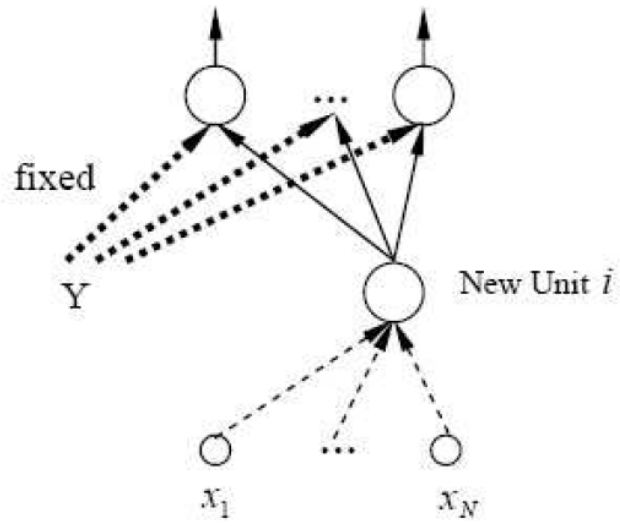

Figure 2.2: Training a new hidden unit in CBP learning. Y represents previously added connections to network output units

2.2.2 Constructive Backpropagation (CBP) Algorithm

In our training course for neural network modules, the Constructive Backpropagation algorithm is used. The CBP can be depicted briefly as follows (Lehtokangas, 1999):

1. *Initialization*: The network has no hidden units. Only bias weights and shortcut connections from the input units to the output units feed the output units. Train the weights of this initial configuration by minimizing the sum of squared errors:

$$E = \sum_{p=1}^{P} \sum_{k=1}^{K} (o_{pk} - t_{pk})^2 \quad , \tag{2.1}$$

where P is the number of training patterns, K is the number of output units,o_{pk} is the actual output value of the k^{th}output unit for the p^{th} training pattern and t_{pk} is the desired output value of the k^{th} output unit for the p^{th} training pattern.

2. *Training a new hidden unit*: Connect inputs to the new unit (let the new unit be the i^{th} hidden unit, $i > 0$) and connect its output to the output units as shown in Figure 2.2. Adjust all the weights connected to the new unit (both input and output connections) by minimizing the modified sum of squared errors:

$$E_i = \sum_{p=1}^{P} \sum_{k=1}^{K} [a(\sum_{j=0}^{i-1} w_{jk} o_{pj} + w_{ik} o_{pi}) - t_{pk}]^2 \quad , \tag{2.2}$$

where w_{jk} is the connection from the j^{th} hidden unit to the k^{th} output unit (w_{0k} represents a set of weights which are the bias weights and shortcut connections trained in step 1), o_{pj} is the output of the j^{th} hidden unit for the p^{th} training pattern (o_{p0} represents inputs to bias weights and shortcut connections), and $a(\cdot)$ is the activation function. Note that in the new i^{th} unit perspective, the previous units are fixed. In other words, we are only training the weights connected to the new unit (both input and output connections).

3. *Freezing a new hidden unit*: Fix the weights connected to the unit permanently.

4. *Testing for convergence*: If the current number of hidden units yields an acceptable solution, then stop the training. Otherwise go back to step 2.

2.3 Decision Tree Classifiers

Decision tree classifiers are also widely used in the classification problems. The basic idea of decision tree classifiers is to break up a complex decision into a number of simpler decisions. It has some similarities to task decomposition algorithms, which also use the concept of "divide-and-conquer".

2.3.1 Review of Decision Tree Classifiers

Decision tree systems exist long compared with neural networks. Researchers have proposed various methods for the tree structure design (Argentiero et al., 1982; Bartolucci, 1976; Casey and Nagy, 1984; Diday and Moreau 1986; Gelfand and Guo,

1991; Gustafson, 1980; Kargupta et al., 2006; Kim and Landgrebe 1990; Kulkarni, 1976; Li and Dong, 2003; Li and Dubes, 1986; Pedrycz and Sosnowski, 2005; Quinlan and Rivest 1989; Rounds, 1980; Yun and Fu, 1983). Some of them had no claim of optimality and utilized a priori knowledge for the design (Argentiero et al., 1982; Gu et al., 1983; Landeweerd, 1983; Wang and Suen, 1987) while others applied mathematical programming methods such as dynamic programming or branch-and bound techniques (Kulkarni, 1976; Payne and Meisel, 1977). There are various heuristic methods to construct decision tree classifiers. They can be grouped into four categories: bottom-up approach, top-down approach, hybrid approach and tree growing-pruning approach. In bottom-up approach, decision trees are created from leaf to root according to certain principles, such as recognizing the most frequently appearing classes first (Landeweerd et al., 1983). In top-down approach, sets of classes are continually divided into smaller subsets of classes (Li and Dubes, 1986). Hybrid methods use both bottom-up and top-down approaches sequentially (Kim and Landgrebe, 1990). Tree growing-pruning approach may be the most popular one. It first grows a huge tree according to bottom-up approach or top-down algorithm, and then prunes unused or unnecessary branches (Breiman et al, 1984; Esposito et al., 1997; Gelfand, 1991; Quinlan, 1993 and 2003).

The most popular decision tree classifier may be Quinlan's ID3, standing for "Iterative Dichotomizer (version) 3" (Pao, 1989). Later versions include C4.5 and C5 (Quinlan, 1993 and 2003). Since various decision trees have similar design principles, now we give a brief review to ID3 to show how decision trees work. Figure 2.3 shows an example for an ID3 decision tree system. The ID3 decision tree learning algorithm computes the Information Gain G on each feature F, for a K-class problem, defined as:

$$G(S,F) = Entropy(S) - \sum_{v \in values(F)} \frac{S_v}{S} Entropy(S_v) \quad , \tag{2.3}$$

where S is the total input data and S_v is the subset of S for which feature F has a value v. The $Entropy(S)$ is given by

$$Entropy(S) = -\sum_{i=1}^{K} p_i log_2(p_i) \quad , \tag{2.4}$$

where p_i represents the probability of class i. The feature with the highest information gain, i.e. B, is chosen as the root node of the tree. Then, the training set is

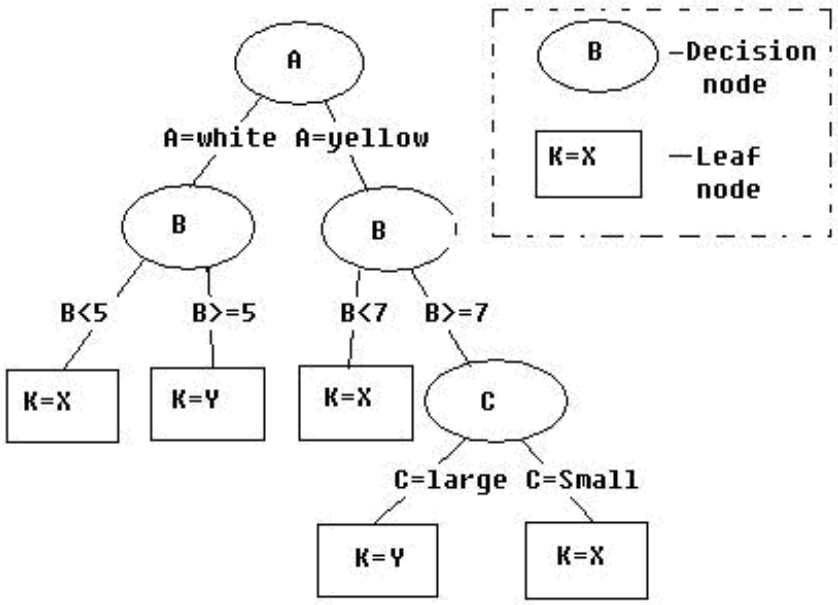

Figure 2.3: An example for an ID3 decision tree classifier

divided into subsets S_B according to the different values of B and a new decision tree is recursively built over each value of B using the corresponding training subset S_B. A leaf-node or a decision-node is formed when all the instances within the available training subset are from the same class. For detecting anomalies, the ID3 decision tree outputs binary classification decision of "0" to indicate normal and "1" to indicate anomaly class assignments to test instances.

2.3.2 Shortcomings of Decision Tree Classifiers

Though various decision tree classifiers have been used to solve the classification problems, they still have some drawbacks. Ordinary decision tree classifiers can guarantee a good classification rate when processing the problems which have simple decision space. However, their performance is downgraded when facing problems with very complex discrimination surfaces. Most decision tree classifiers allow overlaps in order to improve the recognition rate, but too many overlaps will cause the number of terminals to be much larger than the number of classes, thus greatly reducing the efficiency of the classifier. Besides, there are often many levels of nodes in a decision tree system. Thus, when unknown patterns enter the system, the pro-

cessing time is relatively long.

Our PD method can overcome the above shortcomings. The neural network modules in PD networks can handle complex decision surfaces easily. In a PD network, the number of the non-distributor modules (they can be seen as the leaf-node in PDs) can not be larger than the number of classes. The efficiency of such a classifier is much higher than a decision tree classifier. The number of layer of PD networks is also much smaller than that of decision tree classifiers. Thus, the processing time of the PD networks is relatively short.

2.4 Task Decomposition

Task decomposition means the kind of approaches in which we divide a relatively complicated mission into a set of simple tasks and combine their decision in some way. There are mainly three types of task decomposition approaches, namely ensemble learning, data decomposition and class decomposition. Firstly, we look at ensemble learning.

2.4.1 Ensemble Learning

The idea of ensemble learning is based on the assumption that "several minds are better than one". Using learner ensemble, the individual decisions of a set of learners are combined in some way, i.e., using either weighted or unweighted voting, to classify new samples.

Kearns and Valiant (1994) proved that learners can be combined to form an arbitrarily good ensemble hypothesis when enough data is available. Recently, learner ensemble has been shown to be a highly effective approach. Bagging (Breiman, 1996) and boosting (Bauer and Kohavi, 1999; Freund, 1995, 1999 and 2001; Freund and Schapire, 1996, 1997 and 1999; Meir and Ratsch, 2003) introduce diversity in the learners by manipulating the training samples. In bagging, each weak learner randomly makes bootstrap copy of the original training set and using these as new training sets. Some training samples can appear multiple times in the aggregate.

Boosting is commonly known as the best "off the shelf" classifier in literature (Hastie et al., 2001). Like bagging, boosting utilizes the training patterns to create diverse learners. Unlike bagging, however, boosting uses the entire training set to perform the manipulation in each learner. In each iteration, a learner is trained and a hypothesis is returned based on the training set. The error of the hypothesis is used to calculate a corresponding weight for each training patterns based on the concept that more importance is given to the wrongly learnt patterns. The weight will be used for the learner in the next iteration. The final classifier is produced by using a weighted factor on the individual learners.

2.4.2 Domain Decomposition

Domain Decomposition is a category of decomposition methods based on the characteristics of input data space. Domain decomposition has some similarity with learner ensemble. Instead of introducing diversity in the weak learners by manipulating the data and weighing erroneous patterns, data decomposition often removes the patterns which have been learnt correctly and learn the erroneous pattern using new learners (or modules). The advantage is that a finite number of learners or modules are required for learning the patterns. Testing is commonly performed using a sieving network. Some data decomposition algorithms are discussed below.

Mixture-of-experts

We know that strong interference among neural networks will lead to slow learning and poor generalization. The most direct idea for domain decomposition is that if the input data are partitioned into several subspaces and simple systems are trained to fit the local data, the interference will be reduced. Hampshire and Waibel (1989) described a network of this kind that can be used when the division into subtasks is known prior to training. Then Jacobs et al. (1990) developed a related system that allocates instances to experts (or modules) through learning. In the Jacobs' system, during the training process, weight changing is only restricted in the gating network and a few experts if a instance's output is incorrect. The error function used in the above two systems does not encourage localization. Thus, Jacobs et al. revised the

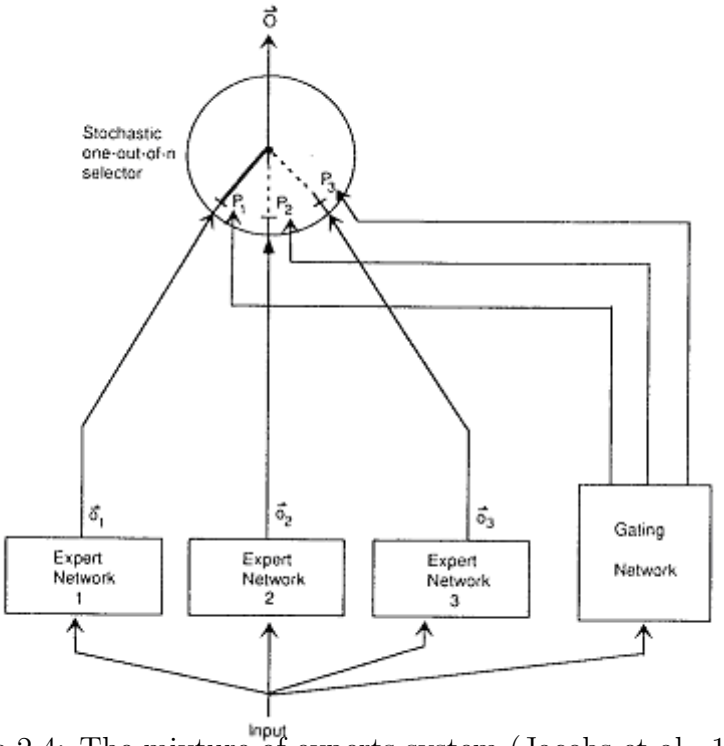

Figure 2.4: The mixture-of-experts system (Jacobs et al., 1991)

error function and developed the famous mixture-of-experts system (Jacobs et al., 1991). See Figure 2.4. The error function is as follows:

$$E = -log \sum_i p_i exp[-\frac{1}{2}(d - o_i)^2] \quad , \tag{2.5}$$

where o_i is the output of expert i, and p_i is the proportional contribution of expert i to the combined output vector and d is the desired output vector. After that, Jordan and Jacobs (1994) designed a hierarchical mixtures-of-experts architecture based on mixture-of-experts and introduced an Expectation-Maximization (EM) algorithm. The EM algorithm decouples the estimation process in a manner that fits well with the modular structure of the architecture. Titsias and Likas (2002) designed a system in which both the gating network units and the specialized experts are suitably defined from the hierarchical mixture.

Multi-sieving

Lu et al. (1994) proposed the multi-sieving neural network, in which patterns are classified by a rough sieve at the beginning and they are reclassified further by finer ones in the subsequent stages. In the algorithm, a neural network is trained using

all the available data until stagnation occurs. At that point, the valid outputs of the patterns are compared with actual outputs. The patterns whose valid outputs are close to the actual outputs are considered learnt and therefore isolated along with their corresponding network. The remaining patterns are further trained using another network and the process is repeated until all the patterns are learnt.

Subset selection

Many papers have been written on the possibility of using a subset of training patterns for training instead of the whole dataset. According to the Mahalannobi-his distances which are close to patterns of other classes, Foody (1998) divided the patterns into border patterns and core patterns and explored different influence of these patterns to the classification accuracy.

The topology based dynamic selection (Gathercole et al., 1994) chooses subsets of training patterns based on their difficulty. The difficulty of a pattern is determined by whether the pattern can be learnt with some accuracy. More and more "difficult" patterns are chosen until a desired subset size is reached. Evolutionary algorithms are used to determine the suitability of a pattern to be part of the subset based on the structure the population induced on the training pattern.

Recursive Percentage-based Hybrid Pattern training Recursive

Percentage-based Hybrid Pattern training proposed by Guan and Ramanathan (2004) uses an efficient recursive combination of global and local search to find a set of pseudo global optimal solutions to a given problem. The hybrid algorithm uses Genetic Algorithms (GA) to find a partial solution with a set of learnt and unlearnt patterns. In each recursion, the GA automatically learns the "easy-to-learn" patterns first while the more "difficult" patterns are passed on to the next recursion. Neural networks are used to learn to perfection the learnt patterns and GA is used again to tackle the previously unlearnt patterns. This is to allow all training patterns to receive attention according to their level of difficulty. The entire process is repeated recursively until a new recursion leads to overfitting. At the end of the training (after N recursions), N solution neural networks would have been trained. Then, a K^{th} Nearest Neighbour algorithm [13] based distributor is used to match

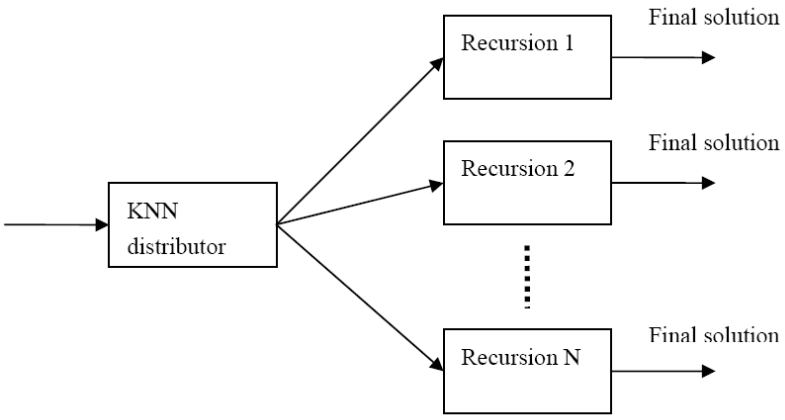

Figure 2.5: A RPHP problem solver

a test pattern to its nearest neighbour. When a test pattern is presented to the system, the system would have to choose one of the N solutions to produce the output. See Figure 2.5 for the network structure. The theory behind their approach is that when training emphasis is given to the difficult patterns in turn, it is possible to obtain an accurate classifier.

2.4.3 Class Decomposition

Another category of decomposition methods is Class Decomposition. Unlike data decomposition, which uses the information of feature space for decomposition, and learner ensemble, which gathers the results from weak learners, class decomposition divides the network based on the characteristics of output space.

Splitting a K-class problem into K 2-class sub-problem

Chen and You (1993) proposed an approach which splits a K-class problem into K two-class sub-problems. One sub-network is trained to learn one sub-problem only. Therefore, each sub-network is used to discriminate one class of patterns from patterns belonging to the remaining classes, and there are K modules in the overall structure. This approach is also introduced by Anand et al. (1993) and Ishihara and Nagano (1994). Such a two-class classification problem often has imbalanced data distribution. Anand et al. (1993) further pointed out that the standard backpropagation algorithm converges slowly for learning these imbalanced two-class problems,

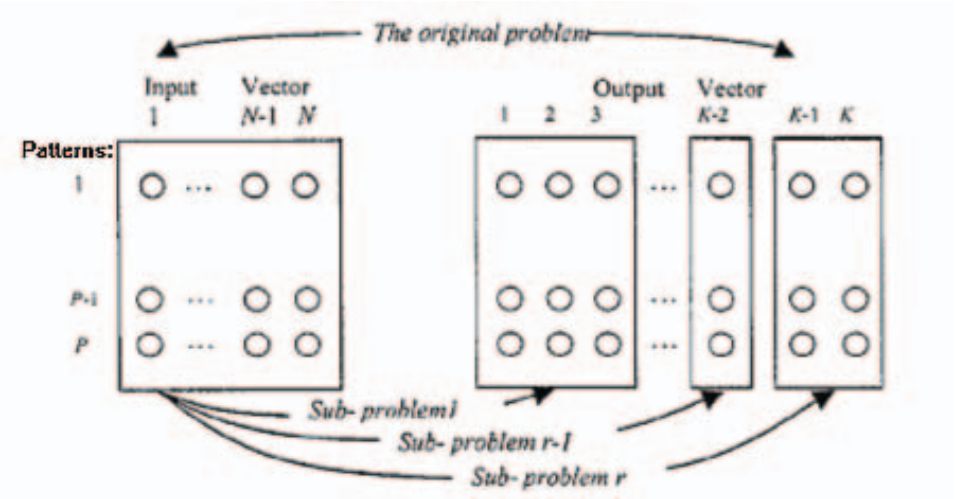

Figure 2.6: Problem decomposition based on Output Parallelism

and thus developed a modified backpropagation algorithm for the imbalanced two-class data set. Their experiments showed that the modified algorithm is faster than the standard one.

Output parallelism

Output parallelism (Guan and Li, 2000, 2002) is a powerful extension to the above class decomposition method. Using output parallelism, a complex problem can be divided into several sub-problems as chosen, each of which is composed of the whole input vector and a fraction of the output vector. Each module (for one sub-problem) is responsible for producing a fraction of the output vector of the original problem. These modules are grown and trained in parallel and incorporated with the constructive backpropagation algorithm (Lehtokangas, 1999). Figure 2.6 shows an example in which a K-class problem is divided into r sub-problems.

The pairwise classifier and the min-max modular network

The pairwise classifier (Friedman, 1996) and the min-max modular network (Lu and Ito, 1999) have similar decomposition idea. Both of them divide a K-class problem into $\binom{K}{2}$ two-class sub-problems. Each of the two-class sub-problems is learned independently while the training data belonging to the other $K-2$ classes are ignored. However, the final combination mechanisms used in the pairwise classifier and the min-max modular network are greatly distinct. In the pairwise classifier,

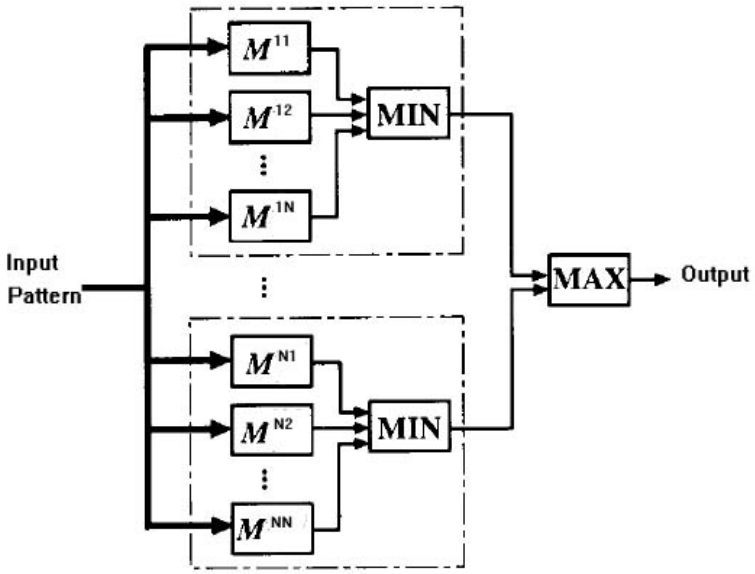

Figure 2.7: An example of the min-max modular network which consist $N \cdot N$ individual modules, N_i MIN unit and one MAX unit

the final output is selected from the $\binom{K}{2}$ decision boundaries by performing the maximizing operation. The combination scheme in the min-max modular network is relatively complicated.

Figure 2.7 shows an example for the min-max modular network. M^{ij} is used to discriminate classes i and j. In these modules (excluding MIN, MAX units), only half of the modules needs to be computed; for the other half are their inverse. The trained modules for each class are integrated using minimization principle. Then the outputs from the MIN units are integrated using maximization principle.

Hierarchical incremental class learning

To make use of the correlation between classes or sub-networks, Guan and Li (2002) proposed an approach named hierarchical incremental class learning. In this approach, a K-class problem is divided into K sub-problems. The sub-problems are learnt sequentially in a hierarchical structure with K sub-networks. Each sub-network takes the output from the sub-network immediately below it as well as the original input as its input. The output from each sub-network contains one more class than the sub-network immediately below it, and this output is fed into the sub-network above it as shown in Fig. 2.8. This method not only reduces harmful

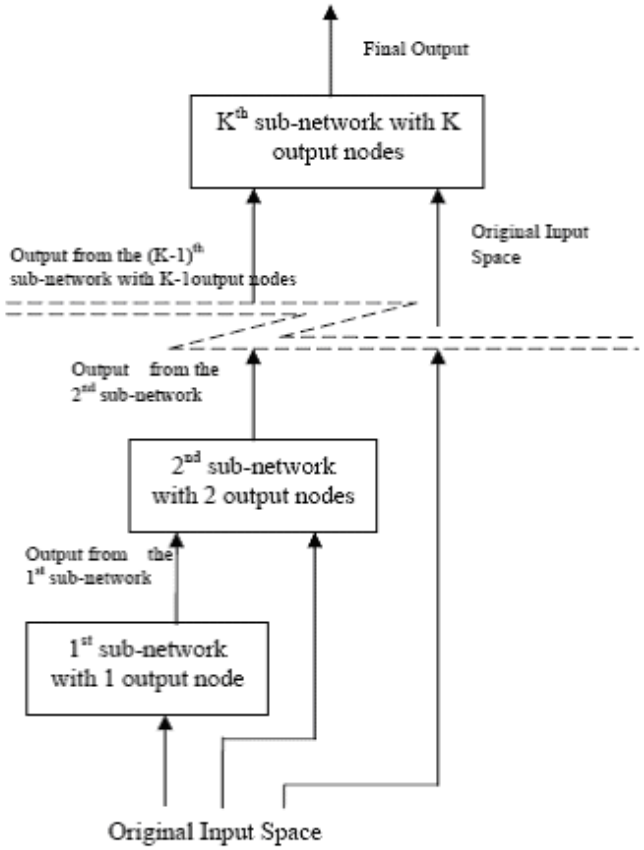

Figure 2.8: The networks structure for hierarchical incremental class learning (Guan and Li, 2002)

interference among hidden layers, but also facilitates information transfer between classes during training. It shows better classification performance than traditional class decomposition methods. Moreover, Guan and Wang (2007) tried to combine hierarchical incremental class learning to Output Parallelism. Experimental results showed the improved recognition rate for classification problems. Later, reduced pattern training method is combined with this method to improve the network performance (Bao and Guan, 2006; Guan and Bao, 2006).

2.4.4 Limitations

We have reviewed various task decomposition methods. All these algorithms are effective ones, yet each of them has strengths and drawbacks. Boosting and bagging can augment the performance of weak learners using a probability based weight sys-

tem, the accuracy of the algorithms depends on the number of weak learners which is problem dependent (Meir and Ratsch, 2003). The number of learners used normally is very large compared with domain decomposition and class decomposition. In our PD networks, the number of modules normally is smaller than the number of classes in the problem. Thus, the modules (or learners) are normally much smaller than those in ensemble learning, so the resources could be saved.

In the domain decomposition methods, subset selection algorithms (Gathercole et al., 1994; Foody, 1998; Lasarzyck et al., 2004) aim to reduce the computational intensity of training by using a subset of the patterns available as a representative of the whole pattern set. The subset used can be either static (Foody, 1998) or dynamic (Gathercole et al., 1994; Lasarzyck et al., 2004). The subsets of patterns are selected using either numerical methods or using evolutionary computation. While the computation intensity is definitely reduced by the use of this algorithm, we should take into account that using a subset of patterns does not guarantee optimal accuracy. Further, the size of the subset plays an important role in the performance of the algorithm, and this again, is a problem dependent value. The mixture-of-experts systems (Jacobs et al., 1991; Jordan and Jacobs, 1994) divide the feature space into many clusters and use a module (or an expert) for each cluster. However, the size and number of the clusters also play an important role in the performance of the algorithm and they depend on the problem itself too. The multi-sieving algorithm (Lu et al., 1995) uses a succession of networks to train the system until all the patterns are learnt. While the algorithm is an efficient one, its accurate performance depends on the value of a predefined error tolerance, which is a problem dependant value. The algorithm, therefore, is not entirely adapted to the problem topology. Recursive Percentage-based Hybrid Pattern training (Guan and Ramanathan, 2004) uses GA to find the suitable subset for recursion modules. However, the computation overhead for this method can not be ignored. Another drawback for these domain decomposition algorithms is that these algorithms can only reduce the size of the data set, but the dimension of the data set does not change. Thus, the internal interferences (that exists within each module due to the coupling of output units) are not reduced. Using the PD method, the selection of subset for each module only depends on the classes, no on the input space, so it is much simpler than that using

domain decomposition methods. The PD method is based on class decomposition, so it can avoid the internal interferences due to the coupling among output units.

Class decomposition methods can effectively reduce the dimension of the problem. However, the reviewed class decomposition methods have other shortcomings. For the algorithms that divide a K-class problem into K two-class sub-problems (Chen and You, 1993; Ishihara and Nagano, 1994; Anand et al., 1995), Output Parallelism (Guan and Li, 2000, 2002b), and the hierarchical incremental class learning network (Guan and Li, 2002a), though the dimension is reduced, the size of each sub-problem's training pattern set is still as large as the original problem. In our PD networks, in most modules, the technique of Reduced Pattern Training is used. Thus, for these modules, the number of patterns used for training and validation is reduced with the final recognition rate either comparable or improved. The pairwise classifier (Friedman, 1996) and the min-max modular network (Lu and Ito, 1999) which splits a K-class problem into $\binom{K}{2}$ two-class sub-problems can reduce the size of training set for each sub-problem. However, if the original K-class problem is complex (K is large), a large number of modules will be needed to learn the sub-problems and thus resulting in excessive computational cost. In the PD networks, the number of modules is normally smaller than the number of classes. Compared with the pairwise classifiers and the min-max modular networks, the PD networks use less modules, especially when K is very large. Thus, the PD method saves computation efforts.

Chapter 3

Single-layer PD Networks

3.1 Design of Single-layer PD Networks

In Chapter 1, A brief description of PD networks is provided. In this chapter, we will focus the discussion on single-layer PD networks.

In a Single-layer PD network, a special module called a distributor module is introduced in order to improve the performance of the whole network. The distributor module and the other modules in the PD network are arranged in a hierarchical structure. The distributor module has a higher position as compared to the other modules in the network. This means an unseen input pattern will be recognized by the distributor module first. The structure of a typical PD network is shown in Figure 3.1. Each output of the distributor module consists of a fraction of the overall output classes of the original problem. The PD method could shorten the training time and improve the generalization accuracy of a network compared with ordinary task decomposition methods.

In this chapter, our discussion is restricted to single-layer PD networks. Section 3.2 presents a theoretical model to compare the performance of a single-layer PD network with the typical task decomposition network - Output Parallelism network. Section 3.3 presents some discussion to the model. In section 3.4, we introduce the Reduced Pattern Training method to improve the PD networks' performance. In Section 3.5, the experimental results are shown and analyzed. Conclusions are presented in Section 3.6.

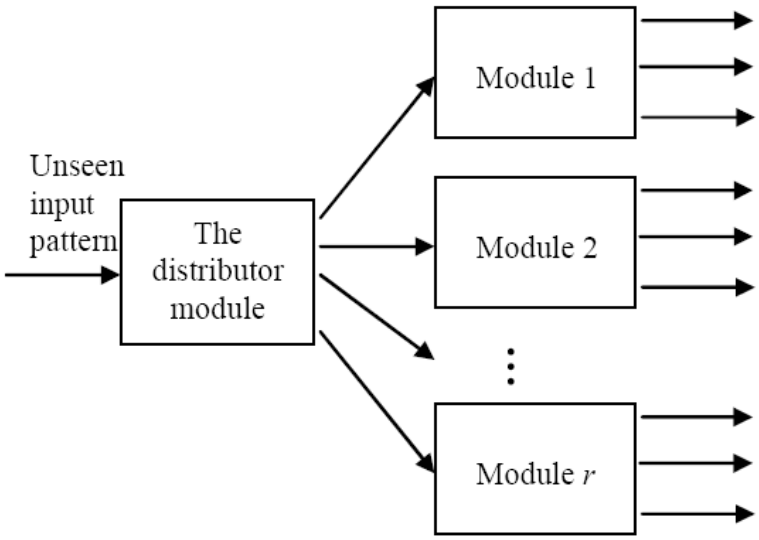

Figure 3.1: A typical Pattern Distributor network

3.2 A Theoretical Model for Single-layer PD Networks

There are two types of modules in a single-layer Pattern Distributor network, distributor module and non-distributor module (for simplicity, non-distributor modules are just called modules). Normally, a PD network consists of one distributor module and several non-distributor modules.

Class decomposition is often used in solving classification problems. Compared with ordinary methods in which only a neural network is constructed to solve the problem, class decomposition divides the problem into several sub-problems and trains a neural network module for each problem. Then the results from these modules are integrated to obtain the solution for the original problem. OP is a typical class decomposition method. Here we present a model to show that the PD method has better performance than the OP method when the recognition rate of the distributor module is guaranteed.

Consider a classification problem with K output classes. To solve the problem, a PD network with one distributor module and r non-distributor modules is constructed. See Figure 3.2 for details. There are r outputs in the distributor module and each non-distributor module is connected to an output of the distributor mod-

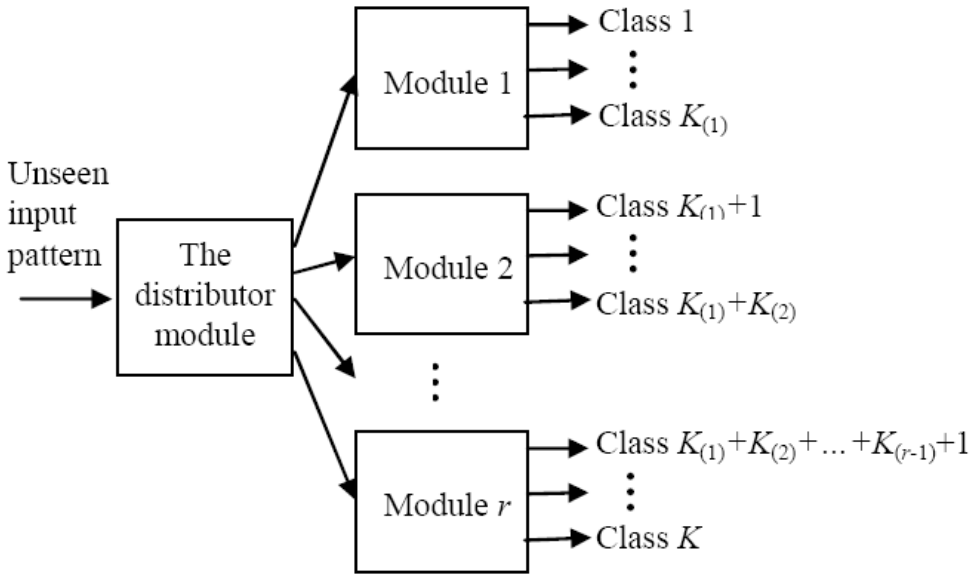

Figure 3.2: A single-layer PD network used to solve a K-class problem

ule. Each output of the distributor module is a combination of several classes. For an unknown pattern, the distributor module only recognizes and dispatches it to one of the outputs. Then the connected non-distributor module will continue the classification to specify which class it belongs to. In other words, a non-distributor module needs to recognize the pattern among several classes. Assume Module j which is a non-distributor module needs to recognize $K_{(j)}$ classes. Different non-distributor modules are assumed to have no overlapping classes, we have the relationship:

$$K = \sum_{j=1}^{r} K_{(j)} \quad . \tag{3.1}$$

Figure 3.3 shows the OP network used to solve the above K-class problem. For the convenience of comparison, we assume that the OP network has the same output grouping as the PD network. There are also r modules in the OP network and Module j needs to recognize $K_{(j)}$ classes among all the patterns. When an unknown test pattern is presented to the OP network, it is processed by each module (Module 1 to Module r), and the final result is obtained by integrating all the results from Module 1 to Module r.

In the PD network, a non-distributor module only recognizes the patterns which have been dispatched to it by the distributor module. These patterns most likely

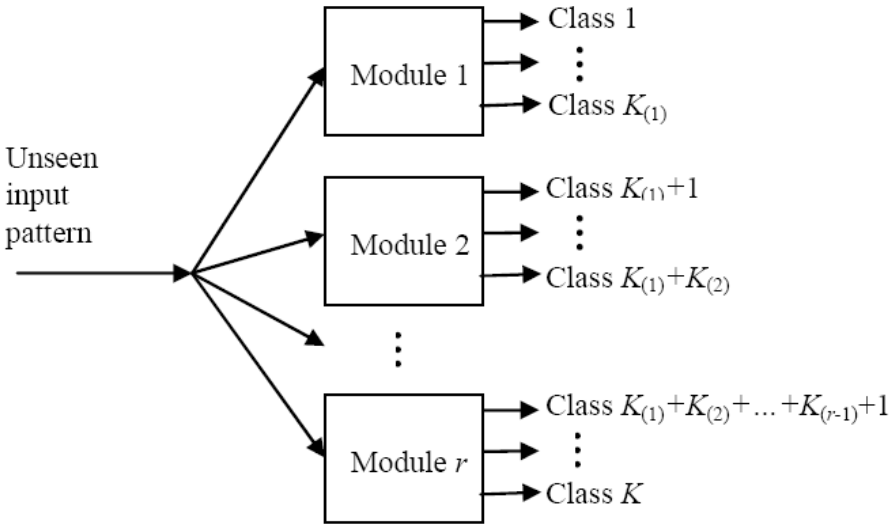

Figure 3.3: The OP network used for a K-class problem

belong to one of the classes covered by that module. Of course, the distributor module may make wrong decisions and mistakenly sends wrong patterns to that module. The OP network is different. Each module needs to recognize all the patterns. In other words, Module j in the OP network needs to differentiate the patterns belonging to it from those patterns which do not. Now we denote the probability of error having Module j processing the patterns that belong to one of the classes of Module i by p_{ji}. If we do not implement winner-takes-all selection, a pattern can be regarded as a wrongly classified pattern if one or more modules give wrong decisions. When a test pattern belonging to one of the classes of module j enters the network, the probability of error in the OP network can be written in the following form:

$$\begin{aligned} p_j^{(OP)} = {} & p_{1j}\prod_{i=2}^{r}(1-p_{ij}) + p_{2j}\prod_{i=1;i\neq 2}^{r}(1-p_{ij}) + \cdots + p_{rj}\prod_{i=1;i\neq r}^{r}(1-p_{ij}) \\ & + p_{1j}p_{2j}\prod_{i=1;i\neq 1,2}^{r}(1-p_{ij}) + p_{1j}p_{3j}\prod_{i=1;i\neq 1,3}^{r}(1-p_{ij}) + \cdots \\ & + p_{(r-1)j}p_{rj}\prod_{i=1;i\neq r-1,r}^{r}(1-p_{ij}) \\ & + \cdots + (p_{1j}p_{2j}\cdots p_{(r-1)j}p_{rj}) \quad . \end{aligned} \tag{3.2}$$

The first r terms represent the probability of a test pattern being classified wrongly by one module. The following $\frac{1}{2}r(r-1)$ terms represent the probability of the test

pattern being classified wrongly by two modules, and so on. Equation (3.2) can be rewritten as:

$$p_j^{(OP)} = p_{1j} + p_{2j} + \cdots + p_{rj} - (p_{1j}p_{2j} + p_{1j}p_{3j} + \cdots + p_{(r-1)j}p_{rj}) + (p_{1j}p_{2j}p_{3j} + \cdots + p_{(r-2)j}p_{(r-1)j}p_{rj}) - \cdots + (-1)^{r-1}(p_{1j}p_{2j}\cdots p_{(r-1)j}p_{rj}) \quad . \tag{3.3}$$

Here r is the number of modules and is normally not a large number. So the number of terms in the above equation is not a large number. p_{ij} is a small positive real number. In other words, $p_{ij}p_{kj}$ is much smaller than p_{ij}. We can ignore the terms of the product of two and more p_{ij}'s. Thus,

$$p_j^{(OP)} \approx p_{1j} + p_{2j} + \cdots + p_{rj} \quad . \tag{3.4}$$

The number of test patterns classified wrongly by the OP network is:

$$N^{(OP)} = \sum_{j=1}^{r} N_j p_j^{(OP)} \approx \sum_{j=1}^{r} N_j(p_{1j} + p_{2j} + \cdots + p_{rj}) = \sum_{j=1}^{r}[N_j \sum_{k=1}^{r} p_{kj}] \quad , \tag{3.5}$$

where N_j is the number of patterns belonging to the classes of Module j. It can also be written as:

$$N^{(OP)} = \sum_{k=1}^{r}(N_1p_{k1} + N_2p_{k2} + \cdots + N_rp_{kr}) \quad . \tag{3.6}$$

Now we define p_{k*} as the probability of error when Module k processes the patterns not belonging to the classes of Module k. Equation (3.6) can be revised as:

$$N^{(OP)} = \sum_{k=1}^{r}[N_kp_{kk} + (N - N_k)p_{k*}] \quad , \tag{3.7}$$

where p_{kk} is the probability of error when Module k processes the patterns belonging to it, N is the number of test patterns and N_k is the number of patterns belonging to the classes of Module k.

It should be mentioned that in the above OP network, each module can be trained separately using all the training patterns, whereas for the PD network, we can also train these modules separately. If we use all the training patterns to train these modules, then the weights and hidden units of the non-distributor modules will be the same as those of the corresponding modules in the OP network. After the training of the PD network is completed, the distributor module will be the first to classify any unseen input pattern. The corresponding output unit in the pattern

distributor will have the largest output value among all the output units. Then only the corresponding module will be activated. After that, the input pattern is presented to this module only and then this module will complete the classification process. Only the distributor module and the corresponding non-distributor module are used in the classification process.

Let p_0 be the probability of error of the distributor module. Then the number of test patterns which are classified wrongly by the distributor module is

$$M_0 = N \cdot p_0 \quad . \tag{3.8}$$

Assume the distributor module classifies patterns wrongly in a uniform manner. In other words, the number of wrongly classified patterns by the distributor module to each non-distributor module is proportional to the number of patterns entering that non-distributor module. The number of correct patterns that enter Module j is $N_j(1-p_0)$. Then, the number of patterns classified wrongly by Module j is written as:

$$M_j = N_j(1-p_0)p_{jj} \quad . \tag{3.9}$$

Thus, the number of patterns classified wrongly by the PD network can be expressed as:

$$M^{(PD)} = \sum_{j=0}^{r} M_j = N \cdot p_0 + (1-p_0)\sum_{j=1}^{r} N_j \cdot p_{jj} \quad . \tag{3.10}$$

Comparing the OP network with the PD network, we have

$$\begin{aligned} N^{(OP)} - M^{(PD)} &= \sum_{j=1}^{r}[N_j p_{jj} + (N-N_j)p_{j*}] - N \cdot p_0 - (1-p_0)\sum_{j=1}^{r} N_j \cdot p_{jj} \\ &= \sum_{j=1}^{r}(N-N_j)p_{j*} - N \cdot p_0 + p_0 \sum_{j=1}^{r} N_j \cdot p_{jj} \quad . \end{aligned} \tag{3.11}$$

Similar to the analysis made earlier, $p_0 p_{jj}$ is much smaller than p_{j*} and p_0, so

$$N^{(OP)} - M^{(PD)} \approx \sum_{j=1}^{r}(N-N_j)p_{j*} - N \cdot p_0 \quad . \tag{3.12}$$

Now we have derived the condition under which the PD network can achieve better classification accuracy than the OP network:

$$\sum_{j=1}^{r}(N-N_j)p_{j*} > N \cdot p_0 \quad . \tag{3.13}$$

We know that each module needs to process all the test patterns in an OP network, while in a PD network each non-distributor module only needs to process a sub-set of the test patterns. Intuitively, if the number of wrongly-classified patterns by the distributor module in the PD network is smaller than the sum of the number of patterns wrongly classified by each module when processing patterns not belonging to it in the corresponding OP network, the PD network will perform better.

3.3 Some Discussion to the Model

1.) Class decomposition can still be applied to the modules of the OP network and PD network so that these modules can be further decomposed into sub-modules. If each sub-module is used to recognize one class from all the patterns, then there will be N sub-modules in the whole OP network. Of course, these sub-modules may belong to different modules. Figure 3.4(a) shows an example of a 6-class OP network. There are two modules that are further partitioned into 6 sub-modules. Figure 3.4(b) shows a fully decomposed OP network for this 6-class problem. In both OP networks, all the training patterns are used to train these sub-modules. So the sub-modules in Figure 4(a) are the same as their counterparts in Figure 4(b). In Figure 4(a), the sub-modules are grouped into two modules. For an unknown pattern, the outputs from Sub-modules 1, 2, 3 are considered together to give the result of Module 1, similar for Module 2. Then the results from Module 1 and Module 2 are considered together to give the final output. In the OP network of Figure 4(b), the outputs from all the sub-modules are considered together to give the final output. In fact, there is little difference between the OP networks in Figures 4(a) and 4(b). Note that the non-distributor modules in the PD network are the same as the counterparts in the OP network. Thus, by decomposing the modules into sub-modules, we can compare the performance of the PD network with that of the fully decomposed OP network. In most of our experiments, we used networks like such.

2.) In Equation (3.4), we have ignored the situation in which two or more modules make wrong decisions at the same time because the situation appears much less frequently compared to the situation in which only one module makes

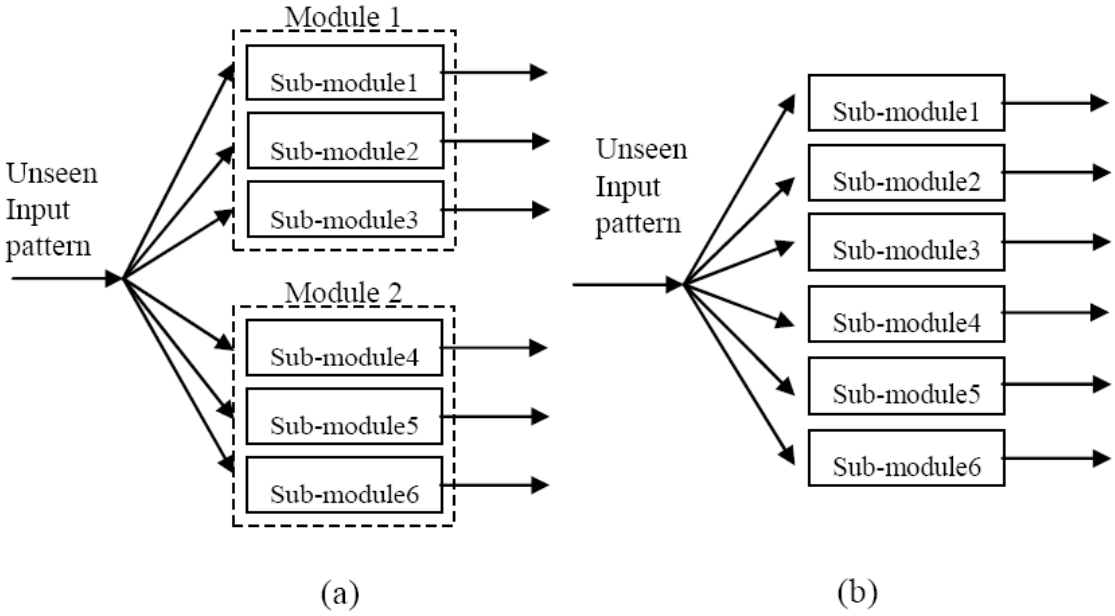

Figure 3.4: Two OP networks for a 6-class problem

wrong decisions. If we do consider that situation,$p_j^{(OP)}$ will be a little smaller than $p_{1j} + p_{2j} + \cdots + p_{rj}$.

3.) In the above model, we do not consider the implementation of winner-takes-all for the OP network. In reality, winner-takes-all is used for selecting a unit among several candidate units to produce the final output. The purpose of a conventional winner-takes-all network is to select a unit with the highest activation strength from a set of candidates. Using winner-takes-all, the network may still choose the correct output even if some modules make wrong decisions. For example, consider a test pattern that belongs to Class A in Module 1. When the pattern enters Module 1 of the OP network, Module 1 produces the correct answer - Class A. However, when the pattern enters Module 2 of the OP network, Module 2 gives an incorrect answer and thinks it belongs to Class B. If the output corresponding to Class A is larger than that of Class B, the OP network can still give a correct decision. Using winner-takes-all will slightly reduce the final classification error of the OP network than not using it.

3.4 Motivation for Reduced Pattern Training

In a PD network, an unseen pattern is firstly classified by the distributor module to decide which module will continue to process it. Then the corresponding module will be activated. Thus only two modules are used to process that pattern. Now we look at all the test patterns. We note that each non-distributor module only processes a subset of the test patterns. In other words, each non-distributor module only needs to recognize the patterns belonging to it if the distributor module classifies all the test patterns correctly. Also, if the distributor module classifies some patterns wrongly, the mistake can not be corrected by the later modules. This motivates us to train a non-distributor module using only the patterns belonging to it. Such a method is called Reduced Pattern Training (RPT). Similarly, the method of using the whole training set to train each non-distributor module is called Full Pattern Training (FPT).

When we train Module j using FPT, the module will carry information of the instances that do not belong to its own classes. Such information does not contribute to the classification accuracy of Module j. So it is useless. Also, training time would be reduced when training using RPT compared with FPT.

Moreover, training Module j together with unnecessary patterns may reduce the ability of Module j to classify the patterns belonging to Module j correctly. There are two aspects. Firstly, the objective of training is to let each module reach its best classification accuracy when processing the patterns dispatched to it. Using FPT, a module may be able to attain its best performance when it needs to process all the test instances. However, it may not attain its best performance when processing only a subset of the test instances. Secondly, for patterns not belonging to Module j it would have the outputs as 0 during the learning process. (In our experiments, if a pattern belongs to some class, the corresponding output is 1, otherwise, 0). With the introduction of those patterns not belonging to Module j, there are much more patterns with an output label 0 than patterns with an output label 1 in the learning process. So the patterns with an output label 0 will be more influential in updating the weights and therefore in computing the training error function. In contrast, the patterns with an output label 1 will become less influential in the decision of weight

updates. After the training process is over, it is likely that the trained network may mislabel some test patterns, in particular those patterns with an output label 1. From the above observations, we conclude those unnecessary patterns are harmful to the module training. Our experimental results confirmed that RPT is crucial for a PD network to obtain good performance.

Reduced pattern training might not be applicable to OP, because the modules in an OP network operate in parallel and each module must deal with all the test patterns in the test process. Training these modules using reduced patterns may lead to information loss. And it may lead to poor accuracy when the test patterns are presented.

3.5 Experimental Results for Single-layer PDs

3.5.1 Experimental Scheme

The CBP algorithm was used to train the network in the experiments (Lehtokangas, 1999). CBP can reduce the excessive computational cost significantly and it does not require any prior knowledge concerning decomposition. In our experiments, RPROP is used. In order to reduce the number of freely adjustable parameters, often leading to a tedious search in parameter space, the increase and decrease factors η^+ and η^- are set to fixed values: $\eta^+ = 1.2$ and $\eta^- = 0.5$. See Riedmiller and Braun's paper for considerations which led to these values (1993). At the beginning of the algorithm, all update-values Δ_{ij} are set to an initial value Δ_0. A good choice may be $\Delta_0 = 0.1$. However, the choice of this parameter is not critical at all, for it is adapted as learning proceeds. In order to prevent the weights from becoming too large, the maximum weight-step determined by the size of the update-value is limited. The upper bound is set by the second parameter of RPROP, Δ_{max}. The default upper bound is set somewhat arbitrarily to $\Delta_{max} = 50$. Usually, the convergence is rather insensitive to this parameter as well. The minimum step size is constantly fixed to $\Delta_{min} = 1.0e-6$. The initial weights selected from $-0.25 \cdots 0.25$ randomly. In order to avoid large computational cost and overfitting, a method called early stopping based on validation set is used as the stopping criteria. Please refer to RPROP algorithm for detail.

The set of available patterns is divided into three sets: a training set is used to train the network, a validation set is used to evaluate the quality of the network during training and to measure overfitting, and a test set is used at the end of training to evaluate the resultant network. The size of the training, validation, and test sets is 50%, 25% and 25% of the problem's total available patterns.

Four benchmark classification problems, namely Vowel, Glass, Segmentation, and Letter Recognition were used to evaluate the performance of the single-layer PD networks. These classification problems were taken from the PROBEN1 benchmark collection (Prechelt, 1994) and University of California at Irvine (UCI) repository of machine learning database (Blake and Merz, 1998). In the set of experiments undertaken, the first three classification problems were conducted 20 times and the Letter Recognition problem was conducted 8 times (due to the long training time). All the hidden units and output units use the sigmoid activation function and E_{th} is set at 0.1 (refer Appendix for detail). When a hidden unit addition was required, 8 candidates were trained and the best one selected. All the experiments were simulated on a Pentium IV - 2.4GHZ PC. The sub-problems were solved sequentially and the CPU time expended was recorded respectively.

For the experiments in later chapters, the scheme is the same as that used here.

3.5.2 Experiments for Single-layer PD Network Based on Full and Reduced Pattern Training

A. Glass

This data set is used to classify glass types. The data set consists of 9 inputs, 6 outputs, and 643 patterns (divided into 321 training patterns, 161 validation patterns, and 161 test patterns). These patterns were normalized and scaled so that each component lies within [0, 1].

Figure 3.5 shows the OP network structure used for this problem. The OP network is composed of 6 sub-modules and each sub-module recognizes one class from

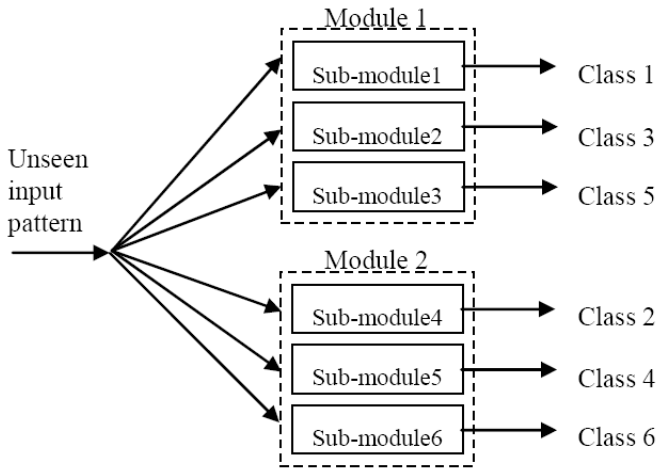

Figure 3.5: The OP network used for the Glass problem

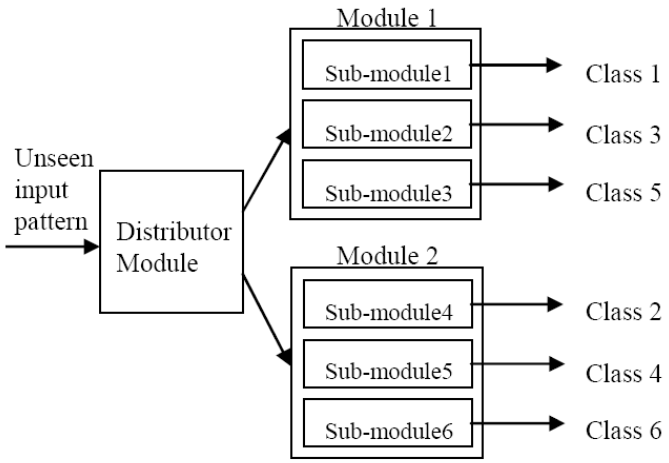

Figure 3.6: The PD network used in the Glass problem

all the patterns. As described in Discussion 1 in Section 2, these sub-modules are combined into 2 modules in the OP network. The sub-modules which recognize class 1, class 3 and class 5 are combined into Module 1 and the remaining sub-modules are grouped into Module 2.

Table 3.1 lists some data which are used in expression (3.13). Here N_i represents the number of patterns in the test data set belonging to the classes of Module i while N denotes the overall number of the patterns. p_{ii} is the probability of error when Module i processes the patterns belonging to Module i and p_{i*} is the probability of error when Module i processes the patterns not belonging to Module i. Now we

Table 3.1: Classification errors in different OP modules for the Glass data

Output Parallelism	N_i	$p_{ii}(\%)$	$N - N_i$	$p_{i*}(\%)$
Module 1	67	8.4142	94	4.7340
Module 2	94	18.1383	67	2.1642

Table 3.2: Results for the Glass data

Method		Training time (s)	Hidden Units	Indep. Param.	C.error (%)
Ordinary method (no task decomposition)		168.1	46	796	16.0870
Output Parallelism (2 modules, 6 sub-modules)		63.7 (in parallel) 197.7 (in series)	253.5	2848.5	14.2547
Pattern Distributor	Distributor module	82.9	30.6	387.2	2.4224
	Overall network (FPT)	85.2 (in parallel) 298.7 (in series)	280.9	3200.5	10.0
	Overall network (RPT)	82.9 (in parallel) 194.3 (in series)	391.2	4413.8	7.8261

show that Discussion 2 in Section 3.3 is reasonable. There are two modules in the OP network. From Table 3.1, we have $p_{11} = 8.4124\%$ and $p_{12} = p_{1*} = 4.7340\%$. So $p_{11} * p_{12} = 0.4\%$, which is much smaller than p_{11} and p_{12}. It is similar that $p_{21} * p_{22} = 0.39\%$, which is much smaller than p_{21} and p_{22}. Ignoring these terms has little effect to the final results. In other words, the situation in which two or more modules making wrong decisions at the same time can be ignored. Now we follow up Discussion 3 in Section 3.3 - the effect of winner-takes-tall. From Table 3.1, we can compute the classification error before the implementation of winner-takes-all, which is $N_1 p_{11} + N_2 p_{1*} + N_1 p_{2*} + N_2 p_{22} = 17.7562\%$. The result is slightly larger than the result using winner-takes-all, which is 14.2547% (see Table 3.2). It also matches our analysis in Discussion 3, Section 3.3.

The PD network structure for this problem is shown in Figure 3.6. The distributor module has two outputs, one has the combination {1,3,5} while the other has {2,4,6}. Module 1 consists of 3 sub-modules, identical to its counterpart in the OP network, and same for Module 2.

Table 3.2 shows the experimental results of the ordinary method, the OP method, the PD method with FPT and the PD method with RPT. The ordinary method is a method in which a single-module neural network was constructed to solve the problem. Constructive Backpropagation (CBP) algorithm is still used in the ordinary method. "Indep. Param." stands for the total number of independent parameters (i.e., the number of weights and biases in the network). "C. Error" stands for classification error. Training time (in parallel) is the maximum training time among all the modules (all modules were trained in parallel). Training time (in series) stands for the sum of training time for all the modules (all modules were trained in series). Using the ordinary and the OP methods, the classification errors were 16.0870% and 14.2547% respectively, while using the PD method, the classification errors were 10% for FPT and 7.8261% for RPT. Comparing with the classification errors from the former two approaches, the classification errors obtained by the PD network are much smaller. It can be also noted that the classification error is further reduced when using RPT instead of FPT.

Now we explain why the PD network can achieve smaller classification error than the other two methods. According to our analysis, if Expression (3.13) is satisfied, the PD network will have better classification accuracy. Using the data in Table 5.1, we have $\sum_{j=1}^{2}(N - N_j)p_{j*} \approx 5.9$. From the classification error Np_0 of the distributor module in Table 3.2, we find $Np_0 \approx 3.9$. Thus, Condition (3.13) is satisfied, which means that using the PD network will get smaller classification error. From Table 2, we can see that the number of hidden units and the number of independent parameters in the PD network are larger than those in the ordinary network and the OP network. This can be attributed to the fact that the PD network has more modules than the other two. From Table 2, we can also note the changes of the training time using the above three methods. With series training, the training time of FPT (298.7s) is larger than those of the ordinary network (168.1s) and the

Table 3.3: Classification errors in different OP modules for the Vowel data

Output Parallelism	N_i	$p_{ii}(\%)$	$N - N_i$	$p_{i*}(\%)$
Module 1	69	9.8551	178	2.9775
Module 2	96	32.0833	151	3.2450
Module 3	82	34.7561	165	5.8788

OP network (197.7s) due to a large number of modules in the PD method. However, the training time of RPT (194s) is reduced compared to that of FPT and is thus comparable to the training time of the other two networks. The reason for this is that the number of training instances used in RPT is smaller than that in FPT. With parallel training, the training time of the PD network (RPT or FPT) is similar to those of the other two methods, and it is even shorter than that of the ordinary method. From the above analysis, we see that the PD method, especially RPT, performs better than the other methods.

B. Vowel

The input patterns of this data set are 10 element real vectors representing vowel sounds that belong to one of 11 classes. It has 990 patterns in total (they are divided into 495 training patterns, 248 validation patterns, and 247 test patterns). The patterns were normalized and scaled so that each component lies within [0, 1]. The distributor module has 3 outputs, {1,2,3}, {4,5,6,7} and {8,9,10,11}. Module 1 recognizes classes 1, 2, 3 and consists of 3 sub-modules. Module 2 recognizes classes 4,5,6,7 and consists of 4 sub-modules, while Module 3 recognizes classes 8,9,10,11 and consists of 4 sub-modules. The OP network has the same Module 1, Module 2 and Module 3 as the PD network.

The experimental results of the ordinary method, the OP method and the PD method for the Vowel data are listed in Table 3.4. Using the ordinary method and the OP method, the classification errors were 37.1660% for the ordinary method and 25.5466% for the OP method respectively, while using the PD method, the classification errors were 24.8987% for FPT and 18.7045% for RPT. The classification error obtained by FPT is much smaller than the classification error of the ordinary method and resembles that of the OP method. While for RPT, the classification er-

Table 3.4: Results for the Vowel data

Method		Training time (s)	Hidden Units	Indep. Param.	C.error (%)
Ordinary method (no task decomposition)		237.9	23.6	640.2	37.1660
Output Parallelism (3 modules, 11 sub-modules)		58.7 (in parallel) 418.9 (in series)	184.4	2333.8	25.5466
Pattern Distributor	Distributor module	117	24.5	376	6.6802
	Overall network (FPT)	117 (in parallel) 534.3 (in series)	210.6	2730.2	24.8987
	Overall network (RPT)	117 (in parallel) 245.6 (in series)	229.4	2955.8	18.7045

ror is decreased to 18.7045%, which is much smaller than those of FPT and the other two methods. We can compute the number of wrongly-classified patterns using the data in Table 3.3 to explain why the PD method can get smaller classification errors than the other two methods. We have $\sum_{j=1}^{3}(N - N_j)p_{j*} \approx 19.9$ while $Np_0 \approx 16.5$. Expression (3.13) is satisfied. Thus the PD network has smaller classification errors. From Table 3.4, we can see that the number of hidden units and the number of independent parameters in the PD network (RPT or FPT) are larger than those in the ordinary and OP networks. Table 3.4 also shows the training time using these methods. Using series training, the training time of FPT (534.3s) is longer than those of the ordinary network (237.9s) and the OP network (418.9s). The training time of RPT (245.6s) is much reduced compared to that of FPT and is also smaller than those of the former two networks. If parallel training is used, the training process of the PD network can save more time. From the above analysis, we see that RPT outperforms the others.

C. Segmentation

This data set consists of 18 inputs, 7 outputs, and 2310 patterns (1155 train-

Table 3.5: Classification errors in different OP modules for the Segmentation data

Output Parallelism	N_i	$p_{ii}(\%)$	$N - N_i$	$p_{i*}(\%)$
Module 1	246	10.4129	331	0.2417
Module 2	331	0.9215	246	0.6098

Table 3.6: Results for the Segmentation data

Method		Training time (s)	Hidden Units	Indep. Param.	C.error (%)
Ordinary method (no task decomposition)		693.8	29	887	5.7366
Output Parallelism (7 sub-modules)		610.2 (in parallel) 1719.6 (in series)	152.1	3175	5.1820
Pattern Distributor	Distributor module	213.4	13.9	329.9	1.0399
	Overall network (FPT)	1002.2 (in parallel) 2219.2 (in series)	128.5	2754.9	5.4419
	Overall network (RPT)	213.4 (in parallel) 706.9 (in series)	128.9	2762.9	4.6101

ing patterns, 578 validation patterns, and 577 test patterns). The patterns were normalized and scaled so that each component lies within [0, 1]. The distributor module has 2 outputs, {3,4,5} and {1,2,6,7}. Module 1 recognizes classes 3, 4, 5 and consists of 3 sub-modules. Module 2 recognizes classes 1, 2, 6, 7 and consists of 4 sub-modules. The OP network has the same module composition as the PD network.

Table 3.6 shows the simulation results of the ordinary method, the OP method, the PD method (FPT and RPT). Using the ordinary method and the OP method, the classification errors were 5.7366% and 5.1820% respectively, while using the PD method, the classification errors were 5.4419% for FPT and 4.6101% for RPT. From Table 3.5, we have $\sum_{j=1}^{2}(N - N_j)p_{j*} \approx 2.3$. From Table 3.6, we find $Np_0 \approx 6.0$. So Expression (3.13) is not satisfied and FPT has a larger classification error than the

Table 3.7: Classification errors in different OP modules for the Letter data

Output Parallelism	N_i	$p_{ii}(\%)$	$N - N_i$	$p_{i*}(\%)$
Module 1	1359	20.833	3641	2.856
Module 2	1333	24.812	3667	1.084
Module 3	1195	25.109	3805	3.035
Module 4	1113	11.051	3889	1.826

OP network. It is also noted that the classification error is decreased when using RPT to replace FPT. From Table 3.6, we can see that the number of hidden units and the number of independent parameters in the PD networks are larger than those in the ordinary and OP networks. From Table 3.6, we also notice changes in training time using the above three methods. Under series training, the training time of FPT (2219.2s) is larger than the training times of the ordinary network (693.8s) and the OP network (1719.6s) due to a large number of modules in the PD network. However, the training time of RPT (706.9s) is reduced compared to that of FPT and the OP network and is thus comparable to the training time of the ordinary method. With parallel training, the training time of RPT is the smallest one. From the above analysis, we see that RPT performs better than the other methods.

D. Letter recognition

The goal of this data is to recognize digitized patterns. Each element of the input vector is a numerical attribute computed from a pixel array containing the letters. This data set consists of 16 inputs, 26 outputs, and 20000 patterns (10000 training patterns, 5000 validation patterns, and 5000 test patterns). All the patterns were normalized and scaled so that each component lies within [0, 1]. The distributor module has 4 outputs, {1,2,3,4,5,6,7}, {8,9,10,11,12,13,14}, {15,16,17,18,19,20} and {21,22,23,24,25,26}. Module 1 recognizes classes 1,2,3,4,5,6,7. Due to the long training time of this problem, Module 1 is not further divided into sub-modules. Module 2 recognizes classes 8,9,10,11,12,13,14, Module 3 recognizes classes 15,16,17,18,19,20 and Module 4 recognizes classes 21,22,23,24,25,26. The OP network has the same module composition as the PD network. For a fair comparison with the PD network, sub-modules are not used in the OP network.

Table 3.8: Results for the Letter data

Method		Training time (s)	Hidden Units	Indep. Param.	C.error (%)
Ordinary method (no task decomposition)		20845.05	73.6	3607	21.672
Output Parallelism (4 modules)		5519 (in parallel) 18112.6 (in series)	173.4	6586.8	19.260
Pattern Distributor	Distributor module	2510 (in parallel) 8497 (in series)	219.5	4019.0	12.195
	Overall network (FPT)	6110 (in parallel) 26723.8 (in series)	386.25	8384.5	20.515
	Overall network (RPT)	2510 (in parallel) 14094.5 (in series)	344.25	7391.0	15.855

The experimental results of the ordinary method, the OP method and the PD method for the Letter data are listed in Table 3.8. Using the ordinary method and the OP method, the classification errors were 21.672% for the ordinary method and 19.260% for the OP method respectively. Using the PD method, the classification error were 20.515% for FPT and 15.855% for RPT. The classification error obtained by FPT resembles the classification errors using the ordinary method and the OP method. Using RPT, the classification error is much smaller than the classification errors of the other three networks. From Table 3.7, we have $\sum_{j=1}^{4}(N - N_j)p_{j*} \approx 330.3$. From Table 3.8, we find $Np_0 \approx 609.8$. So Expression (3.13) is not satisfied, which means that FPT has a larger classification error. From Table 3.8, we see that the number of hidden units and the number of independent parameters in the PD network are larger than those in the ordinary and OP networks. Table 3.8 also shows the training time using these methods. Under series training, the training time of FPT (26723.8s) is larger than those of the ordinary network (18112.6s) and the OP network (20845.05s). The training time of RPT (14094.5s) is greatly decreased compared to that of FPT and is also smaller than those of the former two networks. If parallel training was used, the training process of RPT could save more time. From the above analysis, we can see that RPT performs better than the others.

3.6 Conclusions

In this chapter, we discussed the single-layer PD network. In this design, a special module called the distributor module was introduced in order to improve the accuracy of the whole network. A theoretical model was proposed to compare the performance of PD with that of OP - a typical class decomposition method. The analysis showed that PD can outperform OP when the classification accuracy of the distributor module is guaranteed. The experimental results confirmed this. In order to further improve the performance of the PD network, RPT was introduced. RPT apparently increased the accuracy of the PD network and reduced the whole training time. For these reasons, in our experiments in later chapters, RPT is used in each set of experiments.

Chapter 4

Multi-layer PD Networks

4.1 Introduction

The last chapter discussed single-layer PD networks comprehensively based on a theoretical model. Here we extend the concept of single-layer PD networks to multi-layer PD networks. In single-layer PD networks, the non-distributor modules often need to recognize several classes (i.e. three or more classes). In order to improve the classification rate of these non-distributor modules, OP can be used on these modules. Or these non-distributor modules are further decomposed into sub-modules. Enlightened by the above idea, we may extend the PD method to these non-distributor modules to improve their performance. Multi-layer PD networks are the outcome by applying the PD method to the non-distributor modules of single-layer PD networks.

4.2 Design of Multi-layer PD Networks

If there are two or more distributor modules in a PD network and these distributor modules are arranged in a hierarchical structure, the PD network is called a multi-layer PD network. Figure 4.1 shows a 2-layer PD network. There are two distributor modules in the networks, one sitting in the first level of the network and the other in the second level of the network. For the remaining non-distributor modules, some modules are positioned in Level 2 and others are in Level 3. An imbalanced multi-layer PD network is defined as the multi-layer PD network whose non-distributor modules are positioned in different layers of the network. The network in Figure 4.1 is a typical imbalanced multi-layer PD network.

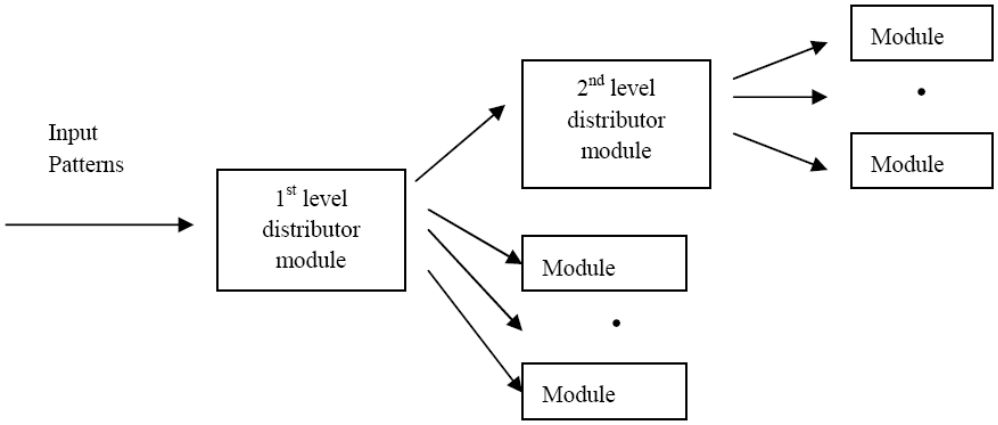

Figure 4.1: An imbalanced 2-layer PD network

When an unseen input pattern enters the network, it is processed by the first level distributor module and classified into one of its outputs. Then the corresponding module will continue to classify it. If the pattern is not classified to the first output of the first level distributor module, a non-distributor module continues to process this pattern. Thus, the number of modules which process the pattern will be two in this situation. If the pattern is classified to the first output of the first level distributor module, the second level distributor module continues to process this pattern. After that, a non-distributor module will continue to recognize it. In this situation, the number of modules which process the pattern will be three.

In fact, besides imbalanced multi-layer PD networks, balanced multi-layer PD networks can also be used. In a balanced multi-layer PD network, all the non-distributor modules are positioned in the same layer. Figure 4.2 shows an example of a balanced 2-layer PD network.

In the above example, after the first level distributor module (Distributor module 1) partitions the original problem into two sub-problems, each sub-problem will then be handled by the corresponding second level distributor module and the second level distributor module will perform the classification task and further separate the sub-problem into smaller sub-problems. Then the other modules will perform the remaining classification tasks.

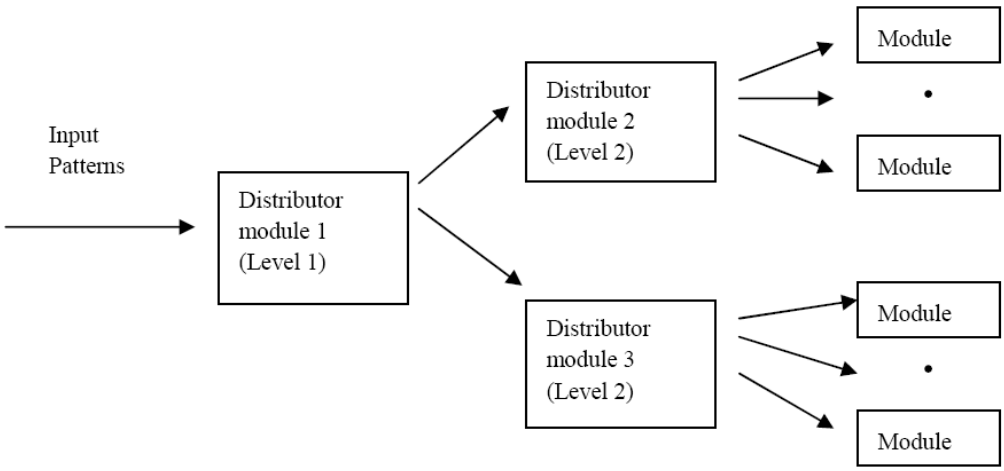

Figure 4.2: A balanced 2-layer PD network

In the above balanced 2-layer PD network, the set of distributor modules can be seen as a whole. In other words, distributor modules 1, 2 and 3 are integrated into a super distributor module. The PD method is used in this super distributor module and the super distributor module is divided into three small modules.

4.3 Theoretical Analysis for Two-layer PD Networks

A Multi-layer PD network can be seen as build-in single-level PD networks in the modules of the original PD network. Firstly, we give some simple analysis for an imbalanced 2-level PD network. Assume a classification problem is solved using a 2-level PD network similar to the one shown in Figure 4.3. The classification problem has K classes. Class 1 to class r_1 are recognized using Module 1, class $r_1 + 1$ to class r_2 are recognized using Module 2, class $r_2 + 1$ to class r_3 are recognized using Module 3 and class $r_3 + 1$ to class K are recognized using Module 4. Assume the number of the patterns which belong to class 1 to class r_1 is N_1, the number of the patterns which belong to class $r_1 + 1$ to class r_2 is N_2, the number of the patterns which belong to class $r_2 + 1$ to class r_3 is N_3, and the number of the patterns which belong to class $r_3 + 1$ to class K is N_4.

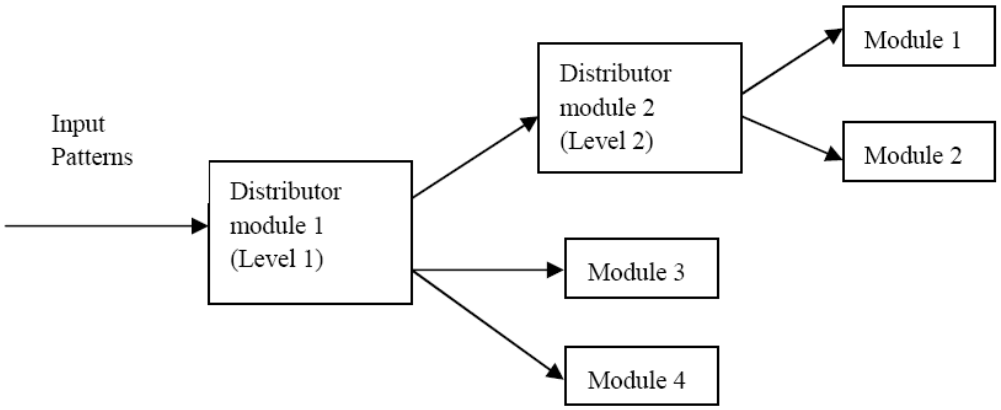

Figure 4.3: An imbalanced 2-layer PD network with 4 non-distributor modules

Let p_1 be the probability of error of Distributor module 1. Then the number of test patterns which are classified wrongly by Distributor module 1 is

$$M_1 = N \cdot p_1 \quad , \tag{4.1}$$

where N is the number of the overall test patterns. Assume the distributor modules classify patterns wrongly in a uniform manner. In other words, the number of wrongly classified patterns by the distributor modules to each output is proportional to the number of patterns entering that non-distributor module. The number of correct patterns that enter Distributor module 2 is $(N_1 + N_2)(1 - p_1)$. The number of patterns classified wrongly by Distributor module 2 is written as:

$$M_2 = (N_1 + N_2)(1 - p_1)p_2 \quad , \tag{4.2}$$

where p_2 be the probability of error of Distributor module 2.

Assume that the probability of error of Module i is denoted by $p_i^{(n)}$ (here n represents a non-distributor module). Let $M_i^{(n)}$ be the number of patterns classified wrongly by Module i. Then we have

$$M_i^{(n)} = N_i(1 - p_1)(1 - p_2)p_i^{(n)} \quad , \tag{4.3}$$

where $i = 1, 2$;

$$M_i^{(n)} = N_i(1 - p_1)p_i^{(n)} \quad , \tag{4.4}$$

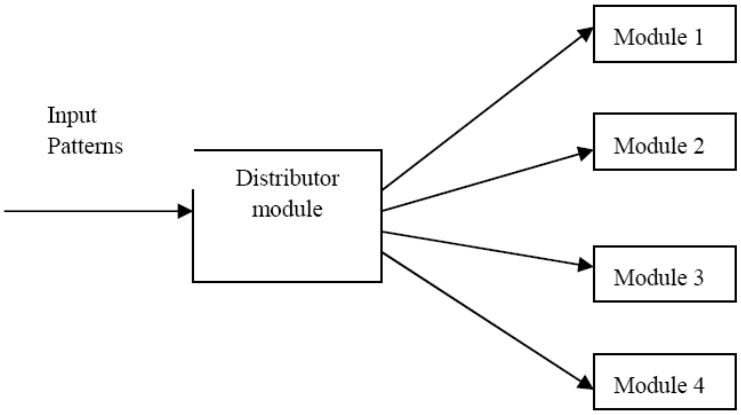

Figure 4.4: A single-layer PD network with 4 non-distributor modules

where $i = 3, 4$. The number of patterns classified wrongly by all the two distributor modules is expressed as:

$$M_2 = N \cdot p_1 + (N_1 + N_2)(1 - p_1)p_2 \quad . \tag{4.5}$$

Because p_i is a small positive real number, $p_i p_j$ is much smaller than p_i. Equation (4.5) can be written as:

$$M_1 + M_2 \approx N \cdot p_1 + (N_1 + N_2)p_2 \quad . \tag{4.6}$$

From Expressions (4.3), (4.4) and (4.6), the number of patterns wrongly classified by the whole imbalanced two-layer PD network is written as

$$\begin{aligned} M^{(iPD)} &= M_1 + M_2 + \sum_{i=1}^{4} M_i^{(n)} \\ &\approx Np_1 + (N_1 + N_2)p_2 + N_1 p_1^{(n)} + N_2 p_2^{(n)} + N_3 p_3^{(n)} + N_4 p_4^{(n)} \quad . \end{aligned} \tag{4.7}$$

Figure 4.4 shows a single-layer PD network used to solve the above problem. In this network, the four non-distributor modules are the same as those in the above 2-layer PD network [1]. Let the probability of error of the distributor module is $p_0^{(s)}$.

[1]Same as those in the imbalanced two-layer PD networks, in the single layer PD, class 1 to class r_1 are recognized using Module 1, class $r_1 + 1$ to class r_2 are recognized using Module 2, class $r_2 + 1$ to class r_3 are recognized using Module 3 and class $r_3 + 1$ to class K are recognized using Module 4.

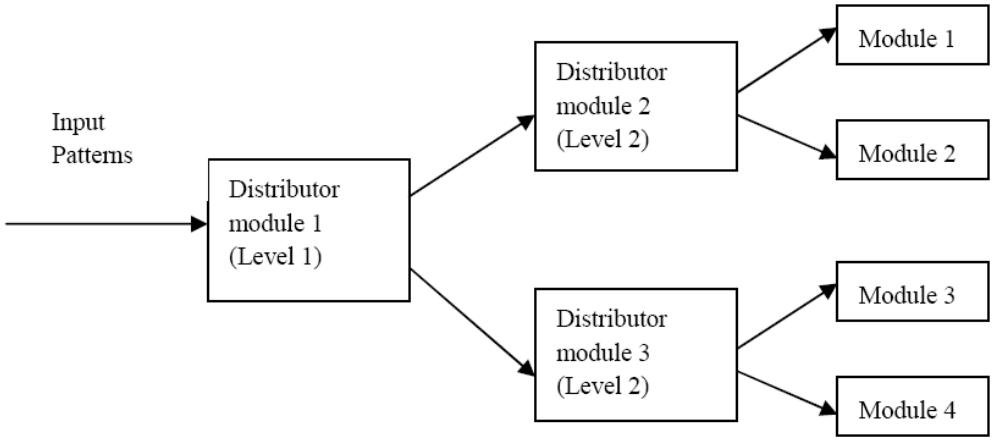

Figure 4.5: A balanced two-layer PD network with 4 non-distributor modules

Then the number of patterns which are wrongly classified by the whole network is

$$\begin{aligned} M^{(sPD)} &= Np_0^{(s)} + \sum_{i=1}^{4} N_i(1-p_0^{(s)})p_i^{(n)} \\ &\approx Np_0^{(s)} + \sum_{i=1}^{4} N_i p_i^{(n)} \quad . \end{aligned} \tag{4.8}$$

From Equation (4.7) and (4.8), we can derive the condition that the imbalanced 2-layer PD network performs better than the single-layer PD network, which is

$$M^{(iPD)} - M^{(sPD)} = Np_1 + (N_1+N_2)p_2 - Np_0^{(s)} < 0 \quad . \tag{4.9}$$

Thus,

$$p_2 < \frac{N(p_0^{(s)} - p_1)}{N_1+N_2} \quad , \tag{4.10}$$

where p_1 and p_2 are the probability of error of Distributor module 1 and 2 of the 2-layer PD network respectively and $p_0^{(s)}$ is the probability of error of the distributor module in the single-layer PD network. When Expression (4.10) is satisfied, the 2-layer PD will have better performance.

Now we look at a balanced 2-layer PD network. See Figure 4.5. We can regard the three distributor modules as an integration. It is obvious that if the distributor modules in the balanced 2-layer network make fewer mistakes than the distributor module in the single-layer network, the 2-layer network will have better performance.

Let $p^{(d)}$ be the probability of error of all the distributor modules in the 2-layer PD network. We have

$$p^{(d)} = \frac{(N_1 + N_2)(p_1 + p_2) + (N_3 + N_4)(p_1 + p_3)}{N} \quad , \tag{4.11}$$

where p_i be the probability of error of Distributor module $i(i = 1, 2, \ldots, 3)$. The condition that allows the balanced 2-layer PD network to outperform the single-layer PD network can be written as:

$$p^{(d)} - p_0^{(s)} < 0 \quad , \tag{4.12}$$

which means that the condition for the 2-layer PD network outperforms the single-layer PD network is that the probability of error of all the distributor modules $p^{(d)}$ in the 2-layer PD network is smaller than the probability of error of the distributor module $p_0^{(s)}$ in the single-layer PD network.

From the discussion above, it can be seen that when the classification error of distributor modules is small enough, the multi-layer PD networks can achieve lower classification error than the single-layer PD networks. For other multi-layer PD configuration, we can reach similar results using similar analysis.

4.4 Experimental Results for Multi-layer PD Networks

Here some experimental results are presented for multi-level PD networks. The set up of these experiments was the same as that in Chapter 3.

4.4.1 Experimental Results for Balanced Two-layer PD Networks

A. Vowel

The structure of the balanced 2-layer PD network is the same as the network in Figure 4.5. Distributor module 1 has 2 outputs, {3,4,6,5,7,11} and {8,9,10,1,2}. The two outputs of Distributor module 2 is {3,4,6} and {5,7,11} and the two outputs of Distributor module 3 is {1,2,10} and {8,9}. Module 1 recognizes classes 3,4,6

Table 4.1: Results of the balanced 2-layer PD for the Vowel data

Method		Training time (s)	Hidden Units	Indep. Param.	C.error (%)
Single-layer Pattern Distributor	Distributor module	51.8 (in parallel) 189.6 (in series)	72.8	917.6	9.1296
	Overall network	51.8 (in parallel) 345.35 (in series)	139.5	1839	18.1983
Two-layer Pattern Distributor	Distributor modules	88 (in parallel) 238.35 (in series)	95.55	1212.6	7.8702
	Overall network	88 (in parallel) 394.1 (in series)	162.25	2134	16.9231

and consists of 3 sub-modules. Module 2 recognizes classes 5,7,11 and consists of 3 sub-modules. It is similar that Module 3 recognizes classes 1,2,10 and consists of 3 sub-modules and Module 4 recognizes classes 8,9 and also consists of 2 sub-modules. For the comparison, a single-layer PD network whose structure is the same as the network in Figure 4.4 is used. The distributor module of the single-layer PD network has 4 outputs, {3,4,6},{5,7,11},{1,2,10} and {8,9} and consists of 4 sub-modules. Its non-distributor modules are the same as their counterparts used in the 2-layer PD network.

The experimental results of the single-layer PD network and the multi-layer PD network using RPT are listed in Table 4.1. According the analysis in Section 4.3, if $p^{(d)} < p_0^{(s)}$, the multi-layer PD network will have higher recognition rate. From Table 4.1, the classification error of the distributor modules (7.8702%) using the multi-layer PD network is smaller than that using the single-layer PD network (9.1296%). Using the single-layer PD network, the classification error is 18.1983%, while using the multi-layer PD network, the classification error is 16.9231%. Thus, the multi-layer PD network performs better than the single-layer PD network and it matches

our analysis. The classification error decreases about 1.3% in this example. From Table 4.1, we can see that the number of hidden units and the number of independent parameters in the multi-layer PD network are a little larger than those in the single-layer PD network because the multi-layer PD has more modules than the single-layer PD. Table 4.1 also shows the training time using these methods. Using series training, the training time of the multi-layer PD (394.1s) is a little longer than that of the single-layer PD (345.35s).

B. Segmentation

The structure of the 2-layer PD network is also the same as the network in Figure 4.5. Distributor module 1 has 2 outputs, {1,2,3} and {4,5,6,7}. The two outputs of Distributor module 2 is {1} and {2,3} and the two outputs of Distributor module 3 is {4,5} and {6,7}. Module 1 recognizes class 1. Module 2 recognizes classes 2,3 and consists of 2 sub-modules. It is similar that Module 3 recognizes classes 4,5 and consists of 2 sub-modules and Module 4 recognizes classes 6,7 and also consists of 2 sub-modules. For comparison, a single-layer PD network whose structure is the same as the network in Figure 4.4 is used. The distributor module of the single-layer PD network has 4 outputs, {1},{2,3},{4,5},{6,7} and consists of 4 sub-modules. Its non-distributor modules are the same as their counterparts used in the 2-layer PD network.

Table 4.2 shows the experimental results of the single-layer PD network and the multi-layer PD network using Reduced Pattern Training (RPT). Using the single-layer PD network, the classification error was 4.7660%, while using the multi-layer PD network, the classification error was 4.6447%. There is just a little improvement for this problem. According the analysis in Section 4.3, if $p^{(d)} < p_0^{(s)}$, the multi-layer PD network will have higher recognition rate. From Table 4.2, the distributor modules' classification error of the balanced multi-layer PD network (3.9515%) is a little smaller than that of the single-layer PD network (4.0035%). Thus, the performance of the multi-layer PD network improves a little. From Table 4.2, we can see that the number of hidden units and the number of independent parameters in the multi-layer PD network are larger than those in the single-layer PD network. This can be attributed to the fact that the multi-layer PD has more modules than

Table 4.2: Results of the balanced 2-layer PD for the Segmentation data

Method		Training time (s)	Hidden Units	Indep. Param.	C.error (%)
Single-layer Pattern Distributor	Distributor module	276.6 (in parallel) 461.7 (in series)	61.4	1304	4.0035
	Overall network	276.6 (in parallel) 521.5 (in series)	128.9	2730	4.7660
Two-layer Pattern Distributor	Distributor modules	220.9 (in parallel) 538.9 (in series)	58.4	1244	3.9515
	Overall network	220.9 (in parallel) 598.7 (in series)	1165.8	3506	4.6447

the single-layer PD. From Table 4.2, we can also note the changes of the training time using the above two methods. With series training, the training time of the multi-layer PD network (598.7s) is a little longer than that of the single-layer PD network (521.5s) due to a large number of modules in the PD method. With parallel training, the training time of the multi-layer PD network is even shorter than that of the single-layer PD network.

C. Pen-Based Recognition

The data set has 16 inputs and 10 outputs, together 7494 instances (3747 training instances, 1873 validation instances and 1874 test instances). The patterns were normalized and scaled so that each component lies within [0, 1]. The structure of the balanced 2-layer PD network is also the same as the network in Figure 4.5. Distributor module 1 has 2 outputs, {1,2,6,10} and {3,4,5, 7,8,9}. The two outputs of Distributor module 2 is {1,6} and {2,10} and the two outputs of Distributor module 3 is {3,4,5} and {7,8,9}. Module 1 recognizes classes 1, 6. Module 2 recognizes classes 2, 10. It is similar that Module 3 recognizes classes 3,4,5 and Module 4 recognizes classes 7,8,9. For comparison, a single-layer PD network whose structure is

Table 4.3: Results of the balanced 2-layer PD for the Pen-Based Recognition data

Method		Training time (s)	Hidden Units	Indep. Param.	C.error (%)
Single-layer Pattern Distributor	Distributor module	966.5 (in parallel) 2914.8 (in series)	6114.25	2124.5	4.8319
	Overall network	966.5 (in parallel) 3145.55 (in series)	276.79	5220.22	5.0961
Two-layer Pattern Distributor	Distributor modules	1160.25 (in parallel) 3077.95 (in series)	196.7	3642.6	1.5792
	Overall network	1160.25 (in parallel) 3308.7 (in series)	359.24	6738.32	1.8490

the same as the network in Figure 4.4 is used. The distributor module of the single-layer PD network has 4 outputs, {1,6}, {2,10}, {3,4,5} and {7,8,9} and consists of 4 sub-modules. Its non-distributor modules are the same as their counterparts used in the 2-layer PD network.

Table 4.3 shows the experimental results of the single-layer PD network and the multi-layer PD network for the Letter recognition data. Using the single-layer PD network, the classification error is 5.0961%, while using the multi-layer PD network, the classification error 1.8490%. The classification error decreases about 7/11. There is significant improvement. According the analysis in Section 4.2, if $p^{(d)} < p_0^{(s)}$, the multi-layer PD network will have higher recognition rate. From Table 4.3, the classification error of the distributor modules is reduced using the multi-layer PD network (1.5792%) compared to that using the single-layer PD network (4.8319%). Thus, the multi-layer PD network outperformed the single-layer PD network. From Table 4.3, we can see that the number of hidden units and the number of independent parameters in the multi-layer PD network are larger than those in the single-layer PD network. This can be attributed to the fact that the multi-layer PD has more

modules than the single-layer PD. From Table 4.3, we can also note the changes of the training time using the above two methods. With series training, the training time of the multi-layer PD network (3308.7s) is a little longer than that of the single-layer PD network (3145.55s) due to a large number of modules in the PD method.

4.4.2 Experimental Results for Imbalanced Two-layer PD Networks

A. Vowel

For this problem, the structure of the imbalanced 2-layer PD network is also the same as the network in Figure 4.3. Distributor module 1 has 2 outputs, {3,4,6,5,7,11}, {1,2,10} and {8,9}. The two outputs of Distributor module 2 is {3,4,6} and {5,7,11}. Module 1 recognizes classes 3,4,6 and consists of 3 sub-modules. Module 2 recognizes classes 5,7,11 and consists of 3 sub-modules. It is similar that Module 3 recognizes classes 1,2,10 and consists of 3 sub-modules and Module 4 recognizes classes 8,9 and also consists of 2 sub-modules. For the ease of comparison, a single-layer PD network whose structure is the same as the network in Figure 4.4 is used. The distributor module of the single-layer PD network has 4 outputs, {3,4,6},{5,7,11},{1,2,10} and {8,9} and consists of 4 sub-modules. Its non-distributor modules are the same as their counterparts used in the 2-layer PD network.

Table 4.4 shows the experimental results for the imbalanced 2-layer PD network and the single-layer PD network. Using the single-layer PD network, the classification error is 18.1983%, while using the multi-layer PD network, the classification error is 17.6316%. According to the analysis in Section 4.2, if Expression (4.10) is satisfied, the multi-layer PD network will have higher recognition rate. Using the data in Table 4.4, $\frac{N(p_0^{(s)}-p_1)}{N_1+N_2} = \frac{247}{140}(9.1296 - 2.5101) \approx 11.68$, which is greater than $p_2 = 10.6071$. Thus, the imbalanced two-layer PD network can achieve better classification accuracy. The classification error decreases about 0.5% in this example, just a minor improvement in this problem. From Table 4.4, we can see that the number of hidden units and the number of independent parameters in the multi-layer PD network are a little larger than those in the single-layer PD network because the multi-layer PD has more modules than the single-layer PD. Table 4.4 also shows the

Table 4.4: Results of the imbalanced 2-layer PD for the Vowel data

Method		Training time (s)	Hidden Units	Indep. Param.	C.error (%)
Single-layer Pattern Distributor	Distributor module	51.8 (in parallel) 189.6 (in series)	72.8	917.6	9.1296
	Overall network	51.8 (in parallel) 345.35 (in series)	139.5	1839	18.1983
Two-layer Pattern Distributor	Distributor module 1	80.4 (in parallel) 173.94 (in series)	53.65	676.8	2.5101
	Distributor module 2	20.45 (in parallel) 40.9 (in series)	24.85	320.2	10.6071
	Overall network	80.4 (in parallel) 370.59 (in series)	145.2	1918.4	17.6316

Table 4.5: Results of the imbalanced 2-layer PD for the Pen-Based Recognition data

Method		Training time (s)	Hidden Units	Indep. Param.	C.error (%)
Single-layer Pattern Distributor	Distributor module	966.5 (in parallel) 2914.8 (in series)	6114.25	2124.5	4.8319
	Overall network	966.5 (in parallel) 3145.55 (in series)	276.79	5220.22	5.0961
Two-layer Pattern Distributor	Distributor module 1	979.5 (in parallel) 2608.4 (in series)	106.2	1962.6	0.8485
	Distributor module 2	135.8 (in parallel) 263.5 (in series)	62.4	1157.2	0.9733
	Overall network	979.5 (in parallel) 3112.3 (in series)	331.14	6215.52	1.4541

training time using these methods. Using series training, the training time of the multi-layer PD (370.59s) is a little larger than that of the single-layer PD (345.35s).

B. Pen-Based Recognition

For the data set, the structure of the imbalanced 2-layer PD network is the same as the network in Figure 4.3. Distributor module 1 has 3 outputs, {1,2,6,10}, {3,4,5} and {7,8,9}. The two outputs of Distributor module 2 is {1,6} and {2,10}. Module 1 recognizes classes 1, 6. Module 2 recognizes classes 2, 10. It is similar that Module 3 recognizes classes 3,4,5 and Module 4 recognizes classes 7,8,9. For comparison, a single-layer PD network whose structure is the same as the network in Figure 4.4 is used. The distributor module of the single-layer PD network has 4 outputs, {1,6}, {2,10}, {3,4,5} and {7,8,9} and consists of 4 sub-modules. Its non-distributor modules are the same as their counterparts used in the 2-layer PD network.

Table 4.5 shows the experimental results for the imbalanced 2-layer PD network and the single-layer PD network. Using the single-layer PD network, the classification error is 5.0961%, while using the multi-layer PD network, the classification error is 1.4541%. According to the analysis in Section 4.2, if Expression (4.10) is satisfied, the multi-layer PD network will have higher recognition rate. Using the data in Table 4.5, $\frac{N(p_0^{(s)}-p_1)}{N_1+N_2} = \frac{1874}{750}(5.0961 - 1.4541) \approx 9.1$, which is much greater than $p_2 = 0.9733$. Thus, the imbalanced two-layer PD network achieves much better classification accuracy. The classification error decreases about 3.5% in this example, about 7/10 of the classification error of the single-layer PD network. It is a greater improvement in this problem. From Table 4.5, we can see that the number of hidden units and the number of independent parameters in the multi-layer PD network are a little larger than those in the single-layer PD network because the multi-layer PD has more modules than the single-layer PD. Table 4.5 also shows the training time using these methods. Using series training, the training time of the multi-layer PD (3112.3s) is close to that of the single-layer PD (3145.55s).

4.5 Discussion and Conclusions

In this chapter, the idea of multi-layer PD networks is discussed. The theoretical analysis shows that the multi-layer PD network can achieve higher recognition rate than single-layer PDs when some condition is satisfied. The experimental results confirm with the analysis. The experimental results also show that the improvement may not be uniform. For the Segmentation data set, only minor improvement is achieved. For the Vowel data set, medium improvement is acquired. While for the Pen-Based Recognition problem, there is major improvement. It should be noted that the multi-layer PD network may not have better classification rate than the corresponding single-layer PD network sometimes, i.e., when Expression (4.10) or (4.12) is not satisfied.

There is a question: Do the multi-layer PD network always have better classification accuracy than the single-layer PD network for a problem? The answer is no. We can not guarantee that the single-layer PD networks always outperform the OP networks. Only when Expression (3.13) is satisfied, the single-layer PD has better

performances than the OP. A multi-layer PD can be regarded as several combinations of several single-layer PDs. If the classification error of one or more distributor modules is large enough, it is possible that the multi-layer PD has low performance.

It is similar that the balanced multi-layer PD does not always outperform the imbalanced one. The overall classification mainly depends on the grouping of classes in the distributor modules, not on the number of distributor modules. Because of the complexity of the feature space, grouping classes with balanced network configuration often leads to large classification error in the distributor module. For the Pen-Based Digits problem, the balanced 2-layer PD has larger classification error than imbalanced ones. In the next two chapters, we will discuss how to find good grouping for a distributor module.

Chapter 5

Greedy Based Class Combination Methods

5.1 Introduction

In Chapter 3, a theoretical model was proposed to analyse the classification rate of a PD network. From there, it can be seen that the performance of a PD network depends greatly on the accuracy of the distributor module. How to group and combine the classes becomes a key issue in designing a PD network.

Firstly, we define two concepts - combination and combination set. If some classes are grouped together and denoted using the same label, we call them a combination. The combination of classes H_1, H_2 and H_3 is represented as $\{H_1, H_2, H_3\}$. Once some classes are combined, they will become a new class in the distributor module. Therefore, Combination $\{H_1, H_2, H_3\}$ is also called class $\{H_1, H_2, H_3\}$ instead. A combination set is an aggregation of combinations where each class in the original problem appears only once. A combination set is denoted by $\{C_1, , C_r\}$. The combination set for a distributor module means that the distributor module uses the elements in that combination set as its outputs. For example, in a 6-class problem, {{1,2,3},{4,5},{6}} is a combination set. Here, $C_1 = \{1, 2, 3\}$, $C_2 = \{4, 5\}$, $C_3 = \{6\}$. Its corresponding distributor module has three outputs, the first one {1,2,3}, the second one {4,5} and the last one {6}.

A distributor module does not always attain high performance if the combination set for that module is arbitrarily chosen. Choosing different combination sets will lead to different classification accuracies of the distributor module. Thus, developing

algorithms to guide the selection of combination sets for the distributor module is necessary. In this chapter and the next, we present several combination methods. In this chapter, we propose combination algorithms based on a greedy approach. The proposed greedy approach is a simple and straightforward approach to solve optimization problems. Greedy algorithms are short-sighted in their approach in the sense that they make decisions on the basis of information at hand without worrying about the effect these decisions may have in the future (Bendall and Margot, 2006; Corman et al., 2001). They are easy to invent, easy to implement and most of the time efficient. Before presenting the algorithms, we take a closer look at the distributor module and the non-distributor modules.

5.2 Analysis of the Distributor Module

As mentioned before, the outputs of the distributor module can be represented as a combination set. Our research shows that the distributor module for a combination set has relations with the neural networks for the elements of the combination set. Here we describe our findings. Before presenting them, we define three concepts - the classification error of a class, the classification error of a combination and the classification error of a combination set.

Definition 5.1—The classification error of class j: The classification error of class j means the classification error of the neural network which is used to classify class j patterns and non-class j patterns. To obtain the classification error of class j, all patterns not belonging to class j are labeled as patterns of class ; a single neural network is then used for the resulting two-class classification problem. Then the classification error of class j is the classification error of that neural network.

Definition 5.2—The classification error of Combination j: Once some classes are combined into Combination j, these classes denoted using the same label. Thus they will become a new class Z in the distributor module. The classification error of combination j means the classification error of the neural network which is used to classify that new class Z patterns and non-Z patterns.

Definition 5.3—The classification error of Combination set W: The classification error of Combination set W is defined as the classification error of the distributor module which uses Combination set W as its outputs.

For example, when the PD method is used to solve a 6-class problem, the classification error of Combination set {{1,2,3},{4,5},{6}} refers to the classification error of the distributor module. The distributor module has 3 outputs. One is {1,2,3}, another is {4,5} and the last output is {6}. Generally, OP is often applied in this distributor module, which means that the distributor module is further decomposed into several sub-modules. In the above example, the distributor module is further divided into 3 sub-modules where each sub-module has one output. Please refer to Discussion 1 in chapter 3 for detail.

Theorem 5.1 *Generally, the classification error of a combination set is equal to half of the sum of the classification errors of the elements in that combination set.*

Because an element in a combination set is a combination, the classification error of an element of the combination set means the classification error of a neural network designed to differentiate the patterns belonging to that combination and the patterns not belonging to that combination (see definition 5.2).

PROOF:

Assume Combination set $W = C_1, C_2, \ldots, C_r$. By applying OP to the distributor module for Combination set W, the distributor module is divided into r sub-modules. Sub-module j is used to select the patterns of combination C_j from all the patterns. For each sub-module, there are two situations in which the sub-module will make wrong decisions on an input pattern: (1) The pattern belongs to the class of that sub-module, while the sub-module does not think so. In other words, after the pattern enters that sub-module, the sub-module outputs a relatively small value. (2) The pattern does not belong to the class of that sub-module, while the sub-module regards it belonging to the class. In other words, the sub-module outputs a relatively large value for the pattern. For the first situation, assume the number of wrongly-classified patterns in sub-module j is A_{j1}, and for the second situation, A_{j2}.

Now we look at all the sub-modules in the distributor module. Winner-takes-all is used in merging the final results from these sub-modules. Thus, to a wrongly-classified pattern, if situation (1) occurs in a sub-module, the situation (2) must appear in another sub-module. In other words, situation (1) and (2) must appear in pairs because of the use of winner-takes-all. Therefore, we have the relationship:

$$\sum_{j=1}^{r} A_{j1} = \sum_{j=1}^{r} A_{j2} \quad . \tag{5.1}$$

Thus, the number of wrongly classified patterns by the distributor module can be written as:

$$M_1 = \sum_{j=1}^{r} A_{j1} = \sum_{j=1}^{r} A_{j2} = \sum_{j=1}^{r} \frac{(A_{j1} + A_{j1})}{2} \quad . \tag{5.2}$$

Now we look at the classification error of Combination C_j. A neural network is set up to classify the patterns of Combination C_j and the patterns of non-Combination C_j. To simplify, we name that neural network NN j. It is similar to Sub-module j in the distributor module. There are also two situations in which NN j will make wrong decisions. (1) The pattern belongs to Combination C_j, while NN j thinks it does not belong to that combination. (2) The pattern does not belong to Combination C_j, while NN j considers it belongs to Combination C_j. For the first situation, assume the number of wrongly classified pattern in NN j is B_{j1}, and for the second situation, B_{j2}. Because the training patterns of NN j are the same as those of Sub-module j, it is obvious that $B_{j1} = A_{j1}$ and $B_{j2} = A_{j2}$. Consider the existence of the complementary class - non-Combination C_j, so the number of wrongly- classified patterns in NN j is

$$P_j = B_{j1} + B_{j2} = A_{j1} + A_{j2} \quad . \tag{5.3}$$

Thus, the sum of the wrongly-classified patterns of the elements in Combination set W is

$$M_2 = \sum_{j=1}^{r} P_j = \sum_{j=1}^{r} (A_{j1} + A_{j2}) = 2M_1 \quad . \tag{5.4}$$

Therefore, the classification error of a combination set is equal to half of the sum of the classification errors of the elements in that combination set. (END PROOF)

Table 5.1: The classification errors of the elements of combination set W(Segmenation data)

	1	2	3	4	5	6	7
C.error(%)	0.70	0	2.94	1.89	4.11	0.11	0.86

Now we illustrate with an actual example to show that Theorem 5.1 is correct. We look at the Segmentation problem which has 7 classes. Assume Combination set $W = \{\{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{7\}\}$. A distributor module is set up for W. We find the classification error of W is $M_1 = 5.18\%$. Then seven neural networks are set up to evaluate the classification errors of the elements of W. See Table 5.1. We can see that the sum of the classification error of these elements is $M_2 = 10.61\%$. Putting aside the error which is brought about by the neural network itself [1] , we can see that M_1 is two times of M_1.

Corollary 5.1 *Assume that the distributor module for Combination set $W = \{C_1, C_2, \ldots, C_r\}$ has the classification error E_w; E_{c-max} is the maximum classification error of Combination $C_1, C_2, \ldots, C_r$, then E_{c-max} and E_w satisfies the following relation:*

$$E_w < \frac{r}{2} E_{c-max} \quad . \tag{5.5}$$

The classification error of Combination C_j means the classification error of the neural network which is used to classify Combination C_j's patterns and non-Combination C_j's patterns. From Theorem 5.1, it can easily reach this corollary.

Corollary 5.1 gives an upper bound of the classification error of a combination set. From this, it can be seen that with the reduction of the number of combinations and the maximum classification error of the combinations in the combination set, the classification error of the distributor module will decrease. An extreme case is that $r = 1$, which means the combination set only has one element or the distributor module for this combination set has only one output. In this case, the distributor module passes all the input patterns to the next module instead of classifying them,

[1]There are factors contributing to the classification error. The first one is the classifier itself. Neural networks are not perfect classifiers. The second one is the initial parameter selection and so on.

thus E_w and E_{c-max} will be zero. The distributor module loses its functions in the case. It is obvious that this extreme case should be avoided in the design of the combination selection algorithms as it will make the distributor module meaningless. Therefore, we prescribe the minimum number of outputs in a distributor module as 2 in the algorithms. In other words, a candidate combination set for a distributor module has at least 2 elements.

Our greedy algorithms are designed to reduce the maximum classification error of combinations in the combination set. Thus, we can find good combination sets. However, the distributor module is not the only factor to dominate the performance of the whole network. Non-distributor modules influence the performance of whole network. Therefore, we discuss the non-distributor modules in the next section.

5.3 Analysis of the Non-distributor Modules

Before starting the discussion, it is necessary for us to describe the characteristics of standard classifiers. In mathematics, a classifier is a device or system that predicts a discrete set of labels Y from a discrete or continuous feature space X. Generally, a classifier can not map all the data perfectly, especially for those unknown patterns. Normally, an actual classifier should attain better discrimination rate when it is used for simple tasks compared with more difficult tasks. Because simple take and difficult one are just relative concepts, we illustrate with the following examples.

For example, we create a classifier for a two-class problem (i.e., we train a neural network for the problem), mapping the feature space into two classes. We say this task is a simple one. Now we add some random noise to the feature space, i.e. these noise data are randomly distributed in the feature space. For comparison, we keep the test data untouched and just add noise to the data for training and validation. Then we use the same method to set up a classifier for this problem (i.e., we train the neural network using the original data with added noise data). The task with noise is relatively difficult compared with the former one. In common sense, the discrimination rate of the classifier will decrease when noise is added in.

For another instance, a classifier (i.e. a neural network) is built for a three-class problem, projecting the feature space into three points. Now we add the patterns which do not belong to any of the three classes to the feature space. These patterns belong to a fourth class and are somewhat like noise, but they are not randomly distributed in the feature space like the noise in the above example. As in the last example, we keep the test data unchanged and just add these patterns to the data for training and validation. Then we set up the classifier using the original data and added data. We can regard the patterns not belonging any of the three classes as the members of an unknown class. Each output of the original classifier is trained to select one class among the three classes, while each output of the new classifier is trained to choose one class among the four classes (though only three outputs exist). Compared with the original classifier, the new classifier has more tasks to do. We can also regard the added patterns as some type of noise. In this sense, the new classifier's task is also harder than the former one. Thus, the discrimination rate of the classifier will also decrease.

Now we illustrate an actual example to confirm this observation. Look at the Segmentation problem. This problem totally has 7 classes. We take out the patterns belonging to class 3, 4 and 5 and group these patterns into a 3-class problem. The data are divided into training set, validation set and test set. Then a MPL NN is trained to classify this 3-class problem using the training set and the validation set. The test set is applied to test the classification error of the neural network, which is 8.62%. After that, we just add the patterns belonging to class 6 to the training set and the validation set, at the same time we keep the test set unchanged. Then we set up the new neural network based on the new training set and the new validation set, and test it using the test set. We find the classification error becomes 11.75%, which is greater than the former classification error. This shows that a classifier can indeed attain better discrimination rate when it is used for simple task compared with relatively difficult task.

Let us go back to the discussion on non-distributor modules.

Theorem 5.2 *Generally, in a non-distributor module, the classification error will*

increase if a new class is added to this module.

RPT is used in the training of a non-distributor module. In other words, only the patterns belonging to the classes of that module are used for training and validation. Thus, when we add a new class to the module, the training patterns of the new class will also join the training set, and same for the validation set. The test patterns of the new class will also join the test set using for evaluating the classification error.

PROOF:

Consider a non-distributor module in a PD network. Before adding the new class to that module, the module needs to recognize m classes. Its classification error is E_1. For convenience, these classes are marked as class 1, class 2, ..., class m. The module has m outputs and each output selects patterns from one class out of all the patterns. Assume using the old test set, the number of wrongly classified patterns is T_1.

Now the patterns of class $m+1$ are added to the module [2]. We train the new module using the original patterns and added patterns. The new module has $m+1$ output nodes. Output 1 to output m's function is the same as that in the old module, which is selecting one class' patterns from all the patterns. But there is still some difference between them. Each output of the old classifier is trained to select one class from m classes, while each output of the new classifier is trained to choose one class from $m+1$ classes. The task in the new module is relatively difficult compared with the old one. According to the discussion earlier, normally, an actual classifier should attain better discrimination rate when it is used for simple tasks compared with more difficult tasks. Thus, if we test the new module using the old test set, output 1 to output m in the new module will normally export more wrongly classified patterns than those in the old one (Of course, in some extreme cases, the class $m+1$ is far away from class 1 to m. It is possible that the number of wrongly-classified patterns may not increase. However, in a normal situation, the number of wrongly-classified patterns will be larger.). Assume using the old test set, the number of wrongly classified patterns exported by output 1 to output m in the new module is T_1'. For the new test set is the old test set added with some

[2] When the new class is added to the module, the training patterns, validation patterns and test patterns of this class are also added to their counterparts.

patterns of class $m+1$, if we test the module using the new test set, assume the number of wrongly classified patterns exported by output 1 to output m in the new module is T_2. Then T_2 should be greater than $T_1^{'}$, because output 1 to output m may give additional wrong results when they process the patterns belonging to class $m+1$.

In the above, we analyzed the change of wrongly classified patterns in output 1 to output m. Assume the error distribution across class 1 to $m+1$ is similar. Now we look at output $m+1$, which select patterns of class $m+1$ from all the patterns. Assume the number of test patterns of class j is N_j. Define n_j as the number of the wrongly classified patterns in output j before adding class $m+1$ and $n_j^{'}$ as the number of the wrongly classified patterns after adding class $m+1$. Because we have no information on the number of wrongly classified patterns in output $m+1$ here, we assume $\frac{n_{m+1}^{'}}{N_{m+1}} = \frac{n_1^{'}+n_2^{'}+\cdots+n_m^{'}}{N_1+N_2+\cdots+N_m}$, which means the rate of wrongly classified patterns of output $m+1$ is equal to the average rate of wrongly classified patterns of the other outputs. It is a reasonable assumption since we have no information of class $m+1$.

Now we compute the overall classification error of the new module.

$$\begin{aligned} E_2 &= \frac{\sum_{j=1}^{m+1} n_j^{'}}{\sum_{j=1}^{m+1} N_j} \\ &= \frac{1}{\sum_{j=1}^{m} N_j} \cdot \frac{\sum_{j=1}^{m} N_j}{\sum_{j=1}^{m+1} N_j} \cdot (\sum_{j=1}^{m} n_j^{'} + n_{m+1}^{'}) \\ &= \frac{\sum_{j=1}^{m} n_j^{'}}{\sum_{j=1}^{m} N_j} + \frac{1}{\sum_{j=1}^{m} N_j \cdot \sum_{j=1}^{m+1} N_j}(n_{m+1}^{'} \sum_{j=1}^{m} N_j - N_{m+1} \sum_{j=1}^{m} n_j^{'}) \\ &= \frac{\sum_{j=1}^{m} n_j^{'}}{\sum_{j=1}^{m} N_j} > E_1 \quad . \end{aligned} \tag{5.6}$$

Thus, we can see that the classification error will increase when adding a new class to a non-distributor module.(End Proof)

Corollary 5.2 *In a non-distributor module, the classification error normally increases with new classes added to that module.*

This theorem can be deduced from Theorem 5.2. Those new classes can be added

one by one. According to Theorem 5.2, the classification error will be increased.

Corollary 5.3 *If a non-distributor module is split into two new non-distributor modules, for each new module, its classification error is smaller than that of the old module.*

PROOF:

Assume the old non-distributor module is called Module 0 and the two new modules are called Module 1 and Module 2. Assume the classification errors of the new modules are E_1 and E_2 respectively and the classification error of the old module is E_0. Module 0 can be regarded as adding new classes to Module 1 or Module 2, we have $E_0 > E_1$ and $E_0 > E_2$.(END PROOF)

5.4 Three Greedy Based Combination Algorithms

5.4.1 Introduction

The objective of these algorithms is to find a combination set so that the distributor module has small classification errors. From Theorem 5.1, it can be seen that if the combination set is composed of the combinations with small classification errors, then the distributor module will have small classification error. Thus, our algorithms in this chapter are based on finding combination with small classification error. After that, we use these combinations to buildup a combination set. In this way, the distributor module will have good performance.

In these algorithms, classification errors will be the measures of the difficulty in discriminating among the patterns of a particular class and the patterns from all the other classes. Classification errors are used to guide the whole process of finding a suitable combination set.

Corollary 5.1 shows that in order to reduce the classification error of the combination set E_w, we should encourage the reduction of the number of elements r in the combination set and the maximum classification error E_{c-max} in the design of the algorithms. Theorem 5.2 and Corollary 5.2 show that in order to reduce the

number of wrongly classified patterns by the non-distributor modules, we should encourage small non-distributor modules, or the non-distributor modules with small classes. However, if the number of elements r in the combination set is very small, the number of classes in some combination(s) will be relatively large. Thus, some non-distributor modules will be very large. Thus, in our design of these algorithms, we should consider this tradeoff and solve this problem.

In this chapter, we present three combination algorithms based on greedy methods. In the first algorithm, Greedy Combination Selection (GCS), we do not consider the number of classes in a combination. It seems that the first algorithm can not avoid the appearance of large non-distributor modules. We know large non-distributor modules can lead to relatively large classification error of the whole network. To improve the performance of the whole network, we must reduce the classification error of the non-distributor modules. Regarding each non-distributor module as an independent sub-problem, we can improve the module's classification rate by applying the PD method to this sub-problem. In this way, we alleviate this problem by building a multi-layer PD network. If a combination has more than 2 classes, we do not connect the corresponding output to a non-distributor module, but connect it to a 2nd-level distributor module. If an output from the 2nd-level distributor module also has many classes, we use the 3rd-level distributor module, and so on. We do this continuously until all the non-distributor modules process simple tasks. By building a multi-layer PD network, we can avoid the appearance of large non-distributor modules.

The second algorithm, namely Simplified Greedy Combination Selection (SGCS), is a simplification of the first algorithm. In the first algorithm, we need to check many classification errors of the combinations. The second one is based on the analysis in Section 5.4.4. The analysis shows that in most cases, the classification error after combination is related to the classification errors before combination. The relation is used to reduce the effort of computing those classification errors. Thus, the second algorithm can save computing time compared with the first one. At the same time, the two algorithms can obtain similar results.

The third algorithm is called Restricted Greedy Combination Selection (RGCS). We mentioned that the distributor module prefers a small r, the number of elements in the combination set, which leads to large non-distributor modules. Large non-distributor modules, however, lead to relatively large classification errors. Therefore, instead using a multi-layer PD network, in this algorithm, we compromise by balancing between the distributor module and the non-distributor modules.

5.4.2 Greedy Combination Selection (GCS) Algorithm

Corollary 5.1 shows that in order to reduce the classification error of the combination set E_w, we should reduce the number of elements r in the combination set and the maximum classification error E_{c-max} in the design of the algorithm. The minimum of r is 2. This algorithm is designed for multi-layer PD networks. If we choose a small r, each element of the combination set will have many classes. Then GCS algorithm is used to these elements. Level by level, GCS algorithm is continuously applied. At last, the size of the whole PD network will be very large. In other words, the network will have many modules. The computation load for setting up the network will be very large. That is not desirable. Thus, in this algorithm, we do not reduce r indefinitely. Once E_{c-max} becomes small enough, the algorithm will stop and not try to reduce r further.

In each epoch of GCS algorithm, the classification errors of the classes (or combinations) are listed. Then the class with the maximum classification error is selected and temporarily combined with one of the other classes. After that, we find the temporary combination with the minimum classification error. If the minimum classification error is smaller than the classification error of the selected class, the temporary combination will be fixed and we proceed to the next epoch. We will prove in Theorem 5.3 that the combination set of the next epoch will be smaller than, at least no more than that of the pervious epoch. Greedy Combination Selection (GCS) Algorithm is described as follows:

1. Find the classification error of each class $E_{\{i\}}$. Record the largest classification error, i.e. E^{init}_{c-max}. Mark all the classes as "non-excluded".

2. Set $epoch = 1$.

3. Find class j with the largest classification error among all the "non-excluded" classes.

4. Choose another "non-excluded" class and temporarily combine it with class j. For instance, class i, combine it with class j, i.e. $\{i, j\}$. Thus, the two classes become one class and measure the classification error $E_{\{i,j\}}$.

5. Repeat Step 4 for all the remaining "non-excluded" classes.

6. Find the combination with the smallest classification error in the temporary combinations which have been measured in Steps 4 and 5, i.e. combination {t,j} with the smallest classification error $E_{\{t,j\}}$.

7. Compare $E_{\{t,j\}}$ with $E_{\{j\}}$. If $E_{\{t,j\}} > E_{\{j\}}$, mark class j as "excluded". Update the class information and go to Step 8. Otherwise, fix combination {t,j}. Update the information of the classes. After fixing combination {t,j}, {t,j} would become one class, the information of the original class t and j is removed and new information of {t,j} is added. Thus, the number of classes is reduced by 1. Mark class {t,j} as "non-excluded".

8. If there is only one "non-excluded" class remaining or there are just two classes remaining or the maximum classification error E_{c-max} in all the updated classes is much smaller than E^{init}_{c-max} [3] , stop. Otherwise, set epoch $epoch = epoch + 1$ and go to step 3.

In the algorithm, a class which is marked as "non-excluded" means that class will be seen as a potential candidate to be combined with other classes. Oppositely, a class marked as "excluded" will be not regarded as a potential candidate for combination.

[3]Assume A and B are positive numbers. If $10A < B$, we can consider that A is much smaller than B. In our experiments, we find that for a multi-class problem, normally, the overall classification error of the whole classifier is larger than, at least close to E^{init}_{c-max}. Thus, if the distributor module has the classification error is smaller than $E^{init}_{c-max}/10$, the wrongly classified patterns by the distributor module will be very small compared with that by the whole network. We can say that the distributor module has a relatively high classification rate.

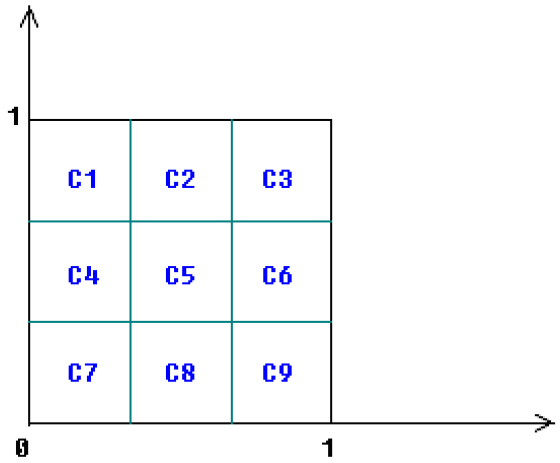

Figure 5.1: A special 9-class problem in a 2-dimensional feature space

In Step 6 of the GCS algorithm, we find the combination (i.e. /t,j/) with the smallest classification error $E_{\{t,j\}}$. In Step 7, we compare $E_{\{t,j\}}$ with E_j and if $E_{\{t,j\}} > E_j$, mark class j as "excluded". In the situation such that $E_{\{t,j\}} > E_j$, the smallest classification error of the temporary combinations is always larger than the selected class j. It hints that the temporary classification error may be increased due to class combination. One may be curious when this situation will appear. Here is an example for this situation.

See Figure 5.1. Assume that a 9-class problem is to be solved in a 2-dimensional feature space. For convenience, the inputs of the patterns are normalized into [0, 1]. Assume the patterns are evenly distributed in the feature space and the wrongly-classified patterns are mainly distributed near the border between classes. Assume the number of wrongly classified patterns is proportional to the length of the border. From the figure, we can see that the border length of class 5 is four units, while the border length of class 1,3,7,9 is two units and the border length of class 2,4,6,8 is three units. Assume that the classification error of class 5 is $4t$, then the classification error of classes 2, 4, 6 and 8 is $3t$ and the classification error of classes 1, 3, 7 and 9 is $2t$. Using GCS, in the first epoch, class 5 is selected as it has the maximum classification error. We need to combine class 5 with another class. If class 5 is combined with class 1, 3, 7 or 9, the classification error will be $6t$. If class 5 is combined with class 2, 4, 6 or 8 instead, the classification error with be $5t$.

No matter which class is chosen for combination, the classification error will increase.

Then we show that our GCS algorithm can find a combination set with relatively small classification error of the distributor module.

Theorem 5.3 *Using GCS algorithm, the classification error of the combination set in the next epoch will be smaller than or no more than that in the previous epoch.*

Here the classification error of the combination set means the classification error of the distributor module for this combination set (see definition 5.3). For example, in a 6-class problem, at the beginning of using GCS algorithm, the combination set is {{1},{2},{3},{4},{5},{6}}. If classes 3 and 4 are combined in the next epoch, then the combination set is {{1},{2},{3,4},{5},{6}}, whose classification error will be no more than the classification error of {{1},{2},{3},{4},{5},{6}}.

PROOF:

In each epoch, the classes or combinations form a combination set. There are two situations. Please see Step 7. (1) The smallest classification error of the temporary combinations is not smaller than the selected class j, class j are marked "excluded" and the combination set will be kept the same as that in the previous epoch. Thus, the overall classification errors will be equal to that in the previous epoch. (2) The smallest classification error of the temporary combinations is smaller than the selected class j. The temporary combination with the smallest classification error is fixed and the overall classification errors of the elements will be smaller than that in the previous epoch. Thus, according to Theorem 5.1, we can see that the classification error of the combination set in next epoch can not be larger than that in the previous epoch.(End Proof)

In our experiments, Situation (2) appears much more often than Situation (1). Thus, in most cases, the classification error of the combination set in the next epoch will become smaller.

Corollary 5.4 *The classification error of the combination set obtained by GCS al-*

gorithm will be no larger than the classification error for the fully decomposed combination set Output Parallelism (OP).

PROOF:

Assume a classification problem which is composed with M classes. Fully decomposed OP can also be regarded as a special distributor module which has M outputs, whose combination set is $\{C_1, C_2, \ldots, C_M\}$. At the beginning of Epoch 1, the combination set is $\{C_1, C_2, \ldots, C_M\}$. According to Theorem 5.3, the classification error of the combination set in next epoch can not be larger than that in the last epoch. Thus, the classification error of the combination set reached by the GCS algorithm will be no larger than the classification error for the fully decomposed combination set.(End Proof)

From the above two theorems, we can see that the GCS algorithm can gradually reach a better combination set.

5.4.3 An Example for GCS Algorithm

Here we will illustrate using an example to show how GCS algorithm progresses step by step. It should be noted that this example simply shows how GCS works while no results of the final PD network are given. The final results of the PD network will be presented in the experiments section. Refer to the Vowel problem. The input patterns of this data set are 10 real vectors representing vowel sounds that belong to one of 11 classes. Now we use GCS to find a combination set for the first-level distributor module.

The classification error of each class is computed and listed here:

Class	1	2	3	4	5	6	7	8	9	10	11
C.error(%)	2.00	2.39	2.00	2.81	5.43	8.40	4.72	3.95	7.45	2.19	6.84

Epoch 1:

Class 6 has the largest classification error. Combine class 6 with the other classes respectively and compute the classification error.

{6,1}/12.02 {6,2}/16.48 {6,3}/11.72 **{6,4}/5.36** {6,5}/6.03
{6,7}/15.36 {6,8}/11.64 {6,9}/16.44 {6,10}/11.2 {6,11}/14.90

Combination {6,4} has the smallest classification error, fix it. Update the class information.

Class	1	2	3	{4,6}	5	7	8	9	10	11
C.error(%)	2.00	2.39	2.00	5.36	5.43	4.72	3.95	7.45	2.19	6.84

Epoch 2:

Class 9 has the largest classification error. Combine class 9 with the other classes respectively and compute the classification error.

{9,1}/7.31 {9,2}/13.60 {9,3}/7.17 {9,4,6}/11.01 {9,5}/13.52
{9,7}/9.62 **{9,8}/2.59** {9,10}/3.36 {9,11}/7.45

Combination {9,8} has the smallest classification error, fix it. Update the class information.

Class	1	2	3	{4,6}	5	7	{8,9}	10	11
C.error(%)	2.00	2.39	2.00	5.36	5.43	4.72	2.59	2.19	6.84

Epoch 3:

Class 11 has the largest classification error. Combine class 11 with the other classes respectively and compute the classification error.

{11,1}/9.09 {11,2}/10.06 {11,3}/7.37 **{11,4,6}/3.44**
{11,5}/13.50 {11,7}/10.10 {11,8,9}/7.71 {11,10}/10.38

Combination {11,4,6} has the smallest classification error, fix it. Update the class information.

Class	1	2	3	{4,6,11}	5	7	{8,9}	10
C.error(%)	2.00	2.39	2.00	3.44	5.43	4.72	2.59	2.19

Epoch 4:

Class 5 has the largest classification error. Combine class 5 with the other classes respectively and compute the classification error.

{5,1}/9.64 {5,2}/7.77 {5,3}/13.91 **{5,4,6,11}/3.37**
{5,7}/8.32 {5,8,9}/10.89 {5,10}/10.95

Combination {5,11,4,6} has the smallest classification error, fix it. Update the class information.

Class	1	2	3	{4,5,6,11}	7	{8,9}	10
C.error(%)	2.00	2.39	2.00	3.37	4.72	2.59	2.19

Epoch 5:

Class 7 has the largest classification error. Combine class 7 with the other classes respectively and compute the classification error.

{7,1}/10.53 {7,2}/9.65 {7,3}/8.18
{7,4,5,6,11}/3.16 **{7,8,9}/2.17** {7,10}/7.19

Combination {7,8,9} has the smallest classification error, fix it. Update the class information.

Class	1	2	3	{4,5,6,11}	{7,8,9}	10
C.error(%)	2.00	2.39	2.00	3.37	2.17	2.19

Epoch 6:

Class {4,5,6,11} has the largest classification error. Combine class {4,5,6,11} with the other classes respectively and compute the classification error.

{4,5,6,11,1}/5.24 {4,5,6,11,2}/9.41 {4,5,6,11,3}/4.57
{4,5,6,11,7,8,9}/0.97 {4,5,6,11,10}/4.80

Combination {4,5,6,11,7,8,9} has the smallest classification error, fix it. Update the class information.

Class	1	2	3	{4,5,6,11,7,8,9}	10
C.error(%)	2.00	2.39	2.00	0.97	2.19

Epoch 7:

Class 2 has the largest classification error. Combine class 2 with the other classes respectively and compute the classification error.

{2,1}/4.15 **{2,3}/1.08** {2,4,5,6,7,8,9,11}/5.63 {2,10}/6.30

Combination {2,3} has the smallest classification error, fix it. Update the class information.

Class	1	{2,3}	{4,5,6,11,7,8,9}	10
C.error(%)	2.00	1.08	0.97	2.19

Epoch 8:

Class 10 has the largest classification error. Combine class 10 with the other classes respectively and compute the classification error.

{10,1}/1.25 {10,2,3}/3.99 {10,4,5,6,7,8,9,11}/5.63

Combination {10,1} has the smallest classification error, fix it. Update the class information.

Class	{1,10}	{2,3}	{4,5,6,11,7,8,9}
C.error(%)	1.25	1.08	0.97

Epoch 9:

Class {1,10} has the largest classification error. Combine class {1,10} with the other classes respectively.

{1,10,2,3}/0.97 {1,10,4,5,6,7,8,9,11}/1.08

Combination {10,1,2,3} has the smallest classification error, fix it. Update the class information.

Class	{1,10,2,3}	{4,5,6,11,7,8,9}
C.error(%)	0.97	0.97

In this example, the GCS algorithm is used to compute the combination set for the Vowel problem. The final combination set is {{1,2,3,10},{4,5,6,7,8,9,11}}. Using theorem 5.1, we can evaluate that the classification error of the distributor module for this combination set is about 0.97%. Totally 65 classification errors of combinations (or classes) are computed in the algorithm, including 11 classification errors of the original classes. Compared with the classification error of the distributor module in Chapter 3, which is 6.68%, 0.97% is very small.

5.4.4 Simplified Greedy Combination Selection (SGCS) Algorithm

When two classes are combined into one, it seems that the classification error after combination is hard to forecast. Sometimes, combining classes can lead to better classification accuracy, while in other times, it will not. Here we explore the reason of the change of classification errors brought about by combining classes.

Combining classes changes the border status in the feature space and thus brings about the change of classification errors. For example, there are two adjacent classes S_1 and S_2 in an N-class problem. We divide the border of class S_1 into two parts according to its relationship with class S_2: (1) the border area which is adjacent to

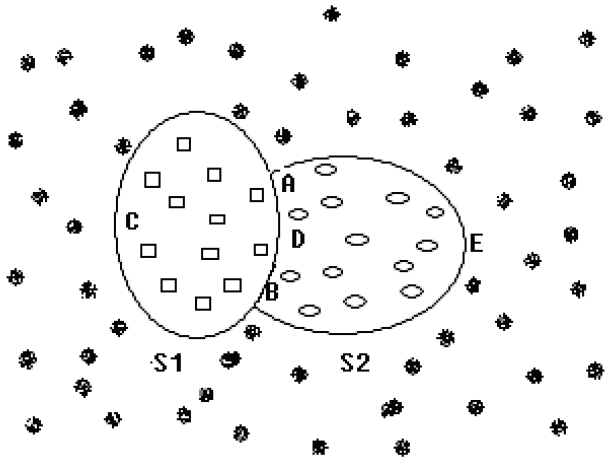

Figure 5.2: The distribution of patterns in the case that the two classes are adjacent

class S_2, denoted as $B_{(S1,S2)}$, (2) the border area which is not adjacent to class S_2, denoted as $B_{(S1,non-S2)}$. Similarly, the border of class S_2 is also divided into two parts: (1) the border area which is adjacent to class S_1, denoted as $B_{(S2,S1)}$. (2) the border area which is not adjacent to class S_1, denoted as $B_{(S2,non-S1)}$. See Figure 5.2. The patterns of the left circle belong to class S_1 and those in the right circle belong to class S_2. we can see that $B_{(S2,S1)} = B_{(S1,S2)} = \widehat{ADB}$, $B_{(S1,non-S2)} = \widehat{ACB}$ and $B_{(S2,non-S1)} = \widehat{AEB}$.

Before combination, the overall border of two classes is $B_{(S2,S1)} + B_{(S1,S2)} + B_{(S1,non-S2)} + B_{(S2,non-S1)}$. After the two classes are combined into one, the border status is changed. The overall border becomes $B_{(S1,non-S2)} + B_{(S2,non-S1)}$.

Normally, in a pattern recognition problem, the patterns around the border are most difficult to recognize and the wrongly-classified patterns are usually distributed around the border. There are many factors which affect classification errors, for example, border situation, the selection of classifiers, the initial parameters of the classifiers and so on. Now we ignore the other factors and focus on the effect of border on the classification error, and we have the following theorem.

Theorem 5.4 *Assume a classifier whose classification error is only determined by the border situation between classes. The classification error of class* S_1 *is denoted*

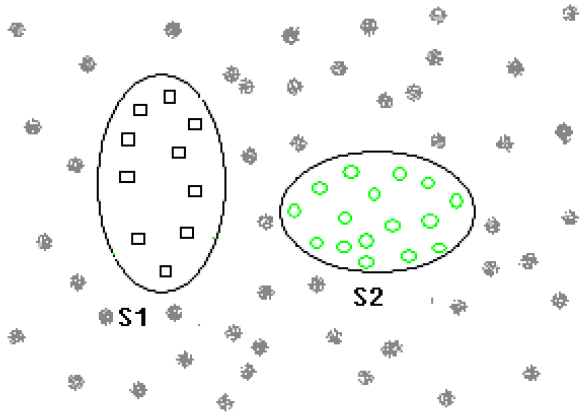

Figure 5.3: The distribution of patterns in the case that the two classes are not adjacent

by E_1 and the classification error of class S_2 is denoted by E_2. Classes S_1 and S_2 are combined into one class. The classification error after combination - E_c satisfies the following relation:

$$|E_1 - E_2| \leq E_c \leq E_1 + E_2 \quad . \tag{5.7}$$

PROOF:

There are three situations of the border between two classes. The first, also the most universal, situation is that the two classes are adjacent. The second one is that the two classes are not adjacent, which means the two classes have not common border. The third one is that one class is fully embedded inside another class, which means the class only has common border with one class and has no common border with the other classes.

We look at the second situation first. The two classes are not adjacent and $B_{(S1,S2)}$ and $B_{(S2,S1)}$ are zero. See Figure 5.3. Because the classification error is only determined by the border, it is obvious that $E_c = E_1 + E_2$.

Then we look at the third situation, the border of one class is fully embedded into the other class. See Figure 5.4. In this case, $B_{(S1,non-S2)}$ or $B_{(S2,non-S1)}$ is zero, we have $E_c = E_1 - E_2$ (in the case $B_{(S2,non-S1)} = 0$) or $E_c = E_2 - E_1$ (in the case $B_{(S1,non-S2)} = 0$).

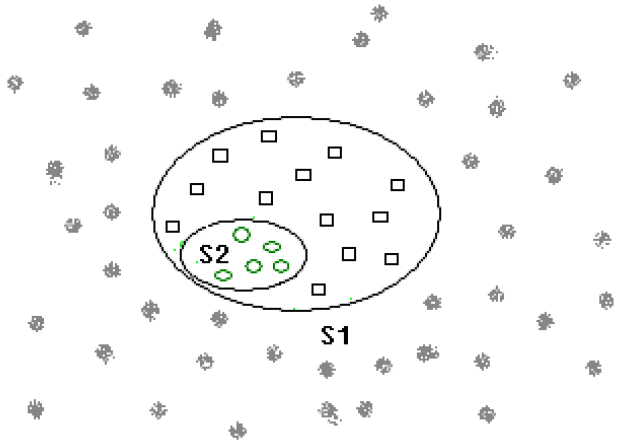

Figure 5.4: The distributions of patterns in the case that class 2 is Fully embedded in class 1

Now we look at the first situation. One can refer to Figure 5.2. Before combination, the overall border is $B_{(S2,S1)} + B_{(S1,S2)} + B_{(S1,non-S2)} + B_{(S2,non-S1)}$, then after combination, the border becomes $B_{(S1,non-S2)} + B_{(S2,non-S1)}$. It is obvious that $B_{(S1,non-S2)} + B_{(S2,non-S1)}$ is smaller than $B_{(S2,S1)} + B_{(S1,S2)} + B_{(S1,non-S2)} + B_{(S2,non-S1)}$ and is greater than $|B_{(S1,non-S2)} - B_{(S2,non-S1)}|$. For the classification error is only determined by the length of the border, we have $|E_1 - E_2| < E_c < E_1 + E_2$.

From the above analysis, we can see that the classification error after the combination is $|E_1 - E_2| \leq E_c \leq E_1 + E_2$. (End Proof)

In Theorem 5.4, we only considered the change of the classification errors brought about by the variety of the bordering conditions. There are also other factors which may affect the classification errors during combination. Classifiers themselves are a factor. Different classifiers tend to have different classification errors for the same problem. In our research, neural networks are used as classifiers. The representations of neural networks also have influence to the classification error of neural networks. For example, in a two-class problem, binary representation and 1-out-2 representation may achieve different results (Lu and Ito, 1995). Moreover, the initial parameters of classifiers also influence the classification errors.

We know neural networks are not perfect classifiers which can utilize all the information carried by the patterns. In a two-class problem, if the border is complicated, it will not be easy for the neural network to plot the perfect border between two classes. Generally, the more complicated the real border, the harder for the neural network to describe it. When two classes are combined into one, the border varies and the complication of border is also changed. In most situations, the border after combination will become longer and more complicated. Of course, it may become simple in some cases and it is only a minority in all the cases. If we ignore this minority cases and count the factor which is brought by neural networks themselves, from expression (5.7), we will have the following expression:

$$E_c \geq |E_1 - E_2| \quad . \tag{5.8}$$

According to our experimental results, Expression (5.8) is satisfied in about 89% cases of all the cases. For example, in section 5.4.3, we computed the classification errors of 54 temporary combinations. Expression (5.8) is satisfied in 50 cases, and only in 4 cases, it does not work.

From Expression (5.8), we can see that if E_1 is much larger than E_2 and vice versa, E_c is unlikely to be small. In GCS algorithm, when the class with the maximum classification error is selected, combining it with the classes having small classification errors will hardly attain lower classification errors. Thus, we do not need to compute all the combination candidates in each epoch of GCS algorithm.

Based on the above analysis, we present Simplified Greedy Combination Selection (SGCS) Algorithm.

1. Find the classification error of each class $E_{\{i\}}$. Record the largest classification error, i.e. E^{init}_{c-max}. Mark all the classes as "non-excluded".
2. Set $epoch = 1$.
3. Find class j with the largest classification error among all the "non-excluded" classes.Order the classification errors of the remaining "non-excluded" classes

in descending order. Use a queue to record the ordering of those classes. Assume the number of classes in the queue is N.

4. Set $p = 1$.

5. Choose the class at the p^{th} position in the queue and temporarily combine it with class j. For instance, class i, combine it with class j, i.e. $\{i, j\}$. Thus, the two classes become one class and measure the classification error $E_{\{i,j\}}$.

6. Find the combination with the smallest classification error in the p temporary combinations which have been measured in this epoch [4].i.e. combination {t,j} with the smallest classification error $E_{\{t,j\}}$.

7. If there are no element in the $(p+1)^{th}$ position, Compare $E_{\{t,j\}}$ with $E_{\{j\}}$.If $E_{\{t,j\}} > E_{\{j\}}$, mark class j as "excluded". Update the class information and go to Step 9. If $E_{\{t,j\}} \leq E_{\{j\}}$, fix combination $\{t, j\}$ and go to Step 8. If there is an element in the $(p+1)^{th}$ position, find the class at the $(p+1)^{th}$ position in the queue, i.e., class m in the queue. Compute $\varepsilon = E_{\{j\}} - E_{\{m\}}$ and compare $E_{\{t,j\}}$ with ε. If $E_{\{t,j\}} > \varepsilon$, set $p = p + 1$ and go to Step 5. If $E_{\{t,j\}} \leq \varepsilon$, fix combination $\{t, j\}$.

8. Update the information of the classes. After fixing combination $\{t, j\}$,$\{t, j\}$ becomes one class, the information of the original class t and j is removed and new information of $\{t, j\}$ is added. Thus the number of classes is reduced by 1. Mark class $\{t, j\}$ as "non-excluded".

9. If there is only one "non-excluded" class remaining or there are just two classes remaining or the maximum classification error in all the updated classes is much smaller than E_{c-max}^{init} [5], stop. Otherwise, set $epoch = epoch + 1$ and go to step 3.

[4]Because Step 5 to 6 may be run for more than one time, it is possible that there are several candidates.

[5]Assume A and B are positive numbers. If $10A < B$, we consider that A is much smaller than B.

5.4.5 An Example for SGCS Algorithm

Here we illustrate using an example to show how SGCS algorithm progresses step by step. It should be noted that this example just shows how SGCS works and we don't give the results of the final PD network until later. Still refer to the Vowel problem. Now we use SGCS algorithm to find a combination set for the distributor module.

The classification error of each class is computed and listed here:

Class	1	2	3	4	5	6	7	8	9	10	11
C.error(%)	2.00	2.39	2.00	2.81	5.43	8.40	4.72	3.95	7.45	2.19	6.84

Epoch 1:

1. Class 6 has the largest classification error. Now we want to find the best class which is suitable to combine with class 6. There are 10 candidates. Order them in descending order, which is 9,11,5,7,8,4,2,10,1,3.

2. Combine class 6 with class 9, compute the classification error $E_{\{6,9\}} = 16.44$ Compute $\varepsilon = E_{\{6\}} - E_{\{11\}} = 8.4 - 6.84 < E_{\{6,9\}}$.

3. Combine class 6 with class 11, compute the classification error $E_{\{6,11\}} = 14.90$ In the two computed classification errors, $E_{\{6,11\}}$ is the minimum one. Compute $\varepsilon = E_{\{6\}} - E_{\{5\}} = 8.4 - 5.43 < E_{\{6,11\}}$.

4. Combine class 6 with class 5, compute the classification error $E_{\{6,5\}} = 6.03$, $E_{\{6,5\}}$ is the minimum in the computed classification errors. Compute $\varepsilon = E_{\{6\}} - E_{\{7\}} = 8.4 - 4.72 < E_{\{6,5\}}$.

5. Combine class 6 with class 7, compute the classification error $E_{\{6,7\}} = 15.36$, $E_{\{6,5\}}$ is the minimum in the computed classification errors. Compute $\varepsilon = E_{\{6\}} - E_{\{8\}} = 8.4 - 3.95 < E_{\{6,5\}}$.

6. Combine class 6 with class 8, compute the classification error $E_{\{6,8\}} = 11.64$, $E_{\{6,5\}}$ is the minimum in the computed classification errors. Compute $\varepsilon = E_{\{6\}} - E_{\{4\}} = 8.4 - 2.81 < E_{\{6,5\}}$.

7. Combine class 6 with class 4, compute the classification error $E_{\{6,4\}} = 5.36$, $E_{\{6,4\}}$ is the minimum in the computed classification errors. Compute $\varepsilon = E_{\{6\}} - E_{\{2\}} = 8.4 - 2.39 > E_{\{6,4\}}$.

There is no need to continue because $\varepsilon > E_{\{6,4\}}$, fix {6,4}. Update the class information.

Class	1	2	3	{4,6}	5	7	8	9	10	11
C.error(%)	2.00	2.39	2.00	5.36	5.43	4.72	3.95	7.45	2.19	6.84

Epoch 2:

1. Class 9 has the largest classification error. Now we want to find the best class which is suitable to combine with class 9. There are 9 candidates. Order them in descending order, which is 11,5,{4,6},7,8,2,10,1,3.

2. Combine class 9 with class 11, compute the classification error $E_{\{9,11\}} = 7.45$ Compute $\varepsilon = E_{\{9\}} - E_{\{5\}} = 7.45 - 5.43 < E_{\{9,11\}}$.

3. Combine class 9 with class 5, compute the classification error $E_{\{9,5\}} = 13.52$, $E_{\{9,11\}}$ is the minimum in the computed classification errors. Compute $\varepsilon = E_{\{9\}} - E_{\{4,6\}} = 7.45 - 5.36 < E_{\{9,11\}}$.

4. Combine class 9 with class {4,6}, compute the classification error $E_{\{9,4,6\}} = 11.01$, $E_{\{9,11\}}$ is the minimum in the computed classification errors. Compute $\varepsilon = E_{\{9\}} - E_{\{7\}} = 7.45 - 4.72 < E_{\{9,11\}}$.

5. Combine class 9 with class 7, compute the classification error $E_{\{9,7\}} = 9.62$, $E_{\{9,11\}}$ is the minimum in the computed classification errors. Compute $\varepsilon = E_{\{9\}} - E_{\{8\}} = 7.45 - 3.95 < E_{\{9,11\}}$.

6. Combine class 9 with class 8, compute the classification error $E_{\{9,8\}} = 2.59$, $E_{\{9,8\}}$ is the minimum in the computed classification errors. Compute $\varepsilon = E_{\{9\}} - E_{\{2\}} = 7.45 - 2.39 > E_{\{9,8\}}$.

Thus, fix {8,9}. Update the class information.

Class	1	2	3	{4,6}	5	7	{8,9}	10	11
C.error(%)	2.00	2.39	2.00	5.36	5.43	4.72	2.59	2.19	6.84

Epoch 3:

1. Class 11 has the largest classification error. Now we want to find the best class which is suitable to combine with class 11. There are 8 candidates. Order them in descending order, which is 5,{4,6},7,{8,9},2,10,1,3.

2. Combine class 11 with class 5, compute the classification error $E_{\{5,11\}} = 13.50$ Compute $\varepsilon = E_{\{11\}} - E_{\{4,6\}} = 6.84 - 5.36 < E_{\{5,11\}}$.

3. Combine class 11 with class {4,6}, compute the classification error $E_{\{11,4,6\}} = 3.44$, $E_{\{4,6,11\}}$ is the minimum in the computed classification errors. Compute $\varepsilon = E_{\{11\}} - E_{\{7\}} = 6.84 - 4.72 < E_{\{4,6,11\}}$.

4. Combine class 11 with class 7, compute the classification error $E_{\{11,7\}} = 10.10$, $E_{\{4,6,11\}}$ is the minimum in the computed classification errors. Compute $\varepsilon = E_{\{11\}} - E_{\{8,9\}} = 6.84 - 2.59 > E_{\{4,6,11\}}$.

Thus, fix {4,6,11}. Update the class information.

Class	1	2	3	{4,6,11}	5	7	{8,9}	10
C.error(%)	2.00	2.39	2.00	3.44	5.43	4.72	2.59	2.19

Epoch 4:

1. Class 5 has the largest classification error. Now we want to find the best class which is suitable to combine with class 5. There are 7 candidates. Order them in descending order, which is 7, {4,6,11},{8,9},2,10,1,3.

2. Combine class 5 with class 7, compute the classification error $E_{\{5,7\}} = 8.32$ Compute $\varepsilon = E_{\{5\}} - E_{\{4,6,11\}} = 5.43 - 3.44 < E_{\{5,7\}}$.

3. Combine class 5 with class {4,6,11}, compute the classification error $E_{\{5,4,6,11\}} = 3.37$, $E_{\{5,4,6,11\}}$ is the minimum in the computed classification errors. Compute $\varepsilon = E_{\{5\}} - E_{\{8,9\}} = 5.43 - 2.59 < E_{\{5,4,6,11\}}$.

4. Combine class 5 with class {8,9}, compute the classification error $E_{\{5,8,9\}} = 10.89$, $E_{\{5,4,6,11\}}$ is the minimum in the computed classification errors. Compute $\varepsilon = E_{\{5\}} - E_{\{2\}} = 5.43 - 2.39 < E_{\{5,4,6,11\}}$.

5. Combine class 5 with class 2, compute the classification error $E_{\{5,2\}} = 7.77$, $E_{\{5,4,6,11\}}$ is the minimum in the computed classification errors. Compute $\varepsilon = E_{\{5\}} - E_{\{10\}} = 5.43 - 2.19 < E_{\{5,4,6,11\}}$.

6. Combine class 5 with class 10, compute the classification error $E_{\{5,10\}} = 10.95$, $E_{\{5,4,6,11\}}$ is the minimum in the computed classification errors. Compute $\varepsilon = E_{\{5\}} - E_{\{1\}} = 5.43 - 2.00 > E_{\{5,4,6,11\}}$.

Thus, fix {4,5,6,11}. Update the class information.

Class	1	2	3	{4,5,6,11}	7	{8,9}	10
C.error(%)	2.00	2.39	2.00	3.37	4.72	2.59	2.19

Epoch 5:

1. Class 7 has the largest classification error. Now we want to find the best class which is suitable to combine with class 7. There are 6 candidates. Order them in descending order, which is {4,5,6,11},{8,9},2,10,1,3.

2. Combine class 7 with class {4,5,6,11}, compute the classification error $E_{\{7,4,5,6,11\}} = 3.16$ Compute $\varepsilon = E_{\{7\}} - E_{\{8,9\}} = 4.72 - 2.59 < E_{\{7,4,5,6,11\}}$.

3. Combine class 7 with class {8,9}, compute the classification error $E_{\{7,8,9\}} = 2.17$, $E_{\{7,8,9\}}$ is the minimum in the computed classification errors. Compute $\varepsilon = E_{\{7\}} - E_{\{2\}} = 4.72 - 2.39 > E_{\{7,8,9\}}$.

Thus, fix {7,8,9}. Update the class information.

Class	1	2	3	{4,5,6,11}	{7,8,9}	10
C.error(%)	2.00	2.39	2.00	3.37	2.17	2.19

Epoch 6:

1. Class {4,5,6,11} has the largest classification error. Now we want to find the best class which is suitable to combine with class {4,5,6,11}. There are 5 candidates. Order them in descending order, which is 2, 10, {7,8,9},1,3.

2. Combine class {4,5,6,11} with class 2, compute the classification error $E_{\{2,4,5,6,11\}} = 9.41$ Compute $\varepsilon = E_{\{4,5,6,11\}} - E_{\{10\}} = 3.37 - 2.19 < E_{\{2,4,5,6,11\}}$.

3. Combine class {4,5,6,11} with class 10, compute the classification error $E_{\{10,4,5,6,11\}} = 4.80$, $E_{\{10,4,5,6,11\}}$ is the minimum in the computed classification errors. Compute $\varepsilon = E_{\{4,5,6,11\}} - E_{\{7,8,9\}} = 3.37 - 2.17 < E_{\{10,4,5,6,11\}}$.

4. Combine class {4,5,6,11} with class {7,8,9}, compute the classification error $E_{\{4,5,6,11,7,8,9\}} = 0.97$, $E_{\{4,5,6,11,7,8,9\}}$ is the minimum in the computed classification errors. Compute $\varepsilon = E_{\{4,5,6,11\}} - E_{\{1\}} = 3.37 - 2.00 > E_{\{4,5,6,11,7,8,9\}}$.

Thus, fix {4,5,6,7,8,9,11}. Update the class information.

Class	1	2	3	{4,5,6,11,7,8,9}	10
C.error(%)	2.00	2.39	2.00	0.97	2.19

Epoch 7:

1. Class 2 is the class with the largest classification error. Now we want to find the best class which is suitable to combine with class 2. There are 4 candidates. Order them in descending order, which is 10,1,3,{4,5,6,7,8,9,11}.

2. Combine class 2 with class 10, compute the classification error $E_{\{2,10\}} = 6.30$ Compute $\varepsilon = E_{\{2\}} - E_{\{1\}} = 2.39 - 2 < E_{\{2,10\}}$.

3. Combine class 2 with class 1, compute the classification error $E_{\{2,1\}} = 4.15$, $E_{\{2,1\}}$ is the minimum in the computed classification errors. Compute $\varepsilon = E_{\{2\}} - E_{\{3\}} = 2.39 - 2 < E_{\{2,1\}}$.

4. Combine class 2 with class 3, compute the classification error $E_{\{2,3\}} = 1.08$, $E_{\{2,3\}}$ is the minimum in the computed classification errors. Compute $\varepsilon = E_{\{2\}} - E_{\{4,5,6,7,8,9,11\}} = 2.39 - 0.97 > E_{\{2,3\}}$.

Thus, fix {2,3}. Update the class information.

Class	1	{2,3}	{4,5,6,11,7,8,9}	10
C.error(%)	2.00	1.08	0.97	2.19

Epoch 8:

1. Class 10 is the class with the largest classification error. Now we want to find the best class which is suitable to combine with class 10. There are 3 candidates. Order them in descending order, which is 1,{2,3},{4,5,6,7,8,9,11}.

2. Combine class 10 with class 1, compute the classification error $E_{\{1,10\}} = 1.25$ Compute $\varepsilon = E_{\{10\}} - E_{\{2,3\}} = 2.19 - 1.08 < E_{\{1,10\}}$.

3. Combine class 10 with class {2,3}, compute the classification error $E_{\{2,3,10\}} = 3.99$, $E_{\{1,10\}}$ is the minimum in the computed classification errors. Compute $\varepsilon = E_{\{10\}} - E_{\{4,5,6,7,8,9,11\}} = 2.19 - 0.97 < E_{\{1,10\}}$.

4. Combine class 10 with class {4,5,6,7,8,9,11}, compute the classification error $E_{\{4,5,6,7,8,9,10,11\}} = 5.63$, $E_{\{1,10\}}$ is the minimum in the computed classification errors.

Because all the combination candidates are used, fix {1,10}. Update the class information.

Class	{1,10}	{2,3}	{4,5,6,11,7,8,9}
C.error(%)	1.25	1.08	0.97

Epoch 9:

1. Class {1,10} is the class with the largest classification error. Now we want to find the best class which is suitable to combine with class {1,10}. There are 2 candidates. Order them in descending order, which is {2,3},{4,5,6,7,8,9,11}.

2. Combine class {1,10} with class {2,3}, compute the classification error $E_{\{1,10,2,3\}} = 0.97$ Compute $\varepsilon = E_{\{1,10\}} - E_{\{4,5,6,7,8,9,11\}} = 1.25 - 0.97 < E_{\{1,10,2,3\}}$.

3. Combine class 10 with class {4,5,6,7,8,9,11}, compute the classification error $E_{\{1,4,5,6,7,8,9,10,11\}} = 1.08$, $E_{\{1,2,3,10\}}$ is the minimum in the computed classification errors.

Because all the combination candidates are used, fix {1,10,2,3}. Update the class information.

Class	{1,10,2,3}	{4,5,6,11,7,8,9}
C.error(%)	0.97	0.97

In this example, the ICSPD algorithm is used to compute the combination set for the Vowel problem. The final combination set is {{1,2,3,10}, {4,5,6,7,8,9,11}}.

Discussion:

Comparing the results using SGCS algorithm with that using the GCS algorithm, we can see that the final combination set is the same. Eight epochs are used for both algorithms. We computed 65 classification errors for those classes or combinations in GCS algorithm, while only 43 classification errors are computed in SGCS algorithm. Because for both algorithms, the time consumed is mainly on the computation of classification errors, the SGCS algorithm saves 1/3 time compared with the GCS algorithm.

5.4.6 The $\sqrt{K}$ Rule-of-thumb

The number of wrongly classified patterns by the whole PD network can be expressed as:

$$T = N \cdot E_w + (1 - E_w)(N_1 E_1 + N_2 E_2 + \cdots + N_r E_r) \quad , \tag{5.9}$$

where N is the number of overall patterns, N_j is the number of patterns belonging to non-distributor Module j. E_w is the classification error of the distributor module and E_j is the classification error of non-distributor Module j.

Assume that the maximum classification error of the non-distributor modules is E_{max}. From Expression (5.9), we have

$$\begin{aligned} T &\leq N \cdot E_w + (1 - E_w)(N_1 E_{max} + N_2 E_{max} + \cdots + N_r E_{max}) \\ &= N \cdot E_w + N \cdot (1 - E_w) E_{max} \quad . \end{aligned} \tag{5.10}$$

From the analysis in chapter 3, $E_w E_{max}$ is much smaller than E_w or E_{max}. Thus, Expression (5.10) can be revised as

$$T \leq N \cdot (E_w + E_{max}) \quad . \tag{5.11}$$

From Expression (5.11), we can see if $E_w + E_{max}$ is minimized, the number of the wrongly classified patterns of the whole PD network T will be a relatively small number. Thus, we need to evaluate the relation between E_{max} and E_w. Seems it is a hard problem to set up the relation between them.

According to Theorem 5.2, a non-distributor module having a relatively large number of classes tends to have larger classification errors. According to Theorem 5.3, a distributor module with a large number of combinations or outputs tends to have larger classification errors. The concrete relation should be problem dependent. For the ease of evaluation, we assume the classification error is proportional to the number of outputs or classes. We have

$$E_w = e_0(r - 1) \quad , \tag{5.12}$$

where r is the number of outputs in the distributor module and e_0 is some constant. If the distributor module only has one output, it is obvious that the classification

error of the distributor module is zero. Thus, in Equation (5.12), E_w is proportional to $(r-1)$.

$$E_{max} = e_1(m-1) \ , \tag{5.13}$$

where m is number of outputs in the non-distributor module. Because the two module is for the same problem, we assume $e_1 = e_0$. Then Expression (5.11) can be revised as

$$T \leq N \cdot e_0 \cdot (m+r-2) \ . \tag{5.14}$$

For $m \cdot r \sim K$, where K is the number of classes in the problem. In order to minimize $m+r$, we have $m = r \sim \sqrt{K}$.

The above analysis shows that the maximum number of classes in a non-distributor module should be close to $\sqrt{K}$. Thus, we set a rule such that the number of classes in each non-distributor module can not exceed $\sqrt{K}$. This rule is called the $\sqrt{K}$ Rule-of-thumb. This rule is used in several of our combination selection algorithms, including the combination selection algorithms presented in the next chapter. In the next chapter, we will verify the rule from experiments.

5.4.7 Restricted Greedy Combination Selection (RGCS) Algorithm

Compared with multi-layer PD networks, the single-layer PD network has the shorter processing time for unknown patterns. In this section, we want to develop a combination selection algorithm for single-layer PD networks. The parameter $\sqrt{K}$ (K is the number of classes in the problem) in fact strikes a balance between the distributor modules and those non-distributor modules. One may wonder whether we should add the restriction $\sqrt{K}$ to the distributor module. Our answer is no. Now we explain the reason.

In the course of deducing the $\sqrt{K}$ rule, we assume the classification error is proportional to the number of outputs or classes for the ease to evaluation. It is an acceptable assumption when we lack the information of classes and combinations. However, in our greedy based combination selection algorithms, we compute the classification error of temporary combinations in each epoch. In other words, we

have the information of combinations. At the same time, for the distributor module, we can estimate the classification error of the temporary combination set in each epoch according to Theorem 5.1. We do not need to assume the classification error of the distributor module is proportional to the number of outputs. Thus, we do not apply the $\sqrt{K}$ rule to the distributor module.

In our actual experiments, some negative effect may be generated if we insist on applying the $\sqrt{K}$ rule to the distributor module. Here we give an example. For the Vowel problem which has 11 classes, firstly, we just apply the $\sqrt{K}$ rule for the non-distributor modules and set the maximum number of classes in a non-distributor module is 3 (Refer Section 5.4.8 for detail). Then RGCS find the combination set {{4,6,11}, {7,8,9}, {1,10}, {2,3}, {5}}, which has 5 combinations. The classification errors for these combinations are listed as follows: {4,6,11}/3.44%, {7,8,9}/2.17%, {1,10}/1.25%, {2,3}/1.08%, {5}/5.43%. Now we want to farther apply the $\sqrt{K}$ rule to the distributor module while keeping the $\sqrt{K}$ rule satisfied in non-distributor modules. The maximum number of outputs of the distributor module is set to be 4. Therefore, we need to combine {5} to {1,10} or {2,3}. Through computation, we find that {5,1,10}/10.12% and {5,2,3}/9.72%. We can estimate the classification error of the combination set according to Theorem 5.1. In either situation, the classification error of the distributor module after combination is larger than that before combination. This example shows that using the $\sqrt{K}$ rule to the distributor module may bring negative effect.

For the above reason, we do not add the restriction $\sqrt{K}$ to the distributor module. We still need to apply the $\sqrt{K}$ rule to these non-distributor modules, because we do not have enough information to trace the classification error of the corresponding non-distributor modules in our greedy based combination algorithms.

The RGCS algorithm is similar to the GCS algorithm. The main difference between them is that we add the restriction to the maximum number of classes in a combination. Before presenting the RGCS algorithm, we define two concepts. N_{oc} is defined as the number of original classes in a combination. For example, the N_{oc} in combination {3,4} is 2. Sometimes combination {3,4} is also called class {3,4} for

that after combination, class 3 and class 4 become one class. N_{oc-max} is defined as the maximum number of original classes in a combination. N_{oc-max} is set according to the $\sqrt{K}$ rule. For simplicity, we just set it as the maximum integer which is equal or smaller than $\sqrt{K}$. An "excluded" class means a class that we no longer choose as the candidate for combining with others. The RGCS algorithm is described as follows:

1. Find the classification error of each class $E_{\{i\}}$. Mark all the classes as "non-excluded".

2. Set $epoch = 1$.

3. Find class j with the largest classification error among all the "non-excluded" classes.

4. Choose another "non-excluded" class and temporarily combine it with class j. For instance, class i, combines it with class j, i.e. $\{i, j\}$. Thus, the two classes become one class. If the N_{oc} in the combination is larger than N_{oc-max}, cancel the combination; otherwise, measure the classification error $E_{\{i,j\}}$.

5. Repeat Step 4 for all the remaining "non-excluded" classes.

6. If all the combinations in Step 4 and 5 are cancelled, mark class j as "excluded" class and go to Step 8. Otherwise, find the combination with the smallest classification error in the temporary combinations which have been measured in Steps 4 and 5, i.e. combination {t,j} with the smallest classification error $E_{\{t,j\}}$.

7. Compare $E_{\{t,j\}}$ with $E_{\{j\}}$. If $E_{\{t,j\}} > E_{\{j\}}$, mark class j as "excluded" and go to Step 8. Otherwise, fix combination {t,j}, thus {t,j} becomes one class. If the N_{oc} in combination {t,j} is equal to N_{oc-max}, exclude {t,j}.

8. Update the information of the classes.

9. If there is only one "non-excluded" class remaining, stop. Otherwise, set epoch $epoch = epoch + 1$ and go to step 3.

5.4.8 An Example for RGCS

Here we will illustrate using an example to show how RGCS algorithm progresses step by step. It should be noted that this example just show how RGCS works and we do not give the results of the final PD network until later. We still look at the Vowel problem. Now we use RGCS to find a combination set for the single-layer PD network. In the problem, $\sqrt{K} = \sqrt{11}$. The maximum integer which is smaller than is 3. Thus, we set $N_{oc-max} = 3$.

The classification error of each class is computed and listed here:

Class	1	2	3	4	5	6	7	8	9	10	11
C.error(%)	2.00	2.39	2.00	2.81	5.43	8.40	4.72	3.95	7.45	2.19	6.84

Epoch 1:

Class 6 has the largest classification error. Combine class 6 with the other classes respectively and compute the classification error.

{6,1}/12.02 {6,2}/16.48 {6,3}/11.72 **{6,4}/5.36** {6,5}/6.03
{6,7}/15.36 {6,8}/11.64 {6,9}/16.44 {6,10}/11.2 {6,11}/14.90

Combination {6,4} has the smallest classification error, fix it. Update the class information.

Class	1	2	3	{4,6}	5	7	8	9	10	11
C.error(%)	2.00	2.39	2.00	5.36	5.43	4.72	3.95	7.45	2.19	6.84

Epoch 2:

Class 9 has the largest classification error. Combine class 9 with the other non-excluded classes respectively and compute the classification error.

{9,1}/7.31 {9,2}/13.60 {9,3}/7.17 {9,4,6}/11.01 {9,5}/13.52
{9,7}/9.62 **{9,8}/2.59** {9,10}/3.36 {9,11}/7.45

Combination {9,8} has the smallest classification error, fix it. Update the class information.

Class	1	2	3	{4,6}	5	7	{8,9}	10	11
C.error(%)	2.00	2.39	2.00	5.36	5.43	4.72	2.59	2.19	6.84

Epoch 3:

Class 11 has the largest classification error. Combine class 11 with the other non-excluded classes respectively and compute the classification error.

{11,1}/9.09 {11,2}/10.06 {11,3}/7.37 **{11,4,6}/3.44**
{11,5}/13.50 {11,7}/10.10 {11,8,9}/7.71 {11,10}/10.38

Combination {11,4,6} has the smallest classification error, fix it.For the number of classes has reach N_{oc-max}, exclude class 4,6,11 and update the class information.

Class	1	2	3	5	7	{8,9}	10	Excluded	{4,6,11}
C.error(%)	2.00	2.39	2.00	5.43	4.72	2.59	2.19		3.44

Epoch 4:

Class 5 has the largest classification error. Combine class 5 with the other non-excluded classes respectively and compute the classification error.

{5,1}/9.64 **{5,2}/7.77** {5,3}/13.91
{5,7}/8.32 {5,8,9}/10.89 {5,10}/10.95

Combination {5,2} has the smallest classification error. However $E_{\{5,2\}} > E_{\{5\}}$, class 5 does not combine with the others. Exclude class {5} and update the class information.

Class	1	2	3	7	{8,9}	10	Excluded	{4,6,11}	5
C.error(%)	2.00	2.39	2.00	4.72	2.59	2.19		3.44	5.43

Epoch 5:

Class 7 has the largest classification error. Combine class 7 with the other non-excluded classes respectively and compute the classification error.

{7,1}/10.53 {7,2}/9.65 {7,3}/8.18 **{7,8,9}/2.17** {7,10}/7.19

Combination {7,8,9} has the smallest classification error, fix it. Exclude {7,8,9} and update the class information.

Class	1	2	3	10	Excluded	{4,6,11}	5	{7,8,9}
C.error(%)	2.00	2.39	2.00	2.19		3.44	5.43	2.17

Epoch 6:

Class 2 has the largest classification error. Combine class 2 with the other non-excluded classes respectively and compute the classification error.

{2,1}/4.15 **{2,3}/1.08** {2,10}/6.30

Combination {2,3} has the smallest classification error, fix it. Update the class information.

Class	1	{2,3}	10	Excluded	{4,6,11}	5	{7,8,9}
C.error(%)	2.00	1.08	2.19		3.44	5.43	2.17

Epoch 7:

Class 10 has the largest classification error. Combine class 10 with the other non-excluded classes respectively and compute the classification error.

{10,1}/1.25 {10,2,3}/3.99

Combination {10,1} has the smallest classification error, fix it. Update the class information.

Class	{1,10}	{2,3}	Excluded	{4,6,11}	5	{7,8,9}
C.error(%)	1.25	1.08		3.44	5.43	2.17

Epoch 8:

Class {1,10} has the largest classification error. Combine class {1,10} with class {2,3}. The N_{oc} after combination is 4, which is larger than N_{oc-max}. So cancel the combination and exclude {1,10}. Update the class information.

Class	{2,3}	Excluded	{4,6,11}	5	{7,8,9}	{1,10}
C.error(%)	1.08		3.44	5.43	2.17	1.25

The final combination set is {{4,6,11}, {7,8,9}, {1,10}, {2,3}, {5}}. According to the analysis in section 5.4.7, the $\sqrt{K}$ rule is not suitable for the distributor module. Therefore, the number of combinations in a combination set can be larger than $\sqrt{K}$.

5.5 Experimental Results for the PDs Using GCS, SGCS and RGCS

We present the experimental results for three data sets here. For each data set, we create multi-level PD networks using the GCS algorithm and SGCS algorithm. For each network, GCS (or SGCS) algorithm is used several times to form a multi-layer PD network (see the Segmentation problem for detail). For RGCS algorithm is designed for single-layer PD networks, we also set up a single-layer PD network using RGCS algorithm.Of course, RGCS may also extend to be used in multi-layer PDs, but here we only restrict it to single-layer PDs.

A. Segmentation

This data set has 7 outputs. Firstly, we set up the network using GCS algorithm. The training data and validation data are combined together, then divided into three new sets - training set 2, validation set 2 and test set 2. The new sets are used by GCS algorithm searching for good combination sets. The procedure is described as follows:

1. GCS algorithm is applied to the new sets and find the combination set {{1,3,4,5}, {2,7}, {6}}. The combination set is for the 1^{st} level distributor module.

2. After that, we select the patterns which belong to class 1,3,4,5, then use GCS to these patterns and find the combination set {{3,4,5}, {1}} for the 2^{nd} level distributor module.

3. We select the patterns which belong to class 3,4,5, then use GCS to these patterns and find the combination set {{3,5}, {4}} for the 3^{rd} level distributor module.

If a combination has 3 or more classes, GCS algorithm will be continuously applied to such a combination to form next-level distributor modules. Thus, after Step 1, we use GCS to form the second-level distributor module, and after Step 2, we use GCS to form the third-level distributor module. If a combination has 2 classes, a non-distributor module will be used to classify the two classes. In this way, we avoid the appearance of large non-distributor modules and make each module's task relatively each to handle.

Then we used the above information set up the network. The network structure is shown on Figure 5.5. From Figure 5.5, we can see that the PD network is composed with three distributor modules and two non-distributor modules. The first level distributor module has three outputs, {1,3,4,5}, {2,7} and {6} and the output {1,3,4,5} connects to the second level distributor module. The second level distributor module has two outputs, {3,4,5}and {1} and the output {3,4,5} connects to the third level distributor module. The third level distributor module has two outputs, {3,5}and {4}. Two non-distributor modules are used in the network to recognize class 2,7 and class 3,5.

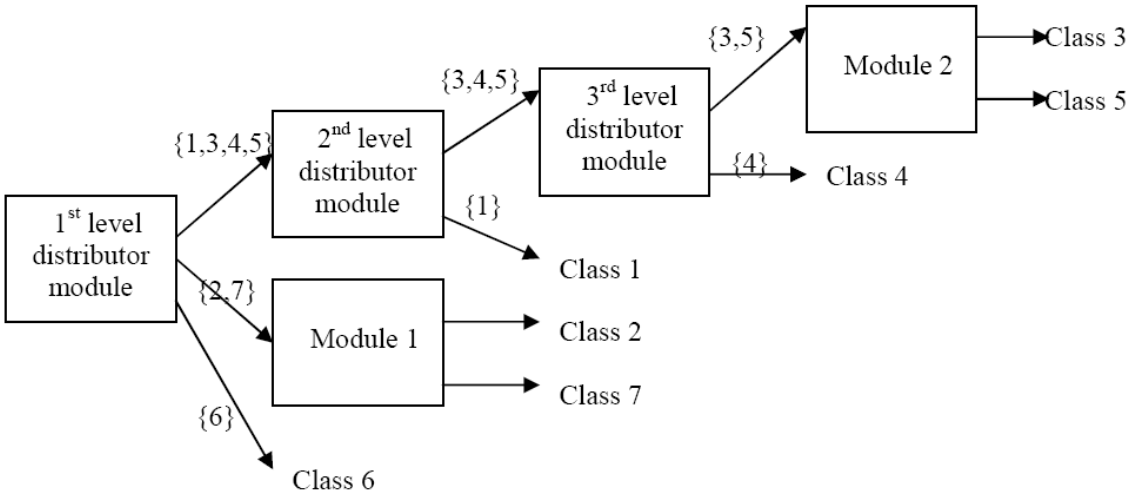

Figure 5.5: The PD network structure based on GCS for the Segmentation problem

It can be seen that this network has an imbalanced module distribution, which means some branches have more depth than the other branches. GCS and SGCS is designed to find a combination set of relatively small classification error, not to find a combination set with balanced class distribution.

Now we set up the network using SGCS algorithm. The network structure is shown in Figure 5.6. We can see that the PD network using SGCS resembles that using GCS. The second level distributor module and the third level distributor module are the same as that using GCS. The first distributor module using SGCS has small difference from that using GCS. The first distributor module using SGCS has the outputs {1,3,4,5}, {6,7} and {2}, while that using GCS has the outputs {1,3,4,5}, {2,7} and {6}.

Then we construct the single-layer PD network using RGCS algorithm. In the problem,$\sqrt{K} = \sqrt{7}$. The maximum integer which is smaller than is 2. Thus, we set $N_{oc-max} = 2$. The network structure is shown on Figure 5.7. The PD network is composed with one distributor module and three non-distributor modules. The distributor module has four outputs, {1,4}, {2,7}, {3,5} and {6}. Three non-distributor modules are used to recognize {1,4}, {2,7} and {3,5} respectively.

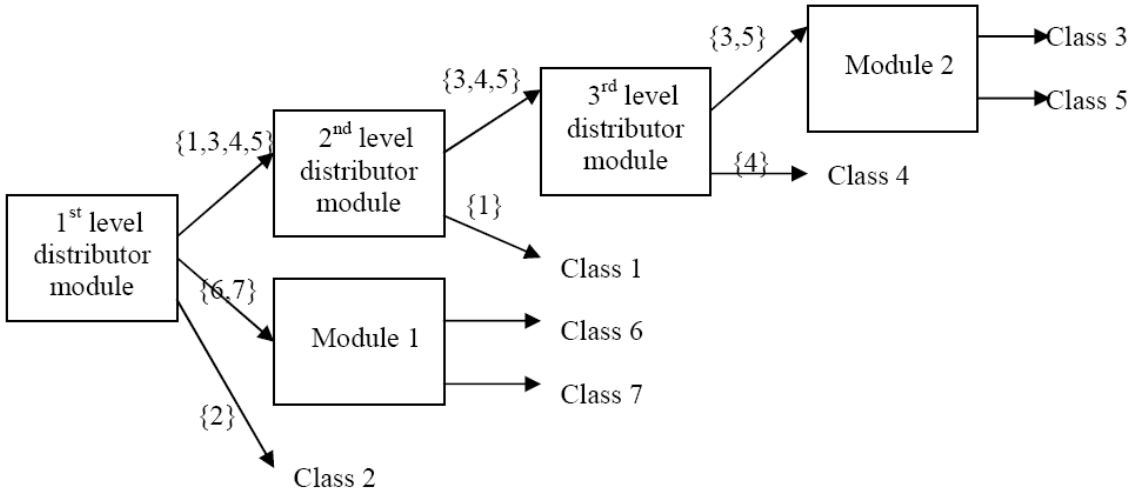

Figure 5.6: The PD network structure based on SGCS for the Segmentation problem

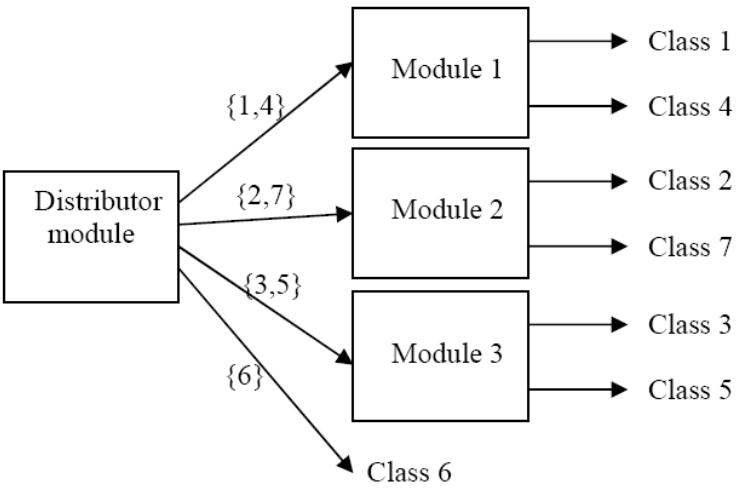

Figure 5.7: The PD network structure based on RGCS for the Segmentation problem

Table 5.2: Results using GCS for the Segmentation problem

Network		Training time (s)	Hidden Units	Indep. Param.	C.error (%)
Distributor Module	The 1^{st} Level	188.5 (in parallel) 314.6 (in series)	37.85	814	0.0260
	The 2^{nd} Level	5.1 (in parallel) 9.95 (in series)	1.95	77	1.2766
	The 3^{rd} Level	41.1 (in parallel) 75.7 (in series)	33.9	716	2.0528
Overall Network		188.5 (in parallel) 432.85 (in series)	92.05	2050	3.8821

Table 5.3: Results using SGCS for the Segmentation problem

Network		Training time (s)	Hidden Units	Indep. Param.	C.error (%)
Distributor Module	The 1^{st} Level	272.3 (in parallel) 387.4 (in series)	61.45	1286	0.01733
	The 2^{nd} Level	5.1 (in parallel) 9.95 (in series)	1.95	77	1.2766
	The 3^{rd} Level	41.1 (in parallel) 75.7 (in series)	33.9	716	2.0528
Overall Network		272.3 (in parallel) 523.9 (in series)	160.45	3418	3.8735

Table 5.4: Results using RGCS for the Segmentation problem

Network	Training time (s)	Hidden Units	Indep. Param.	C.error (%)
Distributor Module {{1,4},{2,7},{3,5},{6}}	261.5 (in parallel) 589.5 (in series)	62.15	1319	1.0225
Overall Network	261.5 (in parallel) 875.35 (in series)	140.65	3003	3.4662

Table 5.5: Results of different methods for the Segmentation problem

Network	Hidden Units	Indep. Param.	C.error (%)	C. error reduction vs. OP(%)
Ordinary method	29	887	5.7366	-
Output Parallelism	152.1	3175	5.1820	-
Arbitrary single-layer PD (in Chapter 3)	128.9	2768	4.6101	0.5719
Arbitrary balanced 2-layer PD (in Chapter 4)	165.8	3506	4.6447	0.5373
Multi-layer PD using GCS	92.05	2050	3.8821	1.2999
Multi-layer PD using SGCS	160.45	3418	3.8735	1.3085
Single-layer PD using RGCS	140.65	3003	3.4922	1.6898

Table 5.2 to Table 5.4 shows the experimental results of the multi-layer PD networks using GCS algorithm and SGCS algorithm and the experimental results of the single-layer PD network using the IGCS algorithm. For comparison, we list the classification errors of these networks, and experimental results in Chapter 3 and Chapter 4 for this problem on Table 5.5.

From Table 5.5, it can be seen that the PD networks, especially the PDs using GCS, SGCS and RGCS algorithms, have much better performance compared with the OP network and ordinary network. It also shows that the networks using the Greedy based combination algorithms have lower classification error than arbitrarily selected PD networks. It is an inspiring signal. It means our Greedy based combination algorithms indeed improved the network's performance. In the next chapter, we will compare the performance of the PD networks using different combination algorithms.

B. Vowel

We set up the network using GCS algorithm. The network structure is shown on Figure 5.8. The network using SGCS algorithm is the same as that using GCS algorithm. From Figure 5.8, we can see that the PD network is composed with six distributor modules and four non-distributor modules. The 1^{st} level distributor module has two outputs, {4,5,6,7,8,9,11} and {1,2,3,10}. The output {4,5,6,7,8,9,11} connects to the 2^{nd} level distributor module 1 and the output {1,2,3,10} connects to the 2^{nd} level distributor module 2. The 2^{nd} level distributor module 1 has two outputs, {4,5,6,11}and {7,8,9}. The output {4,5,6,11} connects to the 3^{rd} level distributor module 1 and the output {7,8,9} connects to the 3^{rd} level distributor module 2. The 2^{nd} level distributor module 2 has two outputs, 1,10and 2,3 and the outputs connects to two non-distributor modules. The 3^{rd} level distributor module 1 has two outputs, {5,6,11}and {4} and the outputs {5,6,11} connects to the 4th level distributor module. The 3^{rd} level distributor module 2 has two outputs, {7,8}and {9}. The 4^{th} level distributor module has two outputs, {6,11}and {5}.

Now we set up the single-layer PD network using RGCS algorithm. In the problem, $\sqrt{K} = \sqrt{11}$. we set $N_{oc-max} = 3$. The network structure is shown on

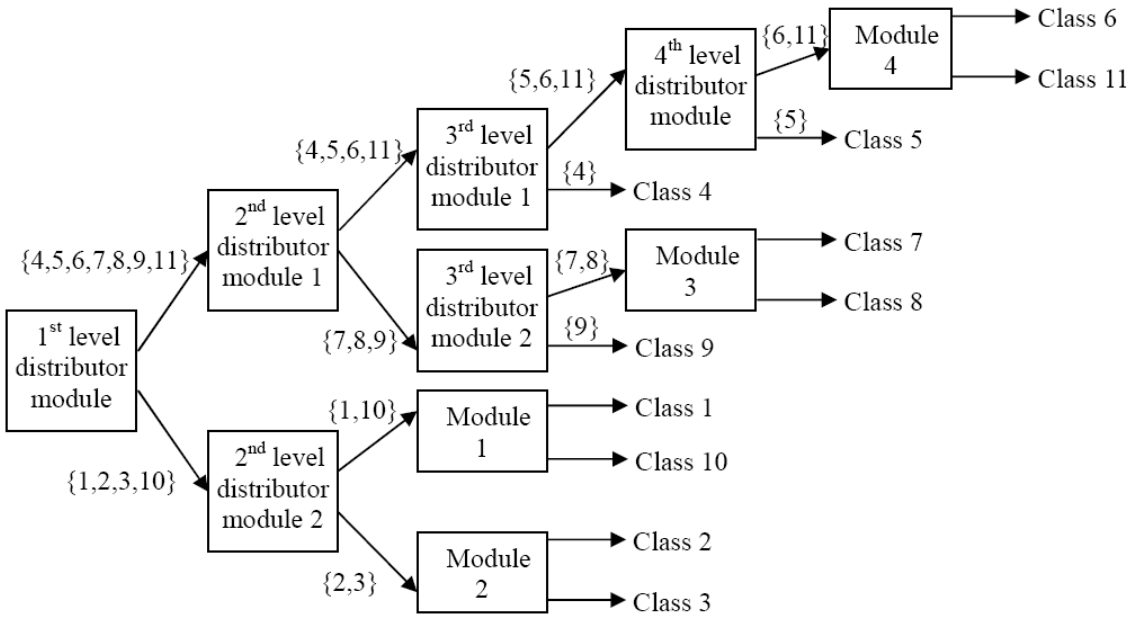

Figure 5.8: The PD network structure based on GCS and SGCS for the Vowel problem

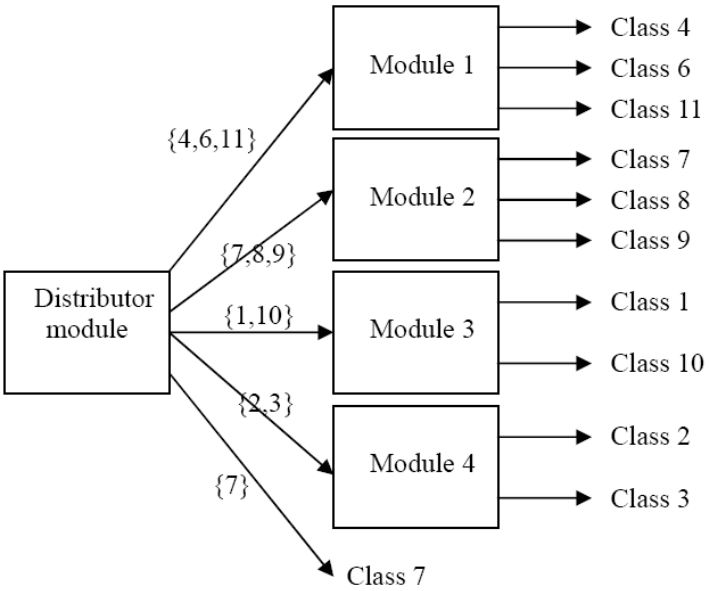

Figure 5.9: The PD network using RGCS for the Vowel problem

Table 5.6: Results using GCS and SGCS for the Vowel problem

Network		Training time (s)	Hidden Units	Indep. Param.	C.error (%)
Distributor Module	The 1^{st} Level	66.9 (in parallel) 121.15 (in series)	35.95	453.4	1.0324
	The 2^{nd} Level 1	46.7 (in parallel) 87.4 (in series)	29.1	371.2	1.4458
	The 2^{nd} Level 2	18.25 (in parallel) 32.75 (in series)	41.4	518.8	1.8519
	The 3^{rd} Level 1	6.1 (in parallel) 11.25 (in series)	20.45	267.4	6.6484
	The 3^{rd} Level 2	7.65 (in parallel) 15.05 (in series)	29.4	374.8	3.7333
	The 4^{th} Level	15.15 (in parallel) 28.55 (in series)	27.7	354.4	7.5758
Overall Network		66.9 (in parallel) 166.9 (in series)	332.85	4137.2	11.6194

Table 5.7: Results using RGCS for the Vowel problem

Network	Training time (s)	Hidden Units	Indep. Param.	C.error (%)
Distributor Module {4,6,11},{7,8,9},{1,10},{2,3},{5}	94.25 (in parallel) 313.95 (in series)	87.7	1107.4	5.8502
Overall Network	94.25 (in parallel) 471.3 (in series)	155.8	2045.6	12.2672

Table 5.8: Results of different methods for the Vowel problem

Network	Hidden Units	Indep. Param.	C.error (%)	C. error reduction vs. OP(%)
Ordinary method	23.6	640.2	37.1660	-
Output Parallelism	184.4	2333.8	25.5466	-
Arbitrary single-layer PD (in Chapter 3)	229.4	2955.8	18.7045	6.8421
Arbitrary balanced 2-layer PD (in Chapter 4)	162.25	2134	16.9231	8.6235
Arbitrary imbalanced 2-layer PD (in Chapter 4)	145.2	1918.4	17.6316	7.915
Multi-layer PD using GCS	332.85	4137.2	11.6194	13.9272
Multi-layer PD using SGCS	332.85	4137.2	11.6194	13.9272
Single-layer PD using RGCS	155.8	2045.6	12.2672	13.2794

Figure 5.9. The PD network is composed with one distributor module and four non-distributor modules. The distributor module has five outputs, {4,6,11}, {7,8,9}, {1, 10}, {2,3} and {5}. Because the output {5} has only one class, we need not use a non-distributor module for combination {5}. Thus, four non-distributor modules are used to recognize {4,6,11}, {7,8,9}, {1, 10} and {2,3} respectively.

The experimental results for the multi-layer PD network using GCS and SGCS algorithms for the Vowel data are listed in Table 5.6 and the experimental results for the single-layer PD network using RGCS algorithm are show on Table 5.7.

For comparison, we list the classification errors of these networks, and experimental results in Chapter 3 and Chapter 4 for this problem on Table 5.8. From Table 5.8, it can be seen that the PD networks, especially the PDs using GCS, SGCS and RGCS algorithms, have much better performance compared with the OP network and ordinary network. It also shows that the networks using the Greedy based combination algorithms have lower classification error than arbitrarily selected PD networks. It means our Greedy based combination algorithms indeed improved the network's performance.

C. Pen-Based Recognition

Now we set up the network using GCS algorithm SGCS algorithm. The network using SGCS algorithm is the same as that using GCS algorithm. The network structure is shown on Figure 5.10. From Figure 5.10, we can see that the PD network is composed with two distributor modules and two non-distributor modules. The 1^{st} level distributor module has seven outputs, {1},{3}, {4}, {5}, {8}, {7,9} and {2,6,10}. The output {2,6,10} connects to the 2^{nd} level distributor module and the output 7,9 connects to Module 1. For the other outputs of the 1^{st} level distributor module only has one class, no modules is needed to classify them. The 2^{nd} level distributor module has two outputs, {6,10}and {2} and the output {6,10} connects to Module 2.

In the problem, $\sqrt{K} = \sqrt{10}$. The maximum integer which is not smaller than is 3, so we set $N_{oc-max} = 3$. The network is set up using the RGCS algorithm. The

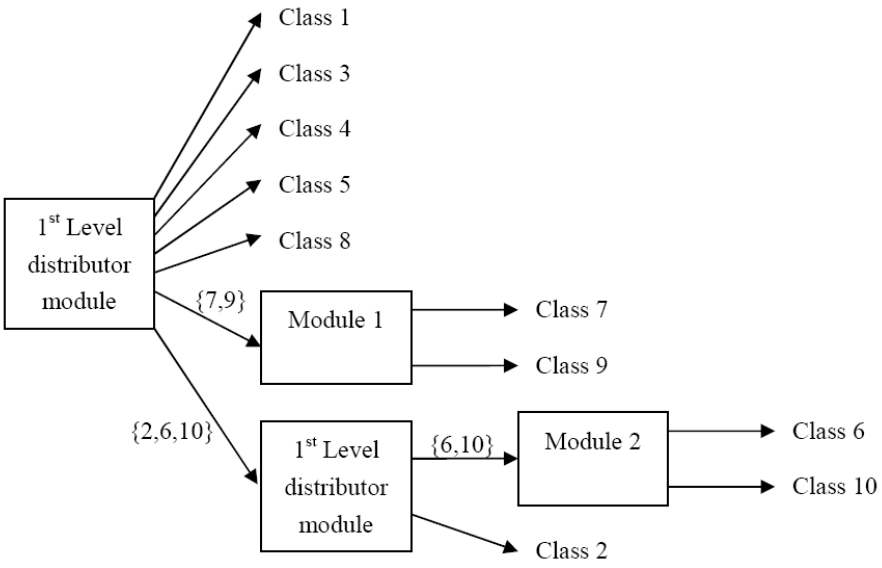

Figure 5.10: The PD network structure using GCS and SGCS algorithm for the Pen-Based Recognition problem

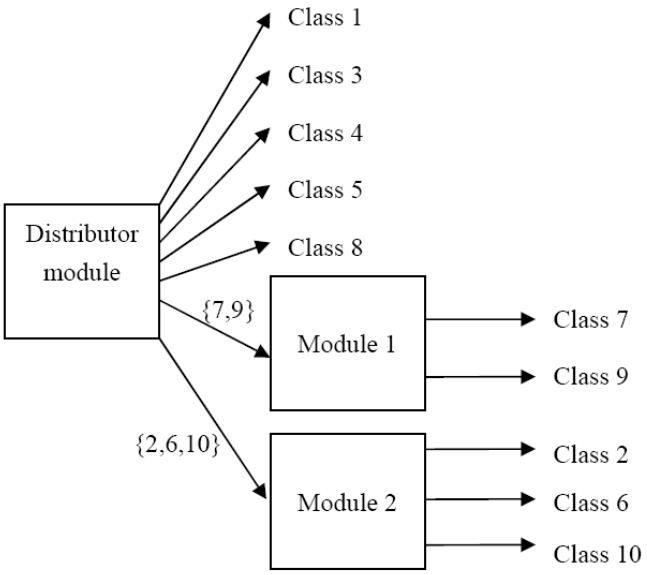

Figure 5.11: The PD network structure using RGCS algorithm for the Pen-Based Recognition problem

Table 5.9: Results using GCS and SGCS for the Pen-based recognition problem

Network		Training time (s)	Hidden Units	Indep. Param.	C.error (%)
Distributor Module	The 1^{st} Level	976.7 (in parallel) 3685.35 (in series)	178.65	3334.7	0.7951
	The 2^{nd} Level	186.2 (in parallel) 358.85 (in series)	49.1	917.8	0.4430
Overall Network		976.7 (in parallel) 4157.55 (in series)	413.35	7661.3	1.0058

Table 5.10: Results using RGCS for the Pen-Based Recognition problem problem

Network	Training time (s)	Hidden Units	Indep. Param.	C.error (%)
Distributor Module {1},{3},{4},{5},{8},{7,9},{2,6,10}	976.7 (in parallel) 3685.35 (in series)	178.65	3334.7	0.7951
Overall Network	976.7 (in parallel) 4024.3 (in series)	299.65	5597.7	1.0592

network structure is shown on Figure 5.11. The PD network is composed with one distributor module and two non-distributor modules. The distributor module has seven outputs, {1}, {3}, {4}, {5}, {8}, {7,9} and {2,6,10}. Two non-distributor modules are used to recognize {7,9} and {2,6,10} respectively.

The experimental results of the PD network based on the GCS or SGCS algorithm for the Pen-based recognition data are shown in Table 5.9 and the results for the IGCS algorithm are listed in Table 5.10.

For comparison, we list the classification errors of the above two networks, to-

Table 5.11: Results of different methods for the Pen-Based Recognition problem

Network	Hidden Units	Indep. Param.	C.error (%)	C. error reduction vs. OP(%)
Ordinary method	42.4	1314.8	4.9493	-
Output Parallelism	239.05	4472.9	5.0560	-
Arbitrary single-layer PD (in Chapter 4)	276.79	5220.22	5.0961	-0.0401
Arbitrary balanced 2-layer PD (in Chapter 4)	359.24	6738.32	1.8490	3.207
Arbitrary imbalanced 2-layer PD (in Chapter 4)	1331.14	6215.52	1.4541	3.6019
Multi-layer PD using GCS	413.35	7661.3	1.0085	4.0475
Multi-layer PD using SGCS	413.35	7661.3	1.0085	4.0475
Single-layer PD using RGCS	299.65	5597.7	1.0592	3.9968

gether with non-task decomposition method (ordinary method), OP network and a PD network for a randomly selected combination set in Table 5.11. The table shows that using GCS, SGCS and RGCS algorithms, the classification errors for this problem is reduced a lot. It shows that our Greedy based combination algorithm can indeed improved the network's performance.

5.6 Discussion

In this chapter, we discussed three greedy based combination selection algorithms. There is a question whether the $\sqrt{K}$ rule can be combined to GCS or SGCS algorithm to further improve the performance of the whole network, i.e. we may restrict the combination set found by GCS or SGCS having $\sqrt{K}$ elements. The answer is no. Now we explain why. When deducing the $\sqrt{K}$ rule, we assume the classification error is proportional to the number of outputs (for the distributor module) or classes (for the non-distributor modules). In actual problems, the relation between them often deviates from the above assumption. The classification error of the distributor modules is also related to the combination set itself. The classification error of a non-distributor module is related to classes in that module, too. For the non-distributor modules, because we have no information of classes and don't know

the classification error, that assumption is acceptable. While for the distributor module, it is different. When GCS or SGCS is used to find a suitable combination set, in each epoch, the classification errors of the combinations is known. Thus, we can estimate the classification error of the temporary combination set according to Theorem 5.1. In other words, we do not need the above assumption that the classification error is proportional to the number of outputs. The $\sqrt{K}$ rule is not suitable for the distributor module. Therefore, we do not restrict the combination set found by GCS or SGCS having $\sqrt{K}$ elements.

Chapter 6

FLD and GA Based Class Combination Methods

6.1 Introduction

In the last chapter, several greedy combination algorithms are proposed to find suitable combination sets for the distributor modules. In this chapter, we continue to discuss class combination in the distributor modules. Two other combination methods will be introduced here. The first one is Cross-talk Table based combination selection for a distributor module (in short, CTCS), which finds a suitable combination set using the cross-talk table based on Fisher's Linear Discriminant method. The second one is called Genetic Algorithm based Combination Selection for a distributor module (in short, GACS), which searches an optimal or near-optimal combination set through an evolutionary method.

Both CTCS and GACS have used the $\sqrt{K}$ rule-of-thumb. In Section 5.4.6, we have deduced this rule from theoretical analysis. In this chapter, we show some experimental results to further validate the rule. These combination methods (including the methods introduced in the last chapter) have advantages and disadvantages. After we introduce all the combination methods, we will compare these combination methods in detail.

6.2 Cross-talk Based Combination Selection (CTCS) Algorithm

The basic idea of this method is to find and combine classes which are hard to discriminate in the feature space.

Normally, the feature space in classification problems has a high dimension and the problems are often very complicated. The distribution of data in the feature space is also usually biased. Thus, it is hard to say which classes are easy to divide and which classes are hard to discriminate directly. Therefore, we use Fisher's linear discriminant (FLD) method to decide the discrimination capacity between classes (Duda, et al., 2000; Friedman, 1989; Mika, et al. 1999). Now we introduce FLD first.

FLD is a widely used feature extraction technique in linear pattern recognition. For two-class problems, it projects a d-dimensional feature space into a one dimensional feature space, where d is the number of features, by the transformation function $y_i = w^t x_i$.

Let a set of N training patterns be X, where $x_i \in R^d$, $i = 1, 2, \ldots, N$. These patterns belong to two classes X_1 and X_2 (X_i has N_i patterns). Y_i is the projection of Class X_i. FLD consists of finding a direction vector $v \in R^d$ such that the projections of the means m_i, $i = 1, 2$ onto the direction v have the largest distance relative to the sum of the projected within-class variance. Mathematically, FLD can be described as follows.

Let m_i be the d-dimensional sample mean given by

$$m_i = \frac{1}{N_i} \sum_{x \in X_i} x \quad . \tag{6.1}$$

Then the sample mean for the projected points is given by

$$\widetilde{m}_i = \frac{1}{N_i} \sum_{x \in X_i} v^t x = v^t m_i \quad . \tag{6.2}$$

It can be seen that $\widetilde{m}_i$ is simply the projection of m_i along v.Define the scatter for the projected samples labeled i by

$$\widetilde{s}_i = \sum_{y \in Y_i} (y - \widetilde{m}_i) \quad , \tag{6.3}$$

where $i = 1, 2$. FLD employs that linear function $v^t x$ for which the fisher criterion function

$$J(v) = \frac{v^t S_B v}{v^t S_w v} \tag{6.4}$$

is maximized, where $S_B = \sum_{i=1}^{2} n_i(m_i - m)(m_i + m)$ is the between-class scatter and $S_B = \sum_{i=1}^{2} S_i$ is the within-class scatter. The optimal projection can be computed by solving the eigenvector problem

$$(S_B - \lambda_i S_w)v_i = 0 \quad , \tag{6.5}$$

where λ_i is the non-zero eigenvalue and v_i is the corresponding eigenvector. The weight vector v which maximizes $J(v)$ can be solved as:

$$v = S_w^{-1}(m_1 - m_2) \quad . \tag{6.6}$$

Cross-talk is originally an English translation of the Chinese term Xiangsheng, meaning a traditional bantering-style dialogue between two comedians rich in puns and allusions. Similarly, the data in the cross-talk table here list the relationship between two classes. To generate the cross-talk table, firstly, we organize all the patterns into $k(k-1)/2$ groups (here k is the number of classes) and each group has the patterns from two classes. Then FLD is applied to each group and the maximum $J(v)$ is recorded in the cross-talk table. We know, the larger $J(v)$, the easier to discriminate the two classes. After the cross-talk table is formed, we will choose and combine those classes that have relatively smaller values from each other in the cross-talk table. $J(v)$ is the Fisher criterion function between two classes and the relatively smaller $J(v)$ implies the two classes are hard to differentiate. If we put the two classes into different combinations, the corresponding distributor module is likely to have relatively large classification error. Thus, we should group them into the same combination. For example, Table 1 shows the cross-talk table for the Vowel problem. See the table 1. The number 2.06 in the column of class 1 and the row of class 2 means that the maximum $J(v)$ between class 1 and class 2 is 2.06, similarity for others.

After the cross-talk table is generated, Cross-talk based Combination Selection (CTCS) algorithm is applied to automatically select the combination set from the

Table 6.1: The cross-talk table for the Vowel problem

class	1	2	3	4	5	6	7	8	9	10	11
1	0	2.06	5.15	8.85	12.15	6.08	6.98	14.68	4.85	3.47	6.15
2	2.06	0	1.70	5.17	6.81	3.08	5.35	14.85	4.91	3.51	3.77
3	5.15	1.70	0	2.75	8.02	3.65	5.46	22.42	8.98	9.08	4.55
4	8.85	5.17	2.75	0	3.36	1.83	4.33	15.11	8.96	15.11	4.59
5	12.15	6.81	8.02	3.36	0	0.78	1.43	9.47	10.72	16.75	5.55
6	6.08	3.08	3.65	1.83	0.78	0	2.18	12.45	7.38	12.97	1.47
7	6.98	5.35	5.46	4.33	1.43	2.18	0	4.22	4.37	9.99	2.12
8	14.68	14.85	22.42	15.11	9.47	12.45	4.22	0	1.65	4.20	9.83
9	4.85	4.91	8.98	8.96	10.72	7.38	4.37	1.65	0	2.35	2.58
10	3.47	3.51	9.08	15.11	16.75	12.97	9.99	4.20	2.35	0	4.70
11	6.15	3.77	4.55	4.59	5.55	1.47	2.12	9.83	2.58	4.70	0

table. In CTCS, the classes which are available for selection are marked as "non-excluded". In each step, the "non-excluded" classes are listed. After that, CTCT selects the class with the minimum $J(v)$ and combines it with its counterpart in $J(v)$. For example, class 4 is selected with (4,6)/1.83. Then class 4 and class 6 will be combined. Of course, the combination is subject to some restriction. We have discussed in the last chapter that the large non-distributor modules often have relatively large classification error. The existence of the large non-distributor modules in a PD network will downgrade the whole network's performance. As we did in Section 5.4.6, we need to restrict the number of classes in a combination. Thus, N_{oc-max}, the maximum number of classes in a combination, is still used in our CTCS algorithm.

The CTCS algorithm is described as follows:

1. Set all the classes as "non-excluded". Find the minimum non-zero value in the table, mark it as T_{min} [1].

2. Set $epoch = 1$.

3. For a single class with the sign "non-excluded", search its column for the class with the minimum value in all the other "non-excluded" classes (except

[1]For example, in Table 6.1, $T_{min} = 0.78$ (between classes 5 and 6).

0). Record it as $(A, B)/value$ (here A is called primary class and B is called assistant class)[2]. If there are no other "non-excluded" classes left, which means all other classes are assigned, set this class as a combination, mark this class as "excluded" and quit.

4. Repeat Step 3 for all the other "non-excluded" classes.

5. List the results in Step 3 and 4 [3].

6. Compare the values in the list with $10T_{min}$. If some value is greater than $10T_{min}$, which means it is relatively easy to discriminate the corresponding class (or primary class) with other classes, set this class as a combination. Then set this class as "excluded" and remove it from the list.

7. Find pairs in the above list, combine them together respectively [4]. Go to Step 9.

8. Find the primary class with the minimum value in the list. Combine this class with the assistant class or the combination which has the assistant class.

9. Update the combination information. If a combination has N_{oc-max} elements, mark all the elements in the combination as "excluded". If there are no "non-excluded" single classes left, quit. Otherwise, set epoch $epoch = epoch + 1$ and go to Step 3.

In the above algorithm, the single class in Step 3 means that class does not belong to any combination. Now we give some explanation to Step 6. For example, in Table 6.1, if class 1, 2, 4, 5, 6, 7, 10, 11 are marked as "excluded", then the list is : (3,9)/9.08, (8, 9)/1.65, (9,8)/1.65, (10,9)/2.35. For class 3, (3,9)/9.08 is greater than . Thus, we set class 3 as combination $\{3\}$, mark it as "excluded" and remove it from the list.

[2]For example, in Table 1, assume all the classes are signed "non-excluded". We search the column of class 1 and find that class 2 with the value 2.06 is the smallest, mark as (1,2)/2.06.

[3]For example, in Table 6.1, if all the classes are signed "non-excluded", the results are listed as follows: (1,2)/2.06, (2,3)/1.70, (3,2)/1.70, (4,6)/1.83, (5,6)/0.78, (6,5)/0.78, (7,5)/1.43, (8,9)/1.65, (9,8)/1.65, (10,9)/2.35, (11,6)/1.37.

[4]In the example in footnote 3, (2,3)/1.70 and (3,2)/1.70, (5,6)/0.78 and (6,5)/0.78, (8,9)/1.65 and (9,8)/1.65 are pairs. Thus, combine {2,3}, {6,5} and {8,9} respectively.

It should be mentioned that after each epoch, the Fisher criterion function $J(v)$ for each class pair will not change. For that $J(v)$ is generated only using the patterns of the two classes and the patterns in other classes is neglected in this course. Thus, combining some classes will not affect the $J(v)$ among the other classes.

It should be mentioned that we do not re-compute $J(v)$ after classes are combined. For example, after classes 2 and 3 are combined into {2,3}, we do not compute $J(v)$ between class 1 and new class {2,3} and still use the old cross-talk table for the next step. The reason is explained as follows.

Using CTCS algorithm, we want to find a combination set whose corresponding distributor module has low classification rate. In other words, the wrongly classified patterns by the distributor module are relatively small. Using CTCS, we select the patterns of classes A and B from a multi-class problem and compute $J(v)$. These patterns from classes A and B form a new 2-class problem. The presupposition for CTCS is that if classes A and B are hard to classify, they should be put together; adding other patterns to the 2-class problem formed by classes A and B will not help to classify the patterns in classes A and B. As we have discussed in Section (5.3), adding unrelated patterns (not belonging classes A and B) normally has negative effect for classifying the patterns between classes A and B. Assume class B is combined with another class C. However, the patterns of class C do not offer useful information to help classify the patterns between classes A and B. If classes A and B are hard to discriminate (or the number of wrongly-classified patterns between A and B is large) before B are combined to C, the wrongly-classified patterns will not reduce after combine B with C. For this reason, we do not recomputed $J(v)$.

An Example for CTCS algorithm:

The descriptions of CTCS algorithm may not be obvious for readers. Here we illustrate with an example to show how CTCS algorithm progresses step by step. It should be noted that this example just show how CTCS works and we don't give the results of the final PD network. We set N_{oc-max} as 3, for $K = 11$ in this problem.

Epoch 1:

List the minimum value $J(v)$ for each single "non-excluded" class with other "non-excluded" classes:

(1,2)/2.06 (2,3)/1.70 (3,2)/1.70 (4,6)/1.83 (5,6)/0.78 (6,5)/0.78
(7,5)/1.43 (8,9)/1.65 (9,8)/1.65 (10,9)/2.35 (11,6)/1.47

In the list, there are three pairs: (2,3)and (3,2), (5,6) and (6,5), (8,9) and (9,8). Combine them together respectively. Thus, we have {2,3}, {5,6}, {8,9}.

Epoch 2:

List the minimum value $J(v)$ for each single "non-excluded" class with other "non-excluded" classes:

(1,2)/2.06 (4,6)/1.83 (7,5)/1.43 (10,9)/2.35 (11,6)/1.47

In the list, (7,5) has the minimum value 1.43. Combine class 7 with combination {5,6}. Thus, {2,3}, {5,6,7}, {8,9}. Because combination {5,6,7} has N_{oc-max} elements, mark class 5, 6, 7 as "excluded".

Epoch 3:

List the minimum value $J(v)$ for each single "non-excluded" class with other "non-excluded" classes:

(1,2)/2.06 (4,3)/2.75 (10,9)/2.35 (11,9)/2.58

In the list, (1,2) has the minimum value 2.06. Combine class 1 with combination {2,3}. Thus, {1,2,3}, {5,6,7}, {8,9}. Because combination {1,2,3} has N_{oc-max} elements, mark class 1,2,3 as "excluded".

Epoch 4:

List the minimum value $J(v)$ for each single "non-excluded" class with other "non-excluded" classes:

(4,10)/3.51 (10,9)/2.35 (11,9)/2.58

In the list, (10,9) has the minimum value 2.35. Combine class 1 with combination {11,9}. Thus, {1,2,3}, {5,6,7}, {8,9,10}. Because combination {8,9,10} has N_{oc-max} elements, mark class 8,9,10 as "excluded".

Epoch 5:

List the minimum value $J(v)$ for each single "non-excluded" class with other "non-excluded" classes:

(4,11)/4.59 (11,4)/4.59

In the list, (4,11) is a pair. Combine them. Thus, {1,2,3}, {5,6,7}, {8,9,10}, {4,11}. Therefore, no single classes which are marked as "non-excluded" left. The final combination set is {1,2,3}, {5,6,7}, {8,9,10}, {4,11}.

6.3 Genetic Algorithm Based Combination Selection (GACS) Method

The basic idea of this method is to find an optimal or near-optimal combination set through evolution.

Genetic Algorithms (GAs) are based on the evolutionary ideas of natural selection and genetics. They are adaptive heuristic search algorithm and represent an intelligent exploitation of a random search used for solving optimization problems (Michalewicz, 1996; Mitchell, 1996; Guan and Zhu, 2003 and 2004). Although randomized, GAs are by no means random. Actually, they direct the search into the region of better performance within the search space through historical information. The fundamental techniques of the GAs are designed to simulate processes in evolutionary biology, specially those following the principles first proposed by Charles Darwin of "survival of the fittest". We know in nature, competition among individuals for scarce resources results in that the fittest individuals dominate over the weaker ones.

Now we introduce our Genetic Algorithm based combination selection (GACS) algorithm for the distributor module. Firstly, we define our chromosome encoding used for evolution. A binary string of specific length is often used to encode a chromosome in canonical genetic algorithms, but it is not suitable in our problems. Thus, we define chromosome according to the following principles. A chromosome

consists of a sequence of combination numbers, wherein each class is encoded with its combination number. The length of a chromosome is equal to the number of classes. Assume chromosome encoding always starts with the smallest class number and increases as follows. For example, 122333 is a chromosome for a 6-class problem. "1" in the first place means class 1 belongs to combination 1. Similarly, number "3" in the fourth place means class 4 belongs to combination 3. The corresponding combination set of this chromosome is {{1}, {2,3}, {4,5,6}}. There is a need for normalization, however. Let us look at another example, chromosome 233111. It is obvious that chromosome 233111 and chromosome 122333 represent the same combination set (though the ordering differs). Therefore, chromosome 233111 can be normalized as 122333.

For convenience, we convert all the chromosomes into a form like 122333. This process is called standardizing the chromosomes. The procedure of standardizing a chromosome is shown as follows:

1. Add a minus sign "-" to all the places. For example, a place with number "3" now becomes "-3". For example, chromosome 233111 becomes (-2)(-3)(-3)(-1)(-1)(-1).

2. Set $t = 1$. Find the number in the first place and find all the places with the same number as the first one. Change the numbers in the first place and all the matching places into "t". In the above example, chromosome (-2)(-3)(-3)(-1)(-1)(-1) becomes 1(-3)(-3)(-1)(-1)(-1).

3. Set $t = t + 1$. Scanning from left to right, find the leftmost place whose number is negative and find all the following places whose number is the same. Change the numbers in these places into "t". In the above example, when t=2, chromosome 1(-3)(-3)(-1)(-1)(-1) becomes 122(-1)(-1)(-1).

4. Repeat Step 3 until all the places have positive numbers inside.

Secondly, we create an initial population of chromosomes. After generating the initial population, each chromosome is evaluated and assigned a fitness value. Here we use a simple neural network [5]. For the evaluation and use the classification error

[5]Here the simple neural network means an ordinary construction backpropagation neural network, no task decomposition methods are applied in this network.

f of the validation data set to calculate the fitness:

$$fitness = 2 - \frac{f}{f_{avg}} \quad , \tag{6.7}$$

where f_{avg} is the average of classification errors based on the validation data set for all the chromosomes in the population. f is also called evaluation value. If $2 - \frac{f}{f_{avg}}$ is smaller than 0, set $fitness = 0$.

The execution of our genetic algorithm can be viewed as a two-stage process. It starts with the current population. Then selection is applied to the current population to generate an intermediate population. After that, mutation and crossover are applied to the intermediate population to create the next population. We use "stochastic universal sampling" to form the intermediate population (Baker, 1987). Assume that the population is laid out in random order as in a pie graph in which each individual is assigned space on the pie graph in proportion to $fitness$. Next an outer roulette wheel is placed around the pie with N equally spaced pointers (N is the number of the population). A single spin of the roulette wheel will now simultaneously pick all N members of the intermediate population.

After the construction of the intermediate population, crossover and mutation are used to generate the next population. Crossover is applied to randomly paired chromosomes with a probability p_c. It must be noted that p_c can be also set as other values. Consider two chromosomes: 112233 and 122123. The random crossover point is chosen, for example, after the 4^{th} place. Then the numbers in the 5^{th} and 6^{th} places are exchanged and new chromosomes are formed. Here the new chromosomes are 112223 and 122133. After crossover, mutation is applied to random chromosomes with a probability p_m. After a chromosome is selected for mutation, a place is randomly selected for mutation and the number in that place is randomly chosen. After the crossover and mutation is complete, standardize the chromosomes. Then the next population is evaluated and becomes the current population. Then the above process is repeated. In our experiments, p_c and p_m should not be over 0.5. If p_c and p_m are too large, the good chromosomes from previous generation can be hardly kept and the number of computations to find the best chromosome will increase. We set $p_c = 0.2$ and $p_m = 0.3$. It should be noted that the values of these parameters can be changed.

Like that in CTCS and RGCS algorithm, N_{oc-max}, the maximum number of classes in a combination is still used in our GACS algorithm to avoiding the appearance of large non-distributor modules. Using this parameter, we kick out some chromosomes directly. For a 6-class problem, if we choose $N_{oc-max} = 3$, then chromosome 121112 will be eliminated, because combination 1,3,4,5 has four classes.

6.4 Experimental Results for the PDs Using CTCS and GACS Algorithms

We present the experimental results for three data sets here. For each data set, single-layer PD networks are created using the CTCS algorithm and GACS method.

A. Vowel

In this problem, the number of classes is 11. As that in the last chapter, we set $N_{oc-max} = 3$ for this problem. Using CTCS, The PD network is composed with one distributor module and four non-distributor modules. The distributor module has four outputs, {1,2,3}, {5,6,7}, {8,9,10} and {4,11}. The experimental results of the single-layer PD network using CTCS algorithm for the Vowel data are listed in Table 6.2. We can see that the overall classification error of the network is 15.3846% for CTCS.

For the GACS algorithm, same as that in CTCS algorithm, we set $N_{oc-max} = 3$. The population number is 20. Due to long computation time, only 40 generations were bred in our experiments. We identified the best chromosome 12234355513, which appeared in Generation 23. The PD network is composed with one distributor module and four non-distributor modules, which is the same as that using RGCS. The distributor module has five outputs, {4,6,11}, {7,8,9}, {1, 10}, {2,3} and {5}. Four non-distributor modules are used to recognize {4,6,11}, {7,8,9}, {1,10} and {2,3} respectively. The experimental results are shown in Table 6.3. We can see that the overall classification error of the network is 12.2672% for GACS.

B. Segmentation

Table 6.2: Results using CTCS and GACS for the Vowel problem

Method	Network	Training time (s)	Hidden Units	Indep. Param.	C.error (%)
CTCS	Distributor Module	73.75 (in parallel) 190.5 (in series)	67.65	855.8	8.0567
	Overall Network	73.75 (in parallel) 277.95 (in series)	246.55	3123.6	15.3846
GACS	Distributor Module	94.25 (in parallel) 313.95 (in series)	87.7	1107.4	5.8502
	Overall Network	94.25 (in parallel) 471.3 (in series)	155.8	2045.6	12.2672

Table 6.3: The cross-talk table for the Segmentation problem

Class	1	2	3	4	5	6	7
1	0	232.9726	9.9478	17.6629	8.8757	40.3356	53.9114
2	232.9726	0	72.5262	17.1233	53.0981	42.9524	518.7841
3	9.9478	72.5262	0	6.7673	1.6476	18.5474	40.6585
4	17.6629	17.1233	6.7673	0	3.0618	4.4578	54.7033
5	8.8757	53.0981	1.6476	3.0618	0	16.3379	29.0228
6	40.3356	42.9524	18.5474	4.4578	16.3379	0	60.9468
7	53.9114	518.7841	40.6585	54.7033	29.0228	60.9468	0

Table 6.4: Results using CTCS and GACS for the Segmenation problem

Method	Network	Training time (s)	Hidden Units	Indep. Param.	C.error (%)
CTCS	Distributor Module	596.7 (in parallel) 1265.05 (in series)	142.1	2937	2.0017
	Overall Network	596.7 (in parallel) 1294.15 (in series)	172.9	3648	4.3934
GACS	Distributor Module	261.5 (in parallel) 589.5 (in series)	62.15	1433	1.0225
	Overall Network	261.5 (in parallel) 875.35 (in series)	190.65	3003	3.4662

Table 6.3 shows the cross-talk table for the Segmentation problem. In the problem, the number of classes is 7. Thus, we set $N_{oc-max} = 2$. We set up the PD network using CTCS algorithm. The network is composed with one distributor module and two non-distributor module. The distributor module has five outputs, {3,5},{4,6},{1},{2} and {7}. The non-distributor modules are used to recognize {3,5},{4,6}.

Now we build the network using GACS method. We set $N_{oc-max} = 2$, which is the same as that using CTCS algorithm. The population number is 20. Due to long computation time, only 40 generations were bred in our experiments. We identified the best chromosome 1231342, which appears at Generation 13. The PD network is composed with one distributor module and three non-distributor modules. The distributor module has four outputs, {1,4},{2,7},{3,5} and {6}. Three non-distributor modules are used to recognize {1,4},{2,7} and {3,5} respectively.

Table 6.4 shows the experimental results of the single-layer PD network using the CTCS and GACS algorithm for this data. We can see that the overall classification

Table 6.5: The cross-talk table for the Pen-Based Recognition problem

class	1	2	3	4	5	6	7	8	9	10
1	0	30.91	59.15	73.45	22.12	13.70	18.57	39.74	5.99	27.61
2	30.91	0	7.07	10.88	18.75	10.49	22.73	8.23	17.77	6.68
3	59.15	7.07	0	83.34	37.33	86.04	45.81	10.73	31.77	39.58
4	73.45	10.88	83.34	0	35.39	9.46	48.03	11.60	22.43	6.80
5	22.12	18.75	37.33	35.39	0	19.07	13.67	22.31	38.44	9.93
6	13.70	10.49	86.04	9.46	19.07	0	15.99	10.82	5.45	3.80
7	18.57	22.73	45.81	48.03	13.67	15.99	0	34.88	15.25	38.59
8	39.74	8.23	10.73	11.60	22.31	10.82	34.88	0	12.49	15.21
9	5.99	17.77	31.77	22.43	38.44	5.45	15.25	12.49	0	11.46
10	27.61	6.68	39.58	6.80	9.93	3.80	38.59	15.21	11.46	0

error of the network is 4.3934% for CTCS and 3.4662% for GACS.

C. Pen-Based Recognition

Now we use the CTCS to set up the network. The cross-talk table for this problem is shown on Table 6.5. In the problem, for the number of classes is 10, we set $N_{oc-max} = 3$. The PD network is composed with one distributor module and three non-distributor modules. The distributor module has four outputs, {1,5,7}, {2,3,8}, {6,9,10} and {4}. Three non-distributor modules are used to recognize {1,5,7}, {2,3,8} and {6,9,10} respectively.

Now we construct the network using GACS method. Same as that in CTCS algorithm, we set $N_{oc-max} = 3$. The population number is 20. Due to long computation time, only 40 generations were bred in our experiments. We identified the best chromosome 1222345441, which appears at Generation 19. The PD network is composed with one distributor module and four non-distributor modules. The distributor module has four outputs, {2,3,4}, {6,8,9}, {1,10} and {5,7}. Four non-distributor modules are used to recognize {2,3,4}, {6,8,9}, {1,10} and {5,7} respectively.

Table 6.6 shows the experimental results of the single-layer PD network using the CTCS and GACS algorithm for this data. We can see that the overall classification error of the network is 1.1633% for CTCS and 1.2060% for GACS.

Table 6.6: Results using CTCS and GACS for the Pen-Based Recognition problem

Method	Network	Training time (s)	Hidden Units	Indep. Param.	C.error (%)
CTCS	Distributor Module	667 (in parallel) 1998.75 (in series)	78.1	1469.8	0.6990
	Overall Network	667 (in parallel) 2737.9 (in series)	273.2	5155.6	1.1633
GACS	Distributor Module	1089.5 (in parallel) 2911.15 (in series)	99.7	1862.6	0.5630
	Overall Network	1089.5 (in parallel) 3607.75 (in series)	335	6268	1.2060

6.5 Validation of the $\sqrt{K}$ Rule-of-thumb

In Section 5.46 of the last chapter, we deduced the $\sqrt{K}$ Rule-of-thumb from theoretical analysis. The $\sqrt{K}$ Rule-of-thumb shows that the maximum number of classes in a non-distributor module should be close to $\sqrt{K}$, where K is the number of classes of the data set, the single-layer PD network will have the best performance. This rule is used in our RGCS, CTCS and GACS methods and serves for the setting of N_{oc-max}, the maximum number of classes in a non-distributor module. Though we have deduced this rule from analysis, one may still have reservation about the validity of this rule when it is applied to real problems. Here we verify the validity of this rule from experiments.

Here we illustrate with an example how the $\sqrt{K}$ Rule works. For this objective, we conducted a set of experiments. In the experiments, we changed N_{oc-max}, and applied it to RGCS algorithm and CTCS algorithm. Due to the long computation

time, we were not able to repeat the experiments using GACS algorithm.

See the Vowel problem. The data set has 11 classes. The results are listed in Table 6.7.

Table 6.7 shows the network performance changes with the value of N_{oc-max} parameter. When $N_{oc-max} = 1$, it means the each output of the distributor module only has one class. The distributor module can classify all the test patterns into their corresponding classes and no non-distributor modules are needed to further classify the patterns. The network is the same as that of the fully partitioned Output Parallelism network. From the table, it also can be seen that for RGCS algorithm, when N_{oc-max} is 5 or 6, the attained classification set is the same as that when N_{oc-max} is 4 and for CTCS algorithm, when N_{oc-max} is 6, the deduced classification set is the same as that when N_{oc-max} is 5. The reason for this is that our RGCS algorithm and CTCS algorithm are designed to find the combination set with a relatively low classification error, not to find the combination set whose max combination has N_{oc-max} elements. N_{oc-max} is just used to restrict the maximum number of elements in a combination. It is possible that no combinations having N_{oc-max} elements exist according to our algorithms. In Table 6.10, we just list the results to $N_{oc-max} = 7$. When $N_{oc-max} > 7$, for the CTCS algorithm, the obtained combination set is the same as $N_{oc-max} = 5$, and for the RGCS algorithm, the obtained combination set is the same as $N_{oc-max} = 7$. For the convenience of readers, we plot the classification errors of the two algorithms in Figure 6.1 and Figure 6.2.

From Figures 6.1 and 6.2, it can be seen that with the increase of N_{oc-max}, the classification errors of the distributor module in both figures decrease. The reason for this is that when N_{oc-max} becomes large, the maximum number of classes in a non-distributor module increases. Thus, the restriction from the non-distributor becomes loose. When we use RGCS algorithm or CTCS algorithm to search a suitable combination set, we need not always worry that the combination reaches its maximum number of classes. Therefore, it is easy to find a combination set whose corresponding distributor module has a relatively low classification error. Figure 6.1 and 6.2 also show that for the whole networks in both figures, the classification errors

Table 6.7: The network performance with the change of N_{oc-max} for the Vowel problem

N_{oc-max}	Method	Combination set	Hidden Units	Indep. Param.	C.error of distributor module(%)	C.error of whole network(%)
1	CTCS	{{1},{2},{3},{4}, {5},{6},{7},{8}, {9},{10},{11}}	184.4	2333.8	25.5466	25.5466
	RGCS	{{1},{2},{3},{4}, {5},{6},{7},{8}, {9},{10},{11}}	184.4	2333.8	25.5466	25.5466
2	CTCS	{{1,10},{2,3}, {5,6},{7,11}, {8,9},{4}}	242.65	3087.8	12.1012	15.3603
	RGCS	{{1,10},{2,3}, {4,6},{8,9}, {5},{7},{11}}	259.4	3277.8	12.0445	14.6154
3	CTCS	{{1,2,3},{5,6,7}, {8,9,10}, {4,11}}	246.55	3123.6	8.0567	15.3846
	RGCS	{{4,6,11},{7,8,9}, {1,10},{2,3}, {5}}	155.8	2045.6	5.8502	12.2672
4	CTCS	{{1,2,3,4}, {5,6,7,11}, {8,9,10}}	222.7	2826.4	6.0121	17.9352
	RGCS	{{1,2,3,10}, {4,5,6,11}, {7,8,9}}	208.2	2652.4	2.9960	16.1538
5	CTCS	{{1,2,3}, {4,5,6,7,11}, {8,9,10}}	241.65	3053.8	3.9474	20.5870
	RGCS	Same as RGCS $N_{oc-max} = 4$				
6	CTCS	Same as CTCS $N_{oc-max} = 5$				
	RGCS	Same as RGCS $N_{oc-max} = 4$				
7	CTCS	Same as CTCS $N_{oc-max} = 5$				
	RGCS	{{4,5,6,7,8,9,11}, {1,2,3,10}}	208.4	2643.8	1.0324	19.6559

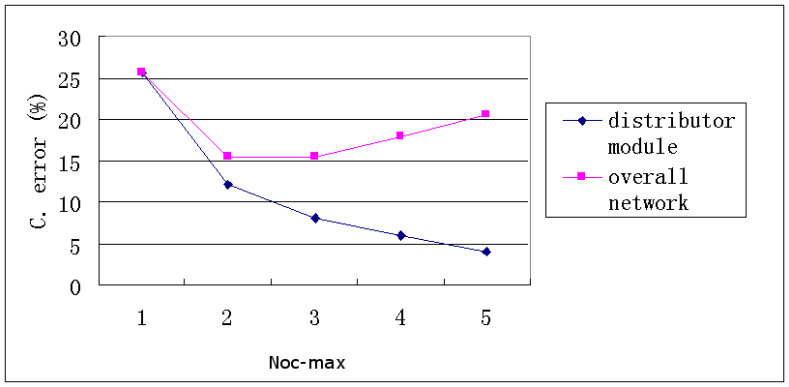

Figure 6.1: The relation between classification error and N_{oc-max} using CTCS algorithm for the Vowel problem

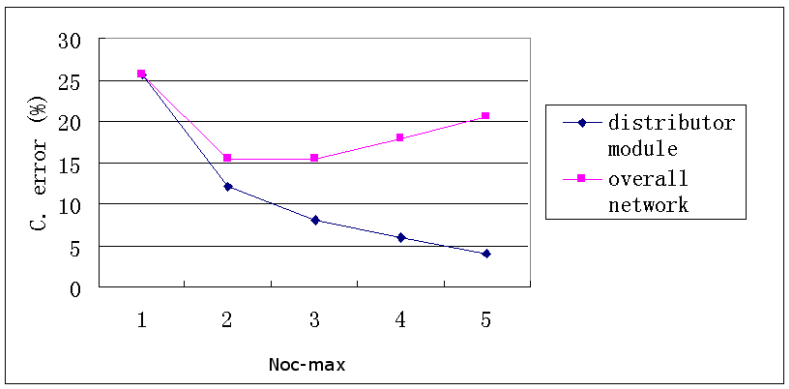

Figure 6.2: The relation between classification error and N_{oc-max} using RGCS algorithm for the Vowel problem

fall first and after that, the classification errors rise. Using RGCS algorithm, the lowest classification error of the whole network 12.2672% appears at $N_{oc-max} = 3$. Using CTCS algorithm, the trough appears at $N_{oc-max} = 2$ and 3, whose classification errors are 15.3603% and 15.3846% respectively. From the two figures, we can see that in order to let the network has relatively low classification error, $N_{oc-max} = 3$ is a good choice. Because the Vowel data set has 11 classes, $\sqrt{11} \approx 3.32$. Thus,$\sqrt{K}$ Rule works for this problem.

From the above example, we can see that the $\sqrt{K}$ Rule is a suitable choice indeed in setting the N_{oc-max} parameter.

6.6 Comparison of the Combination Selection Algorithms

In the last chapter and this chapter, we presented five combination selection methods for the distributor module, namely GCS, SGCS, RGCS, CTCS and GACS. The former three methods are based on greedy methods. In them, GCS and SGCS are designed for multi-layer PD networks and RGCS is designed for single-layer PD networks. The two combination methods which are introduced in this chapter, CTCS and GACS, are also designed for single-layer PD networks. All the three algorithms for single-layer PD networks, RGCS, CTCS and GACS, need an important parameter, N_{oc-max}, which represents the maximum number of original classes in a combination. The $\sqrt{K}$ Rule (K is the number of classes in the problem) is used for the setting of N_{oc-max} parameter.

Now compare the time consumed by these methods. In CTCS algorithm, we apply FLD to project the features to a one-dimensional space, use the obtained information to form a cross-talk table and find a suitable combination set based on the cross-talk table. In the whole process, we do not need to compute the classification errors of combinations or combination sets (please see Definition 5.2 for the classification error of a combination and Definition 5.3 for the classification error of a combination set). While in greedy based combination selection algorithms, we need to compute the classification errors of combinations and in GACS method, we

need to find the classification errors of combination sets. In our actual simulation, the time spent mainly on the computation of classification errors. Thus, compared with that of the other algorithms, the computation time of CTCS algorithm can be neglected.

For the other four algorithms, we need to compute the classification errors in each epoch. We have mentioned earlier that in GCS, SGCS and RGCS algorithms, we compute the classification errors of combinations and in GACS method, we compute the classification errors of combination sets. There is some difference between the classification error of a combination and that of a combination set. The neural network which is used to evaluate the classification error of a combination only has two outputs, while the neural network which is used to evaluate the classification error of a combination set normally has two or more outputs. In average, computing the classification error of a combination set normally need to spend more time than computing the classification error of a combination. However, for the convenience of comparison, we regard the time of computing classification errors as the same.

Definition 6.1—Time unit: The average time of computing a classification error when we use GCS, SGCS, RGCS or GACS algorithm to find a suitable combination set for a data set.

Normally, GACS method needs to spend much more time units than GCS, SGCS and RGCS algorithms. SGCS is a simplification of GCS algorithm. It needs not compute as many as classification errors as GCS algorithm, so SGCS algorithm spends less time units than GCS algorithm. SGCS and GCS algorithms are designed for multi-layer PD networks. To solve a problem, SGCS and GCS algorithms are used several times for the distributor modules in different layers. RGCS algorithm is designed for single-layer PD networks. Thus, for a data set, the time units spent using RGCS algorithm is normally less than that using GCS or SGCS algorithm.

Now we compare the performance of the PD networks using the five algorithms.

A. Vowel

Table 6.8: Results of the different combination selection methods for the Vowel problem

Method	Combination set	Time Units	Hidden Units	Indep. Param.	C.error of distributor module (%)	C.error of whole network (%)
GCS	1st lvl: {{4,5,6,7,8,9,11}, {1,2,3,10}} 2nd lvl 1: {{4,5,6,11}, {7,8,9}} 2nd lvl 2: {{1,10},{2,3}} 3rd lvl:{{5,6,11},{4}} 4th lvl: {{6,11},{5}}	115	332.85	4137.2	–	11.6194
SGCS	Same as GCS	115	332.85	4137.2	–	11.6194
RGCS	{{4,6,11},{7,8,9}, {1,10},{2,3},{5}}	54	155.8	2045.6	5.8502	12.2672
CTCS	{{1,2,3},{5,6,7}, {8,9,10},{4,11}}	0	246.55	3123.6	8.0567	15.3846
GACS	{{4,6,11},{7,8,9}, {1,10},{2,3},{5}}	54	155.8	2045.6	5.8502	12.2672
Arbitrary Selection 1	{{1,2,3},{4,5,6,7}, {8,9,10,11}}	0	229.4	2955.8	6.6802	18.7045
Arbitrary Selection 2	{{1,2,3},{4,5,6}, {7,8,9},{10,11}}	0	190.1	2446.2	7.3041	15.5466
Arbitrary Selection 3	1st lvl:{{1,2,3,4,5,6}, {7,8,9,10,11}} 2nd lvl 1:{{1,2,3},{4,5,6}} 2nd lvl 2:{{7,8,9},{10,11}}	0	217.9	2801.8	–	15.4251

Table 6.9: Improvement of the combination selection methods for the Vowel problem

Method	Avg. of arbitrary of selection	GCS	SGCS	RGCS	CTCS	GACS
C. error.(%)	16.5587	11.6194	11.6194	12.2672	15.3846	12.2672
Improvment compared to arbitrary selection	–	4.9393	4.9393	4.2915	1.1741	4.2915

Table 6.8 shows the performance of the PD networks using different combination selection algorithms, including three arbitrary PD networks, for the Vowel problem. Table 6.9 shows the improvement of the classification errors of these methods compared with arbitrarily selected PD networks. The three arbitrarily selected PD networks include two single-layer PD networks and one multi-layer PD network. Because the multi-layer PD networks have several distributor modules, we do not list the classification errors of these distributor modules. According to the $\sqrt{K}$ rule, the N_{oc-max}parameter for the PD networks using RGCS, CTCS and GACS algorithms is set at 3. For the convenience of comparison, we set $N_{oc-max} = 3$ for the 2^{nd} arbitrary selected PD network.

We can see that the networks using GCS and SGCS algorithm have the lowest classification error in all the eight PD networks, whose classification error is 11.6194%. The average classification error of the three arbitrarily selected networks is 16.5587%. Compared with those of arbitrarily selected networks, the classification accuracy of GCS (or SGCS) increases 5 percent and the classification error decreases 1/3 compared with the arbitrary selected networks. It is a significant improvement.

Now we focus on the five single-layer PD networks. Compared with multi-layer PD networks, the single-layer PD network has the shorter processing time for unknown patterns. In the five networks, four networks have the parameter $N_{oc-max} = 3$, namely RGCS, CTCS, GACS and Arbitrary selection 2. It can be seen that network using RGCS is the same as that using GACS. RGCS and GACS achieved lower classification error than the other two networks, both for the distributor module and the whole network. The performance of the network using CTCS is close to that of the arbitrary selected network 2. We can also note that the classification error of the distributor module in Arbitrary selection 1 is smaller than CTCS. The reason for this is that $N_{oc-max} = 3$ in Arbitrary selection 1 while $N_{oc-max} = 3$ in CTCS.

Then we compared the time consumed for the five methods. It can be seen that GACS spend 800 time units, which is much greater than the other four methods. The time spent by CTCS can be neglected compared with the other methods. Com-

pared with GACS, RGCS achieve the same results, but only spend 54 time units. GCS uses 115 time units to find it network, while its simplified version, SGCS uses 89 time units.

In general, SGCS and RGCS have the best performance for this problem. For multi-layer networks, SGCS achieves the minimum classification error with relatively short computation time. For single-layer networks, RGCS attains the best classification rate and a relatively short consumed time.

B. Segmentation

The experimental results for the Segmentation problem using different combination selection algorithms are shown on Tables 6.10 and 6.11. Same as those in the Vowel problem, the three arbitrary selected PD networks include two single-layer PD networks and one multi-layer PD network. We do not list the classification errors of these distributor modules for multi-layer PD networks. According to the $\sqrt{K}$ rule, the N_{oc-max} parameter is 2 for RGCS, CTCS and GACS. For the convenience of comparison, $N_{oc-max} = 2$ for the 2^{nd} arbitrary selected PD network.

It can be seen that the networks using RGCS algorithm and GACS algorithm are the same and have the lowest classification error in all the eight PD networks, which is 3.4662%. The average classification error of the three arbitrary selected networks is 4.6736%. The classification error of RGCS (or GACS) decreases 1.2% compared with those of arbitrary selected networks. It is a great progress.

Now we focus on the five single-layer PD networks. Compared with multi-layer PD networks, the single-layer PD network has the shorter processing time for unknown patterns. In the five networks, four networks, RGCS, CTCS, GACS and Arbitrary selection 2, have the parameter $N_{oc-max} = 2$. RGCS and GACS achieve lower classification error than the other two networks, both for the distributor module and the whole network. The overall classification error of the network using CTCS is lower than the classification errors of the arbitrarily selected networks. It can be also noted that the classification error of the distributor module in Arbitrary selection 1 is smaller than CTCS. The reason for this is that $N_{oc-max} = 4$ in Arbi-

Table 6.10: Results of the different combination selection methods for the Segmentation problem

Method	Combination set	Time Units	Hidden Units	Indep. Param.	C.error of distributor module (%)	C.error of whole network (%)
GCS	1st lvl: {{1,3,4,5}, {2,7},{6}} 2nd lvl: {{3,4,5},{1}} 3rd lvl: {{3,5},{4}}	39	92.05	2050	–	3.8821
SGCS	1st lvl: {{1,3,4,5}, {6,7},{2}} 2nd lvl: {{3,4,5},{1}} 3rd lvl: {{3,5},{4}}	23	160.45	3418	–	3.8735
RGCS	{{1,4}, {2,7},{3,5}, {6}}	19	140.65	3003	1.0225	3.4662
CTCS	{{3,5},{4,6},{1}, {2},{7}}	0	172.9	3648	2.0017	4.3934
GACS	{{1,4},{2,7},{3,5}, {6}}	800	140.65	3003	1.0225	3.4662
Arbitrary Selection 1	{{3,4,5},{1,2,6,7}}	0	128.9	2768	1.0399	4.6101
Arbitrary Selection 2	{{1},{2,3},{4,5}, {6,7}}	0	128.5	2760	4.0035	4.7660
Arbitrary Selection 3	1st lvl: {{1,2,3}, {4,5,6,7}} 2nd lvl 1: {{1}, {2,3}}} 2nd lvl 2: {{4,5},{6,7}}	0	165.8	3506	–	4.6447

Table 6.11: Improvement of the combination selection methods for the Segmentation problem

Method	Avg. of arbitrary of selection	GCS	SGCS	RGCS	CTCS	GACS
C. error.(%)	4.6736	3.8821	3.8735	3.4662	4.3934	3.4662
Improvment compared to arbitrary selection	–	0.7915	0.8001	1.2074	0.2802	1.2074

trary selection 1 while $N_{oc-max} = 2$ in CTCS.

For the multi-layer PD networks, we can see that GCS and SGCS find the similar network structure. The overall classification errors of the two networks are also very close (3.8821% and 3.8735%), which is much better than the classification error of the arbitrarily selected multi-layer network (4.6447%).

Then we compared the time consumed for the five methods. It can be seen that GACS spend 800 time units, which is much greater than the other four methods. The time spent by CTCS can be neglected compared with the other methods. Compared with GACS, RGCS achieved the same results, but only spend 19 time units. GCS uses 39 time units to find it network, while its simplified version, SGCS uses 23 time units.

In general, RGCS has the best performance for this problem. RGCS attains the best classification rate and a relatively short consumed time.

C. Pen-Based Recognition

Table 6.12 shows the performance of the PD networks using different combination selection algorithms, including three arbitrary PD networks, for the Pen-based Recognition data set. Table 6.13 shows the improvement of the classification errors of these methods compared with arbitrarily selected PD networks. The three arbitrary selected PD networks include two single-layer PD networks and one multi-layer PD network. As those in the last two examples, we don't list the classification errors of these distributor modules. According to the $\sqrt{K}$ rule, the N_{oc-max} parameter for the PD networks using RGCS, CTCS and GACS algorithms is set to be 3. For the convenience of comparison, we set $N_{oc-max} = 3$ for the 2^{nd} arbitrary selected PD network.

We can see that the networks using GCS and SGCS algorithm have the lowest classification error in all the eight PD networks, whose classification error is 1.0058%. The average classification error of the three arbitrarily selected networks is 2.9304%. Compared with those of arbitrarily selected networks, the classification accuracy of

Table 6.12: Results of the different combination selection methods for the Pen-Based Recognition problem

Method	Combination set	Time Units	Hidden Units	Indep. Param.	C.error of distributor module (%)	C.error of whole network (%)
GCS	1st lvl:{{1},{3},{4},{5},{8},{7,9},{2,6,10}} 2nd lvl: {{6,10},{2}}	60	413.35	7661.3	–	1.0058
SGCS	Same as GCS	46	413.35	7661.3	–	1.0058
RGCS	{{1},{3},{4},{5},{8},{7,9},{6,2,10}}	48	299.65	5597.7	0.7951	1.0592
CTCS	{{2,3,4},{6,8,9},{1,10},{5,7}}	0	273.2	5155.6	0.6990	1.1633
GACS	{{1,4},{2,7},{3,5},{6}}	800	335	6268	0.5630	1.2060
Arbitrary Selection 1	{{1,2,3,4,5},{6,7,8,9,10}}	0	287.45	5378.1	0.6190	5.0453
Arbitrary Selection 2	{{1,2,3},{6,9,10},{4,7},{5,8}}	0	258.8	4896.4	1.2540	1.8757
Arbitrary Selection 3	1st lvl: {{1,2,3,4,5},{6,7,8,9,10}} 2nd lvl 1:{{1,2,3},{4,5}} 2nd lvl 2:{{8,9,10},{6,7}}	0	311.95	5887.1	–	1.8703

Table 6.13: Improvement of the combination selection methods for the Pen-Based Recognition problem

Method	Avg. of arbitrary of selection	GCS	SGCS	RGCS	CTCS	GACS
C. error.(%)	2.9304	1.0058	1.0058	1.0592	1.1633	1.2060
Improvment compared to arbitrary selection	–	1.9246	1.9246	1.8712	1.7671	1.7244

GCS (or SGCS) increases about 1.9 percent and the classification error decreases 2/3 compared with arbitrary selected networks. That is a great improvement.

Now we focus on the five single-layer PD networks. Compared with multi-layer PD networks, the single-layer PD network has the shorter processing time for unknown patterns. In the five networks, four networks have the parameter $N_{oc-max} = 3$, namely RGCS, CTCS, GACS and Arbitrary selection 2. We can see that the distributor module of the network using GACS has the lowest classification error (0.5630%) while the whole network using RGCS has the lowest classification error (1.0592%). GACS has the best performance in the distributor module, but not in the whole network. The reason for this is that in the RGCS network, there are only two non-distributor modules, while in the GACS network, there are four non-distributor modules. The wrongly classified test patterns by the non-distributor modules of the GACS network are larger than those of the RGCS network. Thus, RGCS achieve higher classification accuracy in this example. The overall classification error of the CTCS network is a little better than the GACS network. It can be also noted that the classification error of the distributor module in Arbitrary selection 1 is smaller than CTCS and RGCS. The reason for this is that $N_{oc-max} = 5$ in Arbitrary selection 1 while $N_{oc-max} = 3$ in CTCS and RGCS.

Then we compared the time consumed for the five methods. It is obvious that GACS spend 800 time units, which is much greater than the other four methods. The time spent by CTCS can be neglected compared with the other methods. Compared with GACS, RGCS achieve the same results, but only spend 48 time units. GCS uses 60 time units to find it network, while its simplified version, SGCS uses 46 time units.

In general, SGCS and RGCS have the best performance for this problem. For multi-layer networks, SGCS achieves the minimum classification error with relatively short computation time. For single-layer networks, RGCS attains the best classification rate and a relatively short consumed time.

Table 6.14: Comparison to related work

Data set	GCS	SGCS	RGCS	CTCS	GACS	ILOA	HICL	PD
Vowel	11.6194	11.6194	12.2672	15.3846	12.2672	22.57	–	24.355
Segmentation	3.8821	3.8735	3.4662	4.3934	3.4662	–	3.6048	–
Pen-Based Recognition	1.0058	1.0058	1.0592	1.1633	1.2060	–	–	–

6.7 Discussion and Conclusions

Here we introduced two combination methods, CTCS and GACS and compared the combination selection methods in the last and this chapter. CTCS is the fastest method to find a suitable combination set. CTCS utilizes FLD to project the information of feature space into two-dimensional spaces and generate a cross-talk table. Then a combination set is generated based on the cross-talk table. Because some useful information will be lost when we project the feature space from higher dimension to lower dimension, CTCS can hardly find the best combination set. From the experiments, we can see this fact. GACS method try to find an optimum or near-optimum combination set through evolution. It spends more time compared with other methods. From the experimental results, we can see that GACS can always find the combination set for each problem whose distributor module has the smallest classification error.

We have compared our results to the feature selection results reported in the literature such as ILOA (Guan and Thong, 2003), HICL (Guan and Li, 2002b), and PD (Guan and Li, 2002a). For the Vowel data, our PD results are better than the results reported as shown in Table 6.14. For the Segmentation data, our PD results are consistent with the results reported. It should be mentioned that the comparison of the error rates obtained by different methods in Table 6.14 may not be precise (or fair), since the results achieved by different algorithms have not been obtained using the same experimental procedure, network structures and training methods.

From the experiments, we can see that the most successful combination selection algorithm is RGCS for single-layer PD network. For the Vowel and Segmentation problem, it finds the same combination set as GACS and spends much shorter time than GACS. For the Pen-based Recognition problem, though the distributor module's classification error is larger than GACS, the whole networks' classification error is smaller than GACS. Even compared with GCS and SGCS, RGCS is also a good choice. There are three reasons. The first is that the single-layer PD network has shorter processing time when detect unknown patterns compared with multi-layer PD networks. The second is that RGCS often spend less time to find the combination set compared to GCS or SGCS. The third is that the classification error of the final network using RGCS is close to that using GCS or SGCS.

Chapter 7

Conclusions

The main purpose of this thesis is to design neural network classifiers for multi-class problems. A new type of modular neural networks called Pattern Distributor network was proposed. The PD method uses a hierarchical structure to classify unseen patterns. There are two types of PD networks, single-layer PD networks and multi-layer PD networks. In single-layer PD networks, only one distributor module is used. Unknown patterns are first recognized by the distributor module before they are presented to the other modules. We presented a theoretical model to compare the classification rate of a PD network with that of an OP network - a typical class decomposition network developed by Guan and Li (2000 and 2002). The theoretical analysis showed that the PD method can achieve better classification accuracy than the OP method once the performance of the distributor modules is guaranteed. The experiments confirmed our analysis. Multi-layer PD networks are extensions of single-layer PD networks. They can be regarded as using the PD method on non-distributor modules in single-layer PD networks. There are several distributor modules in multi-layer PD networks. Multi-layer PD networks can achieve a better classification rate than the corresponding single-layer PD networks when the performance of those distributor modules is ensured.

How to design highly performing distributor modules is a critical issue in the design of PD networks. Only when the recognition rate of distributor modules is guaranteed do the PD networks have higher classification accuracy than ordinary class decomposition networks. In chapter 5, we presented several theorems and corollaries for class combination in the distributor modules and for the relations in the non-distributor modules. Based on these theorems and corollaries, we pro-

posed three combination algorithms to find good combination sets for the distributor module. The first one is called GCS, the basic idea of which is based on the greedy method (which works by making the decision that seems most promising at any moment). GCS starts from the class with the largest classification error and combines that class with other classes one by one to find a premier combination set. The second one is SGCS algorithm, which is designed to reduce the computation effort of GCS algorithm by considering the border relationship among classes. GCS and SGCS are mainly used in multi-layer PD networks. To customise GCS for single-layer PD networks, we deduced the $\sqrt{K}$ rule-of-thumb (K is the number of classes in the problem), according to which, the maximum number of classes in a non-distributor modules should not exceed $\sqrt{K}$. Based on this rule, we proposed the third algorithm RGCS, the basic idea of which is similar to that of GCS. Restriction of the maximum number of classes in a combination (or an output of the distributor module) is added so as to balance the size of the following non-distributor modules. These three algorithms could find suitable combinations for the distributor module, for they are all based on the greedy method.

In Chapter 6, we presented two combination selection algorithms, CTCS and GACS. The $\sqrt{K}$ rule is used in both algorithms. The basic idea behind CTCS is to find classes which are close in the feature space and combine them together. In CTCS, a Cross-talk table is formed using FLD methods, then the combination is found from the cross-talk table. GACS is a Genetic Algorithm (GA) based algorithm, which finds a suitable combination set through evolution.

We compared the results of the five combination selection algorithms. CTCS is the fastest algorithm, yet the final classification accuracies for data sets in our experiments were not as good as the other algorithms. GACS could find the optimal combination set for the distributor module, but a drawback for GACS is that more computation is needed than the other algorithms. From the experiments, RGCS algorithm has the best performance in all the algorithms. It can attain similar classification accuracies as GACS, while the computation time is much smaller than GACS.

Our research is of significant in that our PD method has good performance for multi-class problems. We wish our study could enlighten other researchers to carry on with further research on better classifiers for multi-class problems.

Future work

There are several directions for future research. The first is the combination methods in the distributor module. Though several combination algorithms have been designed for distributor modules, these combination algorithms have shortcomings. GACS is time-consuming. Though in our experiments, RGCS found the combination set as good as GACS, we can not guarantee the greedy based algorithms can always find the optimum combination. Because greedy algorithms are shortsighted methods which make the decision that seems most promising at present, it is possible that CTCS may not find the premium or near-premium combination sets for distributor modules. How to find an effective and low-computation-effort combination algorithm is still a subject in our future work.

A second study that can be done in the future is further exploration on Reduced Pattern Training. Reduced Pattern Training is only applied in the non-distributor modules. It may be extended to the distributor modules. Foody divided the training patterns for a problem into core patterns and border patterns (1998). Because core patterns are those patterns far away from borders, it is possible to remove some core patterns without losing useful information. Some experiments have been done and the results show that this idea may be valuable. However, a systematic and automatic approach is yet to be found for this. How to reduce the number of training patterns effectively and automatically is still a problem which needs to be addressed in our future study.

A third area of study for the future is the extension of the PD method to regression problems. Unlike classification problems in which outputs are discrete, regression problems have continuous outputs. Thus, distributor modules can not classify patterns with a simple "yes or no" output for regression problems. Decision tree classifiers have been extended to regression problems, i.e. classification and regression tree (CART) system (Steinberg and Colla, 1995). Unlike specifying

priors and misclassification costs in classification trees, regression trees use the sum of squares of the dependent variables for splitting. This idea gives us clues to design PD networks for regression problems. Further work needs to be developed to extend the PD method for regression problems.

References

Akaike, H. (1974). A new look at the statistical model identification. *IEEE Trans. Automatic Control*, vol. 19 (6), pp. 716-723.

Anand, R., Mehrotra, K. G., Mohan, C. K. and Ranka, S. (1993). An improved algorithm for neural-network classification of imbalanced training sets, IEEE Trans. Neural Networks, vol. 4, pp. 962-963.

Argentiero, P., Chin, R. and Beaudet, P. (1982). An automated approach to the design of decision tree classifiers," IEEE Trans. Pattern. *Analysis Machine Intelligence*, vol. PAMI-4, pp. 51-57.

Auda, G., Kamel M. and Raafat, H. (1996). Modular neural network architectures for classification, IEEE International Conference on Neural Networks, vol. 2, pp.1279-1284.

Baker, J. E. (1987). Reducing Bias and Inefficiency in the Selection Algorithm, in *Proceedings of the Second International Conference on Genetic Algorithms and their Application* (Hillsdale), pp. 14-21.

Bao, C. and Guan, S. U. (2006), Reduced Training for Hierarchical Incremental Class Learning, *IEEE Conferences on Cybernetics and Intelligent Systems (CIS) Robotics, Automation and Mechatronics (RAM)*, pp. 1-6.

Bartolucci, L. A., Swain, P. H. and Wu, C. (1976). Selective radiant temperature mapping using a layered classifier, *IEEE Trans. Geosci. Electron.*, vol. GE-14, pp. 101-106.

Bauer, E. and Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, vol. 36(1/2), pp. 105-139.

Bendall, G. and Margot, F. (2006). Greedy Type Resistance of Combinatorial Problems, *Discrete Optimization 3*, pp. 288-298.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Clarendon Press.

Blake C. L. and Merz, C. J. (1998), UCI Repository of machine learning databases [http://www.ics.uci.edu / mlearn/ MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science.

Breiman, L., Friedman, J. H., Olshen, R. A. and Stone, C. J. (1984). *Classification and regression trees*, Belmont, CA: Wadsworth Int..

Casey, R.C. and Nagy, G. (1984). Decision tree design using a probabilistic model, *IEEE Trans. Inform. Theory*, vol. IT-30, pp. 93-99.

Chen, C. H. and You, G. H. (1993). Class-sensitive neural network, *Neural Parallel Scie. Comput.*, vol. 1(1), pp. 93-96.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Steven, C. (2001). *Introduction to Algorithms*, Chapter 16 "Greedy Algorithms", MIT Press.

Diday, E. and Moreau, J. V. (1986). Learning hierarchical clustering from examples -Applications to the adaptive construction of dissimilarity indices, *Pattern Recognition Lett.*, pp. 223 -230.

Duda, R.O., Hart, P.E. and Stork, D.H. (2000). P*attern Classification* (2nd ed.), Wiley Interscience, New York.

Esposito, F., Malerba, D., Semeraro, G. and Kay, J. (1997). A comparative analysis of methods for pruning decision trees, *IEEE Tran. Pattern Analysis and Machine Intelligence*, Vol. 19 (5), pp. 476 - 491.

Feldman, J. (1989). Neural representation of conceptual knowledge, in Nadel and al. (Eds.). *Neural connections, mental computation.* MIT Press. Cambridge, MA.

Freund, Y. (1995). Boosting a weak learning algorithm by majority, *Information and Computation*, vol. 121(2), pp. 256-285.

Freund, Y. (1999). An adaptive version of the boost by majority algorithm. In *Proceedings of the Twelfth Annual Conference on Computational Learning Theory.*

Freund, Y. (2001). An adaptive version of the boost by majority algorithm. *Machine Learning*, vol. 43(3), pp. 293–318.

Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pp. 148–156.

Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, vol. 55(1), pp.119–139.

Freund, Y. and Schapire, R. E. (1999). A Short Introduction to Boosting, *Journal of Japanese Society for Artificial Intelligence*, vol.14(5), pp. 771-780,

Foody, g. M. (1998). Issues in training set selection and refinement for classification by a feedforward neural network, *IEEE international Geoscience and Remote Sensing Symposium Proceeding*, vol. 1(6-10), pp. 409-411.

Friedman, J.H. (1989). Regularized Discriminant Analysis. *Journal of the American Statistical Association*, vol. 84(405), pp. 165–175

Friedman, J. H. (1996). *Another approach to polychotomous classification*, Stanford University, Stanford, CA, Tech. Rep.

Riedmiller, M. and Braun, H. (1993). A direct adaptive method for faster backpropagation learning: the RPROP algorithm, in Proceedings of the IEEE International Conference on Neural Networks, pp. 586-591.

Rreiman, L. (1996). Bagging predictor, *Machine Learning*, vol. 24(2), pp. 123-140.

Gathercole, C., Ross, P. and Bridge, S. (1994). Dynamic training subset selection for supervised learning in genetic programming, *Lecture notes in Computer Science*, 886, pp. 312-321.

Geman, S., Bienenstock, E., and Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation*, vol. 4 (1), pp. 1-58.

Gelfand, S. and Guo, H. (1991). *Tree classifiers with multilayer perceptron feature extraction*, Ph. D. dissertation, School of Elec. Eng., Purdue Univ., W. Lafayette.

Gu, Y. X., Wang, Q. R. and Suen, C.Y. (1983). Application of multi-layer decision tree in a computer recognition of Chinese characters, *IEEE Trans. Putt. Andy. Mach. Intell.*, vol. PAMI-5, pp. 83-89.

Guan, S. U. and Bao, C. (2006). Hierarchical Incremental Class Learning with Reduced Pattern Training, *Neural Processing Letter*, vol. 24(2), pp. 163-177.

Guan, S. U. and Li, S. C. (2000). An approach to parallel growing and training of neural networks, in *Proceeding of 2000 IEEE International Symposium on Intelligent Signal Processing and Communication Systems*, Honolulu, Hawaii, vol. 2, pp. 1101-1104.

Guan, S. U. and Li, S. C. (2002a). Parallel growing and training of neural networks

using output parallelism, *IEEE Trans. Neural Networks*, Vol. 13(3), pp. 542-550.

Guan, S. U.and Li, P. (2002b). A Hierarchical Incremental Learning Approach to Task Decomposition, *Journal of Intelligent Systems*, vol. 12(3), pp. 201-226.

Guan, S. U.and Zhu, F. (2003). Incremental Learning of Collaborative Classifier Agents with New Class Acquisition - An Incremental Genetic Algorithm Approach, *International Journal of Intelligent Systems*, Vol. 18(11), pp. 1173-1192.

Guan, S. U. and Wang, K. (2007). Hierarchical Incremental Class Learning with Output Parallelism, *Journal of Intelligent Systems*, accepted.

Guan, S. U.and Zhu, F. (2004). Class Decomposition for GA-based Classifier Agents - A Pitt Approach, *IEEE Trans. Systems, Man, and Cybernetics, Part B: Cybernetics*, Vol. 34(1), pp. 381-392.

Gustafson, D. E., Gelfand, S. B. and Mitter, S. K. (1980). A nonparametric multiclass partitioning methods for classification, in *Proceedings 5th International conference on pattern Recognition*, pp. 654-659.

Hastie, T., Tibshirani, R. and Friedman, J. (2001). *The elements of statistical learning: Data mining, inferences and perdiction*, Springer Series in Statistics: Springer-Verlag Haykins, Simon.

Haykin, S. (1999). N*eural Networks – A Comprehensive Foundation*. Prentice Hall, 2nd edition.

Ishihara, S. and Nagano, T. (1994). *Text-independent speaker recognition utilizing neural network techniques, Tech. Rep. IEICE*, vol. NC93-121, in Japanese, pp. 71-77.

Jacobs, R.A., Jordan, M.I., Nowlan, S.J. and Hinton, G.E. (1991). *Adaptive mixtures of local experts. Neural Computation*, vol. 3, pp. 79-97.

Jacobs, R.A., and Jordan, M. I. (1991). A competitive modular connectionist architecture, *Neural Information Processing System*, vol. 3, pp. 767-773.

Jordan, M.I. and Jacobs, R.A. (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, vol. 6, pp. 181-214.

Kargupta, K., Park, B. H. and Dutta, H. (2006). Orthogonal decision trees, *IEEE Tran. Knowledge and Data Engineering*, Vol. 18(8), pp. 1028-1042.

Kearns, M. and Valiant, L. (1994). Cryptographic limitations on learning Boolean formulae and finite automata. *Journal of ACM*, vol. 41(1), pp. 67-95.

Kim, B. and Landgrebe, D.A. (1990). *Hierarchical decision tree classifiers in high-dimensional and large class data*, Ph.D. dissertation and Tech. Rep. TR-EE-90-47, School of Elec. Eng., Purdue Univ., W. Lafayette.

Kulkarni, A. V. (1976). *Optimal and heuristic synthesis of hierarchical classifiers*, Ph.D. dissertation, Comput. Sci. Tech. Rep. TR-469., Univ. of Maryland, College Park.

Kwok, T., and Yeung, D. (1997). Constructive algorithms for structure learning in feedforward neural networks for regression problems. *IEEE Trans. Neural Networks*, vol. 8(3), pp. 630-645.

Kwok, T., and Yeung, D. (1997). Objective functions for training new hidden units in constructive neural networks. *IEEE Tran. Neural Networks*, vol. 8(5), pp. 1131-1148.

Landeweerd, G., Timmers, T., Gersema, E., Bins, M. and Halic, M. (1983). Binary tree versus single level tree classification of white blood cells, *Pattern Recog.*, vol. 16, pp. 571-577.

Lehtokangas, M. (1999). Modelling with constructive backpropagation. *Neural Networks*, vol. 12, pp. 707-716.

Li, Y. and Dong, M. (2003). Classifiability based omnivariate decision trees, *Proceedings of the International Joint Conference on Neural Networks*, Vol. 4, pp. 3223 - 3228.

Li, X. and Dubes, R.C. (1986). Tree classifier design with a Permutation statistic, *Pattern Recog.*, vol. 19, pp. 229-235.

Lu, B. L., Kita, H. and Nishikawa, Y. (1994). A multisieving neural-network architecture that decomposes learning tasks automatically, in *Proceedings of IEEE Conference on Neural Networks*, Orlando, FL, 1319-1324.

Lu, B. L. and Ito, K. (1995). A comparative study of output representation schemes for multilayerneural networks, *Proceedings of the 34th SICE Annual Conference.* International Session Papers, pp. 1535-1538.

Lu, B. L. and Ito, M. (1999). Task decomposition and module combination based on class relations: A modular neural network for pattern classification, *IEEE Trans. Neural networks*, vol. 10(5), pp. 1244 - 1256.

Meir, R. and Ratsch, G. (2003). An introduction to boosting and leverageing. *Advanced lectures on machine learning*, Springer, pp. 119-184.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, London, UK.

Mika, S., Ratsch, G., Weston, J., Scholkopf, B. and Mullers, K.R. (1999). Fisher Discriminant Analysis with Kernels. *IEEE Conference on Neural Networks for Signal Processing IX*, pp. 41-48.

Mitchell, M. (1996). A*n Introduction to Genetic Algorithms*, MIT Press, Cambridge,

MA.

Pao, Y. H. (1989). *Adaptive Patter-Recognition and Neural Network*, Reading, MA: Addison-Wesley.

Payne, H. and Meisel, W. (1977). An algorithm for constructing optimal binary decision trees, *IEEE Trans. Comput.*, vol. C-26, pp. 905-916.

Pedrycz, W. and Sosnowski, Z.A. (2005). C-fuzzy decision trees, *IEEE Trans. Systems, Man and Cybernetics, Part C*, Vol. 35(4), pp. 498 - 511.

Poggio, T., and Girosi, F. (1990). Regularization algorithms for learning that are equivalent to multilayer networks. *Science*, 247, 978-982.

Prechelt, L. (1994). PROBEN1: A set of neural network benchmark problems and benchmarking rules, Technical Report 21/94, Department of Informatics, University of Karlsruhe, Germany.

Quinlan, J. R. and Rivest, R. L. (1989). Inferring decision trees using the minimum description length principle, *Information and Computation*, vol. 80(3), pp. 227-248.

Quinlan, J.R. (1993). *C4.5: Programs for Machine Learning*, Morgan Kaufmann.

Quinlan, J.R. (2003). C5.0 Online Tutorial, http://www.rulequest.com.

Reed, R. (1993). Pruning algorithms-a survey. *IEEE Trans. Neural Networks*, vol. 4(5), pp. 740-747.

Riedmiller, M. and Braun, H.(1993). A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm, *Proc. ICNN, San Fransisco.*

Rissanen, J. (1975). Modeling by shortest data description. *Automatica.*, vol. 14, pp. 465-471.

Rounds, E. (1980). A combined non-parametric approach to feature selection and binary decision tree design, *Pattern Recog.*, vol. 12, pp. 313-317.

Rumelhart, D. E., Hinton, G. E. and Williams, R. J. (1986). *Learning internal representations by error backpropagation, in Parallel distributed processing* (D. E. Rumelhart and J. L. McClelland, eds.), The MIT Press, vol. 1, pp. 318-362.

Schapire, R. E. (1997). Using output codes to boost multiclass learning problems, F*ourteenth international conference on machine learning*, San Francisco, pp. 313-321.

Schwartz, G. (1978). Estimating the dimension of a model. *The Annals of Statistics*, vol. 6, pp. 461-464.

Simon, H. (1981). *The sciences of the artifical.* Cambridge, MA: MIT press.

Squires, C. S. and Shavlik, J. W. (1991). Experimental analysis of aspects of the cascade -correlation learning architecture, in Machine Learning Research Group Working Paper 91-1, Department of Computer Science, Univ.Wisconsin, Madison.

Steinberg, D. and Colla., P. (1995), *CART: Tree- structured non parametric data analysis.* San Di ego, Calif., U.S.A.: Sal ford Sys tems.

Titsias, M. K. and Likas, A. (2002). Mixture of Experts Classification Using a Hierarchical Mixture Model, *Neural Computation*, vol. 14, pp. 2221-2244.

Wang, Q. R. and Suen, C. Y. (1987). Large tree classifier with heuristic search and global training, *IEEE Trans. Putt. Anal. Mach. Intell.*, vol. PAMI-9, pp. 91-102.

Yun, S.Q. and Fu, K.S. (1983). A method for the design of binary tree classifiers, *Pattern Recog.*, vol. 16, pp. 593-603.

Appendix

The Early Stopping method using validation set is used as the stopping criteria in training the new modular network (Guan and Li, 2002b). The set of available patterns is divided into three sets: a training set is used to train the network, a validation set is used to evaluate the quality of the network during training and to measure overfitting, and a test set is used at the end of training to evaluate the resultant network. The size of the training, validation, and test set is 50%, 25% and 25% of the problem's total available patterns. The error measure E used is the squared error percentage (Squires and Shavlik, 1991), derived from the normalization of the mean squared error to reduce the dependency on the number of coefficients in the problem representation and on the range of output values used:

$$E = 100 \cdot \frac{o_{max} - o_{min}}{K \cdot P} \sum_{p=1}^{P} \sum_{k=1}^{K} (o_{pk} - t_{pk})^2 \quad , \tag{A.1}$$

where o_{max} and o_{min} are the maximum and minimum values of output coefficients in the problem representation, P is the number of patterns, K is the number of outputs, o_{pk} is the actual output value of the k^{th} output unit for the p^{th} training pattern and t_{pk} is the desired output value of the k^{th} output unit for the p^{th} training pattern.

$E_{tr}(t)$ is the average error per pattern of the network over the training set, measured after epoch t. The value $E_{va}(t)$ is the corresponding error on the validation set after epoch t and is used by the stopping criterion.$E_{te}(t)$ is the corresponding error on the test set; it is not known to the training algorithm but characterizes the quality of the network resulting from training.

The value $E_{opt}(t)$ is defined to be the lowest validation set error obtained in epochs up to epoch :

$$E_{opt}(t) = \min_{t' < t} E_{va}(t') \quad . \tag{A.2}$$

The *generalization loss* (Squires and Shavlik, 1991) at epoch t is defined as the relative increase of the validation error over the minimum so far (in percent):

$$GL(t) = 100 \cdot (\frac{E_{va}(t)}{E_{opt}(t)} - 1) \quad . \tag{A.3}$$

A high generalization loss is one candidate reason to stop training because it directly indicates overfitting.

To formalize the notion of training progress, *a training strip of length* m is defined to be a sequence of m epochs numbered $n+1$, $n+2$, ..., $n+m$, where n is divisible by m(Squires and Shavlik, 1991). The training progress measured after a training strip is:

$$P_m(t) = 100 \cdot (\frac{\sum_{t'=t-m+1}^{t} E_{tr}(t')}{m \cdot \min_{t' \in [t-m+1,t]} E_{tr}(t')} - 1) \quad . \tag{A.4}$$

It is used to measure how much larger the average training error is than the minimum training error during the training strip.

During the process of growing and training individual modules, we adopted the following heuristic overall stopping criteria: $E_{opt} < E_{th}$ OR (*Reduction of training set error due to the last new hidden unit is less than 0.01%* AND *Validation set error increased due to the last new hidden unit*). The first part ($E_{opt} < E_{th}$) means that the optimal validation set error is below the threshold (E_{th}) and the result has been acceptable. The other part means the last insertion of a hidden unit resulted in hardly any progress. The criteria for adding a new hidden unit are as follows: *At least 25 epochs reached for the current network* AND (Generalization loss $GL(t) > 5$ OR Training progress $P_5(t) < 0.1$). The first part means that the current network should be trained for at least a certain number of epochs before a new hidden unit is installed because the error curves may be turbulent at the beginning. The second part means that the current network has been overfitted or training has little progress. It is a bit unsatisfactory that all of these criteria are heuristic.