Efficient sequential and batch learning artificial neural network methods for classification problems

Zhang, Runxuan

2006

Zhang, R. (2006). Efficient sequential and batch learning artificial neural network methods for classification problems. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/3468>

<https://doi.org/10.32657/10356/3468>

Nanyang Technological University

Downloaded on 26 Apr 2023 23:24:24 SGT

Efficient Sequential and Batch Learning Artificial Neural Network Methods for Classification Problems

Zhang Runxuan

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University
in fulfillment of the requirements for the degree of
Doctor of Philosophy

2006

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research and has not been submitted for a higher degree to any other University or Institution.

.....

Date

.....

Zhang Runxuan

To my family.

Acknowledgments

I would like to express my deepest gratitude to my supervisor Prof. N. Sundararajan for his intelligent guidance and patient understanding during the last three years. I have learned so much from his ways of critical thinking. His analytic insights into the problems helped me greatly in the accomplishment of this thesis. I am proud to have a such a great mentor during my research and he has made my stay at Nanyang Technological University a truly valuable experience.

I want to thank Associate Prof. P. Saratchandran and Assistant Prof. Guang-Bin Huang for their numerous helpful comments and enlightening discussions throughout this research work. I have been receiving their help from all angles during my stay at NTU. Their time and assistance were a valuable contribution to this thesis.

Thanks are also dedicated to my colleagues, friends and teachers who were always ready to offer their help and encouragements.

I would also like to dedicate a special thanks to my family, especially my parents, who have always been there for me, whenever and wherever. This research work will not be possible without their love and encouragement.

Finally, I wish to offer my thanks to the Biomedical Instrumentation Laboratory and Bioinformatics Reseach Center and their staff for providing me the research facilities and help. I also acknowledge Nanyang Technological University for their financial support and providing me this precious opportunity of study.

Summary

This thesis focuses on the development and applications of efficient sequential and batch learning artificial neural network methods for classification problems. Emphasis is on applications in the bio-informatics area where the problems have a very high input dimension and a small number of samples. Here, by “efficient methods”, we imply those methods that produce more accurate classification, low training time and a compact network structure.

In this study, sequential learning algorithms based on Radial Basis Function (RBF) neural networks are first investigated for classification problems. In real world classification problems, even though they start initially with large data sets, new training data becomes available from time to time. To handle the training of the network for the new data, batch learning schemes require that the network has to be retrained all over again, resulting in a large training time. A learning scheme that handles these data without retraining with the complete data all over again will be quite useful in real applications. A second problem with the popular learning algorithms of RBF network is that the network structure has to be fixed *a priori* and this may result in an over-determined network. Also, when the data arrives sequentially, fixing the network size *a priori* may not be the right approach. In this case, a growing and pruning strategy that adjusts the number of neurons automatically is more appropriate.

Recently, a sequential learning algorithm for Growing and Pruning Radial Basis Function (RBF) networks called GAP-RBF algorithm has been developed and its performance has been evaluated (only) for function approximation problems. In this thesis, its performance on classification problems is investigated and compared with another well-known sequential learning algorithm, viz., Minimal Resource Allocation

Network (MRAN) as well as the conventional Multilayer Feedforward Networks (MFNs) on three benchmark problems in the classification area from PROBEN1 database. PROBEN1 gives a set of problems for neural network learning in the area of pattern classification and function approximation along with a set of rules and conventions for carrying out the tests with these problems. All the problems in the database represent real problems which ensures that the results are relevant for real applications. The results indicate a better performance of GAP-RBF algorithm in terms of generalization, network size and training speed as well as the classification accuracy for problems with smaller input dimensions.

A difficulty with the GAP-RBF algorithm is in the calculation of the “significance” of the neuron. GAP-RBF assumes that the input data is uniformly distributed and the neuron significance is calculated based on this. If the input distribution is not uniform, especially when the input attributes are discrete, the performance of GAP-RBF algorithm deteriorates. This may be a serious problem for classification applications. To overcome this difficulty, a new scheme of calculating the significance of a neuron based on the recently received M input data is given in this thesis. As the input samples are fed into the network in a random order, these M samples are considered as the representation of the distribution in the whole input space. This method avoids the need for knowing the input distribution and also makes the “significance” computation fast and straightforward.

Another difficulty with the GAP-RBF algorithm in classification problems arise when the input dimension is high. This is because in GAP-RBF algorithm, the parameters of the network are updated using an Extended Kalman Filter (EKF) method. When the number of inputs is large and as the neurons grow, the size of the covariance matrix becomes large and this causes a computational overload. Decoupled EKF (DEKF) is introduced in the GAP-RBF algorithm to overcome this problem by reducing the size of the covariance matrix and thereby making the

computation faster.

A new sequential learning algorithm for RBF networks, referred to as Fast GAP-RBF (FGAP-RBF) algorithm is developed in this thesis. It removes the above two difficulties in the GAP-RBF algorithm, namely, a non-uniform input distribution and also the computational complexity for a large input dimension. The performance of the FGAP-RBF algorithm is compared with that of the GAP-RBF algorithm along with other established sequential algorithms such as MRAN. The comparison is done based on the performances on four real world benchmark classification problems from UCI machine learning repository database and the ESPRIT Basic Research Project. The results indicate that FGAP-RBF algorithm produces higher classification accuracy with a reduced computational complexity.

This thesis also makes another significant contribution to the multi-category classification problems in the bioinformatics area, especially for the applications of micro-array gene expression data based cancer diagnosis. Gene expression-based classification problem is generally considered a difficult task because for such problems, the input dimension is very high and the number of samples is very small. The gene expression data often contains thousands to ten thousands of genes, while the size of the data available is usually below a hundred. The large number of genes of micro-array data directly translates to the number of free parameters. It has been recognized that when the ratio of the number of training samples to the number of free classifier parameters is lower, the generalization capability of the resulting classifier becomes worse. Gene expression based multi-category classification is also more difficult compared to binary classification methods. In the gene expression profiling based classification area, multi-category classifications are mostly done by modifying binary classification methods on a One-Versus-All (OVA) or One-Versus-One (OVO) comparison basis. This modification inevitably involves many classifiers and increases the system complexity. Recently, conventional neural network based

classification methods have been attempted for direct multi-category classification. However, it has been found that these conventional neural network methods usually produce poor performance in terms of classification accuracies and training time.

In this thesis, the newly developed FGAP-RBF is used for a micro-array gene expression-based multi-category classification problem. FGAP-RBF algorithm is the first attempt on the use of any sequential learning algorithm for such classification problems. Results show that FGAP-RBF algorithm achieves better performance in terms of accuracy than the best results described by other methods in the literature. However, when the number of genes used in classification approaches a thousand, FGAP-RBF algorithm causes an updated PC to crash only after a few hidden neurons are added. Therefore, we look into a fast and efficient batch learning algorithm called Extreme Learning Machine (ELM) that can easily handle problems of very large input dimensions.

Extreme Learning Machine (ELM) is a newly proposed batch learning algorithm for Single hidden Layer Feedforward Networks (SLFNs). Its learning speed can be thousands of times faster than traditional feedforward network learning algorithms like back-propagation algorithm and also it produces better generalization performance. In this thesis, the recently developed ELM has also been used to solve the multi-category microarray gene expression based cancer diagnosis problem. Different from traditional learning algorithms, ELM not only tends to reach the smallest training error but also the smallest norm of weights with a good generalization performance. We have made an improvement on the ELM algorithm by introducing a gain parameter λ in the sigmoid activation function to improve the generalization performance on the micro-array gene expression problems where the input is distributed very sparsely.

Study results show that ELM can perform direct classification for such multi-category micro-array problems in a fast and efficient manner. Its performance has

been compared with other methods such as ANN, SANN and SVM algorithms on six gene expression-based cancer diagnosis datasets. Study results show that when the number of categories for the classification task is large, ELM algorithm achieves a higher classification accuracy than the other algorithms with a lesser training time and a smaller network structure. It can also be seen that ELM achieves better and more balanced classification for individual categories as well. For applications with a smaller number of categories, ELM achieves a similar accuracy with a much lesser training time and compact structure compared to SVM-OVO.

Table of Contents

Acknowledgments	i
Summary	ii
Table of Contents	vii
List of Figures	xii
List of Tables	xv
1 Introduction	1
1.1 Motivation	1
1.2 Objectives	4
1.3 Major Contributions of the thesis	5
1.4 Organization of the Thesis	8
2 A Review of Methods for Pattern Classification and Bioinformatics Classification Problems	11
2.1 Conventional Algorithms for Classification	13
2.1.1 Statistical Methods	13
2.1.2 Linear Discriminant Methods	15
2.1.3 Nonlinear Discriminant Methods	16

TABLE OF CONTENTS

viii

2.1.3.1	Decision Trees	17
2.1.4	Artificial Neural Networks	18
2.1.4.1	Multi-Layer Perceptrons	19
2.1.4.2	Radial Basis Function Networks	21
2.2	Overview of Bioinformatics Classification Problems	30
2.3	Problems with the Existing Approaches	40
3	GAP-RBF Based Sequential Learning Scheme for Classification Problems	42
3.1	Brief Review to the GAP-RBF Algorithm	43
3.1.1	<i>Significance of a Neuron</i>	45
3.1.2	Growing Criterion	48
3.1.3	Pruning Criterion	49
3.1.4	Nearest Neuron for Parameter Adjustment and Pruning	50
3.2	A Comparison of GAP-RBF and MRAN	55
3.2.1	Comparison of Parameters	57
3.2.2	Comparison of Growing and Pruning Strategy	58
3.2.3	Comparison of Computational Complexity	58
3.2.4	Comparison of Memory Requirements	59
3.3	Classification Problems from PROBEN1 Databases	59
3.4	Experimental Results	62
3.4.1	Hearta Problem	62
3.4.2	Cancer Problem	65
3.4.3	Gene Problem	67
3.5	Summary	71
4	A New Fast Growing and Pruning-RBF (FGAP-RBF) Algorithm	

TABLE OF CONTENTS

ix

For Classification Problems	73
4.1 Improvement on the Calculation of the Neuron Significance	75
4.1.1 Problems with the calculation of the neuron significance . . .	75
4.1.1.1 Concentric Problem	75
4.1.1.2 Two-shell Problem	77
4.1.2 Simplified Estimation of Neuron Significance	81
4.2 Improvement on the Parameter Adjustment	83
4.2.1 Problem with the EKF Method	83
4.2.2 Parameters Adjusted by Decoupled Extended Kalman Filter (DEKF)	86
4.3 The New FGAP-RBF Algorithm	87
5 Performance Evaluation of FGAP-RBF for Classification Problems	91
5.1 Performance Evaluation of FGAP-RBF for Classification Problems . .	92
5.1.1 Phoneme Problem	92
5.1.2 Segment Problem	93
5.1.3 Satimage Problem	94
5.1.4 DNA Problem	95
5.2 Discussion	97
5.2.1 Impact of Memory Factor M	97
5.2.2 Impact of Decoupling in the EKF	98
5.3 Improvement of Parameter Selection	100
5.3.1 Selection of Overlapping Factor κ	100
5.3.2 Selection of Memory Impact Factor M	102
5.3.2.1 Comparison of MRAN and FGAP-RBF on Memory Impact Factor	107

TABLE OF CONTENTS

x

5.3.3	Selection of Scale of Interest	109
5.3.4	Selection of Learning Accuracy	111
5.3.5	Guidelines for Parameter Selection	114
6	Microarray Gene Expression-Based Multi-category Classification Using Sequential Learning Algorithms	116
6.1	Gene Expression-Based Multi-category Classification Problem	117
6.2	The Sequential Learning Method—FGAP-RBF	119
6.2.1	Experimental Methods	120
6.2.2	Performance Evaluation	121
6.2.2.1	Classification Accuracy	121
6.2.2.2	Network Complexity	123
6.2.2.3	Computational Time	123
6.2.2.4	Classification Performance on Individual Category	124
6.3	Summary	128
7	Microarray Gene Expression-Based Multi-category Classification Using Batch Learning Algorithms	129
7.1	A Brief Introduction of the ELM Algorithm	132
7.1.1	Mathematical Description of Unified SLFN	133
7.1.2	Extreme Learning Machine	134
7.2	Experiments and Results	137
7.2.1	Experiment 1 - Microarray Benchmark Dataset (GCM)	137
7.2.1.1	Gene Selection Method	138
7.2.1.2	ELM Algorithm	138
7.2.1.3	SVM-OVO Algorithm	139
7.2.2	Results and Discussion	140

TABLE OF CONTENTS

xi

7.2.2.1	Classification Accuracy	140
7.2.2.2	Robustness Comparison of ELM and SVM-OVO . .	141
7.2.2.3	Network Complexity	142
7.2.2.4	Training Time	149
7.2.2.5	Classification Performance on Individual Category .	149
7.2.3	Experiment 2 - HBC Dataset	156
7.2.4	Experiment 3 - Lung Dataset	156
7.2.5	Experiment 4 - Lymphoma Dataset	160
7.2.6	Experiment 5 - MLL Dataset	160
7.2.7	Experiment 6 - NCI60 Dataset	170
7.3	Summary	170
8	Conclusions and Future Work	175
8.1	Conclusions	175
8.2	Recommendations for Future work	178
	Author's Publications	181
	Bibliography	182
	Appendix A	195

List of Figures

2.1	An example of MLP with two hidden layers	19
2.2	General architecture of an RBF neural network.	22
3.1	The Basic structure of a GAP-RBF Neural Network	44
3.2	The approximation of Gaussian function	46
3.3	The First and Second Derivatives of Gaussian Function	52
3.4	Learning Speed Comparison between MRAN and GAP-RBF for Cancer3.	67
3.5	Neuron updating progress of MRAN and GAP-RBF for Gene3	70
4.1	Concentric Problem	76
4.2	Testing Samples and the positions of the hidden neurons for Concentric Problem.	77
4.3	Two-Shell Problem	78
4.4	Testing Samples and the positions of the hidden neurons for Two-Shell Problem	79
5.1	The influence of M on the classification error and network structure for the phoneme problem	103
5.2	The influence of M on the classification error and network structure for the segment problem	103

LIST OF FIGURES

xiii

5.3	The influence of M on the classification error and network structure for the satimage problem	104
5.4	The influence of M on the classification error and network structure for the DNA problem	104
5.5	The influence of M on the classification error and training time for the phoneme problem	105
5.6	The influence of M on the classification error and training time for the segment problem	105
5.7	The influence of M on the classification error and training time for the satimage problem	106
5.8	The influence of M on the classification error and training time for the DNA problem	106
5.9	The influence of Memory factor M on MRAN and FGAP-RBF	108
5.10	The influence of Memory factor M on FGAP-RBF	110
6.1	Comparison of the classification accuracies of FGAP-RBF, SANN, SVM-OVA and ANN.	122
6.2	Comparison of individual category classification: Breast, Prostate, Lung, Colorectal, Lymphoma, Bladder and Melanoma.	125
6.3	Comparison of individual category classification: Uterus_Adeno, Leukemia, Renal, Pancreas, Ovary, Mesothelioma and CNS.	126
7.1	Comparison of the classification accuracies of ELM, SVM-OVO, SANN and ANN.	141
7.2	Comparison of the classification accuracies of ELM and SVM-OVO using the best parameters for 14 genes.	142
7.3	Comparison of the classification accuracies of ELM and SVM-OVO using the best parameters for 28 genes.	143
7.4	Comparison of the classification accuracies of ELM and SVM-OVO using the best parameters for 42 genes.	144

LIST OF FIGURES

xiv

7.5	Comparison of the classification accuracies of ELM and SVM-OVO using the best parameters for 56 genes.	145
7.6	Comparison of the classification accuracies of ELM and SVM-OVO using the best parameters for 70 genes.	146
7.7	Comparison of the classification accuracies of ELM and SVM-OVO using the best parameters for 84 genes.	147
7.8	Comparison of the classification accuracies of ELM and SVM-OVO using the best parameters for 98 genes.	148
7.9	Comparison of individual category classification: Breast, Prostate, Lung, Colorectal, Lymphoma, Bladder and Melanoma.	150
7.10	Comparison of individual category classification: Uterus_Adeno, Leukemia, Renal, Pancreas, Ovary, Mesothelioma and CNS.	151
8.1	Comparison of the testing accuracies of ELM, SVM-OVO, SVM-OVA.	197

List of Tables

3.1	Performance comparison for Hearta Problem	64
3.2	Performance comparison for Cancer Problem	66
3.3	Performance comparison for Gene Problem	69
4.1	Performance for concentric problem	76
4.2	Performance for Two-Shell Problem	79
5.1	Details of classification datasets.	92
5.2	Performance comparison for Phoneme Problem	93
5.3	Performance comparison for Segment Problem	94
5.4	Performance comparison for Satimage Problem	95
5.5	Performance comparison for DNA Problem	96
5.6	Algorithm parameters for FGAP-RBF, GAP-RBF and MRAN . . .	101
5.7	Performance of FGAP-RBF, GAP-RBF and MRAN for the bench- mark classification problems.	102
5.8	Algorithm parameters for FGAP-RBF, GAP-RBF and MRAN. . . .	111
5.9	Performance of FGAP-RBF, GAP-RBF and MRAN for the bench- mark classification problems.	112
5.10	Algorithm parameters for FGAP-RBF, GAP-RBF and MRAN (3) . .	112
5.11	Performance of FGAP-RBF, GAP-RBF and MRAN for the bench- mark classification problems.	113

LIST OF TABLES

xvi

6.1	GCM Dataset - Tumor classes and their details	118
6.2	Classification Accuracy (%) for different algorithms.	121
6.3	Number of hidden neurons for GAP-RBF algorithm.	123
6.4	Training time(s) for GAP-RBF networks	124
6.5	Confusion Matrix for the test data for FGAP-RBF algorithm	127
7.1	The detailed information of Datasets	137
7.2	Optimal parameters for ELM and SVM-OVO algorithms for selected genes	140
7.3	Classification accuracy (%) for different algorithms	140
7.4	Training time(s) for ELM and SVM-OVO algorithms	149
7.5	Confusion matrix obtained by ELM for the testing data.	153
7.6	Confusion matrix obtained by SVM-OVO for the testing	154
7.7	Testing Accuracy and standard deviation (%) on different number of genes using different gene selection methods for ELM and SVM for HBC dataset	157
7.8	Best parameter selected for different number of genes using different gene selection methods for ELM and SVM for HBC dataset	158
7.9	Training time(s) for ELM and SVM algorithms for HBC dataset	159
7.10	Testing Accuracy and standard deviation (%) on different number of genes using different gene selection methods for ELM and SVM for Lung dataset	161
7.11	Best parameter selected for different number of genes using different gene selection methods for ELM and SVM for Lung dataset	162
7.12	Training time(s) for ELM and SVM algorithms for Lung dataset	163
7.13	Testing Accuracy and standard deviation (%) on different number of genes using different gene selection methods for ELM and SVM for Lymphoma dataset	164

7.14	Best parameter selected for different number of genes using different gene selection methods for ELM and SVM for Lymphoma dataset . . .	165
7.15	Training time(s) for ELM and SVM algorithms for Lymphoma dataset	166
7.16	Testing Accuracy and standard deviation (%) on different number of genes using different gene selection methods for ELM and SVM for MLL dataset	167
7.17	Best parameter selected for different number of genes using different gene selection methods for ELM and SVM for MLL dataset	168
7.18	Training time(s) for ELM and SVM algorithms for MLL dataset . . .	169
7.19	Testing Accuracy and standard deviation (%) on different number of genes using different gene selection methods for ELM and SVM for NCI60 dataset	171
7.20	Best parameter selected for different number of genes using different gene selection methods for ELM and SVM for NCI60 dataset	172
7.21	Training time(s) for ELM and SVM algorithms for NCI60 dataset . .	173
8.1	Testing accuracy (%) of different algorithms on GCM dataset	196
8.2	Testing accuracy (%) of ELM and SVM-OVO algorithms on Lung dataset	197
8.3	Testing accuracy (%) for ELM and SVM-OVO algorithms on Lymphoma dataset	198

Chapter 1

Introduction

1.1 Motivation

Over the past decade, a number of supervised learning algorithms have been developed for pattern classification problems. These supervised learning algorithms include statistical classification methods, linear discriminant methods as well as nonlinear discriminant methods. Artificial Neural Networks have been widely used in the fields of pattern classification and they show their advantages over other methods in their learning, generalization and adaptation capability as well as their unique power for nonlinear mapping. However, the traditional gradient-based batch learning algorithms, such as scaled conjugate gradients, etc [14] that are most often used in the training process, often cause an artificial neural network to get trapped in a local minimum occasionally and lead to poor classification results. Besides, the training with these gradient based learning methods is generally slow and many iterative learning epochs are required by such learning algorithms to obtain good performance. Therefore, learning methods other than the traditional gradient based batch learning algorithms are studied in this thesis with the aim of developing fast and accurate learning algorithms for classification problems.

One alternative to the traditional gradient based batch learning algorithms is the use of sequential learning algorithms. In batch learning, the input data set is divided into a training set and a validation set. During the training, weights are updated after all the patterns from the training set are processed and the training is done over a number of epochs. The validation set is used to avoid overtraining, and finally the classification accuracy is assessed on the test data. However, in sequential learning, no validation set is required to prevent overfitting. Weights are updated after each pattern presentation rather than after a complete epoch. In sequential learning, only one epoch is needed and the training stops after all the training patterns are processed. One major advantage of sequential learning is that it can perform incremental learning. For a sequentially trained network, when some new training data for the same classification task comes, it can be trained solely on the new data while retaining all the old information of the previously trained data. However, whenever a new data is received, batch learning algorithms require re-training with all the original training set together with the new information. For classification systems that have large datasets for training and need frequent updates with new samples, this incremental learning capability of sequential learning algorithms can save a great computational effort and a large amount of training time. Besides, sequential learning algorithms can perform online classification and analysis, which is impossible for batch learning algorithms. In the area of artificial neural networks, Radial Basis Function (RBF) neural networks have shown their great potential for sequential learning by their unique capability to make any arbitrary mapping between input and output patterns with better generalization and a simpler network structure at the same time. Thus, sequential learning algorithms and RBF neural networks [141] [52] [53] have formed a natural combination to develop neural network methods with good generalization performance and higher training speed for classification problems.

Another alternative to the traditional gradient-based batch learning algorithms is

to use some novel batch learning algorithms that require less computational effort and training time. Recently, Huang *et al* [54, 55, 58] have proposed a new learning algorithm called the Extreme Learning Machine (ELM) for Single-hidden Layer Feedforward neural Networks (SLFNs). In the ELM, one may randomly choose and fix the input weights and the hidden neurons' biases, and then analytically determine the output weights of SLFNs [58]. Input weights are the weights of the connections between the input neurons and the hidden neurons, and output weights are the weights of the connections between the hidden neurons and the output neurons. After the input weights and the hidden layer biases are chosen randomly, SLFNs can be considered as a linear system and the output weights can be analytically determined through a generalized inverse operation of the hidden layer output matrices. Studies [58] show that the new learning algorithm ELM tends to produce smaller training error, obtain a smaller norm of the weights. It also produces good generalization performance and runs extremely fast.

Another motivation of this work is to develop artificial neural network methods that can solve classification tasks efficiently in the bioinformatics domain. For these classification tasks, we are targeting two kinds of problems that usually occur in this area. One type of the problem is when the input distribution is difficult to get or unknown. We know that the distribution of the input space affects classification tasks greatly and many classification methods rely heavily on the distribution of the input patterns. However, in real applications where on-line training is needed, there is little or no prior knowledge of the nature of the datasets. The input distribution of these problems is usually hard to find. Therefore, for such applications, algorithms that are not influenced by the input distribution are preferred. The other problem is that of the high input dimensions, especially for gene expression-based microarray diagnosis problems in the bioinformatics area [79]. Gene expression-based classification is difficult because for such problems the input dimension is very high and the input usually contains thousands to ten thousands of genes. However, the number

of available samples is very small and is usually below 100. The low ratio between the number of samples and the input dimensions directly results in a very sparse input space, which makes achieving good generalization performance difficult for most classifiers. Besides, a large input dimension also brings in a large computational effort. Efficient algorithms are needed to handle these computational burden without jeopardizing the classification performance.

In this work, we first investigate the sequential learning RBF network methods for classification applications. Then the microarray gene expression based diagnosis tasks in the bioinformatics area are studied using both the sequential learning algorithm of the RBF network and the novel batch-learning ELM-based classifiers.

1.2 Objectives

The primary objective of this work is to study artificial neural network based methods for classification problems using both sequential and batch learning algorithms respectively. More specifically, the objectives of the study reported in this thesis can be summarized as:

- To evaluate the recently developed sequential learning Growing and Pruning RBF (GAP-RBF) algorithm [52] for classification problems. The GAP-RBF algorithm is a newly developed sequential growing and pruning algorithm for RBF networks for function approximation problems. It has been evaluated on a number of benchmark problems and confirmed excellent performance in the function approximation area, but its performance for classification problems is generally unknown. One objective of this work is to evaluate its performance for classification problems of different complexities.
- To design a new sequential learning algorithm for RBF networks based on

the GAP-RBF algorithm targeted at complex classification problems with a high input dimension. When the GAP-RBF algorithm is used for complex classification problems with a high input dimension, it often suffers from a low accuracy and a slow training. One objective of this work is to design new learning schemes to improve the classification performance for such problems.

- To explore efficient sequential and batch learning neural network methods for the multi-category classification problems in the bioinformatics area, especially for application to microarray gene expression-based cancer diagnosis problems. Gene expression-based classification problems are generally considered difficult because of their large input dimensions, and multi-category classification is even more difficult than the binary classification problem. In the literature, this multi-category classification is mostly done by modifying binary classification methods [110] [94] [79] [7]. Our objective is to find efficient sequential and batch learning algorithms that can perform multi-category classification directly and achieve higher performance in terms of testing accuracy and training speed on microarray gene expression problems.

1.3 Major Contributions of the thesis

The major contributions of this thesis can be summarized as follows:

In the first part of the thesis, sequential learning algorithms based on RBF neural networks are investigated for classification problems.

- The newly developed sequential learning algorithm for RBF networks, viz. the GAP-RBF algorithm is investigated for classification problems and compared with other well-known sequential learning algorithm—Minimal Resource

Allocation Network (MRAN) as well as the conventional Multilayer Feed forward Networks (MFNs) on three benchmark classification problems from the PROBEN1 database. The results indicate better performance of the GAP-RBF algorithm in terms of generalization, network size as well as training speed for problems with smaller input dimensions.

- Two limitations of GAP-RBF algorithm leading to its poor performance for classification applications with non-uniformly distributed high dimensional input data are identified.
 - In GAP-RBF, the calculation of the *significance* of a neuron is based on the assumption that the input data is uniformly distributed. If the input distribution is not uniform, especially when the input attributes are discrete, then the performance of GAP-RBF is degraded. This may be a serious problem for classification applications.
 - Another difficulty arises when the input dimension is high. In the GAP-RBF algorithm, the parameters of the network are updated using an Extended Kalman Filter (EKF) method. When the number of inputs is large and as the number of neurons grows, the intensive computational effort for the covariance matrix of EKF leads to inefficient training and causes a computational overload.
- A new sequential learning algorithm for RBF networks referred to as Fast GAP-RBF (FGAP-RBF) algorithm is developed to overcome the above drawbacks of the GAP-RBF algorithm.
 - In this thesis, a new scheme of calculating the significance of a neuron based on the most recently received M input samples is presented. As the input samples are fed into the network in a random order, these M samples are considered as a representation of the distribution in the

whole input space. The new scheme offers a faster and more accurate way of calculating the neuron significance and eliminates the requirement of knowing the distribution of the input data *a priori*.

- The Decoupled EKF (DEKF) is introduced into the GAP-RBF algorithm to overcome the problem of computational overload [106] by reducing the size of the covariance matrix.
- The proposed FGAP-RBF algorithm is evaluated together with other sequential algorithms, such as GAP-RBF and MRAN as well as a batch learning algorithm—SVM on four real-world benchmark classification problems. The results show that the FGAP-RBF algorithm achieves a higher classification accuracy than the GAP-RBF and MRAN algorithms for all the four problems and the FGAP-RBF achieves comparable classification accuracies to that of SVM (batch algorithm) on these problems. The influences of the parameters of the FGAP-RBF algorithm are explored and guidelines on how to select these parameters are also given.

In the second part of the thesis, significant contributions to the multi-category classification problems in the bioinformatics area, especially for the applications of microarray gene expression-based cancer diagnosis are presented.

In the gene expression profiling based classification area, multi-category classifications are mostly done by modifying binary classification methods on a One-Versus-All (OVA) or One-Versus-One (OVO) comparison basis. This modification inevitably involves many classifiers and increases the system complexities. Conventional neural network based classification methods have been attempted for direct multi-category classification. However, it is found that these conventional neural network methods usually produce poor performance in terms of classification accuracies and training time.

- The newly developed sequential learning algorithm–FGAP-RBF is used to study the microarray gene expression-based multi-category classification problem. This is the first application of any sequential learning algorithm to such problems. The results show that the FGAP-RBF algorithm can achieve better performance in terms of accuracy than the best results described in the literature. However, when the number of genes used for classification approaches a thousand, FGAP-RBF algorithm causes a PC to crash after only a few hidden neurons are added. This led us to look into a fast and efficient batch learning algorithm called Extreme Learning Machine (ELM) for this application as it can handle problems of very large input dimension.
- Extreme Learning Machine (ELM) is a batch learning algorithm, developed by Huang *et al* [52], and here it is studied for the first time on six multi-category microarray gene expression based cancer diagnosis problems. Its performance has been compared with other methods such as ANN, SANN and SVM algorithms. Study results show that when the number of categories for the classification task is large, ELM algorithm achieves a higher classification accuracy than the other algorithms with a lesser training time and a smaller network structure. It is also seen that ELM achieves a better and more balanced classification for individual categories as well. For applications with lower number of categories, ELM achieves a similar accuracy with much less training time and a compact structure compared to SVM-OVO.

1.4 Organization of the Thesis

The thesis is organized as follows:

Chapter 2 provides a literature review of classification methods and an overview of bioinformatics applications with an emphasis on microarray gene expression based

cancer diagnosis problems.

The thesis is broadly divided into two parts. The first part covers sequential learning algorithms and their use in general classification problems. The second part mainly considers bioinformatics classification problems and here both sequential learning and fast batch learning methods are studied.

In the first part, in chapter 3, the GAP-RBF algorithm is introduced and its performance is then evaluated and compared with another sequential learning algorithm—MRAN and the conventional MFNs on three benchmark problems in the classification area from the PROBEN1 database. The problems considered are : 1) Hearta Problem; 2) Cancer Problem; and 3) Gene Problem. In Chapter 4, two difficulties with the GAP-RBF algorithm for applications with non-uniformly distributed high dimensional inputs are identified and improvement schemes are proposed. A new algorithm which improves GAP-RBF and referred to as Fast GAP-RBF (FGAP) is presented. Chapter 5 presents the performance evaluation of the FGAP-RBF algorithm together with GAP-RBF and MRAN as well as a batch learning algorithm—SVM on four real world benchmark classification problems, viz 1) Phoneme Problem [6]; 2) Segment Problem [15]; 3) Satimage problem [15] and 4) DNA Problem [15]. The influence of the FGAP algorithm parameters and the guidelines for their selection are also explored in this chapter.

In the second part of the thesis, our investigations mainly focus on microarray gene expression based classification problems. In chapter 6, the FGAP-RBF algorithm is used for a microarray gene expression based multi-category classification problem using the GCM dataset. Its performance is also compared with other methods from the literature. In Chapter 7, a comprehensive evaluation of ELM for multi-category microarray classification problems is also carried out on six real benchmark microarray datasets on multi-category cancer diagnosis, viz. the GCM dataset, HBC dataset, Lung dataset, Lymphoma dataset, MLL dataset and NCI60 dataset.

Finally, conclusion and future work are summarized in Chapter 8.

Chapter 2

A Review of Methods for Pattern Classification and Bioinformatics Classification Problems

The term “pattern recognition” is defined as the act of taking in raw data and making an action based on the “category” of the pattern in [26]. The goal of pattern recognition is to classify patterns into a number of categories or classes [132]. It has been an important subject of interest in a variety of engineering applications and scientific disciplines that encompasses a wide range of information processing problems of great significance, such as speech recognition [92], machine vision [124], remote sensing [119], image analysis [66], character recognition [74], biometric recognition [16] (e.g. face, fingerprint or iris identification), medical diagnosis [24], DNA sequence identification [76], microarray gene expression diagnosis [12] and many other applications [133].

There are two kinds of pattern recognition tasks. One is the category of problems where the training samples and the associated class labels are given, and the aim is to build a classifier that can predict the class label for any given sample. In the

context of this thesis, this process of establishing the supervised learning classifier is called pattern classification. The other kind is clustering, where the training samples are not labelled and the aim is to group samples into different clusters according to their natural intrinsic features that distinguish one cluster from another.

Pattern classification is the major focus of the research reported in this thesis. Over the past few decades, a number of supervised learning algorithms have been developed for pattern classification applications. These supervised learning algorithms include statistical classification methods (such as Bayesian analysis [10] and k -nearest neighbor [31]), linear discriminant methods (such as Fisher's linear discriminant [30] and support vector machines [118]) as well as nonlinear discriminant methods such as decision trees [18] and Artificial Neural Networks (ANNs) [46]. ANNs have been widely used in the field of pattern classification and they show their advantages over other methods in their learning, generalization and adaptation capability as well as their unique power for nonlinear mapping. However, the traditional gradient-based batch learning algorithms, which are most often used in the training process, often cause an ANN to get trapped in a local minimum and leads to poor classification results. Besides, a high computational complexity and a large training time are also serious problems when the training dataset is extremely large or the problem is very complex. Therefore, sequential learning algorithms and non-gradient batch learning algorithms, are the natural candidates for solution and arouse more research interest. In this chapter, first, a brief introduction of major pattern classification methods is given, where a detailed review of artificial neural network algorithms for classification problems is presented. Then, an overview of the problems and classification methods in the bioinformatics area are presented. Finally the problems with the current research methods are also discussed.

2.1 Conventional Algorithms for Classification

2.1.1 Statistical Methods

Statistical methods are the most commonly used methods for pattern classification problems. Statistical methods train classifiers based on the probabilistic nature of the data and the decision is made based on the estimated probabilities. A good review of statistical methods for pattern recognition is given in [61]. Among these methods, Bayesian analysis is most commonly used and Bayesian Decision Theory has laid the basic theoretical foundation for most of the algorithms of this category. The earliest application of Bayesian Decision Theory for pattern classification appeared in [23]. Some good references for Bayesian data analysis are [10] [11] [38].

Let $\omega_1, \dots, \omega_c$ be a set of c classes, and let $\mathbf{x}$ be a random feature vector. Bayes' formula can be written as:

$$p(\omega_j|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_j)p(\omega_j)}{p(\mathbf{x})} \quad (2.1)$$

where $p(\omega_j|\mathbf{x})$ gives the probability of the pattern $\mathbf{x}$ belonging to class ω_j . It is also called the class posterior probability. The probability of misclassification is minimized when the class with the largest posterior probability is selected. Therefore, for a pattern $\mathbf{x}$, if $p(\omega_i|\mathbf{x}) > p(\omega_j|\mathbf{x})$ for all $i \neq j$, it should be assigned to class ω_i . $p(\mathbf{x}|\omega_j)$ is the class conditional probability density of $\mathbf{x}$ for class ω_j . It is also referred to as the likelihood. $p(\omega_j)$ is the probability of class ω_j , which is often referred to as prior probability. $p(\mathbf{x})$ is the probability of $\mathbf{x}$ with respect to the whole data set, regardless of the class membership. Since it is the same for the patterns of all the classes, it is often referred to as a normalization factor for the Bayesian decision based approaches. The Bayes formula can also be summarized in the following

form:

$$\text{posterior probability} = \frac{\text{likelihood} \times \text{prior probability}}{\text{normalization factor}} \quad (2.2)$$

For many classification applications, the major problem in applying the Bayesian approach is the difficulty of obtaining the conditional densities $p(\mathbf{x}|\omega_j)$, viz. likelihood. A number of approaches have been proposed for the estimation of the probability density. Most of the approaches fall into two categories, viz., parametric methods and nonparametric methods.

Parametric methods assume a specific functional form for the estimated density and try to optimize the parameters to fit the assumed function to the data. Two commonly used approaches are the maximum-likelihood estimation and Bayesian estimation. The maximum-likelihood method tries to find the best parameters to fit the training data and maximize the probability of obtaining the samples actually observed [26]. In Bayesian estimation, the parameters are considered as random variables having a known prior density and are updated incrementally by the training data, which converts the distribution on the parameters to a posterior probability [26]. The drawback of the methods in this category is that the assumed functional form may not be able to represent the true density of the data precisely no matter how the parameters are optimized and adjusted.

On the other hand, nonparametric methods do not assume a certain form for the density function. They allow the density function to be of any arbitrary form and the probability density is solely decided by the data. One type of nonparametric method is to estimate the likelihood from the training patterns. Parzen window approach was introduced to estimate the density by using a window sampling the training patterns [99] and Specht was the first to use it for the classification applications [125]. Another method is to estimate the posterior probability directly from the training samples. The k -nearest neighbor method introduced by Fix *et al* [31] [32] is a well

known method belonging to this type. It classifies the sample to that class, which has the most members of its k -nearest neighbors. In other words, the decision is made by the vote of the k nearest neighbors' labels [26]. The drawback of nonparametric methods is that the number of parameters in the model grows with the size of data set, so that the model quickly becomes unwieldy [14].

2.1.2 Linear Discriminant Methods

Beside statistical methods based on probability densities, an alternative approach for pattern classification is to use discriminant functions. Discriminant functions are simply a set of functions $y_1(\mathbf{x}), \dots, y_c(\mathbf{x})$ that decide the membership of the pattern $\mathbf{x}$ by the following criterion. If $y_k(\mathbf{x}) > y_j(\mathbf{x})$ for all $j \neq k$, then the input pattern $\mathbf{x}$ is assigned to class C_k . When using discriminant methods, usually a specific discriminant function is chosen and the optimal parameters are obtained by a suitable training algorithm that learns from training patterns.

Linear discriminant functions [30] are the simplest form of discriminant functions and they have been considered widely in the literature as conventional methods for classification [14]. Linear discriminant functions are linear combinations of the input variables and weights. The simple discriminant can also be generalized by transforming the linear combination with a non-linear activation function, or transforming the input variables with nonlinear activation functions before forming a linear combination.

The general approach for linear discrimination is to form a certain criterion function, then perform pseudo inversion of the matrices for small problems, or perform gradient-based learning if the problems are large and complex ones.

Each criterion function has its own advantages and disadvantages related to the computational cost and convergence. There is no clear winner for all circumstances.

The perceptron algorithm [88] uses the inner product of the patterns in each class as the criterion function, while the least-squares method and Fisher's linear discriminant [30] try to minimize the summed squares of the error. Gradient descent is used in perceptron algorithm and least-squares method, while Fisher's linear discriminant performs the pseudo inversion of the input matrices to get the optimal parameters.

The support vector machine [118] is a type of linear discriminant method with very high performance. It is different from other linear discriminant methods in that its input is mapped by a nonlinear function to a high dimensional space first and then an optimal hyperplane that produces the largest separation margin is found. The support vectors are those transformed patterns that determine the margin. They are around the decision boundary and usually, they are the most difficult patterns to classify and the most informative ones for designing a classifier [26].

The drawback of linear discriminant methods is that they may not provide accurate models for some complex problems. They are insufficient to cope with challenging pattern classification problems, such as those involving multimodal or nonconvex densities.

2.1.3 Nonlinear Discriminant Methods

Nonlinear discriminant methods can be considered as a generalized form of linear discriminant methods with transformation of nonlinear activation functions. In theory, nonlinear discriminant methods can handle classification problems with any boundary shape and provide the optimal solution to an arbitrary classification problem. Popular and powerful nonlinear discriminant methods include artificial neural network methods and decision trees. Artificial neural networks are a major focus of this study and their review will be presented in detail in the next section. Here we discuss briefly about the decision tree methods only.

2.1.3.1 Decision Trees

Decision trees [18] differ from other methods in the ability in solving classification problems involving nominal data [26]. Nominal data is different from metric data in that it does not have any natural notions of degree of similarity or ordering that can be measured. Examples of such nominal features are shape of an object, taste of a food, etc.

Decision trees are built upon a sequence of questions. The first question forms the root node of the tree which is connected by successive links to other nodes until the terminal nodes are reached. It is desirable to choose a feature for each node that makes the data reaching the immediate descendant node as pure as possible, so that a simple and compact tree with few nodes can be built [26]. Several mathematical measures of impurity have been proposed, such as misclassification, variance, Gini [81] and entropy. Among these, the entropy impurity has been found of the greatest use. To overcome overfitting and improve generalization performance of decision trees, two methods can be used. One is stopped splitting [18], which declares a node with non zero impurity to be a terminal node, and the other method is pruning [109], which reduces a tree to minimum impurity terminal nodes. The classification of a particular pattern starts from the root node, which asks for the property of a particular feature of the pattern. The different links direct to different possible answers. Based on the answer, we choose the right node to continue the classification process until it reaches a terminal node and its membership is found. In decision trees, the links must be mutually exclusive and exhaustive. One and only one link can be followed. Decision tree based methods include CART [18], ID3 [108] and C4.5 [109], etc. A good overview of the decision tree methods can be found in [89].

One advantage of tree classifiers is that the results are very easy to interpret by following the structure of the trees. Tree classifiers are also very flexible and can be used for a wide range of problems, including those with data that are metric, non-

metric or both. However, decision tree based methods are usually computationally intensive. They also lack continuity in predicted values. Overfitting is another serious problem, which often occurs in decision tree methods, that leads to suboptimal and unstable trees.

2.1.4 Artificial Neural Networks

Artificial Neural Network (ANN) methods [46] [14] [112] are powerful tools to capture and represent complex input-output mappings. ANNs are composed of a number of basic computational units called neurons, where linear or nonlinear transformations of the input data are made. The neurons are connected by synaptic weights. ANNs are capable of representing both linear and non-linear relationships and learning these relationships directly from the data. ANNs usually contain one input layer, one output layer and a number of hidden layers.

ANNs can be broadly classified into two categories according to the flow of the input information, viz., feedforward neural networks and recurrent neural networks. In feedforward neural networks, input information only flows from one direction from one layer to another and there is no feedback loop in any other two layers in the network. In recurrent neural networks, there is at least one feedback loop in the network structure. The presence of feedback loops has a profound influence on the learning capability as well as the performance [46]. The feedback loops can involve the use of unit delay elements that can result in a nonlinear dynamical behavior. Therefore, recurrent networks are mostly found in time series predictions [26]. In this study we only concentrate on feedforward neural networks, which are most often used in the classification area. The single-layer feedforward neural network is a linear discriminant method. However, multilayer neural networks are typically nonlinear discriminant methods. “Multilayer neural networks” refers to neural networks that contain at least one hidden layer. Or put in another way, multilayer neural networks

Figure 2.1: An example of MLP with two hidden layers

refer to the neural networks with two or more layers of adjustable weights. Multilayer perceptrons and Radial Basis function neural networks are the two major network models for multilayer feedforward neural networks.

2.1.4.1 Multi-Layer Perceptrons

The feedforward multi-layer perceptron (MLP) model was proposed by Rumelhart, Hilton and Williams [116] [115]. A graphical representation of an MLP is shown in Figure 2.1. The input patterns are fed into the MLP through the input layer and the weighted inputs are fed forward to the hidden layers. Each hidden layer computes the weighted sum of the outputs from the previous layer, then performs a nonlinear transformation using the activation function and produces an output which serves as the input for the next layer. Different layers are connected by weights. The sigmoid function is the most commonly used activation function for MLPs. The purpose of training such a network is to find the optimal weights that can minimize the error

function. For classification problems, the aim is to find the optimal weights that can produce the least number of misclassifications. There are mainly two error functions for classification problems. One is the sum-of-squares cost function [98] and the other is the cross entropy cost function [8]. The outputs of ANNs can be used to estimate posterior class probabilities for classification problems, but there is a natural pairing of the error function and activation function for output neurons [14]. Only when sum-of-squares error cost function is used together with a linear activation function for the output neuron, or cross-entropy error cost function is used together with a logistic activation function, the outputs of the network can be interpreted as posterior class probabilities.

An evaluation of these two cost functions for training the neural network classifiers has been done by Kline and Berardi [69] and they have found that theoretically cross-entropy can yield statistically better posterior probability estimation compared to the sum-squared-error cost functions. However, the cross entropy is relatively computationally expensive and studies show that when comparing the entropy based objective functions to sum-of-squares errors, there are few differences in the observed error rates [44] [111].

One of the most popular methods for training multilayer neural networks is the back-propagation algorithm [116] [115], which is based on the gradient of error $\partial E / \partial w$. With backpropagation, the input data is repeatedly presented to the neural network. The output of the neural network is compared to the desired output and an error is computed. This error is then fed back (backpropagated) to the neural network and used to adjust the weights such that the error decreases with each iteration and the neural model gets closer and closer to producing the desired output. The training stops when a stopping criterion is satisfied, which is usually a predefined maximum of training epochs or a threshold for the change of the weights. Other methods for this purpose were also proposed, such as cross validation by Stone [128]

and early stopping by Morgan and Bourland [92]. Later improved methods on the backpropagation algorithm include methods using an updating scheme with the momentum [43] term and updating with second order information [20]. More detailed developments of these can be found in Haykin [46] and Bishop [14].

The backpropagation algorithm is straightforward and easy to use. However, the objective function of the MLP model is not convex with respect to its parameters and the backpropagation algorithm often gets stuck in a local minimum. Therefore, several training processes with different initial weights need to be performed in order to find a good solution. This further increase the computational effort and has been considered as the major disadvantage of MLP models and the associated backpropagation algorithm [80].

2.1.4.2 Radial Basis Function Networks

The Radial Basis Function (RBF) [104] neural network is another type of multilayer feedforward neural network. It differs from MLPs in two aspects. First, the activation functions for MLPs are often chosen as sigmoid functions, while for RBF neural networks, radial basis functions are used. Second, the most popular training algorithm for multilayer neural networks is the backpropagation algorithm. However, since RBF networks only have one layer of adjustable weights, the traditional backpropagation algorithm is no longer a good method for training RBF networks.

Radial basis function neural networks were originally motivated by the locally tuned response in biological neurons. The hidden neuron activations are determined by the distances between the input vector and prototype vectors. RBF neural networks have been popular in the past decade due to its fast training and universal approximation capability that it can approximate any continuous functions with arbitrary precision [120].

Figure 2.2: General architecture of an RBF neural network.

Usually RBF networks contain three layers: the input layer, the hidden layer and the output layer. Input samples are fed into the the hidden layer with a unit weight through the input layer. The hidden units provide nonlinear transformations by a set of radial basis functions that constitute the bases for the inputs when they are mapped into the space of the hidden neurons. The output layer of RBF network only computes the linear combination of the outputs from the hidden layer. General architecture of an RBF neural network is shown in Figure 2.2.

Initially, RBF networks were developed for function approximation problems [104]. It can be mathematically expressed as follows: given a set of input and target patterns $(\mathbf{x}_n, y_n)$, $n = 1, 2, \dots, N$, where $\mathbf{x}_n \in R^l$ and $y_n \in R^q$, find a function

$f(.) : R^l \rightarrow R^q$ such that

$$f(\mathbf{x}_n) = y_n, n = 1, 2, \dots, N \quad (2.3)$$

For RBF networks, the function $f(.)$ usually takes the following form:

$$f(\mathbf{x}) = \sum_{k=1}^K \alpha_k \phi_k(\|\mathbf{x} - \boldsymbol{\mu}_k\|) + \phi_0, \quad (2.4)$$

where ϕ_0 is a bias term and $\phi_k(.)$ is a set of arbitrary radial basis functions. $\|\cdot\|$ denotes the norm and it is usually taken as Euclidean distance. The predetermined point $\boldsymbol{\mu}_k \in R^l$, $k = 1, 2, \dots, K$ is ‘the center’ of the radial basis function of the k th hidden neuron. α_k , $k = 1, 2, \dots, K$ is a set of linear weights between the output layer and the k th hidden neuron. Typical choices for radial basis functions are:

1. thin-plate-spline function

$$\phi(\mathbf{x}) = \|\mathbf{x} - \boldsymbol{\mu}\|^2 \log(\|\mathbf{x} - \boldsymbol{\mu}\|); \quad (2.5)$$

2. multiquadratic function

$$\phi(\mathbf{x}) = (\|\mathbf{x} - \boldsymbol{\mu}\|^2 + \beta^2)^{1/2}; \quad (2.6)$$

3. inverse multiquadratic function

$$\phi(\mathbf{x}) = \frac{1}{(\|\mathbf{x} - \boldsymbol{\mu}\|^2 + \beta^2)^{1/2}}; \quad (2.7)$$

4. cosine function

$$\phi(\mathbf{x}) = \frac{\beta}{(\|\mathbf{x} - \boldsymbol{\mu}\|^2 + \beta^2)^{1/2}}; \quad (2.8)$$

5. Gaussian function

$$\phi(\mathbf{x}) = \exp(-\|\mathbf{x} - \boldsymbol{\mu}\|^2/\beta^2). \quad (2.9)$$

Here β is a parameter of the radial basis function to be decided by the users.

Among these radial basis functions, the Gaussian function is the most widely used one. It is found to be capable of making an accurate global mapping with refined local details. Compared with other radial basis functions, the Gaussian function has two advantages. First, the value of the Gaussian function decreases monotonically with the growth of distance from the center, which makes the Gaussian function local in its response. It is more plausible from the biological point of view, because the response is finite. Second, both the position and shape of Gaussian function are more flexible to adjust compared with other radial basis functions. However, it is also hard to adjust formation of prototypes of the Gaussian RBF network using gradient-based methods [63].

To train Gaussian RBF networks, four types of parameters need to be adjusted. They are the number of hidden neurons, the centers of hidden neurons, the widths and the weights that connect the hidden neurons and the output neurons. There are many algorithms that have been proposed for training Gaussian RBF networks and they can be summarized into a few categories according to their schemes to obtain these parameters.

The optimal weights between the hidden layer and the output layer can be obtained by many methods. One is the regulation method which determines the weights by matrix computation [46]. Another type of methods are the gradient-based methods [90]. Least mean squares is one of the most often used gradient methods [60]. The Linear Least Squares method was used in [34] to obtain the optimal weights between the hidden layer and the output layer. In [29], an evolutionary strategy is used to minimize the mean square error to avoid the problem of local minima and

increase the training speed. Karayiannis [63] also proposed an axiomatic method for formulating RBF networks by selecting admissible generator functions, which determine the form and properties of the RBF's. The reformulated RBF networks can be trained by the simple and easily implementable gradient descent learning methods and perform considerably better than the conventional RBF network methods. Criteria for selection of these generator functions and evaluation of different generator functions (linear and exponential) are also discussed in his later work [65].

There are several ways to determine the center and the width of hidden neurons. One way is to select the centers of hidden neurons randomly from the input patterns. The widths are set to a predefined value. In some other methods, the center and width for each hidden neuron are obtained using various clustering methods [34] [60] [21], such as learning vector quantization, the k -means clustering, and C4.5. The width of the neuron is set as the standard deviation of the cluster. In [144], a fast orthogonal estimation algorithm is derived for detection and parameter estimation (centers and weights) of RBF networks. Simulations results demonstrated the improved efficiency of this procedure. In [22], Chen *et al* proposed an orthogonal least square learning algorithm to choose the centres for RBF networks, which maximizes the increment to the energy of the desired output. In [41], Gonzalez *et al* presented a multiobjective evolutionary algorithm to optimize the parameters of RBF network. It yields an improved procedure to adjust the network parameters by including new global mutation operators, which is obtained based on singular value decomposition and orthogonal least squares, to incorporate expert knowledge into the network.

The determination of the number of hidden neurons is the most tricky part of the training algorithm. Too small a number of hidden neurons does not allow for the reduction of the error to a satisfactorily low level, while too large a number of hidden neurons destroys the generalization ability and leads to the problem of overfitting [97]. In some algorithms, the number of hidden neurons is obtained by

an unsupervised clustering method [34] [60]. However, for most supervised learning algorithms, the number of hidden neurons is obtained by trial and error. Therefore, a sequential learning method, which can adjust the number of neurons automatically during the training is more appropriate.

When an RBF network is used for classification problems, the training of the network is generally done by using batch learning algorithms. In batch learning, the input data set is divided into a training set and a validation set and training is done using the training set data over a number of epochs. The validation set is used to avoid overtraining and finally the classification accuracy is assessed based on the test data.

In practical classification applications, even though one starts initially with a large data set, new training data becomes available from time to time and it arrives sequentially. To handle the training of the network for the new data, batch learning schemes require that the network be retrained, resulting in a large training time. A learning scheme that handles the sequential data without retraining with the complete data set will be quite useful in real applications.

A second problem with the popular learning algorithms for RBF networks is that the network structure is to be fixed *a priori*. It is known that the number of hidden neurons should match the complexity of classification applications. In [83], Lowe demonstrated that the complexity of a RBF network can be characterized by its intrinsic degrees of freedom, which can be estimated by a spectral analysis of the output space of hidden neurons. In real applications, the size of the network is always a critical point. Also when the data arrives sequentially, fixing the network size *a priori* may not be the right approach. In this case, a growing and pruning strategy, which can adjust the number of neurons automatically is more appropriate.

In the literature, there are many training methods developed for RBF networks, which change the number of the hidden neurons dynamically [101] [62] [64] [117] [141]. In 1991, the Resource Allocation Network (RAN) was developed. It has

a dynamic network topology by adding hidden neurons sequentially based on the novelty of the input data [101]. A new input pattern is considered as novel if it is far away from the existing centers and if the error between the output and the target is large. If the input pattern does not pass the criteria for novelty, no hidden neuron is added and the network parameters are adjusted using the Least Mean Square (LMS) algorithm. RAN has been used for automated detection, segmentation and classifications of breast cancer nuclei by Lee and Street [73]. They show that this approach provides faster and more accurate nuclei detection and segmentation. The online learning ability of the RAN also gives the analysis system improved performance with continued use.

Improvement on the RAN was made in [62] by using the Extended Kalman Filter (EKF) rather than the LMS method for updating the network parameters, including centers, widths, and weights. This paper shows that compared with LMS, the EKF method improves the convergence rate of RAN and further reduces the network complexity. However, there is a problem with both RAN and RANEKF. Once a neuron is created in the network, it can never be removed. A hidden neuron may be active initially, but it may end up with no contribution to other input patterns. Thus both RAN and RANEKF can lead to overfitting networks with excessive number of hidden neurons.

In 1997, Yingwei *et al* [141] [142] made an improvement on RANEKF by introducing a pruning scheme on the basis of the normalized contribution of each hidden neuron. The new method was named as Minimal RAN (MRAN). MRAN employs the same two criteria for adding neurons, which have been used in the RAN and RANEKF. Besides these two criteria, MRAN also measures a Root Mean Square (RMS) value of the output error over a sliding window in order to reduce the effect of noise and ensure smooth changes of the number of hidden neurons. When a new input pattern satisfies these three conditions, a new hidden neuron is added. If not, the parameters

are updated by the EKF method. After that, MRAN uses a pruning process to get rid of inactive neurons. The contribution of each hidden neuron is measured by the normalized output. If it is below a threshold for a consecutive number of samples, the neuron is removed.

Another method was developed by Rojas *et al.* [113]. The growing process of the network and the pruning process are carried out separately with all the training patterns and in that sense the method is not a real sequential learning algorithm. In the growing process, a new input pattern is deemed as novel when the output error is big and the input pattern is not covered by any of the existing neurons. The network grows until all the training patterns are presented. Then the pruning process begins. The criteria for pruning a neuron include the output of the hidden neuron, the range of the activation region and the activation similarity of the neuron to any other hidden neurons. These pruning strategies help the network to realize a more simple topology and achieve better performances.

It should be noted all these strategies also bring further problems. Firstly, too many criteria means that too many parameters and too many thresholds have to be selected. Choosing the proper parameters can only be done by trial and error simulation studies. They limit the potential applications of these methods. Secondly, although the parameters for these criteria are separately selected, the criteria might be potentially related to each other. Thus, the subjective selection of these parameters is often conflicting and causes oscillations in the hidden neuron numbers. Sometimes the improper matching of the parameters can even disable the scheme and deteriorate the performance of the network completely.

Recently a new method called Growing And Pruning RBF (GAP-RBF), which adds and prunes hidden neurons based on a simple estimation of the neuron “significance” has been developed [52] [53]. Studies for function approximations show that GAP-RBF can realize a much smaller network in a smoother and faster way than

RAN, RANEKF and MRAN. It can achieve a higher learning accuracy and better generalization performance as well.

Major merits of GAP-RBF are summarized as follows:

1. The computational complexity is reduced greatly, by introducing the notion of “significance” of a neuron, and using a piecewise linear approximation to the Gaussian function.
2. For pruning, it has been proved that only the nearest neuron (based on the Euclidean distance to the current input pattern) needs to be checked. This will give GAP-RBF a great advantage in applications where a large number of hidden neurons are needed.
3. When no hidden neuron is to be added, only the parameters of the nearest neuron need to be adjusted and this reduces the overall computation and increases the learning speed.

A detailed description of the GAP-RBF algorithm is given in Chapter 3. Also, the performance evaluation of GAP-RBF for classification problems are presented in this chapter .

As the second part of this thesis focuses on the classification problems in the bioinformatics area, in the next section, we present an overview of the bioinformatics classification applications and the methods proposed for these problems.

2.2 Overview of Bioinformatics Classification Problems

Bioinformatics can be simply defined as the computational storage and manipulation of biological information [45]. In [85], Luscombe *et al.* define bioinformatics as “conceptualising biology in terms of molecules and applying informatics techniques to understand and organize the information associated with these molecules, on a large scale”. As the name ‘bioinformatics’ indicates, this conception includes two aspects [17]. One aspect focuses on the information flow based on molecular biology. DNA sequences are transcribed into mRNA sequences, and mRNA sequences are transcribed into protein sequences. Protein sequences fold into three dimensional structures that have functions. These functions are selected by the environment of the organism that drives the DNA sequences within a population [17]. From this point of view, bioinformatics application is the study of the transfer of information at any stage in this process, including the organization and control of the genes in the DNA sequences, the identification of transcriptional units in DNA, the prediction of protein structures from the sequences, and the analyses of molecular function. Generally speaking, bioinformatics applications involve all the problems in biology above the cellular level.

The other aspect of bioinformatics focuses on the informatics techniques, which are scientific methods derived from disciplines such as applied maths, computer science and statistics, that are used to understand and organize the information associated with these molecules [85]. These methods mainly serve three purposes. First, to organize data in a way that allows researchers to access existing information and to submit new entries as they are produced. Second, to develop tools and resources that aid in the analyses of the data. Third, to use these analysis tools and interpret the results in a biologically meaningful manner.

One major property of bioinformatics problems is their extremely large quantities of data [76]. For example, a raw DNA sequence is a string of the four base-letters comprising genes, each typically 1,000 bases long. The GenBank [9] repository of nucleic acid sequences holds more than 12.5 billion bases in 11.5 million entries [85]. Most bioinformatics analyses mainly focus on three sources of primary data [85]:

- DNA or protein sequences. For DNA sequences, investigations mainly involve gene recognition, gene identification and gene structure prediction [135]. For protein sequences, investigations include sequence comparison [87], multiple sequence alignments [40] and searching for functional domains from conserved sequence motifs in such alignments.
- Macromolecular structures. Investigations of structural data include secondary and tertiary protein structure prediction, producing methods for 3D structural alignments [96], examining protein geometries using distance and angular measurements, calculations of surface and volume, and analyses of protein interactions with DNA, RNA and smaller molecules.
- The results of functional genomics experiments, such as gene expressions. Research includes characterization of protein contents and metabolic pathways between different genomes, identification of interacting proteins, assignments and prediction of gene products, and large scale analyses of gene expression levels.

The intensive interest in bioinformatics has been driven by the emergence of experimental techniques that generate great amounts of data, such as DNA sequencing, mass spectrometry, and microarray expression analysis [17]. Bioinformatics depends on the availability of these large data sets and bioinformatics applications usually have to deal with these data sets. These problems are so big and so complex that it is impossible for them to be analyzed by people manually.

Microarray technologies [19] [71] allow the monitoring of expression levels of thousands to ten thousands of genes simultaneously in any given cell of organisms, cell lines or human tissues. The study of microarray data has attracted more and more interest and it has been an important part of bioinformatics research over the last few years [39] [77] [59] [1].

In the tumor and cancer classification area, gene expression profiling technologies have attracted more and more interest over conventional methods relying macro and microscopic histology and tumor morphology for a number of reasons. First, gene expression profiles offer more information than conventional methods, especially where the histopathologic appearance are similar but the clinical course and response to treatment vary significantly. An accurate classification of different tumor types is essential to the treatment and toxicity minimization in clinical practice. Second, it allows the monitoring of expression levels for thousands of genes simultaneously, which can create a more comprehensive overview of the change of expression levels for each individual gene in a specific tissue under various conditions. Hence it provides more information for a better and more reliable classification. Third, the gene expression levels can provide extremely useful biological information. They are known to contain the keys to fundamental problems, including the prevention and cure of diseases, biological evolution mechanisms and drug discovery. Some discriminant genes in classifying tissue types can be found of great value for further investigations of the disease and therapies.

There have been many classification methods used for cancer classifications both from statistical and machine learning area, but some unique characteristics of gene expression data have made this classification a difficult task. First, gene expression data usually has a very high dimension and often contains thousands to tens of thousands of genes. Second, the amount of the data available is usually very small, often below 100. It is recognized that the higher the ratio of the number of training

samples to the number of free classifier parameters, the better the generalization capability of the resulting classifier. The large number of attributes of microarray data are directly translated into the number of free parameters. Therefore, for microarray data classifications, the ratio of the number of training samples to the number of free classifier parameters is extremely low. It is considered as the major difficulty for microarray data classification problems. Third, most of the genes are irrelevant to cancer classification. This large number of useless genes not only increase the computational complexity and cost, but also bring in noise and compromise the generalization performance. Fourth, the amplification of mRNA from a single cell is extremely difficult under the current state of technology. Therefore, tissues seemingly with the same function are pooled together to obtain a certain amount of mRNA. Expression levels are the mean values of all the cells in the pool [79]. Fifth, genetic variability affects gene expression. Expressions of the same gene for two individuals can be different. Last, noise is introduced into the system at various points of the experiment of getting the micro-array data, which could affect the outcome.

Over the past few years, binary sample classifications using gene expression data have been extensively studied [39] [77] [59] [1] [33] [102] [143]. However for multi-category classifications involving more than 2 classes, only a small amount of work has been done so far [110] [94] [79] [7]. Studies also indicate that multi-class classification problems are much more difficult than the binary ones and need more and further investigation. Classification accuracies drop quickly with an increasing number of classes. Datasets with a large number of classes are especially difficult and the prediction accuracies are very low.

So far some classification methods have been applied to applications using gene expression profiling. Some commonly used algorithms include weighted voting of informative genes [39], discriminant analysis [59] [94] [4], Decision Trees [25] [138],

Nearest Neighbor methods [7] [77], Support Vector Machines [35] [110] [139] and Neural Networks [68] [82] [131].

The weighted voting method was proposed by Golub *et al* [39] in 1999. The assignment of the class for a new sample is done based on the weighted votes of a set of informative genes. The informative genes are selected according to their correlation with the class labels. Each of the informative genes casts a weighted vote for one class and the votes are summed together to obtain the total votes for each class. The sample will be assigned to the class with the highest total weight if the prediction confidence exceeds a predetermined threshold. Golub *et al* have applied this method to Leukemia dataset and achieved very good results. However, this method is only applicable to binary classification problems. It does not work well on datasets where there are unequal number of genes that favor the two classes with the same correlation strength [84].

Recently, Support Vector Machines (SVM) have been widely used for cancer classification problems using microarray data [35] [110] [139] [102] [75]. Furey *et al* [35] applied an SVM algorithm for the analysis of microarray gene expression data on ovarian cancer, leukemia [39] and colon tumor datasets [3]. The results are promising and comparable to those previously obtained. However, the SVM algorithm originally was developed for binary-class problems and it can not be applied to multi-category classification problems for microarray gene expression data directly. Some combination schemes have been used to modify SVM for multi-category classification problems. Ramaswamy *et al* [110] have used a one-versus-all scheme to perform multi-classifications using the SVM. For a c -category classification problem, c binary classifiers should be built for SVM to distinguish one class from all the rest of the classes. This method has potential drawbacks when there is considerable overlapping between the classes. Similarly, it can also be implemented in a one-versus-one fashion. When using a one-versus-one comparison approach, $c(c-1)/2$

binary classifiers should be built for the SVM to distinguish between every pair of classes. Thus, it can be seen that when the number of classes c increases, the complexity of the overall classifier also increases. Besides, the one-versus-one approach often exhibits large variability [75]. Although Lee *et al* [75] have proposed a SVM method for microarray multi-category classification problems, it is computationally expensive and its classification performance is only comparable to other methods. However, the effectiveness of the SVM algorithms for microarray classification applications, is still an on-going research problem [84].

Neural network classifiers have been well established for their capability to model any linear or nonlinear mappings between the input and output. Compared with the SVM, neural networks try to classify the inputs directly into a number of classes. Another advantage of neural network methods is that they can be easily adapted to predict continuous variables instead of discrete class labels. This can be applied to the cases where we need to predict the level of a medical indicator rather than classify the samples into binary categories. The first application of ANNs for diagnostic classification of cancer using gene expression data was presented in [68]. Khan *et al* [68] used a two-layer feed forward NN to classify the small, round blue-cell tumors (SRBCT) into 4 diagnostic categories based on gene expression signatures. The ANN method correctly classifies all the samples, even for the samples that often present diagnostic difficulties in clinical practice. Recently Linder *et al.* developed a new neural network-based algorithm for multi-category microarray analysis [82]. The benchmark dataset GCM [110] has been studied using both a single feed-forward neural network and a new ANN based algorithm called as the Subsequent ANN (SANN). The SANN method performs a preselection by a simple ANN at the first stage that narrows down the decision scope by selecting the two most preferred classes with highest activities at the corresponding output neurons. After that a second ANN is applied for the final decision on these two selected classes at the second stage. The second stage of the SANN is also a pairwise comparison in essence.

The results show that simple ANN produces very poor classification accuracies and SANN produces a high accuracy that beats the best classifier described in [110]. However, SANN causes a great increase in the network complexity and a big loss in the training speed.

Many comparisons of different classification algorithms using microarray data analyses have been presented in the literature. Yeang *et al* [139] have made a comparison of three binary classifiers, viz., the k -nearest neighbors method, the weighted voting method and the support vector machine, in conjunction with three combination schemes that comprise of one-versus-all, one-versus-one and hierarchical partitioning schemes. The simulation results show that the one-versus-all support vector machine produces the smallest cross validation error and test error when all the genes are used. However, for weight voting and the k -nearest neighbor method, the one-versus-one method tends to outperform one-versus-all methods when a fixed number of selected genes is used. Dudoit *et al* has made a comparison of classifications methods using discriminant analysis methods, nearest-neighbor classifiers, decision trees and aggregating classifiers [27]. Three datasets, the lymphoma dataset [2], leukemia dataset [39], and NCI60 dataset [114] are used in this study. In the main comparison, the nearest neighbor method and the diagonal linear discriminant method have the lowest error rates and the Fisher linear discriminant method has the biggest error when an intermediate number of genes selected by the "between group and within group" ratio are used. A comparison of classification methods for multi-category gene expression problems has been carried out by Li *et al* [79]. This paper compares various feature selection methods together with many classification methods on various multi-category gene expression dataset. The classification methods include support vector machines, naive Bayes, k -nearest neighbors and decision trees. Eight microarray gene expression datasets, Leukemia dataset [39], ALL dataset [140], GCM dataset [110], SRBCT dataset [68], MLL dataset [5], Lymphoma dataset [2], NCI60 dataset [114] and HBC dataset [47] are

studied. The results show that the SVM is the best classifier for gene expression based classification. It outperforms all the other classifiers for almost all the datasets. k -nearest neighbor method is the second best classification method and it performs better than decision trees and naive Bayes for most of the datasets. For gene selection methods, there is no clear winner for all datasets. The study also shows that gene expression-based multi-category classification problems are far more difficult than its binary counterparts and the results deteriorate with an increase in the number of the categories. In 2005, Statnikov *et al* presented a comprehensive evaluation of multi-category classification methods for gene expression cancer diagnosis problems [126]. The paper compares several SVM-based methods together with the k -nearest neighbor method, weight voting method, backpropagation Neural Networks as well as Probabilistic Neural Networks. This study was conducted upon nine multi-category datasets and two binary datasets, which are the 11_Tumors dataset [129], GCM dataset [110], 9_Tumors dataset [127], Brain_Tumor1 dataset [103], Brain_Tumor2 dataset [95], Leukemia dataset [39], MLL dataset [5], Lung_Cancer dataset [13], SRBCT dataset [68], Prostate_Tumor dataset [123] and DLBCL dataset [122]. Experimental results show that generally SVM-based classifiers are the best performers both with and without gene selections. Decision tree methods and the weight voting are the worst performers, while the k -nearest neighbor method, backpropagation neural network and probabilistic neural network method rank in the middle. As to the four gene selection methods, there is no clear winner for all datasets. Lee *et al* [72]. also conducted an extensive comparison of recent classification methods for microarray datasets. 21 classification methods, seven microarray datasets and three gene selection methods are used in this study. These classification methods mainly fall into four categories.

1. Classical methods that include linear discriminant analysis, diagonal linear and quadratic discriminant analysis, k -nearest neighbor, logistic regression

and generalized partial least square methods.

2. Decision trees and aggregation methods that include CART, bagging, boosting and logic boosting and random forest.
3. Machine learning methods, such as support vector machines and neural network methods.
4. Generalized algorithms such as flexible discriminant analysis, penalized discriminant analysis, mixture discriminant analysis and shrunken centroid methods.

The datasets used are all publicly available benchmark datasets, viz. Leukemia dataset [39], Lymphoma dataset [2], NCI60 dataset [114], Colon Cancer dataset [3], Lung Cancer dataset [37], SRBCT dataset [68] and Yeast dataset [28]. It is found that the SVM classifier performs the best in most datasets and the gene selection method has little effect on the performance of the SVM. Linear discriminant analysis and k -nearest neighbor perform well compared to sophisticated methods on the datasets with small variance within the class, but diagonal quadratic discriminant analysis performs better on the datasets with big variance within the class. k -nearest neighbor method performs well when the number of the classes is small, but it is outperformed by other methods when the number of the classes is big. For tree methods, aggregating methods such as bagging, boosting and random forest improve the performance of basic tree method significantly and random forest is the best method among the tree methods when the number of the classes is moderate. Generally speaking, more sophisticated classifiers, such as SVM, neural networks and aggregating methods give better performance than traditional methods, such as k -nearest neighbor method and linear discriminant analysis. Gene selection has a big influence on the performance of most of the classifiers and thus the classifiers and the gene selection methods should be considered together.

As we discussed above, due to the unique nature of microarray data, gene selection is a very important procedure before the classification is performed. First, it can reduce the dimension of problem. For most microarray data sets, there are thousands of genes being monitored and recorded. Gene selection can reduce this number to a few or less than one hundred. Thus great computational effort can be saved. Also, many algorithms that can not deal with high dimension can be applied to microarray classification problems after gene selection. Second, gene selection can improve the classification accuracy. Since most of the genes in the data set are irrelevant to the cancer distinction, they cannot help with the classification but only introduce noise and deteriorate the performance. Removing these noisy genes and selecting the most informative ones by gene selection can improve the efficiency as well as the prediction accuracy.

Many gene selection methods have been proposed. All these methods fall into two categories, viz., individual gene ranking methods and gene subset ranking methods [84].

For individual ranking methods, the correlation of each gene with the class is measured individually according to a certain criterion and the top ones are selected. Most gene selection methods belong to this category, including ratio of between-group to within-group sums of squares [27], neighborhood analysis [39], t-statistics [137], information gain [130], likelihood gene selection [67], weights of linear discriminant functions [143]. This kind of methods have several shortcomings. First, genes selected in this way may have high correlations among themselves and thus contain redundancy. Other complementary genes which are useful in the discrimination may not be selected if their individual correlation is not high. Second, since they are measured as individual genes, the combination of the genes may not give the best prediction accuracy. Generally speaking, there is no clear winner of the gene selection methods that can select the best set of genes and produce

the best results for all the dataset. Different datasets favor different gene selection criteria.

For subset ranking methods, a group of genes that serve together to achieve the best predication accuracy is sought instead of searching for single individual genes. The methods in this category either remove the genes one by one, such as Recursive Feature Elimination [42] [110], or add the genes one by one, such as Forward Stepwise Selection and Monte Carlo methods [137]. Then the effect of removing or adding this gene is monitored and the genes with the best classification performance are selected. The drawback of these methods is the heavy computational burden. However, they can select the genes with complementary characters and achieve better classification results. When the number of selected genes is quite small, the computational burden will not pose a problem and gene selection methods in this category will be much more efficient than the individual ranking methods.

2.3 Problems with the Existing Approaches

Neural networks have been widely used in the past as pattern and statistical classifiers in many applications [49, 70, 91, 97, 100, 136]. However, most of the past work is based on batch learning algorithms [49, 91, 97, 100, 136]. Batch learning algorithms require re-training whenever a new data is received and thus can not be used for applications that needs online learning. Another problem with the current ANN classification applications is the difficulty in fixing the right structure of the network [36, 48, 97]. It is known that the number of hidden neurons should match the complexity of the mapping. Whenever some new data comes, the complexity of the mapping is changed. The number of hidden neurons, which the actual mapping needs, is also changed. Hence, a sequential learning algorithm with a growing and pruning strategy, which can adjust the number of neurons automatically is more

appropriate for these applications.

In the first part of this thesis, focus is on investigations of sequential learning algorithms. A new algorithm viz. Fast GAP-RBF (FGAP) is developed and the performance of this algorithm with other sequential methods is compared.

In the second part, we carry out a detailed investigations for gene expression based multi-category classification problems. Many classification methods have been applied for multi-category problems, but most of them are done by modifying binary classifiers [79] [110] [126] [139] [82], which requires more computational efforts.

In this thesis, we present two fast and efficient methods for training neural networks that can perform classification directly for multi-category microarray problems. A sequential learning algorithm—FGAP-RBF and a batch learning algorithm—ELM are used for microarray gene expression based multi-category classification problems. Their performances are compared with other neural network methods as well as two SVM based methods.

In the next chapter, we present the investigations of sequential learning algorithms by evaluating the newly developed GAP-RBF algorithm for classification problems.

Chapter 3

GAP-RBF Based Sequential Learning Scheme for Classification Problems

The GAP-RBF (Growing and Pruning RBF) algorithm [52] is a newly developed sequential growing and pruning algorithm for RBF networks for function approximation problems. It has shown excellent performance for a number of benchmark problems in the function approximation domain, but its performance for classification problems has not been evaluated so far.

The performance of GAP-RBF for classification applications is investigated for the first time in this thesis. It is tested on three benchmark problems in the classification area from PROBEN1 database [105], together with another well-known sequential learning algorithm—Minimal Resource Allocation Network (MRAN) as well as the conventional Multilayer Feedforward Networks (MFNs). The three benchmark problems used in the investigation are: 1) Hearta Problem; 2) Cancer Problem; and 3) Gene Problem. The first two problems are of low dimensions with roughly uniform input distributions and the last problem is of comparatively higher dimensions with

a highly non-uniform input distribution. The results indicate better performance of the GAP-RBF algorithm in terms of generalization, network size and training speed as well as the classification error for the first two problems. Yet, for the Gene problem, GAP-RBF performs inferior to other algorithms in terms of classification accuracy. Limitations of GAP-RBF algorithm for classification problems with inputs of non-uniform distribution and large dimensions are identified and indicated through the experiments. This leads to further improvements and the proposal of a Fast GAP-RBF (FGAP-RBF) algorithm in the next chapter .

Section 3.1 describes the principal ideas behind the GAP-RBF algorithm. Section 3.2 makes a comparison between GAP-RBF and MRAN. Section 3.3 gives a brief introduction to the standard database—PROBEN1 from where the benchmark problems are taken. Section 3.4 presents a quantitative performance result comparisons for GAP-RBF with the other algorithms based on the benchmark problems. Section 3.5 summarizes the conclusions from this study.

3.1 Brief Review to the GAP-RBF Algorithm

In this section, the main idea behind the GAP-RBF algorithm for growing and pruning hidden neurons for a RBF network are described.

The basic structure of the GAP-RBF neural network is shown in Figure 3.1. It consists of only three layers: input layer, hidden layer and output layer. Input attributes are fed into the the hidden layer linearly with a unit weight through the input layer. The hidden units provide a nonlinear transformation by a set of functions that constitutes the basis for the input when they are mapped into the space of the hidden neurons. Here in the GAP-RBF network, Gaussian function is used as the basis function. The output layer of RBF network only computes the linear combination of the outputs from the hidden layers.

Figure 3.1: The Basic structure of a GAP-RBF Neural Network

The output of a RBF network with K neurons is given by

$$f(\mathbf{x}) = \sum_{k=1}^K \alpha_k \phi_k(\mathbf{x}) \quad (3.1)$$

where $\phi_k(\mathbf{x})$ is the response of the k -th hidden neuron for an input vector $\mathbf{x} \in \mathbf{R}^l$:

$$\phi_k(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - \boldsymbol{\mu}_k\|^2}{\sigma_k^2}\right) \quad (3.2)$$

α_k is the weight connecting hidden neuron k and the output neurons, and $\boldsymbol{\mu}_k \in \mathbf{R}^l$ and σ_k are the center and width of the k -th hidden neuron, respectively, $k = 1, \dots, K$.

3.1.1 Significance of a Neuron

The algorithm introduces the notion of *significance* for the hidden neurons based on their average contribution over all inputs seen so far.

In sequential learning, a series of training samples $(\mathbf{x}_i, y(\mathbf{x}_i))$, $i = 1, 2, \dots$, are randomly drawn and presented one by one to the network. We assume that these input samples, $\mathbf{x}_i$, have a uniform distribution and the range for the input is X . After sequentially learning n observations, assume that a RBF network with K neurons has been obtained. The network output for an input $\mathbf{x}_i$ is given by :

$$f_1(\mathbf{x}_i) = \sum_{j=1}^K \alpha_j \exp \left(-\frac{1}{\sigma_j^2} \|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2 \right) \quad (3.3)$$

If the k th neuron is removed, the output of the RBF network with the remaining $K - 1$ neurons is:

$$f_2(\mathbf{x}_i) = \sum_{j=1}^{k-1} \alpha_j \exp \left(-\frac{1}{\sigma_j^2} \|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2 \right) + \sum_{j=k+1}^K \alpha_j \exp \left(-\frac{1}{\sigma_j^2} \|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2 \right) \quad (3.4)$$

Thus, for an observation $\mathbf{x}_i$, the error resulting from removing neuron k is the absolute difference between $f_1(\mathbf{x}_i)$ and $f_2(\mathbf{x}_i)$, that is,

$$E(k, i) = |f_1(\mathbf{x}_i) - f_2(\mathbf{x}_i)| = |\alpha_k| \cdot \exp \left(-\frac{1}{\sigma_k^2} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2 \right), \quad i = 1, \dots, n \quad (3.5)$$

Then, the average error for all n sequentially learned inputs due to removing neuron k will be

$$E_{\text{ave}}(k) = \frac{\sum_{i=1}^n E(k, i)}{n} = \frac{|\alpha_k|}{n} \cdot \sum_{i=1}^n \exp \left(-\frac{1}{\sigma_k^2} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2 \right) \quad (3.6)$$

The *significance* of a neuron is defined as its average output over the number of all the input samples it has received so far, as given by the above equation. However,

calculation of $E_{\text{ave}}(k)$ in a sequential learning scheme would require storing all the past inputs and their corresponding outputs. This will be a time consuming operation and is not ideal for sequential real time applications. A simplified approach to calculate $E_{\text{ave}}(k)$ without storing all the past inputs and their corresponding outputs is given below.

Figure 3.2: The approximation of Gaussian function

Figure 3.2 shows the Gaussian output of neuron k . From the figure, it is found that when $\|\mathbf{x} - \boldsymbol{\mu}_k\| > 1.7\sigma_k$, $\exp\left(-\frac{1}{\sigma_k^2}\|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2\right) < 0.0556$. Then, the average error $E_{\text{ave}}(k)$ mainly results from losing outputs for the observations $\mathbf{x}_i$ located in the impact area $\{\mathbf{x} : \|\mathbf{x} - \boldsymbol{\mu}_k\| \leq 1.7\sigma_k\}$ of neuron k . The average error can then be approximated as

$$E_{\text{ave}}(k) \approx \frac{|\boldsymbol{\alpha}_k|}{n} \cdot \sum_{i=1}^m \exp\left(-\frac{1}{\sigma_k^2}\|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2\right), \quad \mathbf{x}_i \in \{\mathbf{x} : \|\mathbf{x} - \boldsymbol{\mu}_k\| \leq 1.7\sigma_k\} \quad (3.7)$$

To simplify this further, a piecewise linear approximation to the Gaussian function

as shown in Figure 3.2 is used here. The corresponding equation is given below,

$$\exp\left(-\frac{1}{\sigma_k^2}\|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2\right) \approx \begin{cases} \frac{0.77}{1.2\sigma_k}(\mathbf{x}_i - \boldsymbol{\mu}_k + 1.7\sigma_k), & \text{if } \boldsymbol{\mu}_k - 1.7\sigma_k \leq \mathbf{x}_i < \boldsymbol{\mu}_k - 0.5\sigma_k, \\ 0.77 + \frac{0.23}{0.5\sigma_k}(\mathbf{x}_i - \boldsymbol{\mu}_k + 0.5\sigma_k), & \text{if } \boldsymbol{\mu}_k - 0.5\sigma_k \leq \mathbf{x}_i < \boldsymbol{\mu}_k, \\ 0.77 - \frac{0.23}{0.5\sigma_k}(\mathbf{x}_i - \boldsymbol{\mu}_k - 0.5\sigma_k), & \text{if } \boldsymbol{\mu}_k \leq \mathbf{x}_i < \boldsymbol{\mu}_k + 0.5\sigma_k, \\ -\frac{0.77}{1.2\sigma_k}(\mathbf{x}_i - \boldsymbol{\mu}_k - 1.7\sigma_k), & \text{if } \boldsymbol{\mu}_k + 0.5\sigma_k \leq \mathbf{x}_i < \boldsymbol{\mu}_k + 1.7\sigma_k, \\ 0, & \text{otherwise.} \end{cases} \quad (3.8)$$

Since there are m input samples located in the impact area $(\boldsymbol{\mu}_k - 1.7\sigma_k, \boldsymbol{\mu}_k + 1.7\sigma_k)$ of neuron k and those observations are uniformly drawn from X , there are $\frac{(1)\sigma_k}{(3.4)\sigma_k}m$ observations located in the area $(\boldsymbol{\mu}_k - 0.5\sigma_k, \boldsymbol{\mu}_k + 0.5\sigma_k)$ and $\frac{(2.4)\sigma_k}{(3.4)\sigma_k}m$ input samples located in the area $(\boldsymbol{\mu}_k - 1.7\sigma_k, \boldsymbol{\mu}_k - 0.5\sigma_k) \cup (\boldsymbol{\mu}_k + 0.5\sigma_k, \boldsymbol{\mu}_k + 1.7\sigma_k)$. Then, by approximations (3.7) and (3.8), we have

$$E_{\text{ave}}(k) \approx \frac{|\boldsymbol{\alpha}_k|}{n} \cdot \left(\frac{1}{3.4}m \cdot \frac{1+0.77}{2} + \frac{2.4}{3.4}m \cdot \frac{0.77}{2}\right) \approx 0.53 \frac{m}{n} \cdot |\boldsymbol{\alpha}_k|, \quad (3.9)$$

Similarly, for l -dimensional input case, we can have,

$$E_{\text{ave}}(k) \approx \frac{|\boldsymbol{\alpha}_k|}{n} \cdot \left(\frac{1}{3.4}m \cdot \frac{1+0.77}{2} + \frac{2.4}{3.4}m \cdot \frac{0.77}{2}\right) \approx 0.53^l \cdot \frac{m}{n} \cdot |\boldsymbol{\alpha}_k|, \quad (3.10)$$

Since all n observations are uniformly distributed in the range X , $\frac{m}{n} = \frac{(3.4\sigma_k)^l}{S(X)}$, where $S(X)$ is the size of the range X the training samples are drawn from, we have $E_{\text{ave}}(k) \approx \left|\frac{(1.8\sigma_k)^l \boldsymbol{\alpha}_k}{S(X)}\right|$, which is the contribution of neuron k to the overall performance of the RBF network. Thus, the *significance* of neuron k can be quantified as

$$E_{\text{sig}}(k) = \left|\frac{(1.8\sigma_k)^l \boldsymbol{\alpha}_k}{S(X)}\right|, \quad (3.11)$$

where l is the dimension of the input space.

This definition of *significance* of a neuron is used for the growing and pruning criteria in a RBF network as indicated below.

3.1.2 Growing Criterion

The learning process of GAP-RBF involves allocation of new hidden neurons as well as adaptation of network parameters. The RBF network begins with no hidden neurons. As observations are received during training, some of them may initiate new hidden neurons based on a growing criterion. The decision as to whether an observation $(\mathbf{x}_n, \mathbf{y}_n)$ should give rise to a new hidden neuron depends on the novelty in the data which is decided using the following three conditions:

$$\left\{ \begin{array}{l} \|\mathbf{x}_n - \boldsymbol{\mu}_{nr}\| > \epsilon_n \\ |\mathbf{e}_n| > e_{\min} \\ \frac{(1.8 \cdot \kappa \|\mathbf{x}_n - \boldsymbol{\mu}_{nr}\|)^l |\mathbf{e}_n|}{S(X)} > e_{\min} \end{array} \right. \quad (3.12)$$

where $\mathbf{e}_n = \mathbf{y}_n - f(\mathbf{x}_n)$ and $\boldsymbol{\mu}_{nr}$ is the center of the hidden neuron which is nearest to $\mathbf{x}_n$. If the above three conditions are satisfied, then the data is deemed to have novelty and a new hidden neuron is added. The first condition says that the input must be far away from all the centers. The distance ϵ_n represents the scale of resolution in the input space. $\epsilon_{\max}$ is chosen as the largest scale of interest in the input space with a nonzero probability. The distance ϵ_n is typically reduced exponentially in the entire input space as $\epsilon_n = \max\{\epsilon_{\max}\gamma^n, \epsilon_{\min}\}$, $0 < \gamma < 1$, where γ is a decay constant. The value for ϵ_n is reduced until it reaches $\epsilon_{\min}$. The exponential decay of the distance criterion allows fewer basis functions with large widths (smoother basis functions) initially and with an increasing number of observations more basis functions with smaller widths are allocated to fine tune the approximation. The second

condition says that the error between the network output and the target output must be significant. The third condition says the hidden neuron to be added must make significant contribution, where κ is an overlapping factor that determines the amount of overlap of the responses of the hidden neurons in the input space. Before a hidden neuron is added, this criterion is used to prevent neurons with trivial contributions to be added so that the oscillation of the number of hidden neurons can be reduced. It has been proved that Condition 2 is always contained in Condition 3 [53]. Thus, whether a new observation $(\mathbf{x}_n, \mathbf{y}_n)$ should give rise to a new hidden neuron depends on the novelty in the data which is decided using the following two conditions [53]:

$$\begin{cases} \|\mathbf{x}_n - \boldsymbol{\mu}_{nr}\| > \epsilon_n \\ E_{\text{sig}}(K+1) > e_{\min} \end{cases} \quad (3.13)$$

The parameters associated with the new hidden neuron are set as follows:

$$\begin{cases} \boldsymbol{\alpha}_{K+1} = \mathbf{e}_n \\ \boldsymbol{\mu}_{K+1} = \mathbf{x}_n \\ \sigma_{K+1} = \kappa \|\mathbf{x}_n - \boldsymbol{\mu}_{nr}\| \end{cases} \quad (3.14)$$

When an observation $(\mathbf{x}_n, \mathbf{y}_n)$ does not pass the novelty criteria, a hidden neuron is not added but the network parameters $\boldsymbol{\mu}$, σ and $\boldsymbol{\alpha}$ are adapted using the EKF algorithm to fit that observation.

3.1.3 Pruning Criterion

If the significance of neuron k to the overall performance of the RBF network is less than the chosen $e_{\min}$, neuron k is insignificant and should be removed. Otherwise, neuron k is significant. Thus, we have a new pruning criterion, which is independent of the true function to be learned. For neuron k , given the desired approximation

accuracy $e_{\min}$, if

$$E_{\text{sig}}(k) = \left| \frac{(1.8\sigma_k)^l \alpha_k}{S(X)} \right| < e_{\min} \quad (3.15)$$

then the average contribution made by neuron k in the whole range X is less than the expected accuracy $e_{\min}$ and the neuron k is insignificant, thus, neuron k can be removed.

3.1.4 Nearest Neuron for Parameter Adjustment and Pruning

The GAP-RBF algorithm uses the Extended Kalman Filter (EKF) as its parameter adjustment method. The details are summarized below:

When the criteria for adding a neuron are not satisfied, the network parameters $\mathbf{w} = [\alpha_1, \mu_1^T, \sigma_1, \dots, \alpha_K^T, \mu_K^T, \sigma_K]^T$ are updated by the following equation:

$$\mathbf{w}_n = \mathbf{w}_{n-1} + \mathbf{K}_n \mathbf{e}_n, \quad (3.16)$$

among which $\mathbf{K}_n$ is the Kalman gain matrix given by:

$$\mathbf{K}_n = \mathbf{P}_{n-1} \mathbf{B}_n [\mathbf{R}_n + \mathbf{B}_n^T \mathbf{P}_{n-1} \mathbf{B}_n]^{-1}, \quad (3.17)$$

where $\mathbf{B}_n = \nabla_{\mathbf{w}} f(\mathbf{x}_n)$ is the gradient of the function $f(\mathbf{x}_n)$ with respect to the parameter vector $\mathbf{w}$ evaluated at $\mathbf{w}_{n-1}$. Therefore,

$$\begin{aligned} \mathbf{B}_n = & [\phi_1(\mathbf{x}_n), \phi_1(\mathbf{x}_n) \frac{2\alpha_1}{(\sigma_1)^2} (\mathbf{x}_n - \mu_1)^T, \phi_1(\mathbf{x}_n) \frac{2\alpha_1}{(\sigma_1)^3} \|\mathbf{x}_n - \mu_1\|^2, \dots, \\ & \phi_K(\mathbf{x}_n), \phi_K(\mathbf{x}_n) \frac{2\alpha_K}{(\sigma_K)^2} (\mathbf{x}_n - \mu_K)^T, \phi_K(\mathbf{x}_n) \frac{2\alpha_K}{(\sigma_K)^3} \|\mathbf{x}_n - \mu_K\|^2]^T. \end{aligned} \quad (3.18)$$

$\mathbf{R}_n$ is the variance of the mean noise and the error covariance matrix $\mathbf{P}_n$ is updated by

$$\mathbf{P}_n = [\mathbf{I} - \mathbf{K}_n \mathbf{B}_n^T] \mathbf{P}_{n-1} + Q_0 \mathbf{I}, \quad (3.19)$$

where Q_0 is a scalar that determines the random step in the direction of gradient vector. When a new hidden unit is added, $\mathbf{P}_n$ is initialized by

$$\mathbf{P}_n = \begin{pmatrix} \mathbf{P}_{n-1} & 0 \\ 0 & P_0 \mathbf{I} \end{pmatrix} \quad (3.20)$$

where P_0 is a matrix which gives an estimate of the uncertainty in the initial values assigned to the parameters.

In order to increase the training speed further, instead of adjusting the parameters for all neurons after each observation, one need only to adjust parameters for the nearest neuron if no new neuron is added and only need to check the nearest neuron for pruning. It is neither necessary to adjust the parameters for all neurons nor necessary to check all neurons for possible pruning. The rationale for this is given as follows.

As shown in Figure 3.3, the functions $y_1(x) = \phi(x) = \exp(-\frac{x^2}{\sigma^2})$, $y_2(x) = \phi(x) \frac{x}{\sigma^2} = \exp(-\frac{x^2}{\sigma^2}) \frac{x}{\sigma^2}$ and $y_3(x) = \phi(x) \frac{x^2}{\sigma^3} = \exp(-\frac{x^2}{\sigma^2}) \frac{x^2}{\sigma^3}$ will approach zero very quickly when $x > \sigma$. Therefore, the elements in $\mathbf{B}_n$ approach zero quickly when $x > \sigma$, except $\phi_{nr}(x_n)$, $\phi_{nr}(x_n) \frac{2\alpha_{nr}}{\sigma_{nr}^2} (x_n - \boldsymbol{\mu}_{nr})^T$, $\phi_{nr}(x_n) \frac{2\alpha_{nr}}{\sigma_{nr}^3} \|x_n - \boldsymbol{\mu}_{nr}\|^2$. Thus, for the parameter adjustment, the adjustment will be mainly done on the parameters of the nearest neuron and the parameters of the rest neurons will be changed only slightly. To increase the learning speed and reduce the computational complexity without loss of accuracy, one can just adjust the parameters for the nearest neuron only. Thus, the gradient $\mathbf{B}_n$ will become a sparse matrix with much less elements. What's

Figure 3.3: The First and Second Derivatives of Gaussian Function

more, the Kalman filter matrix $\mathbf{K}_n$ can be calculated from a simple sparse matrix operation, which can increase the learning speed greatly. Furthermore, we can see from the definition of the *significance* of a neuron that its value is independent of the mapping the network is approximating and it only depends on the parameters of that neuron, such as α_k , σ_k and μ_{nr} . Therefore, the significance of a neuron will not change unless these parameters are adjusted. From the reasoning above, we know that GAP-RBF only adjusts the nearest neuron of the input pattern. Then only the parameters for the nearest neuron may be changed and hence it is necessary only to check the nearest neuron for the pruning purposes. It is easy to see that both of these procedures will contribute greatly to the increase of the learning speed as well as the reduction of the computational burdens for the GAP-RBF algorithm.

Suppose that after sequentially learning n observations, a RBF network with K neurons is obtained. Obviously all these K neurons should be significant since insignificant neurons would have been pruned after learning the n -th observation.

If a new $(n + 1)$ -th observation $(\mathbf{x}_{n+1}, \mathbf{y}_{n+1})$ arrives and the growing criteria (3.13) are satisfied, a new significant neuron $K + 1$ will be added. Since the parameters of all the rest neurons remain unchanged, these neurons will remain significant after learning the $(n + 1)$ -th observation. The new added neuron is also significant, thus, pruning checking need not be done after a new neuron is added. If a new observation $(\mathbf{x}_{n+1}, \mathbf{y}_{n+1})$ arrives and the growing criteria (3.13) are not satisfied, no new neuron will be added and only the parameters of the nearest neuron will be adjusted. Since the parameters of all the neurons except for the nearest one remain unchanged, those neurons will remain significant after learning the $(n + 1)$ -th observation. After the parameters of the nearest neuron are adjusted, if the nearest neuron becomes insignificant it should be removed. That means, if only the parameters of the nearest neuron are adjusted after each observation during sequential learning, one needs to check only whether the nearest neuron becomes insignificant after adjustment. As for the parameter adjustment, the adjustment will be done for the parameters of the nearest neuron using the EKF algorithm. In the EKF algorithm, the Kalman gain matrix computation is simpler because we are adjusting only one neuron.

Thus, we have a new simple efficient growing and pruning RBF algorithm as follows:

GAP-RBF Algorithm [52]:

Given an approximation error $e_{\min}$, for each observation $(\mathbf{x}_n, \mathbf{y}_n)$, where $\mathbf{x}_n \in \mathbf{R}^l$, do

1. **compute** the overall network output:

$$f(\mathbf{x}_n) = \sum_{k=1}^K \alpha_k \exp\left(-\frac{1}{\sigma_k^2} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2\right) \quad (3.21)$$

where K is the number of hidden neurons.

2. **calculate** the parameters required in the growth criterion:

$$\begin{aligned} \epsilon_n &= \max\{\epsilon_{\max} \gamma^n, \epsilon_{\min}\}, \quad (0 < \gamma < 1) \\ \mathbf{e}_n &= \mathbf{y}_n - f(\mathbf{x}_n) \end{aligned} \quad (3.22)$$

3. **apply** the criterion for adding neurons:

$$\text{If } \|\mathbf{x}_n - \boldsymbol{\mu}_{nr}\| > \epsilon_n \text{ and } \frac{(1.8 \cdot \kappa \|\mathbf{x}_n - \boldsymbol{\mu}_{nr}\|)^l |\mathbf{e}_n|}{S(X)} > e_{\min}$$

allocate a new hidden neuron $K + 1$ with

$$\begin{aligned} \boldsymbol{\alpha}_{K+1} &= \mathbf{e}_n \\ \boldsymbol{\mu}_{K+1} &= \mathbf{x}_n \\ \sigma_{K+1} &= \kappa \|\mathbf{x}_n - \boldsymbol{\mu}_{nr}\| \end{aligned} \quad (3.23)$$

Else

adjust the network parameters $\boldsymbol{\alpha}_{nr}$, $\boldsymbol{\mu}_{nr}$, σ_{nr} for the nearest neuron only, using the EKF method.

check the criterion for pruning the hidden neuron:

$$\text{If } \left| \frac{(1.8 \sigma_{nr})^l \boldsymbol{\alpha}_{nr}}{S(X)} \right| < e_{\min}, \text{ where } S(X) \text{ is the estimated size of the range}$$

where the training samples are drawn from,

remove the nr -th hidden neuron

reduce the dimensionality of EKF

Endif

Endif

3.2 A Comparison of GAP-RBF and MRAN

Minimal Resource Allocation Network (MRAN) is a sequential growing and pruning learning algorithm proposed by Yingwei *et al* [141] and it has been used in a number of areas including the function approximation and classification domains [141, 142].

MRAN uses a sliding data window in the growing and pruning criteria to identify the neurons that contribute relatively little to the network output. Selection of the appropriate size for these windows critically depends on the distribution of the input data. In MRAN, choosing proper window sizes can only be done by trial and error based on experimental studies. In this part, we present a comparison of GAP-RBF algorithm and the MRAN algorithm.

A brief but complete description of the MRAN algorithm is given below for reference:

MRAN Algorithm:

Given an approximation error $e_{\min}$, for each observation $(\mathbf{x}_n, \mathbf{y}_n)$, where $\mathbf{x}_n \in \mathbf{R}^l$, do

1. **compute** the overall network output:

$$f(\mathbf{x}_n) = \alpha_0 + \sum_{k=1}^K \alpha_k \exp\left(-\frac{1}{\sigma_k^2} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2\right) \quad (3.24)$$

where α_0 is the bias and K is the number of hidden neurons.

2. **calculate** the parameters required in the growth criterion:

$$\begin{aligned}\epsilon_n &= \max\{\epsilon_{\max}\gamma^n, \epsilon_{\min}\}, \quad (0 < \gamma < 1) \\ \mathbf{e}_n &= \mathbf{y}_n - f(\mathbf{x}_n) \\ e_{rms} &= \sqrt{\frac{\sum_{i=n-(n_w-1)}^n |e_i|^2}{n_w}}\end{aligned}\tag{3.25}$$

where e_{rms} is the RMS value of the output error over a sliding window n_w .

3. **apply** the criterion for adding neurons:

$$\text{If } |\mathbf{e}_n| > e_{min}, \|\mathbf{x}_n - \boldsymbol{\mu}_{nr}\| > \epsilon_n \text{ and } e_{rms} > e'_{min}$$

allocate a new hidden neuron $K + 1$ with

$$\begin{aligned}\boldsymbol{\alpha}_{K+1} &= \mathbf{e}_n \\ \boldsymbol{\mu}_{K+1} &= \mathbf{x}_n \\ \sigma_{K+1} &= \kappa \|\mathbf{x}_n - \boldsymbol{\mu}_{nr}\|\end{aligned}\tag{3.26}$$

Else

adjust the network parameters $\boldsymbol{\alpha}_0$ and $\boldsymbol{\alpha}_k, \boldsymbol{\mu}_k, \sigma_k, (k = 1, \dots, K)$ using the EKF method.

Endif

where e_{min} and e'_{min} are different thresholds selected for the first and third criterion.

4. **check** the criterion for pruning the hidden neuron:

Compute the hidden unit outputs $|\boldsymbol{o}_k^n|, (k = 1, \dots, K)$ using

$$\boldsymbol{o}_k^n = \boldsymbol{\alpha}_k \exp\left(-\frac{1}{\sigma_k^2} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2\right)\tag{3.27}$$

Find the largest absolute hidden unit output value $|\boldsymbol{o}_{max}^n|$ and compute the

normalized output values r_k^n , ($k = 1, \dots, K$) using

$$r_k^n = \frac{|\mathbf{o}_k^n|}{|\mathbf{o}_{max}^n|} \quad (3.28)$$

If $r_k^n < \delta$ for n_w consecutive observations, then

remove the k -th hidden neuron

reduce the dimensionality of EKF

Endif

where δ is another parameter to be selected.

3.2.1 Comparison of Parameters

To achieve a proper size of network, MRAN introduces many parameters such as RMS error threshold e'_{min} (for smooth growth and noise resistance purposes), pruning threshold δ and growing and pruning sliding windows of size n_w . However, there is no guidance on how to select these parameters. These parameters critically depend on the application and cannot be selected intuitively, as these parameters will have a certain connection with the learning accuracy e_{min} . Unfortunately, the quantitative representation of this connection is not known. Due to this connection, not only it is necessary to select the value of the parameters according to the need of the applications but also they should be picked to match each other. Thus, it is very difficult to select the right values for all the parameters of the applications quickly. If one parameter is not set properly, the whole performance would deteriorate greatly.

3.2.2 Comparison of Growing and Pruning Strategy

In MRAN, the hidden neurons which are inactive consecutively over a number of training patterns will be pruned. It is realized by comparing the normalized outputs of hidden neurons with a threshold δ . If any of them falls below the threshold for a number of consecutive training samples, this hidden neuron would be pruned. Besides, the fact that the size of the sliding window n_w and the threshold δ are difficult to be selected, MRAN may prune several neurons at one time when the normalized outputs of several neurons fall below that threshold and this causes a sharp change in the network and accuracy.

As mentioned above, there is no fixed relationship between the threshold δ and the expected accuracy e_{min} . The pruning strategy used in MRAN is actually not accuracy-dependent. What's more, the adding and pruning strategy for the same network is not based on the same criterion, which might cause oscillations for the change in the size of the RBF network if the parameters are not selected properly.

For GAP-RBF, its learning scheme ensures that not only the newly added neurons are significant but also at most only one neuron (the nearest neuron) can be pruned at a time. Furthermore, the growing and pruning strategy of GAP-RBF are all based on the same criterion—the significance of the neurons and it is linked to the average of the errors directly.

3.2.3 Comparison of Computational Complexity

Comparing with GAP-RBF, MRAN needs additional $O(n_w)$ operations to ensure the smooth growth and $O(n_w K - 1)$ operations for sorting and comparison in pruning phase. Its computational burden is also higher than GAP-RBF, since the matrix in the EKF parameter adjustment is sparse in GAP-RBF. What's more, the computational complexity of MRAN increases sharply with the number of hidden neurons

and the size of the growing and pruning windows.

GAP-RBF achieves a fast learning speed because only the nearest neuron is checked for significance. It needs only one step operation for pruning checking and a simple sparse matrix for parameter adjustment using EKF method.

3.2.4 Comparison of Memory Requirements

MRAN has to remember the n_w consecutive instance errors e_i , ($i = n - (n_w - 1), \dots, n$), and the $n_w K$ values for n_w consecutive normalized outputs of K hidden neurons. While GAP-RBF algorithm does not have to remember anything except the centers, widths and impact factors of neurons.

3.3 Classification Problems from PROBEN1 Databases

To examine the performance of GAP-RBF on classification applications, three benchmark problems in the classification area have been studied. All the data for these three applications is taken from PROBEN1.

PROBEN1 is a set of problems for neural network learning in the area of pattern recognition and function approximation plus a set of rules and conventions for carrying out benchmark tests. These benchmark problems are available on http://www.ipd.uka.de/~prechelt/?NIPS_bench.html. The PROBEN1 database contains 15 datasets from 12 domains. All databases represent real problems which could be called as *diagnosis tasks*. All but one consist of real world data. Real data is preferred over artificial data in PROBEN1 because the choice of real data guarantees to get results relevant for at least a few real domains and the results obtained tell us about the behavior of our system on real world tasks.

The *diagnosis tasks* in PROBEN1 can be described as follows:

- The input attributes used in the data sets are similar to those that used by human beings to solve the same problem.
- The outputs represent either a classification into a small number of understandable classes or prediction of a small set of understandable quantities.
- In practice, the same problems are often solved by human beings.
- Examples are expensive to get. This is why the training sets are not very large.
- Often some attribute values are missing.

Together with the data sets, PROBEN1 also includes a set of rules on how to conduct valid benchmark tests and how to document the experiments as well as the results. This is to ensure the validity of the results and reproducibility by other researchers.

Firstly, each data set is pre-partitioned into training, validation and test examples. Data used for performing benchmark testing on neural network problems should at least be divided into 2 parts [105]: one part is used for training, which is called the training data, and the other part is used to measure the performance of the resulting network, which is called the test data. The performance of the network on the test data estimates its effectiveness in real practice. Therefore, no information about the test data should be available during the training process. Otherwise, the benchmark is not valid. In many cases, the training data is further subdivided into the actual training samples and a so-called validation set. The latter serves as a pseudo test set in order to evaluate the performance of the network during training. For PROBEN1, the size of training, validation and the test set is 50%, 25%, 25% respectively for all examples. PROBEN1 also contains three different permutations of the data set to resolve the variance brought by different partitioning.

Secondly, input and output representations are also one of the key factors influencing

the result obtained from network. Each problem may have several kinds of attributes that must be represented. Since there are many ways to represent these attributes, PROBEN1 designates the way of the representation of the attributes by the following rules:

- Real-valued attributes are re-scaled by some function that maps the value into the range of $(0 \dots 1)$ or $(-1 \dots 1)$ in a way that makes a roughly even distribution with that range.
- Integer-valued attributes are processed as real value.
- Ordinal attributes with m different values are either mapped into m equidistant scale making them pseudo-real-valued or represented by $m - 1$ inputs of which the leftmost k have the value of 1 to denote the k th attribute value while all others are 0.
- Nominal attributes with m different values are represented by either a 1-of- m code or a binary code.
- Missing attribute values can be replaced by a fixed value, such as the mean of the non-missing values for that attribute, or can be represented by a value out of the range to denote that the value is missing.

Thirdly, PROBEN1 also provides a standard error measure which is called **square error percentage E** ,

$$E = 100 * \frac{O_{max} - O_{min}}{P * N} \sum_{n=1}^N \sum_{p=1}^P (o_{np} - t_{np})^2, \quad (3.29)$$

where O_{max} and O_{min} are the maximum and minimum values of the output coefficients, P is the number of output units of the network and N is the number of examples used in the test set. o_{np} and t_{np} are the actual and target output values of

the p th output node for the n th example. Classification performance is documented in percentage of incorrect classified samples, i.e., classification error.

3.4 Experimental Results

In this section, the performance of the GAP-RBF learning algorithm is compared with another well-known sequential learning algorithm—MRAN as well as MFNs on three benchmark problems in the classification area. The benchmark problems are: 1) Hearta Problem; 2) Cancer Problem; and 3) Gene problem. All the data sets are from the UCI machine learning database. The first two problems are low dimensional problems with roughly uniform input distribution and the last problem is of higher dimension and has an extremely non-uniform input distribution.

All the experiments for GAP-RBF and MRAN are carried out in the Matlab 6.5 environment running in a Pentium 4, 2.4 GHZ CPU with 512 MB memory. The results of MFNs are for the suggested architectures in PROBEN1 and quoted from Table 11 in [105].

3.4.1 Hearta Problem

Hearta problem is a heart disease diagnosis problem. The single continuous output predicts heart disease and decides the number of major vessels which are reduced in diameter by more than 50% and the **hearta** data sets have 45% patients with ‘no vessel is reduced’. The decision is made on the personal data, which serves as the input attributes. There are a total of 13 attributes which include age, sex, smoking habits, subjective patient pain descriptions, and results of many medical examinations such as blood pressure and ECG results. The **hearta** data sets is a union of four data sets from Cleveland Clinic Foundation, Hungarian Institute of

Cardiology, V.A. Medical Center Long Beach, and University Hospital Zurich.

In the original **hearta** data set given in PROBEN1, the 13 attributes are described by 35 input units based on the rules described above. To make the input data more compact, the nominal attributes are mapped with m different values onto m equidistant scale in the range from 0 to 1 rather than described as the 1-of- m code. The missing values are replaced by a fixed value, which is the mean of the non-missing attributes for that attribute. Therefore, the 13 input attributes can be represented by 13 input units. In the total of 920 examples of the **hearta** data set, 690 examples are used for training and the rest 230 are used for testing. Among these 920 examples, only 299 records are complete. This preprocessing is applied to both GAP-RBF and MRAN algorithms. In addition, both of these two algorithms used 13 input units and 1 single continuous output unit to represent the five levels of the output attribute. The performance of both algorithms are evaluated on the three different permutations of the **hearta** data set given in PROBEN1, namely **hearta1**, **hearta2**, **hearta3**.

The performance of GAP-RBF together with the performance of MRAN and MFNs are shown in Table 3.1. The results of MFNs are for the architectures suggested in PROBEN1 for **Hearta** data sets of no shortcut connection, and quoted from Table 11 of [105]. We use the same values for the parameters for MRAN and GAP-RBF except e_{min} , as the technical meaning of e_{min} is not known and still needs to be identified for classification problems. The parameters are chosen as: $\epsilon_{max} = 4.0$, $\epsilon_{min} = 1.5$, $\gamma = 0.99$, $\kappa = 0.4$, $p_0 = 1.0$, $R_n = 1.0$, $q_0 = 0.00001$, together with $e_{min} = 4$ for GAP-RBF and $e_{min} = 0.15$, $e'_{min} = 0.26$, $n_w = 60$ and $\delta = 0.000001$ for MRAN.

From experimental results, we can see that, compared with the MRAN, the GAP-RBF algorithm achieves a similar accuracy but with a smaller neuron number. GAP-

Benchmark Data	Network Used	Network Architecture	Final Squared Error Percentage for Testing Set	Number of Network Parameters adjusted at each iteration	Time Used(s)	Number of Training Epochs
Hearta1	MFN	35-32-1	4.55	1185	N/A	47
	MRAN	13-3-1	4.48	46	1.98	1
	GAPRBF	13-3-1	4.52	15	0.78	1
Hearta2	MFN	35-16-1	4.33	593	N/A	54
	MRAN	13-5-1	3.89	76	3.98	1
	GAPRBF	13-3-1	4.09	15	0.91	1
Hearta3	MFN	35-32-1	4.89	1185	N/A	46
	MRAN	13-3-1	4.93	46	1.93	1
	GAPRBF	13-2-1	4.55	15	0.83	1

Table 3.1: Performance comparison for Hearta Problem

RBF also saves more than 50% cpu computation time than that of MRAN in all the experiments where the number of the hidden neurons is small and the computational burden is rather light. It can be seen that the number of parameters to be adjusted at each iteration in GAP-RBF is far less than that of MRAN. When the complexity of the application increases, the number of neurons will grow quickly. The number of parameters to be adjust at each iteration would also increase dramatically for MRAN, while for GAP-RBF the number of parameters stays unchanged no matter how many hidden neurons are added and how complex the problem is. The speed of GAP-RBF is also much faster than MRAN. For MFNs, the network structure is far more complicated than GAP-RBF. Sometimes the number of its hidden neurons can be as much as 10 times of that for GAP-RBF in some cases in **hearta** problem. The number of network parameters adjusted by GAP-RBF at each iteration is one or two order less of MFNs. What's more, GAP-RBF only needs the data to be presented once while MFNs require more than 40 learning epochs for convergence in all cases.

3.4.2 Cancer Problem

Cancer problem is a breast cancer diagnosis problem based on the “Wisconsin breast cancer database” from the UCI repository machine learning databases, trying to classify a tumor as benign or malignant based on cell descriptions gathered by microscopic examination. The output represents the classification result. The decisions are made based on the input attributes which include the clump thickness, the uniformity of cell size and cell shape, the amount of marginal adhesion, and the frequency of bare nuclei.

In the original **cancer** data set given in PROBEN1, the binary output is represented by two variables to show the tumor is benign or malignant. To make the output data more compact, one data (0 or 1) is used to represent this binary output attribute. In the total of 699 examples of the **cancer** data set, 525 examples are used for training and the remaining 174 are used for testing. There are 16 missing values for attribute 6. The missing values are replaced by a fixed value, which is the mean of the non-missing values for that attribute. This preprocessing work is applied to both the GAP-RBF and MRAN algorithms. In addition, both of these two algorithms use 9 input units and 1 output unit. The performance of both algorithms is evaluated on the three different permutations of the **cancer** data set, namely **cancer1**, **cancer2**, **cancer3** given in PROBEN1.

The comparisons of the performance of GAP-RBF with MRAN and MFNs are shown in Table 3.2. The results of MFNs are for the suggested architectures for **Cancer** data sets to be using for training of no shortcut connection networks in PROBEN1 and quoted from Table 10 of [105]. The parameters for these two algorithms are chosen as: $\epsilon_{max} = 3.0$, $\epsilon_{min} = 0.5$, $\gamma = 0.97$, $\kappa = 0.8$, $p_0 = 1.0$, $R_n = 1.0$, $q_0 = 0.00001$, together with $e_{min} = 400$ for GAP-RBF and $e_{min} = 0.4$, $e'_{min} = 0.28$, $n_w = 50$ and $\delta = 0.000001$ for MRAN only.

Benchmark Data	Network Used	Network Architecture	Final Squared Error Percentage for Testing Set	Classification Error for Test Sets	Number of Network Parameters adjusted at each iteration	Time Used (s)	Number of Training Epochs
Cancer1	MFN	9-4-4-2	1.32	1.38	56	N/A	46
	MRAN	9-2-1	2.13	1.72	23	0.99	1
	GAPRBF	9-2-1	2.00	1.72	11	0.39	1
Cancer2	MFN	9-8-4-2	3.47	4.77	126	N/A	54
	MRAN	9-2-1	4.14	4.02	23	1.26	1
	GAPRBF	9-2-1	3.41	3.45	11	0.64	1
Cancer3	MFN	9-16-8-2	2.60	3.70	314	N/A	54
	MRAN	9-3-1	4.66	6.89	34	0.88	1
	GAPRBF	9-2-1	3.36	4.95	11	0.55	1

Table 3.2: Performance comparison for Cancer Problem

It can be seen that GAP-RBF produces smaller classification errors with a more compact network than MRAN. GAP-RBF also saves more than 50% cpu computation time than that of MRAN in all the experiments. It can be seen that the number of parameters to be adjusted at each iteration in GAP-RBF is far less than that of MRAN. The training speed of GAP-RBF is much faster than MRAN and it would be a great advantage in the real clinical applications, where the online monitoring require the system to identify abnormal situations as soon as possible. Figure 3.4 is an illustration of the learning speed comparison between MRAN and GAP-RBF on **Cancer3** dataset. For MFNs, the network structure is far more complicated than GAP-RBF. Sometimes it even has 2 hidden layers and the number of its hidden neurons can be as much as 10 times of that for GAP-RBF in some cases in **cancer** problem. The number of network parameters adjusted at each iteration is one or two order more of GAP-RBF's. More importantly, GAP-RBF only needs the data to be presented once or twice while MFNs require more than 40 learning epochs for the convergence in all the cases. Although the algorithm is developed based on

the assumption of uniform input distribution, the input distribution for the Cancer Problem is not completely uniform. From this experimental result, we can see that for this low dimensional problem with continuous input attributes, the GAP-RBF algorithm still achieves very good classification results in spite of not-so-uniform input distribution.

Figure 3.4: Learning Speed Comparison between MRAN and GAP-RBF for Cancer3.

3.4.3 Gene Problem

Splice junctions are points on a DNA sequence at which superfluous DNA is removed during the process of protein creation in higher organisms. The problem posed in this dataset is to recognize, given a sequence of DNA, the boundaries between the exons (the parts of the DNA sequence retained after splicing) and introns (the parts of the DNA sequence that are spliced out). This problem consists of

two subtasks: recognizing exon/intron boundaries (referred to as EI sites) and recognizing intron/extron boundaries (IE sites). In the biological community, IE borders are referred to as acceptors while EI borders are referred to as donors.

For the **Gene** data set 3 outputs indicate the class of the given gene sequence: IE, EI or none of them. 120 inputs represent 60 nucleotides in the gene sequence. In the original dataset, the input values used are -1 and 1. In the experiments, 0 and 1 are used to represent the nucleotides instead of -1 and 1. Four categories of nucleotides are encoded in binary numbers. In the total of 3175 examples, 2382 examples are used for training and 793 used for testing. There are no missing values for the problem. This preprocessing method is applied to both GAP-RBF and MRAN algorithm. The performance of both algorithms are evaluated on the three different permutations of the **Gene** data set, namely **Gene1**, **Gene2**, **Gene3** given in PROBEN1.

The final results for all the algorithms are shown in Table 3.3. The results of MFNs are for the suggested architectures for **Gene** data sets used for training of no shortcut connection networks in PROBEN1 and quoted from Table 10 of [105]. The parameters for these two algorithms are chosen as: $\epsilon_{max} = 7.0$, $\epsilon_{min} = 1.0$, $\gamma = 0.99$, $\kappa = 0.8$, $p_0 = 1.0$, $R_n = 1.0$, $q_0 = 0.0001$, together with $e_{min} = 1e124$ for GAP-RBF and $e_{min} = 1.0$ for MRAN and $e'_{min} = 0.5$, $n_w = 60$, $\delta = 0.000001$, 3 parameters for MRAN only.

Figure 3.5 is an illustration of how the neurons updates during the training progress for Gene3 case. GAP-RBF also has the least number of network parameters adjusted which is almost one order less than MRAN and MFNs. It also saves six seventh of the training time compared to MRAN in **Gene1**. It also achieves similar square error percentage and classification accuracy for testing sets with other algorithms in **Gene1**. However, in **Gene2** and **Gene3**, the classification performance is not so satisfactory. It can be seen that GAP-RBF algorithm does not work well

3.4 Experimental Results

69

Benchmark Data	Network Used	Network Architecture	Final Squared Error Percentage for Testing Set	Classification Error for Test Sets	Number of Network Parameters adjusted at each iteration	Time Used (s)	Number of Training Epochs
Gene1	MFN	120-4-2-3	8.66	16.67	503	N/A	124
	MRAN	120-10-3	9.26	13.60	643	6170	1
	GAPRBF	120-5-3	12.75	22.92	64	812	1
Gene2	MFN	120-4-2-3	9.54	18.41	503	N/A	321
	MRAN	120-8-3	8.67	12.34	515	3226	1
	GAPRBF	120-6-3	14.30	34.38	64	1381	1
Gene3	MFN	120-4-2-3	10.84	21.82	503	N/A	262
	MRAN	120-9-3	9.37	12.97	579	4549	1
	GAPRBF	120-7-3	15.05	38.29	64	2113	1

Table 3.3: Performance comparison for Gene Problem

on the **Gene** Problem, which is a high dimensional problem with discrete input attributes and an extremely non-uniform input distribution. All these factors lead to the problem of poor classification performance of the algorithm. For many online applications with high dimensional and discrete decision variables, the input distribution is expensive to get. They are not all necessarily uniform. The assumption of uniform input distribution will make the algorithm inapplicable for many real applications. Further work is needed to modify the algorithm to work for any arbitrary input distributions.

Figure 3.5: Neuron updating progress of MRAN and GAP-RBF for Gene3

3.5 Summary

In this chapter, the feasibility and effectiveness of GAP-RBF algorithm for classification applications have been explored. GAP-RBF was used to solve three classification problems from PROBEN1 database: **hearta**, **cancer** and **gene** problem. Comparison of the performance of GAP-RBF with MRAN and MFNs has also been made.

Results show that the GAP-RBF achieves a similar generalization performance with a more compact structure and a faster training than other neural network methods. Compared with MRAN, the GAP-RBF algorithm achieves a similar accuracy with a smaller network. GAP-RBF also saves more than 50% CPU training time in all the experiments. For MFNs, the network structure is far more complicated than GAP-RBF. Sometimes it even has more than one hidden layer and the number of its hidden neurons can be as much as 10 times of that of GAP-RBF algorithm. The number of network parameters adjusted at each iteration is one or two order more than that of MFNs. It should also be noted that GAP-RBF only needs the data to be presented once while MFNs require more than 40 learning epochs for the convergence in all the cases of these two problems.

The GAP-RBF algorithm is developed based on the assumption of uniform input distribution, but the input distributions for the Hearta Problem and Cancer Problem are not all uniform. From the experimental results, we can see that for low dimensional problems with continuous input attributes, the GAP-RBF algorithm still achieves very good classification results in spite of a non-uniform input distribution.

However, it can be seen that the GAP-RBF algorithm does not work well on the **Gene** Problem, which is a high dimensional problem with discrete input attributes and an extremely non-uniform input distribution. This clearly shows that some

problems exist for GAP-RBF algorithm.

First, the selection of the value of e_{min} is very difficult. For function approximation problems, e_{min} is a measure of expected learning accuracy and it is a threshold for adding or pruning a hidden neuron. It is defined as its average output over all the input samples. If the training and testing data have been scaled to the range of $[0, 1]$, its value should be no more than 1. Interestingly for the three classification problems we tried, two cases, viz Hearta Problem and Gene Problem, e_{min} has to use a value bigger than one. Especially for Gene Problem, the value of e_{min} are selected as big as $e_{min} = 10^{124}$. At this point, the value of e_{min} is far from the estimation of a neuron significance.

Second, heavy computation is a big problem for large input dimensional problems, like Gene Problem. When the number of hidden neurons grows to around 30, the network is trained at a very slow speed and a PC with 2.4GHz CPU and 512M memory crashes because of lack of memory. This problem may be attributed to the large requirement for memory and computational effort of the EKF method. When the numbers of inputs and neurons are big, large covariance matrix causes a computational overload. Therefore, for large input dimensional problems, we need to find methods which are less demanding in computational and memory requirements without much compromise in the terms of learning accuracy to replace the existing EKF method.

Further work is still needed to 1) find a more accurate way for the estimation of neuron significance, 2) modify the algorithm to work for any arbitrary input distributions, and 3) find a method which is less demanding in computational and memory requirements without much compromise in the terms of learning accuracy for the parameter adjustment.

In the next chapter, two improvement schemes are proposed to overcome these problems and a new algorithm named Fast GAP-RBF (FGAP-RBF) is developed.

Chapter 4

A New Fast Growing and Pruning-RBF (FGAP-RBF) Algorithm For Classification Problems

In the previous chapter, we have evaluated GAP-RBF for three classification problems. Compared with previous sequential learning networks, GAP-RBF produces a more compact network with similar generalization when the problems have small input dimensions. However, GAP-RBF may face difficulty in applications where the input dimension is high or the complexity of target mappings necessitates a large number of hidden neurons in the network. Although compared with other sequential learning algorithms GAP-RBF is able to reduce the number of neurons [52], in some cases the networks obtained by GAP-RBF are still large.

Another difficulty with the GAP-RBF network may be in the calculation of the significance of the neuron. It assumes that the input data is uniformly distributed

and the neuron significance is calculated based on this. If the input distribution is not uniform then the performance of GAP-RBF will be degraded.

To avoid this difficulty, a new scheme of calculating the significance of a neuron based on the most recently received M input data points is given in this chapter. As the input samples are fed into the network in a random order, these M samples are considered as the representation of the distribution in the whole input space. This method avoids the need for knowing the input distribution and also makes the significance computation fast and straightforward.

In the GAP-RBF algorithm, the parameters of the network are updated using an Extended Kalman Filter (EKF). When the number of inputs is large and as neurons grow, the size of the covariance matrix becomes large and this causes a computational overload. Decoupled EKF (DEKF) [106, 107] is introduced into the new algorithm to overcome this problem. The key feature of DEKF is to ignore the interdependencies of mutually exclusive groups of neurons, i.e., the cross correlation terms of the error covariance matrix $\mathbf{P}$. When a neuron's parameters are adjusted in the DEKF method, the cross correlation elements of that neuron to all the other neurons as well as all the other elements of the error covariance matrix $\mathbf{P}$ are assumed zero. In this chapter, DEKF is based on the assumption that there is no coupling between hidden neurons. Also, for a single neuron, its centers, width and weights are assumed uncorrelated. Thus, we obtain a significant reduction in computational cost per training instance and in storage requirements for the error covariance matrix.

In Section 4.1 the limitations of GAP-RBF algorithm are illustrated in detail by examining two bi-dimensional classification problems and Section 4.2 describes two improvement schemes to modify the GAP-RBF algorithm for a faster and more accurate implementation. Section 4.3 summarizes the solutions and finally the FGAP-RBF algorithm is presented.

4.1 Improvement on the Calculation of the Neuron Significance

4.1.1 Problems with the calculation of the neuron significance

As we mentioned in the previous chapter, the selection of the threshold value e_{min} for neuron significance is very difficult. Neuron significance is defined as its average output over all the input samples. If the training and testing data have been scaled to the range of $[0, 1]$, its value should be no more than 1. Interestingly for the three classification problems we tried, two cases, viz Hearta Problem and Gene Problem, have to use a value bigger than one. Especially for Gene Problem, the value of e_{min} was selected as big as $e_{min} = 10^{124}$. At this point, the value of e_{min} is too big to represent the real estimation of neuron significance.

In order to have a detailed look and find out the reasons causing this problem, we look at two simple two dimensional problems, which can help us to find out the problem and how it becomes serious when it comes to large input dimensional problems.

4.1.1.1 Concentric Problem

The data belongs to a two-class problem of uniform distributions with concentric circular form. The input samples are entirely contained in the square from $(0, 0)$, to $(1, 1)$. There are 2500 instances out of which 1579 is in class 1 and 921 in class 2. The points of class 1 and class 2 are uniformly distributed into a circle of radius 0.3 centered on $(0.5, 0.5)$ and an area with internal and external radius respectively equal to 0.3 and 0.5 respectively. The graphical presentation of this dataset is shown

in figure 4.1.

Figure 4.1: Concentric Problem

In the **Concentric** data set, the two coordinates are described by the two inputs and the two outputs represent the probability of belonging to each class. In the total of 2500 examples of the **Concentric** data set, 2000 examples are used for training and the rest 500 are used for testing.

The performance of GAP-RBF is shown in Table 4.1. The parameters chosen are as: $\epsilon_{max} = 0.4$, $\epsilon_{min} = 0.01$, $\gamma = 0.99$, $\kappa = 0.8$, $p_0 = 1.0$, $R_n = 1.0$, $q_0 = 0.0001$, and $e_{min} = 0.01$. $S(X) = 1$ for GAP-RBF algorithm.

	Number of Neurons	Classification Error for the testing set(%)	Training Time(s)
GAP	28	5.6	16.4990

Table 4.1: Performance for concentric problem

Here we plot the the data points as well as the positions of the hidden neurons on the

same figure as shown in Figure 4.2. The dots represent testing samples from class 1 and crosses represent testing samples from class 2. The stars represent the centers of hidden neurons and the blue circles are drawn corresponding to each individual centers μ_n and the radius is set as $r = \sigma_n$

Figure 4.2: Testing Samples and the positions of the hidden neurons for Concentric Problem.

4.1.1.2 Two-shell Problem

The **two-shell** problem is pattern recognition problem where the data contains two classes that are not separable by linear or quadratic discrimination functions. The distributions of the classes are described by a system of two quadratic equations with certain boundary constraints, as shown in the following

$$(y - y_0)^2 = \frac{b}{a}(x - x_0)^2 + G(0, c) \quad \text{with} \quad x_u \leq x \leq x_v, \quad (4.1)$$

where (x_0, y_0) , (x_u, x_v) and (a, b, c) are three sets of parameters that determine the location and shape of the sample distribution. $G(0, c)$ is a Gaussian noise function with a mean value equal to zero and a variance equal to a constant value c . Here c is set to 50 to create unclear boundaries that make the classification task even harder.

In the **two-Shell** data set, there are 1000 samples in each class. The two coordinates are described by two inputs and the two outputs represent the probability of belonging to each class. In the total of 2000 examples of the **Two-Shell** data set, 1500 examples are used for training and the rest 500 are used for testing. The figure presentation of this dataset is shown in figure 4.3.

Figure 4.3: Two-Shell Problem

The performance of GAP-RBF is shown in Table 4.2. The parameters are chosen as: $\epsilon_{max} = 0.4$, $\epsilon_{min} = 0.01$, $\gamma = 0.99$, $\kappa = 0.8$, $p_0 = 1.0$, $R_n = 1.0$, $q_0 = 0.0001$, and $e_{min} = 0.001$. $S(X) = 416.80$.

	Number of Neurons	Classification Error for the testing set(%)	Training Time(s)
GAP-RBF	14	2.6	5.53

Table 4.2: Performance for Two-Shell Problem

Here we plot the the data points as well as the positions of the hidden neurons on the same figure as shown in Figure 4.4. The dots represent testing samples from class 1 and crosses represent testing samples from class 2. The stars represent the centers of hidden neurons and the blue circles are drawn corresponding to individual centers μ_n and the radius is set as $r = \sigma_n$.

Figure 4.4: Testing Samples and the positions of the hidden neurons for Two-Shell Problem

From Figures 4.2 and Figure 4.4, we can see that the estimation of the neuron significance based on the uniform input distribution is far from accurate.

In GAP-RBF algorithm, the neuron significance is approximated by

$$E_{\text{sig}}(k) = \left| \frac{(1.8\sigma_k)^l \alpha_k}{S(X)} \right| \quad (4.2)$$

while assuming that all the input samples are distributed uniformly in the space of $S(X)$ and $\frac{m}{n} = \frac{(3.4\sigma_k)^l}{S(X)}$. However, when the points are more sparsely distributed within the activation area of a hidden neuron and more densely distributed outside, $\frac{m}{n} < \frac{(3.4\sigma_k)^l}{S(X)}$ and the real value of the neuron significance is smaller than the one we get by using Equation 4.2. The consequence is that a neuron with little contribution is added. Yet when the points are more densely distributed within Gaussian and more sparsely distributed outside, $\frac{m}{n} > \frac{(3.4\sigma_k)^l}{S(X)}$ and the real value of neuron significance is bigger than the one we get by using Equation 4.2. The consequence is that a neuron with a big contribution is pruned or not added.

These figures explain another problem with the GAP-RBF algorithm, which is the excessive value of $e_{\min}$. From Figure 4.2 and 4.4, we can see that not all the spaces within the hidden neuron are filled with uniformly distributed input samples. Big space within the neurons space are actually empty. However, with the uniform input distribution assumption, these non-existing input samples contributes greatly to the calculation of the neuron significance. In some cases, the neuron may take a larger space than the input space $S(X)$ itself. Thus the neuron significance we get will be bigger than 1. Since these are only two dimensional problems, for applications with large input dimensionalities, this estimation of the neuron significance would be very big, such as in Gene problem, where a value of 10^{124} is used for $e_{\min}$. What's more, when the range of the measure is so large, it is also very difficult to find the optimal parameter by trial and error. It has a bigger chance to select the suboptimal parameters. The generalization performance of the algorithm will be jeopardized as a consequence.

Especially when the number of hidden neurons is small, the distance between the

input sample and the nearest neuron is relatively big. If a new hidden neuron is added, it is more likely to cover a large space outside the input domain. For large input dimensional problems, the excessive computational demand of EKF does not allow many hidden neurons to be added. Thus the number of hidden neurons has to stay low.

This mutual influence of EKF and the inaccurate neuron's significance estimation will jeopardize the performance of GAP-RBF even further for classification problems.

To overcome this problem, we can either try to divide the input space into many small parts that can be assumed to have uniform distribution and then apply the GAP-RBF algorithm to these small parts, or we try to develop a new scheme that completely eliminates the assumption of uniform input distribution. For the first option, the estimation of the value of $S(X)$ is critical. Small changes in $S(X)$ can cause great oscillations of the number of hidden neurons in GAP-RBF algorithm. Besides, the accurate estimation of $S(X)$ is very computationally expensive or in some cases impossible to get. The difficulty in choosing the right value for $S(X)$ is also a reason that causes the GAP-RBF algorithm to perform inconsistently and end up with improper number of hidden neurons. In the next section, we propose a new scheme for the estimation of neuron significance using the second method.

4.1.2 Simplified Estimation of Neuron Significance

As we had discussed above, one of the problems with the GAP-RBF algorithm is in the calculation of the significance of neurons. It assumes that the input data is uniformly distributed. If the input distribution is not uniform then the performance of GAP-RBF will be affected. To overcome this, an alternate method to calculate the significance for any input distribution has been developed in Huang *et al* [53]

and is referred to as Generalized GAP-RBF. However, one should still know the distribution of the input samples beforehand.

In many real world problems, the input data distribution does not match the form of any known distribution function. In this case computation of significance is difficult. This calls for a new scheme that avoids the difficulty of knowing the input distribution as prior knowledge. The FGAP-RBF algorithm is developed aiming at overcoming this problem.

Since the neuron significance is an average contribution of the neuron to the output based on all the received input data, we can estimate it using a certain limited number of input samples. Suppose that the input observations are randomly generated and arrive with the same sampling distribution, the neuron significance $E_{\text{sig}}(k)$ (3.6) can be estimated by the most recently received M training samples as long as the memory factor M is large enough:

$$E_{\text{sig}}(k) \approx \frac{\|\alpha_k\|}{M} \sum_{i=n-M+1}^n \phi_k(\mathbf{x}_i). \quad (4.3)$$

If M is big enough, the distribution of these M recently received samples may represent the distribution of all the input data. The computation of neuron significance based on the M recently received samples will be more representative than blindly assuming a uniform input distribution or other form of distributions. From a statistical point of view, as the memory factor M grows bigger, the equation above will give a more accurate estimation of the neuron significance. When the memory factor M is as big as the input sample size n , the true value of the neuron significance will be given.

4.2 Improvement on the Parameter Adjustment

4.2.1 Problem with the EKF Method

Another problem with the GAP-RBF algorithm is the extremely heavy computational burden for problems of large input dimensions. This problem may attribute to the requirement for large memory and computational effort of EKF method. When the number of inputs is large and as the number of neurons grows, the size of the covariance matrix becomes large and this causes a computational overload. Furthermore, the GAP-RBF algorithm takes a long time and large memory space for applications where the target mapping is very complex and a large number of hidden neurons are required. Therefore, for problems of large input dimensions, we need to find methods which are less demanding in computational and memory requirements without much compromise in terms of learning accuracy to replace the existing EKF method.

As analyzed by Huang *et al* [52], the complexity of GAP-RBF is much less than other sequential learning algorithms since only the nearest neuron is checked for pruning. GAP-RBF needs only one step (times and division) operation for pruning checking and a simple sparse matrix operation for parameter adjustment at each step using the EKF. It should be noted that in GAP-RBF when the parameters of the nearest neuron are adjusted, all the cross correlation terms between the nearest neuron and all other neurons need to be updated as well. All remaining elements of the error covariance matrix $\mathbf{P}$ are unchanged from their previous values. If the obtained RBF network is not big, this EKF based parameter adjustment is efficient. However, if the network size or the input dimension is big, the cross correlation terms between the nearest neuron and all other neurons in the error covariance matrix $\mathbf{P}$ could be very big. Thus, the decoupled EKF (DEKF) algorithm proposed by Puskorius and Feldkamp [106, 107] is introduced to adjust network parameters to reduce the

computational complexity in this work. The key feature of DEKF is to ignore the interdependencies of mutually exclusive groups of neurons, i.e, the cross correlation terms of the error covariance matrix $\mathbf{P}$. When a neuron's parameters are adjusted in the DEKF based learning algorithm, the cross correlation elements of that neuron to all the other neurons of the error covariance matrix $\mathbf{P}$ are assumed zero.

The major difference between EKF and DEKF is the error covariance matrix $\mathbf{P}(n)$. Standard $\mathbf{P}(n)$ in EKF, as in most algorithms such as MRAN, is denoted as

$$\mathbf{P}(n) = \begin{bmatrix} \mathbf{P}_1(n) & \mathbf{P}_{12}(n) & \dots & \mathbf{P}_{1N_{hd}}(n) \\ \mathbf{P}_{21}(n) & \mathbf{P}_2(n) & \dots & \mathbf{P}_{2N_{hd}}(n) \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{P}_{N_{hd}1}(n) & \mathbf{P}_{N_{hd}2}(n) & \dots & \mathbf{P}_{N_{hd}}(n) \end{bmatrix}$$

where N_{hd} is the number of hidden neurons and $\mathbf{P}_{ab}(n)$ is the error covariance matrix between hidden neuron a and b . For a RBF network with N_{In} inputs, N_o outputs, size of $\mathbf{P}_{ab}(n)$ is $(N_{In} + N_o + 1)^2$.

In GAP-RBF algorithm, the complexity has been reduced by adjusting the parameters for the nearest neurons only. Therefore only the cross validation terms between the nearest neuron and all the other hidden neurons needed to be updated as shown here:

$$\mathbf{P}(n) = \left[\begin{array}{ccc|c|ccc} \mathbf{P}_1(n) & \mathbf{P}_{12}(n) & \dots & \mathbf{P}_{1nr}(n) & \dots & \mathbf{P}_{1N_{hd}}(n) \\ \mathbf{P}_{21}(n) & \mathbf{P}_2(n) & \dots & \mathbf{P}_{2nr}(n) & \dots & \mathbf{P}_{2N_{hd}}(n) \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \hline \mathbf{P}_{nr1}(n) & \mathbf{P}_{nr2}(n) & \dots & \mathbf{P}_{nr}(n) & \dots & \mathbf{P}_{nrN_h}(n) \\ \hline \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathbf{P}_{N_{hd}1}(n) & \mathbf{P}_{N_{hd}2}(n) & \dots & \mathbf{P}_{N_{hd}nr}(n) & \dots & \mathbf{P}_{N_{hd}}(n) \end{array} \right]$$

where nr is the number of the nearest neuron to the current input $\mathbf{x}(n)$.

Like the GAP-RBF algorithm, the FGAP-RBF algorithm also adjusts the parameters of the nearest neuron only. For the DEKF method in FGAP-RBF algorithm, all the cross validation terms between different neurons of error covariance matrix $\mathbf{P}(n)$ are assumed to be zero. Only the term of $\mathbf{P}_{nr}(n)$ needs to be updated.

$$\mathbf{P}(n) = \begin{bmatrix} \mathbf{P}_1(n) & 0 & \dots & 0 & \dots & 0 \\ 0 & \mathbf{P}_2(n) & \dots & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \mathbf{P}_{nr}(n) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & \dots & \mathbf{P}_{N_{hd}}(n) \end{bmatrix}$$

Furthermore, the parameters of a single neuron are decoupled into three groups: weights, center and width. These three groups of parameters are also assumed uncorrelated and only the elements on the diagonal of $\mathbf{P}_{nr}(n)$ need to be adjusted as shown in the following equation.

$$\mathbf{P}_{nr}(n) = \begin{bmatrix} \mathbf{P}_{\alpha_{nr}}(n) & 0 & 0 \\ 0 & \mathbf{P}_{\mu_{nr}}(n) & 0 \\ 0 & 0 & P_{\sigma_{nr}}(n) \end{bmatrix}$$

where $\mathbf{P}_{\alpha_{nr}}(n)$, $\mathbf{P}_{\mu_{nr}}(n)$ and $P_{\sigma_{nr}}(n)$ are the error covariance matrices for weights, center and width of neuron nr respectively.

For example, assuming a RBF network with N_{In} inputs, N_o outputs and N_{hd} hidden neurons, the size of the error covariance matrix $\mathbf{P}(n)$ for EKF is $[N_{hd} \times (N_{In} + N_o + 1)]^2$ and the number of values updated in $\mathbf{P}(n)$ during each network parameter adjustment for GAP-RBF algorithm is $(2N_{hd} - 1) \times (N_{In} + N_o + 1)^2$. However, for DEKF in FGAP-RBF algorithm the size of all the error covariance matrices $\mathbf{P}_i(n)$ summed together is $N_{hd} \times (N_{In}^2 + N_o^2 + 1)$ and the number of values updated in $\mathbf{P}_i(n)$

during each network parameter adjustment is $N_{In}^2 + N_o^2 + 1$. From the above calculations we can see that for the applications when the input and output dimensions are high, with the network growing bigger, the computational burden is also growing for updating the parameters. Therefore the training will slow down. DEKF saves a large computational effort and a large amount of memory space compared with EKF. Moreover, the training time for each sample is not affected by the growing network structure at all.

4.2.2 Parameters Adjusted by Decoupled Extended Kalman Filter (DEKF)

Given a network with N_{In} inputs, N_o output nodes and N_{hd} hidden neurons, partition all the parameters into g groups, with h_i parameters in group i . The parameters update for a training instance at time step n of DEKF is given by:

$$\mathbf{A}(n) = [\mathbf{R}(n) + \sum_{i=1}^g \mathbf{B}_i^T(n) \mathbf{P}_i(n) \mathbf{B}_i(n)]^{-1} \quad (4.4)$$

$$\mathbf{K}_i(n) = \mathbf{P}_i(n) \mathbf{B}_i(n) \mathbf{A}(n) \quad (4.5)$$

$$\mathbf{w}_i(n+1) = \mathbf{w}_i(n) + \mathbf{K}_i(n) \mathbf{e}(n) \quad (4.6)$$

$$\mathbf{P}_i(n+1) = [\mathbf{I} - \mathbf{K}_i(n) \mathbf{B}_i^T(n)] \mathbf{P}_i(n) + \mathbf{Q}_i(n) \quad (4.7)$$

In these equations, $\mathbf{R}(n)$ is a diagonal N_o -by- N_o matrix denoting the error covariance matrix for group i . $\mathbf{B}_i(n)$ is a h_i -by- N_o matrix containing the partial derivatives for the output node signal with respect to the parameters of group i . $\mathbf{P}_i(n)$ is a h_i -by- h_i matrix defined as the approximated conditional error covariance matrix for group i . $\mathbf{A}(n)$ is a N_o -by- N_o matrix referred as the global scaling matrix. $\mathbf{K}_i(n)$ is a h_i -by- N_o matrix containing the Kalman gains for parameter group i . $\mathbf{w}_i(n)$ is

a vector of length h_i containing the parameter values of group i . $\mathbf{e}(n)$ is the error vector of the network output layer. $\mathbf{Q}_i(n)$ is the artificial process noise added in to avoid convergence to local minimum.

Here in the FGAP-RBF algorithm, $g = 3 \times N_{hd}$ since the correlation matrix is first decoupled between hidden neurons into N_{hd} groups. Then parameters of a single neuron are decoupled into weights, center and width, three groups.

When the number of groups $g = 1$, the DEKF method degenerates into the EKF method. When the parameters of the network are decoupled, the elements of the error covariance matrix $\mathbf{P}(n)$ corresponding to parameters from different groups are ignored. When the number of groups is large, we obtain a significant reduction in computational cost per training instance and in storage requirements for $\mathbf{P}(n)$.

4.3 The New FGAP-RBF Algorithm

Based on the analysis above, we present the new Fast GAP-RBF(FGAP-RBF) algorithm.

The FGAP-RBF algorithm improves on GAP-RBF. As the way in which the GAP-RBF approximates the neuron significance brings in a larger error, we propose a new way to estimate the significance of the neuron by using a memory factor. This only remembers the past M input samples and estimates the neuron significance based on these samples. As the samples are shuffled into random orders each time when they are fed to the neural network, it is reasonable to consider that these M samples represent the input distribution so long as M is big enough. This new scheme avoids the need for knowing the input distribution and also makes the significance computation fast and straightforward.

To improve the training speed as well as avoid the excessive computation and mem-

ory request of the EKF, the DEKF (Decoupled EKF) is introduced into the new algorithm to adjust the parameters of the neural network.

Unlike the EKF, the DEKF only considers the interdependence of the weights within the group, rather than the interdependence of all the weights in the network [106,107]. The key features of DEKF is that the elements of the error covariance matrix corresponding to weights from different groups are ignored. When the number of groups is large, a significant reduction in computational cost and in storage requirements for the error covariance matrix is obtained. Thus the DEKF will retain the features of the EKF while providing flexibility in choosing the complexity of the computation [106,107]. In FGAP-RBF algorithm, DEKF is used to adjust the parameter of the nearest neuron only, therefore there is no coupling between hidden neurons. Also for a single neuron, its center, width and weights are also assumed uncorrelated. Thus, we obtain a significant reduction in computational cost per training instance and in storage requirements for error covariance matrix.

Thus, we have a new fast and efficient growing and pruning RBF (FGAP-RBF) algorithm as follows:

FGAP-RBF Algorithm:

Given an approximation error $e_{\min}$, for each observation $(\mathbf{x}_n, \mathbf{y}_n)$, where $\mathbf{x}_n \in \mathbf{R}^l$, do

1. **compute** the overall network output:

$$f(\mathbf{x}_n) = \sum_{k=1}^K \alpha_k \exp\left(-\frac{1}{\sigma_k^2} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2\right) \quad (4.8)$$

where K is the number of hidden neurons.

2. **calculate** the parameters required in the growth criterion:

$$\begin{aligned} \epsilon_n &= \max\{\epsilon_{\max} \gamma^n, \epsilon_{\min}\}, \quad (0 < \gamma < 1) \\ \mathbf{e}_n &= \mathbf{y}_n - f(\mathbf{x}_n) \end{aligned} \quad (4.9)$$

3. **apply** the criterion for adding or pruning neurons:

$$\text{If } \|\mathbf{x}_n - \boldsymbol{\mu}_{nr}\| > \epsilon_n \text{ and } \frac{\|\mathbf{e}_n\|}{M} \cdot \sum_{i=n-M+1}^n \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_n\|^2}{(\kappa \|\mathbf{x}_i - \boldsymbol{\mu}_{nr}\|)^2}\right) > e_{\min}$$

allocate a new hidden neuron $K + 1$ with

$$\begin{aligned} \alpha_{K+1} &= \mathbf{e}_n \\ \boldsymbol{\mu}_{K+1} &= \mathbf{x}_n \\ \sigma_{K+1} &= \kappa \|\mathbf{x}_n - \boldsymbol{\mu}_{nr}\| \end{aligned} \quad (4.10)$$

Else

adjust the network parameters α_{nr} , $\boldsymbol{\mu}_{nr}$, σ_{nr} for the nearest neuron only, using the DEKF method.

check the criterion for pruning the hidden neuron:

$$\text{If } \frac{\|\alpha_{nr}\|}{M} \cdot \sum_{i=n-M+1}^n \exp\left(-\frac{\|\mathbf{x}_i - \boldsymbol{\mu}_{nr}\|^2}{\sigma_{nr}^2}\right) < e_{\min}$$

remove the nr -th hidden neuron

reduce the dimensionality of DEKF

Endif

Endif

In this chapter, we examined two simple two dimensional classification problems to illustrate in detail the limitations of GAP-RBF algorithm. Then two simple ideas are proposed to reduce the complexity of GAP-RBF algorithm: 1) calculation of neuron significance based on recently received M input observations, and eliminating the uniform input distribution assumption in GAP-RBF algorithm; and 2) reduction of the computational load of EKF by using a Decoupled EKF (DEKF) scheme for adjusting the parameters of the network. The last part of this chapter presents a new sequential learning RBF network, referred to as Fast GAP-RBF (FGAP-RBF).

In the next chapter, the performance of the FGAP-RBF algorithm for classification problems is given. The FGAP-RBF algorithm is compared with other sequential learning algorithms, such as GAP-RBF and MRAN along with a batch learning classifier SVM on four benchmark classification problems. The influence of parameters and guidelines on the parameter selections are also studied.

Chapter 5

Performance Evaluation of FGAP-RBF for Classification Problems

In this chapter, the performance of the FGAP-RBF algorithm is compared with the GAP-RBF algorithm along with MRAN [141, 142]. Support Vector Machine (SVM) is a popular *batch* learning algorithm for classification problems. We also present a comparison of classification accuracies of FGAP-RBF with those of the SVM. The comparison is done based on the performance for the following real world benchmark classification problems, viz 1) Phoneme Problem [6]; 2) Segment Problem [15]; 3) Satimage problem [15]; and 4) DNA Problem [15]. The results indicate that FGAP-RBF produces higher classification accuracy with reduced training time compared with other sequential learning algorithms and obtains comparable classification accuracy to the batch learning algorithm—SVM. The influence of the parameter selection is studied and simpler settings for the parameters are explored, which makes the new algorithm easy and efficient to use.

5.1 Performance Evaluation of FGAP-RBF for Classification Problems

In this section, we present experimental results on four real benchmark classification problems: Phoneme, Segment, Satimage and DNA. All the datasets except for the Phoneme problem [6] are from the UCI machine learning repository [15]. A brief description of the problem details is given in Table 5.1. All the experiments in this study were carried out in a MATLAB environment on a Pentium 4, 2.4GHz PC with 512MB memory. For each problem, the number of output neurons for the RBF network is equal to the number of classes. The output neuron with the highest output indicates the class membership of the corresponding input. The parameters are obtained by trial and error. All the experimental results given represent the average of the results of ten trials. The data was reshuffled for each trial with random and different input orders.

Datasets	No. of training data	No. of testing data	No. of classes	No. of attributes
Phoneme	3603	1801	2	5
Segment	1600	710	7	19
Satimage	1500	500	6	36
DNA	2000	1186	3	180

Table 5.1: Details of classification datasets.

5.1.1 Phoneme Problem

This data set was in use in the European ESPRIT 5516 project [6]. The aim of this project is to develop and implement a real time analytical system for French and Spanish speech recognition. The present data set is to distinguish between nasal and oral vowels. The first five harmonics, normalized by the total energy were chosen to

characterize each vowel. A harmonic is signed as positive when it corresponds to a local maximum of the spectrum and negative otherwise.

For all the sequential algorithms, the parameters were chosen as: $\epsilon_{max} = 3$, $\epsilon_{min} = 0.1$, $\gamma = 0.99$. For MRAN algorithm, the learning accuracy was chosen as $e_{min} = 0.6$, pruning threshold was set as $e'_{min} = 0.4$. The overlapping factor was chosen as $\kappa = 0.8$. For the GAP-RBF algorithm, the parameters were chosen as $e_{min} = 0.01$ and $\kappa = 0.8$. The range of the input space $S(X)$ was calculated each time based on the training set before the learning started. For FGAP-RBF algorithm, the parameters were chosen as $e_{min} = 0.001$ and $\kappa = 0.6$. The number of the sliding window was chosen as $M = 20$. For SVM, the cost parameter $C = 8$ and kernel parameter $\gamma = 1$ were obtained by grid searching.

	Number of Neurons(SV)	Testing Accu- racy(%)	Standard De- viation(%)	Training Time(s)
MRAN	73	85.46	4.58	905.02
GAP-RBF	136.9	82.70	5.26	2947.6
FGAP-RBF	2613.3	89.02	0.69	314.84
SVM	1025.1	88.61	0.86	14.08

Table 5.2: Performance comparison for Phoneme Problem

5.1.2 Segment Problem

The Segment problem [15] is an image segmentation problem taken from the STAT-LOG database [86]. The problem posed in this dataset is to predict a classification based on the statistical features of the image. The samples were drawn randomly from a database of 7 outdoor images. The images were segmented manually to create a classification for every pixel. Each sample is a 3x3 region. The outputs represent different image classes, viz. brickface, sky, foliage, cement, window, path and grass. The decisions are made based on 19 input attributes, which include the row and column of the center pixel of the region, the number of pixels, the average

intensity, a 3-d nonlinear transformation of RGB and the contrast of horizontal or vertical adjacent pixels, etc.

The performance of FGAP-RBF together with the performance of GAP-RBF, MRAN as well as the SVM are shown in Table 5.3. We set the parameters $\epsilon_{max} = 60$, $\epsilon_{min} = 0.1$, $\gamma = 0.99$. For MRAN, the parameters are chosen as $e_{min} = 0.2$, $e'_{min} = 0.2$ and $\kappa = 0.8$. For GAP-RBF algorithm, the parameters were chosen as $e_{min} = 10^{28}$ and $\kappa = 0.8$. $S(X)$ was calculated each time based on the training data before the learning started. For FGAP-RBF algorithm, the parameters were chosen as $e_{min} = 0.005$, $\kappa = 0.9$ and $M = 20$. For SVM, the cost parameter $C = 64$ and kernel parameter $\gamma = 1$ are obtained by grid searching.

	Number of Neurons(SV)	Testing Accu- racy(%)	Standard De- viation(%)	Training Time(s)
MRAN	100.1	89.59	3.89	33614
GAP-RBF	92.8	82.37	2.58	15163
FGAP-RBF	916	95.69	0.68	83.79
SVM	378.3	96.95	0.46	0.78

Table 5.3: Performance comparison for Segment Problem

5.1.3 Satimage Problem

The Satimage data set [15] was generated from landsat multi-spectral scanner image data and it consists of the multi-spectral values of pixels in 3x3 neighborhoods in a satellite image, and the classification associated with the central pixel in each neighborhood. The aim is to predict this classification based on the multi-spectral values. The seven classes are red soil, cotton crop, grey soil, damp grey soil, soil with vegetation stubble, mixed classes and very damp grey soil. The samples in mixture class have all been removed because of doubts about their validity. The data is given in random order. In each line of data, the four spectral values for the

top-left pixel are given first, followed by the four spectral values for the top-middle pixel and then those for the top-right pixel, and so on. The pixels are read out in a sequence of left-to-right and top-to-bottom.

The performance of FGAP-RBF together with the performance of GAP-RBF as well as MRAN are shown in Table 5.4. The parameters are chosen as: $\epsilon_{max} = 10$, $\epsilon_{min} = 0.1$, $\gamma = 0.99$. For the MRAN algorithm, the learning accuracy was chosen as $e_{min} = 0.2$, pruning threshold was set as $e'_{min} = 0.25$. The overlapping factor was chosen as $\kappa = 0.8$. For the GAP-RBF algorithm, the parameters were chosen as $e_{min} = 10^{25}$ and $\kappa = 0.8$. The range of the input space $S(X)$ was calculated each time before the learning started. For the FGAP-RBF algorithm, the parameters were chosen as $e_{min} = 0.008$ and $\kappa = 0.9$. The memory factor was chosen as $M = 60$. For SVM, the cost parameter $C = 8$ and kernel parameter $\gamma = 1$ were obtained by grid searching.

	Number of Neurons(SV)	Testing Accuracy(%)	Standard Deviation(%)	Training Time(s)
MRAN	23.4	85.42	6.40	2738.3
GAP-RBF	31.6	81.20	9.69	3253.7
FGAP-RBF	436.3	90.2	1.42	64.31
SVM	690.5	90.7	1.42	1.25

Table 5.4: Performance comparison for Satimage Problem

5.1.4 DNA Problem

The problem posed in this data set is to recognize the boundaries between the exons (the parts of the DNA sequence retained after splicing) and introns (the parts of the DNA sequence that are spliced out) based on a sequence of DNA. DNA problem [15] consists of 3 subtasks: recognizing exon/intron boundaries (referred to as EI sites), recognizing intron/extron boundaries (IE sites) and neither of them.

For the DNA data set, three outputs are used for all the algorithms indicating the probability of the current input instance belonging to each class: IE, EI or none of them. 180 inputs represent 60 nucleotides in the gene sequence. Four categories of nucleotides are encoded by 3 binary indicator variables.

The performance of FGAP-RBF, GAP-RBF and MRAN are shown in Table 5.5. We set the parameters $\epsilon_{max} = 8$, $\epsilon_{min} = 1$, $\gamma = 0.99$. For MRAN algorithm, the learning accuracy was chosen as $e_{min} = 0.9$, pruning threshold was set as $e'_{min} = 0.5$. The overlapping factor was chosen as $\kappa = 0.8$. For GAP-RBF algorithm, the parameters were chosen as $e_{min} = 10^{190}$ and $\kappa = 0.8$. $S(X)$ was calculated each time before the learning started. For FGAP-RBF algorithm, the parameters were chosen as $e_{min} = 0.07$, $\kappa = 0.8$ and $M = 180$. For SVM, the cost parameter $C = 8$ and kernel parameter $\gamma = 2^{-6}$ are obtained by grid searching.

	Number of Neurons(SV)	Testing Accu- racy(%)	Standard De- viation(%)	Training Time(s)
MRAN	6	90.13	3.55	6273.4
GAP-RBF	6.125	79.89	8.94	3917.2
FGAP-RBF	453.9	93.46	0.06	1385.7
SVM	1096.9	95.45	0.03	6.41

Table 5.5: Performance comparison for DNA Problem

From the above results, we can see that the FGAP-RBF algorithm produced a much smaller classification error on the testing set than both the the GAP-RBF and MRAN algorithm. The FGAP-RBF algorithm reduced much of the training time while achieving a much better generalization performance.

Compared with the FGAP-RBF algorithm, the MRAN and GAP-RBF algorithm not only took long training time but also required excessive memory. For the DNA problem, when MRAN and GAP-RBF grew a network of no more than 30, it caused the PC with 2.4GHz cpu and 512M memory to crash down. This limitation prevented MRAN and GAP-RBF from implementing a more complex network structure

and achieving accurate mapping where a large number of hidden neurons are needed and as a consequence its generalization ability deteriorates greatly .

Compared with the SVM, FGAP-RBF algorithm obtained a similar classification accuracy on the testing set. The training time was also comparable, since SVM was running on C platform and it usually takes up to 50 times as much as the time to run the same program in MATLAB. *It should be noted that SVM is not a sequential learning algorithm and it needs to retrain whenever some new data comes. The aim of putting the results of SVM among a group sequential learning algorithms is to show that even compared with the most popular batch classifier, FGAP-RBF algorithm can still produce comparable results in terms of both accuracy and training time.*

5.2 Discussion

5.2.1 Impact of Memory Factor M

One of the difficulties of the GAP-RBF algorithm from a computational point of view is in the calculation of the significance of neurons, especially for higher dimensional problems. In FGAP-RBF, the neuron significance calculation is simplified by using only the most recent M observations.

$$E_{\text{sig}}(k) \approx \frac{\|\alpha_k\|}{M} \sum_{i=n-M+1}^n \phi_k(\mathbf{x}_i) \quad (5.1)$$

where M is the memory factor. M represents the number of past observations to be remembered. It is worth noting that the GAP-RBF is a memoryless learning algorithm and does not need any past observations for the calculation of significance; however it needs the knowledge of the distribution of the input data.

When the input distribution of the training data is known or easily estimated, these memoryless learning schemes are efficient. If the distribution is unknown or difficult to estimate, then memoryless learning becomes inefficient especially for discrete or high dimensional inputs.

The sequential learning algorithm MRAN also makes use of a sliding data window n_w to make the neuron growth smooth. Basically, MRAN looks at the past n_w output errors of the network whereas FGAP-RBF looks at past M input observations. MRAN uses a sliding window in the growing and pruning criteria to identify the neurons that contribute little to the network output. Selection of the appropriate size of these windows critically depends on the distribution of the data. In MRAN, choosing proper window sizes can only be done by trial and error based on experimental studies. The influence of the memory factor on the MRAN and FGAP-RBF algorithm will be studied and compared later in detail to assess the impact on classification performance.

5.2.2 Impact of Decoupling in the EKF

For parameter adjustment, MRAN algorithm uses EKF to adjust all the hidden neurons and in each epoch, MRAN is checking all the hidden neurons for pruning. For GAP-RBF algorithm, only the nearest neuron (which has the smallest Euclidean distance with the current input) is adjusted by EKF and only this adjusted neuron needs to be checked for pruning. That means newly added neurons is free from pruning-checks.

However, EKF algorithm demands a great computational effort and large memory. This prevents the algorithms, including GAP-RBF and MRAN, from getting a bigger network structure and achieving a higher classification accuracy, especially for large input dimensional problems.

Unlike EKF, DEKF only considers the interdependence of the weights within the group, rather than the interdependence of all the weights in the network [106, 107]. The key features of DEKF is that the elements of the error covariance matrix corresponding to weights from different groups are ignored. When the number of groups is large, a significant reduction in computational cost and in storage requirements for error covariance matrix is obtained. Thus DEKF will retain the features of EKF while providing flexibility in choosing the complexity of the computation [106, 107]. As a general rule, DEKF will produce a classification performance that approaches the EKF but is not expected to outperform it. On the other hand, DEKF is always computationally less demanding than the EKF [46].

Here, our new algorithm inherits the characteristics of GAP-RBF—adjusting the nearest neuron only. The weights are further decoupled within the neuron. Weights of each neuron are divided into 3 groups: weights connecting the hidden neuron and the output layer, center and width. The conclusion that the performance of DEKF is not supposed to surpass the performance of EKF, is valid only when both of them are feasible for that application. However, for some problems with a large number of training samples and high input dimensions, the excessive computational demand of EKF prevents the algorithms from making an accurate mapping from the input to the output space. Even the most up-to-date PC will crash after a certain number of neurons are added. Another issue is that even when EKF is feasible for some of these problems, the long training time makes these sequential learning algorithms completely lose the time advantage to batch learning. DEKF not only saves computational effort and memory requirement, but also make an accurate input-output mapping feasible to be implemented for complex problems. The sequential learning algorithms using DEKF also have a training speed comparable to SVM, which is one of the most popular batch learning classifiers these days. Thus it is reasonable that we got better results with DEKF while with EKF the generalization performance was much worse in most of the cases.

5.3 Improvement of Parameter Selection

A common problem with sequential learning algorithms is that there are too many parameters in the algorithm to be selected by trial and error. This makes the algorithm difficult and impractical to use in real world problems. However, compared with other sequential methods, the parameter selection for FGAP-RBF is much more intuitive and more straight forward.

In this section, we try to explore how these parameters influence the performance of FGAP-RBF and compare it with other sequential methods. Our aim is to find a simple way for setting the parameters without significantly affecting the performance in terms of testing accuracy and training time. In this part, we examine the parameters one by one in the following sequence: overlapping factor κ , memory impact factor M , scale of interest viz ϵ_{max} , ϵ_{min} , γ , and learning accuracy e_{min} and finally some guidelines for choosing them are given.

5.3.1 Selection of Overlapping Factor κ

In this set of experiments, we first take some measures to reduce the variations due to the different scales of various problems. For all the problems, the input attributes are linearly scaled to $[0, 1]$. Then we fix the parameter of overlapping factor κ as 0.8, since we noticed that in most of the classification cases, the algorithms perform best in term of classification accuracy when κ is set to 0.8 or 0.9. Experimental results show that the performance is not affected so much if κ is set to 0.8 for all the problems while tuning the other parameters. Therefore, to make it easy we fixed the overlapping factor κ and set it as 0.8 for all the experiments that followed. The rest of the setup of the experiment is the same as that of the previous experiments. The common parameters of GAP-RBF, FGAP-RBF and MRAN are fixed for each problem of all the four classification problems, viz Phoneme, Segment, Satimage,

DNA problem individually as: $\epsilon_{\max} = 0.5$, $\epsilon_{\min} = 0.01$, and $\gamma = 0.99$; $\epsilon_{\max} = 1.8$, $\epsilon_{\min} = 0.1$, and $\gamma = 0.99$; $\epsilon_{\max} = 1.5$, $\epsilon_{\min} = 0.1$, and $\gamma = 0.99$; $\epsilon_{\max} = 8$, $\epsilon_{\min} = 1$, and $\gamma = 0.99$. The rest of the parameters of FGAP-RBF, GAP-RBF and MRAN algorithms are given in Table 5.6, which are case dependent.

Datasets	Algorithms	Parameters
Phoneme	FGAP-RBF	$e_{\min} = 0.001$, $M = 20$
	GAP-RBF	$e_{\min} = 0.001$, $S(X) = 0.1411$
	MRAN	$e_{\min} = 0.6$, $e'_{\min} = 0.4$, $M = 50$
Segment	FGAP-RBF	$e_{\min} = 0.001$, $M = 280$
	GAP-RBF	$e_{\min} = 0.001$, $S(X) = 0.597$
	MRAN	$e_{\min} = 0.2$, $e'_{\min} = 0.2$, $M = 60$
Satimage	FGAP-RBF	$e_{\min} = 0.005$, $M = 60$
	GAP-RBF	$e_{\min} = 10$, $S(X) = 0.113$
	MRAN	$e_{\min} = 0.2$, $e'_{\min} = 0.3$, $M = 70$
DNA	FGAP-RBF	$e_{\min} = 0.07$, $M = 180$
	GAP-RBF	$e_{\min} = 1e190$, $S(X) = 1$
	MRAN	$e_{\min} = 0.8$, $e'_{\min} = 0.45$, $M = 50$

Table 5.6: Algorithm parameters for FGAP-RBF, GAP-RBF and MRAN

Table 5.7 shows the performance comparison of the three algorithms for the four classification problems. From these results, we can see that with these two measures to simplify the parameter selection, we still achieve similar classification accuracies as before. Therefore, we can take a further step to search for the easy way of setting other parameters.

However, we can see that even after normalization, the learning accuracy $e_{\min}$ of GAP-RBF algorithm for Satimage and DNA problem is still unreasonably big. This is due to two factors. One is the assertion that the input distribution is universally uniform, which is hardly true in real problems and cause great errors when estimating the neuron significance. The other one is the difficulty of obtaining proper value of the sample range $S(X)$, especially for high-dimensional problems, which is strongly related to the input distributions of the problems. Thus in the following experiments, we will not discuss the performance of the GAP-RBF algorithm on

Datasets	Algorithms	Training Time (s)	Testing Accuracy (%)	Standard Deviation (%)	No. of Neurons
Phoneme	FGAP-RBF	244.95	88.96	0.84	2622.6
	GAP-RBF	3815.8	76.96	3.24	177.4
	MRAN	5452.5	81.24	2.94	181.6
Segment	FGAP-RBF	58.39	96.32	0.60	408.6
	GAP-RBF	3393.1	88.61	1.39	56.2
	MRAN	3979.3	93.04	1.66	42
Satimage	FGAP-RBF	87.00	90.68	1.55	521.2
	GAP-RBF	10054	82.96	4.28	53.2
	MRAN	649.46	86.22	3.33	15
DNA	FGAP-RBF	1385.7	93.46	0.08	453.9
	GAP-RBF	30849	81.32	3.80	13.5
	MRAN	6273.4	90.13	1.05	6

Table 5.7: Performance of FGAP-RBF, GAP-RBF and MRAN for the benchmark classification problems.

two high-dimensional problems, namely Satimage problem and DNA problem until we find a good way to estimate the neuron significance accurately for the GAP-RBF algorithm.

5.3.2 Selection of Memory Impact Factor M

In this section, the influence of memory impact factor M is explored for the FGAP-RBF algorithm. The influence of M on the classification error, training time as well as the final network structure is studied. Here we allow M to range from 20 to 300 with an interval of 20, while keeping the other parameters unchanged.

Figure 5.1 to 5.4 shows the change of the classification error and the number of hidden neurons with M . From these figures, we can see that the number of hidden neurons goes all the way down with M increasing. This is because initially when M is small, the calculation of neuron significance is not accurate, which causes great oscillation of the number of hidden neurons and the network ends up with many

Figure 5.1: The influence of M on the classification error and network structure for the phoneme problem

Figure 5.2: The influence of M on the classification error and network structure for the segment problem

Figure 5.3: The influence of M on the classification error and network structure for the satimage problem

Figure 5.4: The influence of M on the classification error and network structure for the DNA problem

Figure 5.5: The influence of M on the classification error and training time for the phoneme problem

Figure 5.6: The influence of M on the classification error and training time for the segment problem

Figure 5.7: The influence of M on the classification error and training time for the satimage problem

Figure 5.8: The influence of M on the classification error and training time for the DNA problem

more hidden neurons. With the increase of M , the significance calculation becomes more accurate. The oscillation of the number of hidden neurons is also reduced and the network ends up with fewer hidden neurons.

Figure 5.5 to 5.8 shows the variation of classification error and the training time with M . The classification error goes up and down rather randomly with M . However the difference between maximum and minimum of the classification error is less than 2.5% for all the four applications. For the training time, we can see that the time is quite high initially and it goes down when M increases until the training time reaches its minimum and then begins to increase. This should be explained together with the previous figure. Initially when M is small, the network ends up with a large number of hidden neurons. Generally, it takes longer time to construct a bigger network structure. With the increase in M , the significance calculation becomes more accurate. The oscillation of the number of hidden neurons is reduced and the network ends up with fewer hidden neurons. The training time is also reduced. However, when M keeps growing, the computational effort for calculating the neuron significance also grows. The training time is the result of a balance between the effort of building a larger network with less neuron significance estimation workload and building a smaller network with larger neuron significance estimation workload. Usually with M growing, the size of the network will stabilize eventually and after that, the training time will keep growing as shown in most cases.

5.3.2.1 Comparison of MRAN and FGAP-RBF on Memory Impact Factor

The sequential learning algorithm MRAN also makes use of a sliding data window n_w to make the neuron growth smooth. Basically, MRAN looks at the past n_w output errors of the network whereas FGAP-RBF looks at past M input observations. Here we explore the influence of the chosen value of memory factor on both MRAN

and FGAP-RBF based on the Phoneme Problem. The experimental setting is the same in Table 5.6 only with M ranging from 20 to 180 in an interval of 20. From

Figure 5.9: The influence of Memory factor M on MRAN and FGAP-RBF

Figure 5.9, it is found that FGAP-RBF is not as sensitive to the chosen value of the memory factor as MRAN. Even when M is varied from 20 to 300, the difference in classification accuracies of FGAP-RBF is less than 1%. However, for MRAN we found that the classification accuracy varies by around 4% for n_w in the range of $[20, 180]$.

Therefore, we can set a universal value of M for the FGAP-RBF algorithm for all the classification applications. As for the n_w in the MRAN algorithm, we still have to figure out the best parameters through trial and error.

5.3.3 Selection of Scale of Interest

In this section, the impact of the choice of scale of interest ϵ , will be investigated and the possibility of simplifying selection of these parameters is explored.

The parameters deciding the scale of interest ϵ include 3 specific parameters, namely ϵ_{max} , ϵ_{min} and γ . ϵ_{max} is the largest scale of interest, typically the size of the entire input space of nonzero probability. γ is a decaying constant between 0 and 1. ϵ_{min} is the smallest scale. ϵ decays exponentially from ϵ_{max} until it reaches ϵ_{min} . The decaying of the distance allows a few basis functions with large width (smoother basis functions) initially, and with an increasing number of observations, more and more basis functions with smaller width are allocated to fine tune the mapping between the inputs and outputs.

Practically, these three parameters are much more related to the size, dimensionality and density distribution of the input space, etc. than the algorithms themselves. So we set the same parameters for all these three algorithms but different parameters for different applications. However, through the experiments, we notice that after normalization, the variation of the values for the scale parameters is greatly reduced for different applications. Here, we look into more experiments to see if we can set the same values for the scale parameters for all the four classification problems.

Experimental results shows that when the decaying speed γ is constant, say $\gamma = 0.99$, if ϵ_{max} and ϵ_{min} are big, the resulting network is not fine tuned and cannot produce precise and accuracy classification results. While keeping them relatively small within the range of acceptable values, we can make the algorithms maintain their performance by tuning other parameters, such as e_{min} .

Therefore, we take the smallest value of ϵ_{max} and ϵ_{min} in all the four applications as the universal parameters. Under these circumstances, we observe little change in the performance of these algorithms. We also examined the influence of the memory

impact factor on the FGAP-RBF algorithm for all the four classification problems. See Figure [5.10].

Figure 5.10: The influence of Memory factor M on FGAP-RBF

We can see that here the performance of the FGAP-RBF algorithm is still not sensitive to the choice of the memory impact factor M as before. So in the following experiments, we set $M = 40$ in all the applications of the FGAP-RBF algorithm, but choose the best value of n_w by trial and error for MRAN. In this set of experiments, the experimental setting is almost the same as before. All the input attributes are normalized to the range $[0, 1]$. The common parameters of GAP-RBF, FGAP-RBF and MRAN are fixed for all the four classification problems as: $\epsilon_{\max} = 0.5$, $\epsilon_{\min} = 0.01$, $\kappa = 0.80$, and $\gamma = 0.99$. The rest of the parameters of the FGAP-RBF, GAP-RBF and MRAN algorithms, which are case dependent, are given in Table 5.8. The performance comparison has been done based on the average of ten trials.

We compared the results from Table 5.9 with Table 5.7 and found that we can

Datasets	Algorithms	Parameters
Phoneme	FGAP-RBF	$e_{\min} = 0.001, M = 40$
	GAP-RBF	$e_{\min} = 0.01, S(X) = 0.1411$
	MRAN	$e_{\min} = 0.6, e'_{\min} = 0.4, M = 50$
Segment	FGAP-RBF	$e_{\min} = 0.005, M = 40$
	GAP-RBF	$e_{\min} = 0.001, S(X) = 0.597$
	MRAN	$e_{\min} = 0.2, e'_{\min} = 0.2, M = 60$
Satimage	FGAP-RBF	$e_{\min} = 0.005, M = 40$
	MRAN	$e_{\min} = 0.2, e'_{\min} = 0.3, M = 70$
DNA	FGAP-RBF	$e_{\min} = 0.1, M = 40$
	MRAN	$e_{\min} = 0.9, e'_{\min} = 0.5, M = 50$

Table 5.8: Algorithm parameters for FGAP-RBF, GAP-RBF and MRAN.

set the same parameters for the scale of interest for all the application by proper normalization and adjustment of other parameters. Thus the parameter selection process can be simplified and the FGAP-RBF algorithm can be used in a fast, easy and efficient way.

5.3.4 Selection of Learning Accuracy

Up to now, we have fixed all the other parameters except the learning accuracy $e_{\min}$. We can see from the experiments that the FGAP-RBF algorithm always selected a smaller value than the MRAN and GAP-RBF algorithms and ended up with a larger network with better generalization performance. Here we shall explore whether it is the choice of the learning accuracy $e_{\min}$ that cause this difference between FGAP-RBF and other sequential algorithms.

In this set of experiments, we set the same learning accuracy for all the algorithms for the same problem while keeping the rest of the parameters the same to the previous one as noted in Table 5.8. All the input attributes were normalized to the range $[0, 1]$. The common parameters of GAP-RBF, FGAP-RBF and MRAN are fixed for all the four classification problems as: $\epsilon_{\max} = 0.5$, $\epsilon_{\min} = 0.01$, $\kappa = 0.80$,

5.3 Improvement of Parameter Selection

112

Datasets	Algorithms	Training Time (s)	Testing Accuracy (%)	Standard Deviation (%)	No. of Neurons
Phoneme	FGAP-RBF	203.47	88.55	0.65	1903
	GAP-RBF	2661.2	76.70	3.64	150.4
	MRAN	5452.5	81.24	2.94	181.6
Segment	FGAP-RBF	85.15	96.65	0.55	766.3
	GAP-RBF	3291.3	88.42	1.88	55.4
	MRAN	7004.5	93.30	1.80	53.1
Satimage	FGAP-RBF	113.82	90.30	1.92	948.6
	MRAN	2469.4	86.36	2.09	20.4
DNA	FGAP-RBF	1741.3	93.41	0.05	633.3
	MRAN	6079	86.85	1.56	5

Table 5.9: Performance of FGAP-RBF, GAP-RBF and MRAN for the benchmark classification problems.

and $\gamma = 0.99$. The rest of the parameters of FGAP-RBF, GAP-RBF and MRAN algorithms are given in Table 5.10, which are case dependent. The performance comparison has been done based on the average of ten trials.

Datasets	Algorithms	Parameters
Phoneme	FGAP-RBF	$e_{\min} = 0.01, M = 40$
	GAP-RBF	$e_{\min} = 0.01, S(X) = 0.1411$
	MRAN	$e_{\min} = 0.01, e'_{\min} = 0.01, M = 50$
Segment	FGAP-RBF	$e_{\min} = 0.01, M = 40$
	GAP-RBF	$e_{\min} = 0.01, S(X) = 0.597$
	MRAN	$e_{\min} = 0.01, e'_{\min} = 0.01, M = 60$
Satimage	FGAP-RBF	$e_{\min} = 0.01, M = 40$
	MRAN	$e_{\min} = 0.01, e'_{\min} = 0.01, M = 70$
DNA	FGAP-RBF	$e_{\min} = 0.1, M = 40$
	MRAN	$e_{\min} = 0.1, e'_{\min} = 0.1, M = 50$

Table 5.10: Algorithm parameters for FGAP-RBF, GAP-RBF and MRAN (3)

In this set of experiments, the MRAN algorithm can finish none of the training tasks. The number of neurons grows with more and more training samples feeding into the network. The “out of memory” error would occur when a certain number of hidden neurons are added. For the phoneme problem, the experiment was deliberately

Datasets	Algorithms	Training Time (s)	Testing Accuracy (%)	Standard Deviation (%)	No. of Neurons
Phoneme	FGAP-RBF	151.74	86.60	1.16	970.7
	GAP-RBF	2661.2	76.70	3.64	150.4
	MRAN*	210510	83.23	—	511
Segment	FGAP-RBF	47.607	96.21	0.91	423
	GAP-RBF	1724.3	89.93	1.07	44.2
	MRAN*	2343	89.88	—	157.6
Satimage	FGAP-RBF	78.65	89.58	0.76	630.6
	MRAN*	497.77	82.16	—	90.4
DNA	FGAP-RBF	1741.3	93.41	0.048	633.3
	MRAN*	19.44	56.40	—	25.9

* Out of memory during the training. The results are based on one trial and obtained by reducing the number of training samples.

Table 5.11: Performance of FGAP-RBF, GAP-RBF and MRAN for the benchmark classification problems.

stopped when the number of hidden neurons is 511 and the number of trained samples is 3080, because it took too much time already. For the segment problem, the PC was out of memory when the number of hidden neurons is 183 and the number of trained samples is 283. When the number of training samples is set as 280, it was not always working, so it was set as 250. For the satimage problem and DNA problem, the PC was out of memory when the numbers of hidden neurons are 117 and 27, the numbers of trained samples are 153 and 27 respectively. Therefore the number of training samples were set to 120 and 26 individually. The number of testing samples remained the same.

As a matter of fact, these problems lie in EKF which is a part of the mechanism of MRAN algorithm. These results support of what was discussed earlier about the EKF. Generally DEKF will only produce a performance that may approach the EKF but usually not surpass it. However, for some problems with large number of training samples or high input dimensions, the excessive computational demand of EKF prevents the algorithms from making an accurate mapping from the input to

the output space. Even the most up-to-date PC will crash after a certain number of neurons are added. Another issue is that even when EKF is feasible for this kind of problems, the long training time makes these sequential learning algorithms completely lose the time advantage over batch learning. DEKF not only saves computational effort and memory requirement, but also make an accurate input-output mapping feasible to implement.

5.3.5 Guidelines for Parameter Selection

From the investigations we have done above, we find that although the FGAP-RBF algorithm has several parameters to select, most of them are not difficult to select since the generalization performance of the network is not sensitive to the selection of these parameters.

The overlapping factor κ is preferred to be big. It can be set as $\kappa = 0.8$ or $\kappa = 0.9$ for most of the cases. Experiments show that after normalization these three parameters, viz. ϵ_{max} , ϵ_{min} and γ , can take fixed numbers as universal values for all classification problems. The memory impact factor M depends on the size of the input samples. 5% to 20% of the training sample number can already produce good results and there are no big differences in the classification accuracies if M varies within this range. The learning accuracy e_{min} is the only parameter that needs to be selected by trial and error.

In this chapter, the performance of the FGAP-RBF network is compared with the GAP-RBF and MRAN along with SVM. The comparison is done based on the performance on four real world benchmark classification problems and the results show that FGAP-RBF produces higher classification accuracy with reduced training time compared with other sequential learning algorithms and obtains comparable classification accuracy compared with batch learning algorithm—SVM. The influence

of the selection of different parameters is also studied. The aim of discussing the selection of the parameter for FGAP-RBF algorithm is to show that the compared with previous sequential learning methods, such as MRAN, FGAP-RBF algorithm is less sensitive to the selection of the parameters. Some guidelines for the selection of parameters are also given.

In the next chapter, we will study specifically multi-category classification problems in the bioinformatics area, viz. microarray gene expression based cancer diagnosis problems. The gene expression-based classification problem is generally considered as a difficult task because the input dimension is usually very high and the number of samples is very small. The FGAP-RBF algorithm is applied for a multi-category cancer diagnosis problem based on microarray data.

Chapter 6

Microarray Gene Expression-Based Multi-category Classification Using Sequential Learning Algorithms

In the past, gene expression-based multi-category classification problems have mostly been addressed by modifying binary classification methods to a one-versus-all (OVA) or one-versus-one (OVO) comparison basis. This modification inevitably involves many classifiers and increases system complexities. They are also based on batch learning algorithms.

One method to overcome this problem is to use the efficient sequential learning algorithm-FGAP-RBF directly for multi-category classifications. The FGAP-RBF algorithm can perform incremental learning on the additional data directly. No training of the previous data is needed. This characteristic can reduce the learning complexity and improve the learning efficiency and is greatly favored in the real

implementation of a gene expression-based cancer diagnosis system.

In this chapter, we evaluate the classification performance of the FGAP-RBF algorithm on a real benchmark multi-category cancer diagnosis microarray data, namely the GCM dataset [110]. This is the first attempt in applying sequential learning algorithm for such problems. Results show that the FGAP-RBF algorithm can achieve better performance in terms of accuracy than the best results quoted in the literature. However, the FGAP-RBF algorithm always crashes after a few hidden neurons are added, when the number of genes used for classification approaches a thousand. Therefore, the microarray classification problems are further investigated using fast and efficient batch learning algorithms in the next chapter. Before describing the performance of FGAP-RBF, a brief description of the GCM problem is given in the following section.

6.1 Gene Expression-Based Multi-category Classification Problem

The GCM dataset is a collection of microarray data for snap frozen human tumor and normal tissue specimens, spanning 14 different tumor classes obtained from 6 institutions and hospitals. This is the first single reference database that covers the cancer diagnosis across all the common malignancies [110]. Ramaswamy *et al* [110] have made the data available on the website: http://www.broad.mit.edu/cgi-bin/cancer/publications/pub_paper.cgi?mode=view&paper_id=61. Linder *et al* [82] have performed the classification task on the GCM dataset using their newly developed neural network based method called SANN and achieved very good results. SANN differs from a conventional neural network in that it performs a pre-selection by a simple ANN at the first stage. Then it narrows down the decision scope by selecting the two most preferred classes with the highest activities at the

Tumor Type	Abbr.	Sample Size	# Training Data	# Testing Data
Breast	BR	8	6	2
Prostate	PR	8	6	2
Lung	LU	8	6	2
Colorectal	CO	8	6	2
Lymphoma	LY	16	13	3
Bladder	BL	8	6	2
Melanoma	ML	8	6	2
Uterus_Adeno	UT	8	6	2
Leukemia	LE	24	19	5
Renal	RE	8	6	2
Pancreas	PA	8	6	2
Ovary	OV	8	6	2
Mesothelioma	ME	8	6	2
CNS	CNS	16	13	3

Table 6.1: GCM Dataset - Tumor classes and their details

corresponding output neurons. Then a second ANN is used for the final decision on these two selected classes. The second stage of the SANN is also a pairwise comparison in essence. For the purpose of comparison, we carried out the classification on the same data using the experimental methods similar to [82] and is described in subsections 6.2.1. The data is from the file named GCM_Training.res, which can be downloaded from the above mentioned website. This file contains expression profiles comprising 16,063 genes and 144 primary tumor samples spanning 14 common tumor types. The detailed information on the pattern numbers of each class and the number of training patterns and testing patterns for each class of tumors are shown in Table 6.1. For the FGAP-RBF, ELM and SVM-OVO algorithms all the input attributes are scaled to $[0, 1]$.

For the purpose of comparison, we use the same gene selection method as was used in Linder *et al* [82] and Ramaswamy *et al* [110], namely the recursive feature elimination method. As introduced in Ramaswamy *et al* [110], for a microarray dataset with n genes, each SVM-OVA classifier produces a hyperplane w , which is a vector of n elements each corresponding to the expression of a particular gene.

The absolute magnitude of each element in w can be considered as a measure of the importance of the corresponding gene. Each SVM-OVA classifier is first trained with all the genes, then genes corresponding to the bottom 10% $|w_i|$ are removed. Each classifier is retrained after the removal of the genes. This process is repeated iteratively and the rank of all the genes based on the statistical significance of each class can be obtained. The most significant 14, 28, 42, 56, 70, 84 and 98 genes selected by this method can be found in the file OVA_MARKERS.xls from the same website. It also gives the most significant m genes for each of these 14 classes ($m = 1, \dots, 7$). Similar to Linder *et al* [82], the FGAP-RBF algorithm is also evaluated for these most significant genes.

6.2 The Sequential Learning Method—FGAP-RBF

Sequential learning algorithms can perform incremental learning and do not need to retrain the network on all the previous data whenever a new data point arrives. They also have a mechanism to retain information and reduce noise by adjusting the network structure, which makes it more ready for the future training samples and less necessary to find the optimal parameters for the classifier every time training is performed. Due to the lack of training samples and the rapid development and propagation of microarray technology, the need to retrain the classifier with new samples could occur frequently. For batch learning algorithms, finding the optimal parameters would be a time consuming procedure, which makes it hard to update of the classifiers as frequently as needed in practice.

Also, sequential learning algorithms can perform gene selection on a small set of samples, and perform the training on a much bigger set of samples using the same initialization of parameters. This is because the parameters of the network structure, which are associated with gaining information and reduction of noise, are adjusted

online sequentially. While for batch learning algorithms, the best parameters for gene selection on the small set of samples mostly perform badly on the big set of real learning samples. Thus, there are two effects. First, optimal parameters for the classifier should be searched twice for both gene selection and real training. Second, with new data coming, gene selection with more training data should be performed more frequently for batch learning algorithms than sequential ones. Especially under the condition that gene selection should be performed among thousand of genes, it would be another heavy computational burden for the system.

Therefore, sequential algorithms make the system more applicable in practice and it could be a good method for training the classifiers for microarray applications.

In this chapter, the FGAP-RBF algorithm is used for direct multi-category classification for a benchmark multi-category cancer diagnosis problem based on microarray data. Other sequential learning algorithms such as GAP-RBF and MRAN are not capable of handling the higher input dimensionality of gene expression-based classification problems. In this chapter, the results of the FGAP-RBF algorithm are compared with the results of other batch learning algorithms in the literature for the same problem.

6.2.1 Experimental Methods

In our experiments, all the input attributes were scaled in the range $[0, 1]$. 1-of-c coding was used for the target variable. The number of output neurons was equal to the number of classes. The parameters of the FGAP-RBF algorithm selected for the problems are: $\epsilon_{\max} = 0.5$, $\epsilon_{\min} = 0.01$, $\gamma = 0.99$, $e_{\min} = 0.01$, $M = 10$ and $\kappa = 0.90$. The proportion of the division between training data and testing data is also 4:1 which is the same as in [82]. For the FGAP-RBF algorithm, each training sample is presented to the network only once for training. Therefore, there is only one learning

#Genes	FGAP-RBF	SD(%)	SANN	SVM-OVA	ANN
14	65.45	6.67	68.75	69	50
28	69.09	5.39	71.53	70.5	59.72
42	75.15	5.55	72.92	70.5	64.58
56	79.39	4.66	79.17	71.5	70.14
70	80.30	3.89	76.4	71.5	59.72
84	82.12	3.94	80.56	71.75	70.83
98	82.42	5.88	77.08	72	72.22

Table 6.2: Classification Accuracy (%) for different algorithms.

epoch for the whole training data. The results are obtained based on the average of 10 trials for each selected number of genes. The 10 trials are of different shuffles between the training and testing samples while keeping the proportion between the number of training samples and testing samples unchanged.

6.2.2 Performance Evaluation

6.2.2.1 Classification Accuracy

The FGAP-RBF algorithm was evaluated for the benchmark dataset GCM. The results are presented and compared with other results from literature in Table 6.2 and Figure 6.1. All the results of SANN, SVM-OVA and ANN algorithms are quoted from [82].

From the result we can see that FGAP-RBF and SANN algorithm achieved a much higher classification accuracy on the testing data than the SVM-OVA and ANN method for genes greater than 28. For these two algorithms, the accuracy tends to grow with the increase of the number of genes used. FGAP-RBF exceeds SANN in accuracy for all the 5 experiments with bigger numbers of genes. The inferior performance of FGAP-RBF algorithm on the smaller gene numbers may due to its decoupling effect of the DEKF method used for parameter adjustment, which jeopardizes its accuracy in simple mappings. However, with more genes used, FGAP-

Figure 6.1: Comparison of the classification accuracies of FGAP-RBF, SANN, SVM-OVA and ANN.

Number of Genes	FGAP-RBF
14	55.9
28	74.8
42	80.5
56	81
70	82.2
84	81.3
98	83.4

Table 6.3: Number of hidden neurons for GAP-RBF algorithm.

RBF shows more advantage to handle complex classification problems.

It should be noted that for all the 7 experiments with different number of genes we fixed the same settings of the parameters of the FGAP-RBF algorithm. That means for experiments even without tuning of the parameters the results are still better or quite comparable to other algorithms. While for algorithms that have sensitive parameters, such as SVM, the search for the optimal parameters might be carried out first for each experiment.

6.2.2.2 Network Complexity

The size of the network structure is shown in Table 6.3. In [82], there are 1 ANN and up to 91 SANN have to be trained for each experiment. For each network, there are 5 modules each consisting of 10 hidden neurons. That means for each experiment there are up to 4600 hidden neurons involved in the training process. While for FGAP-RBF algorithm, only around 80 hidden neurons are involved in the training.

6.2.2.3 Computational Time

The training time (CPU time) of the FGAP-RBF algorithm for different number of genes are shown in Table 6.4. It only takes one epoch for FGAP-RBF to finish the training process. However, for the SANN algorithm up to 1000 epochs are needed

Number of Genes	FGAP-RBF
14	0.454
28	0.757
42	1.165
56	1.594
70	2.147
84	2.974
98	4.386

Table 6.4: Training time(s) for GAP-RBF networks

to finish the training process. Also, it only takes a few seconds for FGAP-RBF algorithm to finish the training. Yet for SANN algorithm, it takes 150 minutes only to train and test the data with 14 genes 5 times, when a Pentium 4 double processor (2×1 GHz) is used. It is quite comparable to the Pentium 4 2.4GHZ processor we used for the experiment.

6.2.2.4 Classification Performance on Individual Category

Furthermore, the classification power of the FGAP-RBF algorithm on each tumor class is investigated. The FGAP-RBF algorithm was trained and tested with 98 genes for 10 times with different shuffles of the training and testing data. Specific classification results for each category are shown in Figure 6.2 and Figure 6.3. The results from [82] are also plotted on the figure. From the results we can see that the FGAP-RBF algorithm exceeds SANN for almost every tumor category. For some classes, such as Ovary and Pancreas, SANN failed to classify correctly for more than half of the samples. Yet FGAP-RBF algorithm still classifies these classes with a high accuracy. This indicates that for this multi-category classification with 14 classes, SANN has shown some preference to patterns in other classes over the ones of Ovary and Pancreas during the training. This results in poor classification of the patterns in the latter classes. The FGAP-RBF algorithm obviously has a better balance among the patterns of all the classes during the training for multi-category

Figure 6.2: Comparison of individual category classification: Breast, Prostate, Lung, Colorectal, Lymphoma, Bladder and Melanoma.

classification with a big number of classes.

The confusion matrix of FGAP-RBF algorithm for testing data only is also given in Table 6.5. This is also based on the average of 10 trials on different shuffle of the training and testing data with 98 genes. From this table we can see that the accuracies for the tumor classes with large number of patterns usually achieve a much higher classification accuracy than other classes. This indicates that with more tumor samples available we can further improve the classification accuracy and make this classifier more reliable.

Figure 6.3: Comparison of individual category classification: Uterus_Adeno, Leukemia, Renal, Pancreas, Ovary, Mesothelioma and CNS.

	BR(%)	PR(%)	LU(%)	CO(%)	LY(%)	BL(%)	ML(%)	UT(%)	LE(%)	RE(%)	PA(%)	OV(%)	ME(%)	CNS(%)
BR	80		5								10	5		
PR	10	75	5											10
LU		20	75										5	
CO				95									5	
LY					96.7				3.3					
BL			5	5		75	5	5				5		
ML						5	80	10					5	
UT							30	55				15		
LE									100					
RE	5							10		85				
PA	10	5		5				5			70			
OV			5			10		25		20		40		
ME				5			5	5					85	
CNS														100

Table 6.5: Confusion Matrix for the test data for FGAP-RBF algorithm

6.3 Summary

In this section, the FGAP-RBF algorithm has been compared with the best results in the literature on a real benchmark multi-category cancer diagnosis problem. Study results show that the FGAP-RBF algorithm achieves a higher classification accuracy than the previous best results with a lesser training time and a smaller network structure. In this study, we also find that although compared with the GAP-RBF algorithm, the FGAP-RBF algorithm has improved the capability in handling the problems of high input dimensions up to a few hundred, FGAP-RBF can not handle the problems with dimensions up to a thousand. When the classification with inputs more than a thousand is performed using the FGAP-RBF algorithm, it always causes a Pentium 4, 2.4GHZ PC with 512MB memory to crash only after a few hidden neurons are added. Therefore, in the next chapter we look into a fast and efficient batch learning algorithm that can handle problems of very large input dimensions.

Chapter 7

Microarray Gene Expression-Based Multi-category Classification Using Batch Learning Algorithms

As mentioned in the previous chapter, the FGAP-RBF algorithm has difficulties in handling the problems with large input dimensions. In this chapter, we examine the performance of a recent, fast and efficient batch learning algorithm—the Extreme Learning Machine (ELM) algorithm for microarray gene expression-based multi-category classification problems. Besides the GCM dataset, a comprehensive evaluation of ELM for multi-category microarray classification problems for cancer diagnoses is performed on a wider basis using three gene selection methods on five real benchmark microarray datasets. The datasets studied are : the HBC dataset, Lung dataset, Lymphoma dataset, MLL dataset and NCI60 dataset.

Linder *et al* [82] have evaluated the SANN method on the GCM dataset using

different number of genes and compared their results with SVM-OVA and ANN methods. However, they have carried out this study using only the 144 training samples by performing a 5 fold internal cross validation. The resulting validation accuracy was then compared with the testing accuracy reported by Ramaswamy *et al* [110]. As pointed by Linder *et al* [82] it should be noted that the results of [82] can not be directly compared with those of [110] as [82] has reported the internal validation accuracy on 144 training data whereas [110] has reported the testing accuracy on the 144+46 samples.

Hence, in order to compare on the same basis, we performed both internal validation as done by Linder *et al* [82] and also external validation as carried out by Ramaswamy *et al* [110]. Then the comparison can be done on the same basis. In this chapter, only the experimental results using the internal validation results are reported. Experimental results for three datasets, viz. GCM dataset, Lung dataset and Lymphoma dataset, using external validation are presented in Appendix A.

Before a detailed performance comparison is done, it is worth while to recapitulate here the meaning of the internal and external validation procedures. For internal validation:

1. The dataset is randomly divided into two parts: a training set and a testing set. The training is performed on the training set, then the testing is performed on the testing set.
2. The dataset is reshuffled 10 times and Step 1 is performed 10 times for each reshuffled dataset. An average testing accuracy is calculated.
3. For Each pair of parameter, step 2 is performed. Therefore, there is an average testing accuracy corresponding to each specific pair of parameter.
4. Finally the best accuracy together with the parameter is reported.

For external validation:

1. The dataset is randomly divided into two parts: a training set(a) and a testing set.
2. The training set(a) is divided into a new training set(b) and a validation set. The training is performed on a training set(b), then the testing is performed on a validation set.
3. The training set(a) is reshuffled 10 times and step 2 is performed 10 times for each reshuffled dataset(a). An average validation accuracy is calculated.
4. For Each pair of parameter, step 3 is performed. There is an average validation accuracy corresponding to each specific pair of parameter.
5. The best pair of parameters are selected according to the validation accuracy.
6. The training is performed on a training set(a) using the selected parameter, then the testing is performed on a testing set.
7. Step 1-6 are repeated for 100 times for different division of training and testing sets. (Note that each time, the best parameter selected according to the validation accuracy will not be the same.)
8. Finally an average testing accuracy over 100 times is reported. No best parameter can be reported.

In this chapter, for the GCM dataset, we have compared the results of ELM with that of Linder *et al* [82] using similar conditions as used by them. We have also used SVM-OVO in our comparison study. For the other data sets, the comparison has been done between ELM and SVM-OVO using the internal validation.

The results using internal validation indicate that ELM can perform direct classification for such multi-category microarray problems in a fast and efficient manner.

ELM produces higher classification accuracies than those obtained by SANN and SVM-OVO with a more compact network structure and a shorter training time for GCM dataset.

Studies on other data sets indicate that the total training time for ELM is always smaller than that of SVM-OVO. For the classification accuracy when the numbers of genes used are large, ELM always performs better than SVM-OVO. Similar results can be obtained using external validation in Appendix A. Before precessing with the results,a brief overview of ELM is given. For further details, refer to [57].

7.1 A Brief Introduction of the ELM Algorithm

Generally, in a feedforward ANN training scheme, parameters (like weights and biases) of all the layers need to be tuned by the learning algorithm. Gradient descent-based methods and their variations such as Back-Propagation (BP) have formed the backbone of most of the learning algorithms of feedforward neural networks over the last two decades. However, it should be noted that these gradient descent based learning methods are generally slow due to improper learning step size and may converge to local minima. Also, many iterative learning epochs are required by such learning algorithms to obtain good performance.

Recently, Huang *et al* [58] and [54,55] have proposed a new learning algorithm called Extreme Learning Machine (ELM) for Single-hidden Layer Feedforward neural Networks (SLFNs). In ELM, one may randomly choose and fix the input weights and the hidden neurons' biases and then determine the output weights of SLFNs [58]. Input weights are the weights of the connections between the input neurons and hidden neurons and output weights are the weights of the connections between hidden neurons and output neurons. After the input weights and the hidden layer biases are chosen randomly, SLFN can be considered as a linear system and the

output weights can be determined through a generalized inverse of the hidden layer output matrices. Studies have shown [58] that ELM has good generalization performance and can be implemented easily. Many nonlinear kernel functions can be used in ELM, like sigmoid, sine, hardlimit [56], radial basis functions [54, 55] and complex activation functions [78], etc. Activation functions used in ELM may be nondifferentiable or even discontinuous.

7.1.1 Mathematical Description of Unified SLFN

The output of a SLFN with $\tilde{N}$ hidden nodes (additive or RBF nodes) can be represented by

$$f_{\tilde{N}}(\mathbf{x}) = \sum_{i=1}^{\tilde{N}} \beta_i G(\mathbf{a}_i, b_i, \mathbf{x}), \quad \mathbf{x} \in \mathbf{R}^n, \mathbf{a}_i \in \mathbf{R}^n \quad (7.1)$$

where $\mathbf{a}_i$ and b_i are the learning parameters of the hidden nodes and β_i the weight connecting the i -th hidden neuron to the output node. $G(\mathbf{a}_i, b_i, \mathbf{x})$ is the output of the i -th hidden node with respect to the input $\mathbf{x}$. For additive hidden node with the activation function $g(x)$ (e.g sigmoid or threshold), $G(\mathbf{a}_i, b_i, \mathbf{x})$ is given by

$$G(\mathbf{a}_i, b_i, \mathbf{x}) = g(\mathbf{a}_i \cdot \mathbf{x} + b_i), \quad (7.2)$$

where $\mathbf{a}_i$ is the weight vector connecting the input layer to the i -th hidden node and b_i is the bias of the i -th hidden node. $\mathbf{a}_i \cdot \mathbf{x}$ denotes the inner product of vectors $\mathbf{a}_i$ and $\mathbf{x}$ in $\mathbf{R}^n$.

For a RBF hidden node with an activation function $g(x)$ (e.g Gaussian), $G(\mathbf{a}_i, b_i, \mathbf{x})$ is given by

$$G(\mathbf{a}_i, b_i, \mathbf{x}) = g(b_i \|\mathbf{x} - \mathbf{a}_i\|), \quad b_i \in R^+ \quad (7.3)$$

where $\mathbf{a}_i$ and b_i are the center and impact factor of i -th RBF node. R^+ indicates positive real value. The RBF network is a special case of SLFN with RBF nodes in

its hidden layer. Each RBF node has its own centroid and impact factor, and its output is given by a radially symmetric function of the distance between the input and the center.

7.1.2 Extreme Learning Machine

In supervised batch learning, the learning algorithms use a finite number of input-output samples for training. Here, we consider N arbitrary distinct samples $(\mathbf{x}_i, \mathbf{t}_i) \in \mathbf{R}^n \times \mathbf{R}^m$, where $\mathbf{x}_i$ is a $n \times 1$ input vector and $\mathbf{t}_i$ is a $m \times 1$ target vector. If a SLFN with $\tilde{N}$ hidden nodes can approximate these N samples with zero error, it then implies that there exist β_i , $\mathbf{a}_i$ and b_i such that

$$f_{\tilde{N}}(\mathbf{x}_j) = \sum_{i=1}^{\tilde{N}} \beta_i G(\mathbf{a}_i, b_i, \mathbf{x}_j) = \mathbf{t}_j, \quad j = 1, \dots, N. \quad (7.4)$$

Equation (7.4) can be written compactly as:

$$\mathbf{H}\boldsymbol{\beta} = \mathbf{T} \quad (7.5)$$

where

$$\mathbf{H}(\mathbf{a}_1, \dots, \mathbf{a}_{\tilde{N}}, b_1, \dots, b_{\tilde{N}}, \mathbf{x}_1, \dots, \mathbf{x}_N) = \begin{bmatrix} G(\mathbf{a}_1, b_1, \mathbf{x}_1) & \cdots & G(\mathbf{a}_{\tilde{N}}, b_{\tilde{N}}, \mathbf{x}_1) \\ \vdots & \cdots & \vdots \\ G(\mathbf{a}_1, b_1, \mathbf{x}_N) & \cdots & G(\mathbf{a}_{\tilde{N}}, b_{\tilde{N}}, \mathbf{x}_N) \end{bmatrix}_{N \times \tilde{N}} \quad (7.6)$$

$$\boldsymbol{\beta} = \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_{\tilde{N}}^T \end{bmatrix}_{\tilde{N} \times m} \quad \text{and} \quad \mathbf{T} = \begin{bmatrix} \mathbf{t}_1^T \\ \vdots \\ \mathbf{t}_N^T \end{bmatrix}_{N \times m} \quad (7.7)$$

$\mathbf{H}$ is called the hidden layer output matrix of the network [50]; the i th column of

$\mathbf{H}$ is the i -th hidden node's output vector with respect to inputs $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$ and the j -th row of $\mathbf{H}$ is the output vector of the hidden layer with respect to input $\mathbf{x}_j$.

The ELM algorithm is based on the following two principles.

1. When the number of training samples equals the number of hidden nodes, i.e, $N = \tilde{N}$, one can randomly assign the parameters of the hidden nodes (the input weights and biases for additive hidden nodes or the centers and impact factors for RBF) and based on this calculate the output weights by simply inverting $\mathbf{H}$ and realize a zero training error. Calculation of the output weights is done in a single step here. There is no need for any time-consuming training procedure where the network parameters are adjusted iteratively with control parameters (learning rate and learning epochs, etc).
2. When the number of training samples is greater than the number of hidden nodes, i.e, $N > \tilde{N}$, one can still randomly assign the parameters of hidden nodes and calculate the output weights by using a pseudo-inverse of $\mathbf{H}$ to give a small non-zero training error $\epsilon > 0$. Here also the output weights' calculation is done in a single step.

In real applications, the number of hidden nodes $\tilde{N}$ will always be less than the number of training samples N and hence the training error can not be made exactly zero but can approach a nonzero training error ϵ . The hidden node parameters $\mathbf{a}_i$ and b_i (input weights and biases or centers and impact factors) of SLFNs *need not be tuned during training and may simply be assigned with random values*. Equation (7.5) then becomes a linear system and the output weights $\boldsymbol{\beta}$ are estimated as:

$$\hat{\boldsymbol{\beta}} = \mathbf{H}^\dagger \mathbf{T} \quad (7.8)$$

where $\mathbf{H}^\dagger$ is the Moore-Penrose generalized inverse [121] of the hidden layer out-

put matrix $\mathbf{H}$. The ELM algorithmⁱ only consists of three steps and can then be summarized as:

ELM Algorithm: Given a training set $\aleph = \{(\mathbf{x}_i, \mathbf{t}_i) | \mathbf{x}_i \in \mathbf{R}^n, \mathbf{t}_i \in \mathbf{R}^m, i = 1, \dots, N\}$, activation function $g(x)$, and hidden neuron number $\tilde{N}$,

1. Assign random hidden nodes by randomly generating parameters $(\mathbf{a}_i, b_i)$, $i = 1, \dots, \tilde{N}$.
2. Calculate the hidden layer output matrix $\mathbf{H}$.
3. Calculate the output weight β :

$$\beta = \mathbf{H}^\dagger \mathbf{T} \quad (7.9)$$

Universal approximation capability of ELM has been analyzed in Huang, *et al* [51] using an incremental method and it has been shown that single SLFNs with randomly generated additive or RBF nodes with a range of activation functions can universally approximate any continuous functions in any compact subset of the Euclidean space $\mathbf{R}^n$.

In this study, the activation function used in ELM is the sigmoid function $g(x) = \frac{1}{1+e^{-\lambda x}}$. Originally ELM has a fixed unit value for the parameter λ . It works well for all the applications of function approximation and classification problems except when the input dimensionality is high and the number of available samples are small. We found that ELM does not work well for this kind of problem and we solved this problem by using flatter sigmoid functions by changing the values of λ . The performance has improved based on this modification.

ⁱThe source codes of ELM can be downloaded from <http://www.ntu.edu.sg/home/egbhuang/>

Dataset	# Gene	# Sample	# Class
GCM	16063	144	14
HBC	3226	22	3
Lung	918	73	5
Lymphoma	4026	62	3
MLL	12582	72	3
NCI60	5244	61	8

Table 7.1: The detailed information of Datasets

7.2 Experiments and Results

In this chapter, in order to evaluate the performance of ELM algorithm for multi-category microarray classification problems, six benchmark microarray datasets on multi-category cancer diagnosis, namely GCM dataset, HBC dataset, Lung dataset, Lymphoma dataset, MLL dataset and NCI60 dataset have been studied . In the experiments, the missing data are imputed by a k -nearest neighbor algorithm, which finds k other genes that are most similar to the genes with a missing value and then the weighted average of the corresponding values are used as the estimates of the missing values [134]. All the input attributes are scaled in the range of $[0, 1]$. The detailed information about each dataset and their characteristics are shown in Table 7.1.

All the studies were carried out in Matlab(Ver 7.1) environment in a Pentium 4, 2.4GHZ PC with 512MB memory.

7.2.1 Experiment 1 - Microarray Benchmark Dataset (GCM)

GCM dataset is a collection of microarray data for snap frozen human tumor and normal tissue specimens, spanning 14 different tumor classes obtained from six institutions and hospitals. This is the first single reference database that covers the cancer diagnosis across all the common malignancies [110]. Ramaswamy *et al* [110] have made the data available on the website: <http://www.broad.mit.edu/cgi-bin/>

cancer/publications/pub_paper.cgi?mode=view\&paper_id=61. The training data is from the file named GCM_Training.res, which can be downloaded from the above mentioned website. This file contains expression profiles comprising of 16,063 genes and 144 primary tumor samples spanning 14 common tumor types.

7.2.1.1 Gene Selection Method

For the purpose of comparison, we use the same gene selection method as used in [82] and [110], namely the recursive feature elimination method. As introduced in [110], for a microarray data with n genes, each SVM-OVA classifier produces a hyperplane w , which is a vector of n elements each corresponding to the expression of a particular gene. The absolute magnitude of each element in w can be considered as measure of importance for each corresponding gene. Each SVM-OVA classifier is first trained with all genes, then genes corresponding to the bottom 10% $|w_i|$ are removed. Each classifier is retrained after the removal of genes. This process is repeated iteratively and a rank of all the genes based on the statistical significance of each class can be obtained. The most significant 14, 28, 42, 56, 70, 84 and 98 genes selected by this method can be found in the file OVA_MARKERS.xls from the same website.

7.2.1.2 ELM Algorithm

In ELM, the number of output neurons is equal to the number of classes of the problem. The number index of the output neuron with the highest output indicates the class number of the corresponding input. The activation function used in ELM is a sigmoid function $f(x) = \frac{1}{1+e^{-x}}$. We made an extension of the ELM algorithm by introducing the gain parameter λ into the activation function $f(x) = \frac{1}{1+e^{-\lambda x}}$ because experimental results show that flatter sigmoid function gives better generalization performance when the ratio between input dimension and the number of

training samples per class is high. Therefore, we need to choose two parameters: gain parameter λ that decides the flatness of the sigmoid function and the number of hidden neurons $\tilde{N}$. The best combination of these two parameters is obtained by a grid search for each number of genes. We study the generalized accuracy using different combinations of the hidden neurons number $\tilde{N}$ and gain parameter λ : $\tilde{N} \in \{10, 15, 20, \dots, 90, 95, 100\}$ and $\lambda \in \{10^{-1}, 10^{-2}, \dots, 10^{-9}, 10^{-10}\}$. The maximum number of neurons for ELM is set to 100, because there are only 111 training samples available. Therefore, for each problem we try $19 \times 10 = 190$ combination of parameters $(\tilde{N}, \lambda)$ for ELM. The best parameters for each number of genes are shown in Table 7.2. The average results of 10 trials for each selected gene number are shown Table 7.3. At each trial the training and testing data are randomly generated, however, the proportion between the training samples and testing samples remains unchanged as shown in Table 6.1.

7.2.1.3 SVM-OVO Algorithm

We carried out the experiments using the SVM One-Versus-One (SVM-OVO) classifier from the toolbox http://www.ece.osu.edu/~maj/osu_svm/. This toolbox implements SVM classifiers in C++ based on LIBSVM algorithm <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>. The cost parameter C and kernel parameter γ of SVM-OVO are obtained by grid searching for each number of genes used in this experiment. We estimate the classification accuracy using different combinations of cost parameter C and kernel parameter γ : $C \in \{2^{12}, 2^{11}, \dots, 2^{-1}, 2^{-2}\}$ and $\gamma \in \{2^4, 2^3, \dots, 2^{-9}, 2^{-10}\}$. Therefore, for each gene selection we try $15 \times 15 = 225$ combination of parameters (C, γ) for SVM. The best parameters for each number of genes are shown in Table 7.2.

#Gene	ELM ($\tilde{N}, \lambda$)	SVM-OVO	
		#SVs	(C, γ)
14	$(15, 10^{-2})$	93.3	$(2^2, 2^1)$
28	$(35, 10^{-2})$	91.6	$(2^1, 2^1)$
42	$(30, 10^{-3})$	92.5	$(2^2, 2^{-2})$
56	$(30, 10^{-6})$	94.6	$(2^2, 2^{-2})$
70	$(35, 10^{-4})$	98.7	$(2^1, 2^{-2})$
84	$(40, 10^{-8})$	100.3	$(2^2, 2^{-2})$
98	$(50, 10^{-5})$	102.5	$(2^2, 2^{-2})$

Table 7.2: Optimal parameters for ELM and SVM-OVO algorithms for selected genes

#Genes	ELM	SD	SVM-OVO	SD	SANN	ANN
14	78.79	4.7	74.24	8.2	68.75	50
28	82.73	4.8	75.15	4.9	71.53	59.72
42	81.82	7.3	79.39	5.4	72.92	64.58
56	83.33	6.4	80	6.4	79.17	70.14
70	84.55	5.1	81.82	5.6	76.4	59.72
84	84.85	3.3	82.73	5.3	80.56	70.83
98	86.36	4.6	83.94	4.1	77.08	72.22

Table 7.3: Classification accuracy (%) for different algorithms

7.2.2 Results and Discussion

7.2.2.1 Classification Accuracy

The classification performance of different algorithms for the benchmark dataset GCM is presented in Table 7.3 and Figure 7.1. These algorithms include ELM, SVM-OVO, SANN and ANN. All the results of SANN and ANN algorithms are quoted from Linder *et al* [82].

As observed from Table 7.3 and Figure 7.1, ELM, SVM-OVO and SANN algorithm achieve a much higher classification accuracy than ANN. For ELM, SVM-OVO and SANN algorithms, classification accuracy tends to grow with the number of genes selected. It can be noted that for all these gene selections ELM achieves the highest classification accuracy. SVM-OVO achieves the second highest accuracy amongst

Figure 7.1: Comparison of the classification accuracies of ELM, SVM-OVO, SANN and ANN.

the algorithms compared.

7.2.2.2 Robustness Comparison of ELM and SVM-OVO

Since ELM and SVM-OVO produce the best classification accuracy, robustness of these two algorithms with respect to the number of genes selected are further evaluated in this subsection. Here, we first choose the number of genes and then for this gene number the best parameters of (C, γ) for SVM-OVO and $(\tilde{N}, \lambda)$ for ELM are selected. These parameters are then frozen and used for all the other gene selections and the resulting classification accuracy is evaluated. If the performance for all numbers of genes does not vary drastically, it then indicates that the algorithm is robust in the aspect that the parameters are not sensitive to the number of genes selected. For example, when the number of genes selected is 14, we tune the parameters to get the best results. Then we apply the same parameter values for all other gene numbers, namely, 28, 42, 56, 70, 84, 98 and compare the performance of the two algorithms. We repeat this process for each of the selected number of the genes.

Figure 7.2: Comparison of the classification accuracies of ELM and SVM-OVO using the best parameters for 14 genes.

Using the above method for robustness evaluation the classification performance of the two algorithms are presented in Figure 7.2 to Figure 7.8. As observed from these figures, ELM produces a flatter curve than SVM-OVO indicating that ELM's performance is less sensitive to the number of genes selected, i.e, it is more robust. It also can be seen that accuracy produced by ELM are generally higher than that of SVM-OVO.

7.2.2.3 Network Complexity

Table 7.2 also presents the number of hidden neurons for ELM and the number of support vectors of SVM-OVO corresponding to the best classification performance for each gene number. It can be seen that the number of hidden neurons for ELM is always smaller than the number of support vectors for SVM-OVO. If one compares

Figure 7.3: Comparison of the classification accuracies of ELM and SVM-OVO using the best parameters for 28 genes.

Figure 7.4: Comparison of the classification accuracies of ELM and SVM-OVO using the best parameters for 42 genes.

Figure 7.5: Comparison of the classification accuracies of ELM and SVM-OVO using the best parameters for 56 genes.

Figure 7.6: Comparison of the classification accuracies of ELM and SVM-OVO using the best parameters for 70 genes.

Figure 7.7: Comparison of the classification accuracies of ELM and SVM-OVO using the best parameters for 84 genes.

Figure 7.8: Comparison of the classification accuracies of ELM and SVM-OVO using the best parameters for 98 genes.

#Genes	Training time (s)		Parameter selection time (s)	
	ELM	SVM-OVO	ELM	SVM-OVO
14	0.00156	0.0062	21.25	28.5
28	0.00469	0.00781	18.86	31.91
42	0.00625	0.00625	19.42	36.06
56	0.00625	0.00938	22	41.69
70	0.00469	0.00782	25.10	48.72
84	0.00313	0.01406	25.38	52.25
98	0.0125	0.0125	25.75	54.64

Table 7.4: Training time(s) for ELM and SVM-OVO algorithms

ELM with SANN [82], it can be seen that in SANN there is 1 ANN and up to 91 SANNs to be trained for each experiment. For each network, there are 5 modules each consisting of 10 hidden neurons. This means for each experiment up to 4600 hidden neurons are needed for the training process, while for ELM, the network is much more compact with less than 50 hidden neurons.

7.2.2.4 Training Time

The training time for ELM and SVM-OVO along with the time taken for finding the best parameters for these algorithms are shown in Table 7.4. The time given for ELM is based on Matlab while that for SVM is based on a C++ implementation. In spite of this, ELM takes much smaller training time and parameter selection time than SVM-OVO especially when the number of genes selected is high. Compared with the training time for SANN given in [82] ELM takes significantly lower training time.

7.2.2.5 Classification Performance on Individual Category

For a multi-category classifier, although the overall classification performance is important one may have to look at the classification performance on individual classes as well. A good classifier is one that produces a good overall classification as well

Figure 7.9: Comparison of individual category classification: Breast, Prostate, Lung, Colorectal, Lymphoma, Bladder and Melanoma.

as equally good performance for individual classes. To assess this, the classification accuracy of all the studied algorithms on each tumor class has been investigated.

Figures 7.9 and 7.10 show the classification results for each category for the different algorithms. The figures show the number of samples for each category and the number of successful classifications, called hits, for each of the algorithms. The hits shown in the figures are based on the average of 10 trials.

As observed from Figures 7.9 and 7.10 ELM and SVM-OVO outperform other algorithms for every tumor category. For some classes, such as Ovary and Pancreas, SANN fails to classify correctly more than half of the samples. This indicates that for this multi-category classification with 14 classes, SANN has shown some preference to patterns in other classes over Ovary and Pancreas whereas ELM and SVM-OVO possess a better balance among the classes.

Figure 7.10: Comparison of individual category classification: Uterus_Adeno, Leukemia, Renal, Pancreas, Ovary, Mesothelioma and CNS.

The confusion matrices for ELM and SVM-OVO algorithms are shown in Table 7.5 and Table 7.6, respectively. This is based on the average of 10 trials on different shuffles of the training and testing data with 98 selected genes. The diagonal elements of the confusion matrix indicate the correct classification percentages and the off diagonal elements give the mis-classification percentages. It can be seen that the classification accuracies for both ELM and SVM-OVO are higher when the tumor category has a large number of patterns such as Lymphoma, Leukemia and CNS. This indicates that with more tumor samples the classification accuracy of ELM and SVM-OVO can be improved.

Regarding misclassification, after a careful comparison of the two tables, it can be seen that these are 56 cases of misclassifications out of 182 possibilities. Among these 56 cases, there are 16 cases where both algorithms happen to misclassify the same cases with a similar probability. For example, the probability of Breast tumor being categorized as Pancreas tumor and Prostate tumor being categorized as Breast tumor is quite high for both the algorithms. This indicates that these cases are difficult cases for classification for all the algorithms.

For the rest of the datasets, we use three gene selection methods in this study, namely BSS/WSS method, Entropy based method and the Wilcoxon rank-based statistics. BSS/WSS method sorts the genes in a descending order by the ratio of between group to within group sum of squares as in [27]. Entropy based method orders genes descendingly based on the entropy of each gene. Wilcoxon rank-based statistics uses a one versus all (OVA) approach and compares each class separately against other groups and measures the wilcoxon statistics between one and other groups of each gene [72]. Wilcoxon statistics measures the similarities between two vectors. The similar two vectors, the higher Wilcoxon statistics value. Genes are sorted ascendingly based on the average OVA Wilcoxon statistics values for each individual gene.

	BR(%)	PR(%)	LU(%)	CO(%)	LY(%)	BL(%)	ML(%)	UT(%)	LE(%)	RE(%)	PA(%)	OV(%)	ME(%)	CNS(%)
BR	75			5						5	10		5	
PR	10	70			5		5			5			5	
LU	5		90									5		
CO				95									5	
LY					100									
BL	5		5			75				5		5		
ML				5		5	80					5		5
UT							5	85		5		5		
LE									100					
RE			5					10		80		5		
PA		5				5					85	5		
OV	5				5	10		15		10		55		
ME						5				10			85	
CNS														100

Table 7.5: Confusion matrix obtained by ELM for the testing data.

	BR(%)	PR(%)	LU(%)	CO(%)	LY(%)	BL(%)	ML(%)	UT(%)	LE(%)	RE(%)	PA(%)	OV(%)	ME(%)	CNS(%)
BR	80						10				10			
PR	10	75	5											10
LU	5		80									10	5	
CO				90			5						5	
LY					96.7				3.3					
BL				5		75	10					5	5	
ML	5			10			80						5	
UT				5		5	15	50		5		15	5	
LE									100					
RE							15			70		15		
PA	5	5				5					85			
OV	5					5	10	5				75		
ME				15			5						80	
CNS														100

Table 7.6: Confusion matrix obtained by SVM-OVO for the testing

data.

For all the datasets, 10, 20, 50, 70, 100, 150, 200, 400, 800 genes are selected using each of 3 gene selection methods and used in the simulation. Finally, the total number of genes of each dataset without selection but just a ordering based on the three gene selection criteria is used for simulation of both ELM and SVM-OVO algorithms.

In ELM, 1-of- c coding is used for outputs, where c is number of classes. We study the generalized accuracy using different combinations of the hidden neurons number $\tilde{N}$ and gain parameter λ : $\tilde{N} \in \{10, 20, 30, \dots, 480, 490, 500\}$ and $\lambda \in \{10^{-1}, 10^{-2}, \dots, 10^{-14}, 10^{-15}\}$. The maximum number of neurons for ELM is set to 500, because experiments show that increasing the number of neurons does not gain any classification accuracy for all the datasets. Therefore, for each problem we try $50 \times 15 = 750$ combination of parameters $(\tilde{N}, \lambda)$ for ELM.

The cost parameter C and kernel parameter γ of SVM-OVO are obtained by grid searching using different combinations of cost parameter C and kernel parameter γ : $C \in \{2^{12}, 2^{11}, \dots, 2^{-1}, 2^{-2}\}$ and $\gamma \in \{2^4, 2^3, \dots, 2^{-9}, 2^{-10}\}$. Therefore, for each gene selection we try $15 \times 15 = 225$ combination of parameters (C, γ) for SVM-OVO.

The best parameters for each number of genes are shown together with the results of each dataset. The average results of 10 trials for each selected gene number of all the 5 datasets are reported. The partition of the training set and testing set is kept at a fixed proportion of 4:1 and the selection of the training and testing data is done on a random basis. At each trial the training and testing data are randomly generated, however, the proportion between the training samples and testing samples remains unchanged.

7.2.3 Experiment 2 - HBC Dataset

The HBC dataset contains the original data with 22 samples, consisting of 3226 genes. The data set was gathered to understand how tumors resulting from two different mutations differ in their genetic basis. Among these 22 samples, 7 of them are known to have the BRCA1 mutation, 8 are known to have the BRCA2 mutation and 7 have no cancer, being labeled “Sporadic”. The dataset can be downloaded from <http://www.columbia.edu/~xy56/project.htm>.

The classification performance for the benchmark dataset HBC is presented in Table 7.7. The selected parameters for each gene number and gene selection method, that produce the best testing accuracy for ELM and SVM algorithm are shown in Table 7.8.

The training time for ELM and SVM along with the time taken for finding the best parameters for these algorithms are shown in Table 7.9. In the table, training refers to the training time, while total refers to the time taken for searching the best parameters for these algorithms.

7.2.4 Experiment 3 - Lung Dataset

The Lung dataset consists of 73 samples spanning 5 different classes, each with 918 genes. These samples include 41 adeno carcinomas (AC) samples, 17 squamous cell carcinomas (SCC) samples, 4 large cell lung cancers (LCLC) samples, 5 small cell lung cancers (SCLC) samples and 6 normal lung cells samples. This dataset is available at http://genome-www.stanford.edu/lung_cancer/adeno/.

The classification performance for the Lung dataset is presented in Table 7.10. The selected parameters for each gene number and gene selection method for ELM and SVM-OVO algorithm are shown in Table 7.11. The training time for ELM and SVM-

Algorithm	Gene selection	Number of genes									
		10	20	50	70	100	150	200	400	800	3226
ELM	BSS/WSS	80	77.5	82.5	90	87.5	90	92.5	92.5	92.5	92.5
	SD	12.2	17.5	17.5	19.5	16.0	18.7	12.2	13.5	15	17.5
	Entropy	62.5	77.5	82.5	85	92.5	95	95	92.5	92.5	92.5
	SD	17.5	18.7	15	15	17.5	12.2	11.5	11.5	19.5	16.8
	Wilcoxon	72.5	85	95	95	100	100	92.5	95	97.5	97.5
	SD	18.7	18.7	12.2	12.5	0	0	12.2	11.5	16.6	12.2
SVM	BSS/WSS	80	80	85	85	90	87.5	90	90	92.5	87.5
	SD	15.0	18.7	16.6	12.2	12.2	12.2	12.5	12.2	11.5	12.5
	Entropy	57.5	77.5	80	80	85	90	92.5	92.5	92.5	87.5
	SD	19.5	17.5	10	15	12.2	12.2	11.5	11.5	11.5	12.5
	Wilcoxon	72.5	92.5	95	95	95	90	85	90	92.5	95
	SD	13.5	11.5	10	10	10	12.2	16.6	12.2	11.5	10

Table 7.7: Testing Accuracy and standard deviation (%) on different number of genes using different gene selection methods for ELM and SVM for HBC dataset

Algorithm	Gene selection	Parameter	Number of genes									
			10	20	50	70	100	150	200	400	800	3226
ELM	BSS/WSS	Neuron #	340	50	350	330	180	350	320	250	240	170
		lamda	1e-012	1e-013	0.1	0.1	0.1	0.001	0.0001	0.01	1e-005	0.001
	Entropy	Neuron #	350	290	80	160	200	470	500	380	370	410
		lamda	0.1	1e-012	1e-005	0.1	0.1	1e-009	0.1	0.01	0.1	0.01
Wilcoxon	Neuron #	Neuron #	440	340	80	350	200	120	470	220	330	410
		lamda	1e-012	1e-012	1e-009	0.1	0.0001	0.1	0.1	0.001	0.01	0.001
SVM	BSS/WSS	C	0.5	4096	4096	4096	0.5	4096	4096	4096	4096	4096
		Gamma	0.5	0.5	0.125	0.25	0.125	0.0625	0.03125	0.01562	0.00195	0.00781
	Entropy	C	4	4	4096	4096	4096	4	4096	4096	4096	4096
		Gamma	0.125	0.25	0.5	0.25	0.125	0.01562	0.03125	0.00391	0.00391	0.00781
	Wilcoxon	C	0.5	1	4096	4096	4096	4096	1	1	2	4096
		Gamma	2	0.5	0.25	0.125	0.25	0.125	0.01562	0.00781	0.00781	0.00391

Table 7.8: Best parameter selected for different number of genes using different gene selection methods for ELM and SVM for HBC dataset

Algorithm	Gene selection	Time	Number of genes									
			10	20	50	70	100	150	200	400	800	3226
ELM	BSS/WSS	Training	0	0	0.00313	0.00625	0.00625	0.00625	0.00625	0.01563	0.02656	0.0875
		Total	2.6219	3.1344	3.6547	3.9031	4.4922	6.5187	7.8437	14.192	28.392	129.45
	Entropy	Training	0.0015	0.0047	0	0.0047	0.0078	0.0171	0.0154	0.0294	0.0517	0.2623
		Total	2.922	3.3233	3.5386	4.0945	4.9336	6.8042	8.3744	14.67	28.478	121.44
	Wilcoxon	Training	0.00625	0.00156	0.00156	0.00781	0.00625	0.00469	0.01406	0.0125	0.03281	0.17344
		Total	2.2125	2.4156	2.8359	3.4	3.8063	4.9734	5.8359	9.9234	18.786	80.994
SVM	BSS/WSS	Training	0.00156	0	0.00156	0	0.00156	0	0	0.00156	0.00469	0.01875
		Total	0.13594	0.16406	0.18125	0.20625	0.22187	0.3	0.26875	0.53906	0.89375	5.1109
	Entropy	Training	0	0	0.0031	0.0015	0	0.0031	0.0016	0.0032	0.0031	0.0201
		Total	0.2186	0.1588	0.2423	0.2132	0.2106	0.2727	0.3011	0.5293	1.0004	5.1614
	Wilcoxon	Training	0	0	0	0	0.00313	0.00313	0	0.00313	0.00625	0.01719
		Total	0.18594	0.18594	0.22812	0.22187	0.26094	0.2625	0.26719	0.45781	0.81875	4.0641

Table 7.9: Training time(s) for ELM and SVM algorithms for HBC dataset

OVO along with the time taken for finding the best parameters for these algorithms are shown in Table 7.12.

7.2.5 Experiment 4 - Lymphoma Dataset

The Lymphoma dataset is concerned with the three most prevalent adult lymphoid malignancies. It contains 62 samples consisting of 4026 genes spanning 3 classes, which include 42 Diffuse Large B-Cell Lymphoma (DLBCL) samples, 9 Follicular Lymphoma (FL) samples and 11 B-cell Chronic Lymphocytic Leukemia (B-CLL) samples. The dataset can be found at <http://genome-www.stanford.edu/lymphoma/>.

The classification performance, selected parameters and training time for the Lymphoma dataset is presented in Table 7.13, 7.14 and Table 7.15 respectively.

7.2.6 Experiment 5 - MLL Dataset

The MLL dataset contains 72 samples consisting of 12582 genes spanning 3 different leukemia classes. The cell lines include 24 Acute Lymphoblastic Leukemia (ALL), 20 Mix-Lineage Leukemia (MLL) and 28 Acute Myelogenous Leukemia (AML). The dataset can be downloaded at http://research.dfci.harvard.edu/korsmeyer/Supp_pubs/Supp_Armstrong_Main.html.

The classification performance, selected parameters and training time for the MLL dataset is presented in Table 7.16, Table 7.17 and Table 7.18 respectively.

Algorithm	Gene selection	Number of genes									
		10	20	50	70	100	150	200	400	800	918
ELM	BSS/WSS	89.286	91.429	93.571	93.571	92.857	93.571	94.286	93.571	92.143	92.143
	SD	6.7	2.9	5.9	8.3	8.1	7.9	8.1	7.5	7.3	7.3
	Entropy	77.857	83.571	90	91.429	92.857	92.143	93.571	94.286	95	94.286
	SD	10.8	7.7	5.8	7.2	8.3	9.1	5.9	8.6	6.2	5.0
Wilcoxon		75.714	82.143	90	92.857	92.143	92.143	92.857	93.571	93.571	93.571
	SD	11.1	10	7.2	6.6	6.5	4.6	4.8	7.2	6.4	5.8
SVM	BSS/WSS	88.571	91.429	90.714	92.143	91.429	91.429	91.429	92.143	91.429	91.429
	SD	3.5	2.9	7.2	6.7	6.2	7.0	6.2	6.7	6.2	6.2
	Entropy	85	88.571	88.571	88.571	89.286	92.143	91.429	92.857	92.857	92.857
	SD	6.7	4.7	6.5	6.5	8.6	8.7	8.3	6.4	7.1	7.1
Wilcoxon		82.143	84.286	89.286	88.571	89.286	91.429	91.429	92.143	91.429	90.714
	SD	6.6	6.2	6.6	3.5	5.8	5.3	4.3	5.9	5.3	4.6

Table 7.10: Testing Accuracy and standard deviation (%) on different number of genes using different gene selection methods for ELM and SVM for Lung dataset

Algorithm	Gene selection	Parameter	Number of genes										
			10	20	50	70	100	150	200	400	800	918	
ELM	BSS/WSS	Neuron #	300	280	390	390	290	180	500	310	270	350	
		lamda	1e-012	1e-012	1e-012	1e-012	1e-012	1e-011	1e-007	1e-009	1e-009	0.001	
	Entropy	Neuron #	20	90	350	350	460	370	490	300	480	190	
		lamda	0.001	0.0001	1e-012	1e-012	1e-012	1e-012	1e-012	1e-005	0.01	1e-013	
	Wilcoxon	Neuron #	60	100	270	290	340	420	190	440	450	410	
		lamda	0.0001	0.0001	1e-012	1e-012	1e-012	1e-012	0.001	0.1	1e-012	1e-008	
SVM	BSS/WSS	C	1	1	16	4096	4096	4096	4096	16	4096	4096	
		Gamma	2	0.25	0.03125	0.25	0.125	0.0625	0.03125	0.00391	0.00781	0.00781	
	Entropy	C	8	4	4096	4096	4096	4096	4096	4	8	4096	
		Gamma	0.5	0.25	0.5	0.25	0.125	0.0625	0.0625	0.01563	0.00391	0.00781	
	Wilcoxon	C	32	2	4096	4096	4096	4096	4096	8	4096	4096	
		Gamma	0.125	1	0.25	0.5	0.03125	0.03125	0.125	0.00781	0.00195	0.00391	

Table 7.11: Best parameter selected for different number of genes using different gene selection methods for ELM and SVM for Lung dataset

Algorithm	Gene selection	Time	Number of genes									
			10	20	50	70	100	150	200	400	800	918
ELM	BSS/WSS	Training	0.01563	0.01563	0.025	0.01875	0.02188	0.01719	0.05	0.034375	0.05625	0.076563
		Total	16.233	16.502	17.798	19.305	20.794	22.995	25.417	33.973	54.316	60.498
	Entropy	Training	0.00156	0.00625	0.01875	0.01563	0.02813	0.025	0.039063	0.042188	0.092188	0.04375
		Total	11.42	11.191	12.605	13.969	15.088	16.82	18.517	24.692	37.456	41.987
	Wilcoxon	Training	0.00469	0.01094	0.01719	0.01563	0.023438	0.03125	0.025	0.04375	0.082812	0.089063
		Total	11.486	11.02	12.375	13.472	14.666	16.322	17.898	24.675	37.745	41.841
SVM	BSS/WSS	Training	0.00156	0	0.00156	0.00469	0	0.00781	0.00781	0.0125	0.03281	0.042188
		Total	0.40938	0.50313	0.64688	0.81094	1.0063	1.3781	1.7437	3.3547	10.261	13.07
	Entropy	Training	0.00156	0.00313	0.00469	0.00625	0.00313	0.00313	0.00625	0.0125	0.02813	0.035937
		Total	0.4625	0.52656	0.77031	0.93125	1.0531	1.4797	1.7859	3.3187	8.9391	11.198
	Wilcoxon	Training	0	0.00156	0.00313	0.00469	0.00156	0.00156	0.00781	0.01563	0.02813	0.03281
		Total	0.39375	0.44063	0.70156	0.83281	1.0266	1.3859	1.7078	3.1937	8.9828	11.183

Table 7.12: Training time(s) for ELM and SVM algorithms for Lung dataset

Algorithm	Gene selection	Number of genes									
		10	20	50	70	100	150	200	400	800	4026
ELM	BSS/WSS	98.333	99.167	100	100	100	100	100	100	100	100
	SD	5.0	2.5	0	0	0	0	0	0	0	0
	Entropy	92.5	97.5	99.167	100	100	100	100	100	100	100
	SD	5.8	3.8	.5	0	0	0	0	0	0	0
	Wilcoxon	85	87.5	93.333	97.5	94.167	99.167	100	100	100	100
SVM	SD	8.7	8.5	6.9	3.8	6.9	2.5	0	0	0	0
	BSS/WSS	100	99.167	99.167	100	100	100	100	100	100	99.167
	SD	0	2.5	2.5	0	0	0	0	0	0	0
	Entropy	94.167	97.5	100	100	100	100	100	100	100	100
	SD	7.5	3.8	0	0	0	0	0	0	0	0
Wilcoxon		85	89.167	94.167	95	96.667	99.167	100	100	100	100
	SD	7.3	7.5	7.5	4.1	5.5	2.5	0	0	0	0

Table 7.13: Testing Accuracy and standard deviation (%) on different number of genes using different gene selection methods for ELM and SVM for Lymphoma dataset

Algorithm	Gene selection	Parameter	Number of genes									
			10	20	50	70	100	150	200	400	800	4026
ELM	BSS/WSS	Neuron #	90	180	180	10	20	170	360	200	110	140
		lamda	1e-012	1e-012	1e-012	1e-006	0.01	0.1	0.1	0.1	0.1	0.1
	Entropy	Neuron #	10	260	320	480	170	180	160	110	230	180
		lamda	1e-005	1e-012	1e-012	1e-012	1e-012	0.1	0.1	0.1	0.1	0.1
	Wilcoxon	Neuron #	20	40	450	110	80	170	330	430	160	140
		lamda	1e-012	1e-012	1e-011	1e-012	1e-012	1e-012	1e-012	0.01	0.1	0.1
SVM	BSS/WSS	C	4096	4096	4096	4096	4096	4096	1	2	4096	4096
		Gamma	2	2	0.5	0.125	0.25	0.0625	0.0625	0.03125	0.00781	0.00195
	Entropy	C	8	4096	4096	4096	4096	4096	4096	4096	4096	4096
		Gamma	2	1	0.25	0.25	0.125	0.125	0.0625	0.0625	0.03125	0.00391
	Wilcoxon	C	1	4096	4096	4096	4096	4096	4096	4096	4096	4096
		Gamma	4	1	0.5	0.5	0.125	0.25	0.0625	0.0625	0.03125	0.00391

Table 7.14: Best parameter selected for different number of genes using different gene selection methods for ELM and SVM for Lymphoma dataset

Algorithm	Gene selection	Time	Number of genes									
			10	20	50	70	100	150	200	400	800	4026
ELM	BSS/WSS	Training	0.00938	0.00938	0.00938	0.00156	0.00156	0.01563	0.02656	0.025	0.026563	0.11094
		Total	12.956	12.664	15.034	15.478	16.848	18.677	20.747	28.808	45.003	192.23
	Entropy	Training	0.00156	0.00938	0.01875	0.01875	0.0125	0.01094	0.01094	0.01406	0.04375	0.13594
		Total	7.9828	7.7594	9.4281	10.028	10.881	12.281	13.978	19.814	32.333	142.29
	Wilcoxon	Training	0.0016	0.0048	0.025	0.0032	0.0142	0.0157	0.0298	0.0545	0.0391	0.1469
		Total	10.552	10.541	12.302	13.43	14.671	16.64	18.583	26.514	43.32	192.48
SVM	BSS/WSS	Training	0	0	0	0	0.00156	0.00156	0.00313	0.00781	0.01094	0.11406
		Total	0.175	0.25937	0.37031	0.4125	0.53594	0.74219	0.94063	1.7687	5.0109	35.555
	Entropy	Training	0.00156	0.00156	0.00156	0.00156	0.00313	0.00313	0.00313	0.00781	0.026563	0.12812
		Total	0.28125	0.32813	0.4625	0.56406	0.67344	0.84688	1.0203	1.8578	5.2438	35.266
	Wilcoxon	Training	0.0016	0.0015	0.003	0.0032	0.003	0.0031	0.0015	0.0126	0.0234	0.1563
		Total	0.2731	0.2419	0.4164	0.48	0.6144	0.8258	1.0493	2.0951	6.0494	42.789

Table 7.15: Training time(s) for ELM and SVM algorithms for Lymphoma dataset

Algorithm	Gene selection	Number of genes									
		10	20	50	70	100	150	200	400	800	12582
ELM	BSS/WSS	94	94	95.333	94.667	95.333	96.667	97.333	98.667	97.333	97.333
	SD	6.7	6.7	7.3	5.3	7.3	5.2	6.3	4.3	3.3	3.3
	Entropy	73.333	80.667	84.667	84	88	88	89.333	94	95.333	98
	SD	11.2	7.4	8.0	6.1	6.8	7.3	6.0	7.5	5.0	2.3
	Wilcoxon	76	74	79.333	82.667	86.667	92.667	94	97.333	97.333	97.333
SVM	BSS/WSS	94.667	94.667	95.333	94.667	95.333	94	94.667	95.333	96.667	92
	SD	7.2	5.0	4.3	4.0	5.2	4.0	4.3	4.3	3.3	5.0
	Entropy	74	82.667	84.667	86	88	88.667	88	91.333	90.667	92.667
	SD	7.6	8.5	7.3	7.0	5.0	8.5	4.0	6.0	6.1	6.3
	Wilcoxon	72.667	74.667	77.333	81.333	87.333	90	94.667	94.667	95.333	94
	SD	10.1	7.8	15.3	12.2	7.6	10	5.8	5.8	6.7	5.5

Table 7.16: Testing Accuracy and standard deviation (%) on different number of genes using different gene selection methods for ELM and SVM for MLL dataset

Algorithm	Gene selection	Parameter	Number of genes									
			10	20	50	70	100	150	200	400	800	12582
ELM	BSS/WSS	Neuron #	250	290	330	20	260	260	160	220	270	490
		lamda	1e-012	1e-012	1e-012	0.01	1e-012	1e-010	1e-012	0.01	0.1	1e-007
	Entropy	Neuron #	260	160	200	250	240	430	430	240	310	390
		lamda	1e-012	1e-012	1e-012	1e-012	1e-012	1e-008	1e-009	0.01	1e-009	1e-012
	Wilcoxon	Neuron #	40	80	220	220	200	410	380	410	490	460
		lamda	0.0001	1e-012	1e-012	1e-012	1e-012	1e-006	1e-012	1e-009	1e-006	1e-006
SVM	BSS/WSS	C	0.5	2	4096	0.5	4096	4096	4096	4096	4096	4096
		Gamma	2	0.25	1	0.5	0.01563	0.25	0.125	0.01563	0.03125	0.00098
	Entropy	C	8	8	256	4096	16	8	4096	4096	4096	4096
		Gamma	0.0625	0.125	0.00195	0.0625	0.01563	0.01563	0.01563	0.00195	0.00098	0.00098
	Wilcoxon	C	32	16	4096	4096	4096	32	4096	32	4096	4096
		Gamma	0.25	0.125	0.03125	0.03125	0.03125	0.00781	0.00195	0.00195	0.00391	0.00098

Table 7.17: Best parameter selected for different number of genes using different gene selection methods for ELM and SVM for MLL dataset

Algorithm	Gene selection	Time	Number of genes									
			10	20	50	70	100	150	200	400	800	12582
ELM	BSS/WSS	Training	0.01563	0.01563	0.01875	0.00469	0.01875	0.023438	0.01875	0.03281	0.0625	1.2953
		Total	15.931	15.331	17.314	18.817	20.17	22.777	25.073	34.128	53.661	667.79
	Entropy	Training	0.0174	0.0127	0.0157	0.0202	0.0233	0.0389	0.0517	0.045	0.0782	1.2935
		Total	13.428	12.954	14.848	16.212	17.675	19.702	21.901	31.656	50.649	619.87
	Wilcoxon	Training	0.00469	0.00625	0.01406	0.01563	0.01563	0.03125	0.03281	0.05	0.095313	1.1922
		Total	10.467	10.15	12.077	12.889	13.764	15.692	17.527	24.297	39.228	499.05
SVM	BSS/WSS	Training	0.00156	0.00313	0	0	0.00156	0	0.00469	0.00938	0.03281	0.67031
		Total	0.27969	0.29375	0.53906	0.57969	0.74063	1.0063	1.2078	2.3078	6.9563	168.33
	Entropy	Training	0.003	0.0016	0.0031	0.0031	0.003	0.0032	0.0064	0.0111	0.0314	0.6765
		Total	0.418	0.4479	0.6244	0.7273	0.8797	1.2132	1.6178	2.992	8.0732	171.21
	Wilcoxon	Training	0	0	0.00313	0.00156	0.00156	0.00313	0.00313	0.00313	0.01875	0.67188
		Total	0.43438	0.41406	0.56875	0.67188	0.83125	1.0328	1.3234	2.525	7.4203	168.73

Table 7.18: Training time(s) for ELM and SVM algorithms for MLL dataset

7.2.7 Experiment 6 - NCI60 Dataset

The NCI60 data contains 61 samples, consisting of 5244 genes spanning 8 different cancer cell lines. The cell lines include 9 breast (including one cell line analyzed by 3 independent micro array experiments), 6 central nervous system (CNS), 7 colon, 8 leukemia (including one cell line analyzed by 3 independent micro array experiments), 8 melanoma, 9 non-small cell lung carcinoma, 6 ovarian, 8 renal. The dataset is available at <http://genome-www.stanford.edu/nci60>.

The classification performance, selected parameters and training time of the NCI60 dataset are presented in Table 7.19, 7.20 and 7.21.

7.3 Summary

In this chapter, a fast and efficient classification method called ELM algorithm for a multi-category cancer diagnosis problem based on microarray data is presented. Its performance has been compared with other methods such as ANN, SANN and SVM on GCM dataset. SVM for multi-category classifications is done by modifying the binary classification method of SVM on a one-versus-all or one-versus-one comparison basis. This inevitably involves more classifiers, bigger system complexities, greater computational burden and longer training time. ELM can perform the multi-category classification directly without any modification. Study results show that when the number of categories for the classification task are big, ELM algorithm achieves a higher classification accuracy than other algorithms with a lesser training time and a smaller network structure. It can also be seen that ELM achieves better and more balanced classification for individual categories as well.

Studies on other data sets indicate that the total training time for ELM is always lower than that of SVM-OVO. For the classification accuracy when the numbers of

Algorithm	Gene selection	Number of genes									
		10	20	50	70	100	150	200	400	800	5244
ELM	BSS/WSS	49.231	56.923	68.462	66.154	68.462	71.538	72.308	72.308	75.385	75.385
	SD	9.3	13.1	8.5	11.6	7.8	13.1	9.9	13.8	13.1	11.3
	Entropy	44.615	52.308	62.308	65.385	64.615	66.154	69.231	72.308	72.308	76.154
	SD	11.9	13.3	9.8	13.0	12.1	9.9	12.0	9.2	10.2	10.8
	Wilcoxon	43.077	46.923	56.923	64.615	66.154	67.692	70	69.231	72.308	74.615
SVM	SD	14.0	12.4	8.4	11.4	11.4	8.3	9.8	9.2	9.8	6.9
	BSS/WSS	56.154	59.231	70	68.462	71.538	70.769	71.538	72.308	73.077	76.154
	SD	10.9	9.8	10.6	10.6	12.9	12.8	13.7	12.5	10.5	11.1
	Entropy	46.154	56.923	62.308	62.308	63.846	64.615	66.923	71.538	71.538	76.154
	SD	11.4	10.4	12.6	11.1	9.8	9.9	10.9	9.8	9.3	11.1
Wilcoxon		43.846	52.308	63.077	65.385	64.615	68.462	66.923	70	72.308	70.769
	SD	8.5	7.5	10.2	9.3	11.0	10.6	10.4	8.4	6.2	6.7

Table 7.19: Testing Accuracy and standard deviation (%) on different number of genes using different gene selection methods for ELM and SVM for NCI60 dataset

Algorithm	Gene selection	Parameter	Number of genes									
			10	20	50	70	100	150	200	400	800	5244
ELM	BSS/WSS	Neuron #	170	200	250	280	330	470	380	420	490	330
		lamda	1e-012	1e-012	1e-012	1e-012	1e-012	1e-012	1e-005	1e-006	1e-010	0.0001
	Entropy	Neuron #	60	20	280	230	380	480	450	470	450	490
		lamda	0.0001	0.01	1e-012	1e-012	1e-012	1e-009	1e-012	1e-007	1e-012	0.1
	Wilcoxon	Neuron #	80	330	300	480	500	210	470	330	320	350
		lamda	0.0001	0.1	1e-12	1e-12	1e-12	1e-6	1e-12	0.01	0.01	1e-10
SVM	BSS/WSS	C	4096	4	16	4096	16	8	16	4096	4096	4096
		Gamma	2	1	0.03125	0.0625	0.01563	0.01563	0.00391	0.03125	0.00781	0.00195
	Entropy	C	4	4096	4096	2	4096	4096	4096	4096	2	4096
		Gamma	0.25	0.5	0.25	0.125	0.01562	0.0623	0.0623	0.01563	0.01563	0.00195
	Wilcoxon	C	8	1	4	4096	4096	2	2	4096	4096	4096
		Gamma	0.125	1	0.25	0.25	0.25	0.0625	0.0625	0.01562	0.01563	0.00391

Table 7.20: Best parameter selected for different number of genes using different gene selection methods for ELM and SVM for NCI60 dataset

Algorithm	Gene selection	Time	Number of genes									
			10	20	50	70	100	150	200	400	800	5244
ELM	BSS/WSS	Training	0.00781	0.00938	0.01094	0.01406	0.01875	0.02656	0.025	0.0375	0.07031	0.25938
		Total	38.047	38.048	39.523	40.484	41.936	42.844	43.244	46.02	59.444	208.14
	Entropy	Training	0.00781	0.00313	0.0125	0.01094	0.01719	0.02813	0.025	0.04688	0.07031	0.45469
		Total	7.3453	7.4687	9.0938	9.7922	10.553	12.23	13.956	19.861	32.188	181.28
	Wilcoxon	Training	0.00938	0.01406	0.01406	0.01719	0.025	0.01406	0.03125	0.03281	0.05156	0.32969
		Total	8.2266	8.1922	9.3969	10.073	10.817	12.417	14.106	19.783	31.809	182.12
SVM	BSS/WSS	Training	0	0.00469	0.00469	0.00313	0	0.00313	0.00625	0.01094	0.02188	0.27187
		Total	0.49063	0.5125	0.74219	0.83125	0.93125	1.2172	1.4984	2.6844	5.9781	62.003
	Entropy	Training	0.00313	0.00625	0.00469	0.00469	0.00312	0.00625	0.00625	0.01563	0.02656	0.23438
		Total	0.61719	0.63125	0.7875	0.90781	1.0812	1.4391	1.6891	2.65	5.6125	52.65
	Wilcoxon	Training	0.00469	0	0.00625	0.007813	0.00469	0.00625	0.007819	0.01406	0.02813	0.2375
		Total	0.60938	0.68594	0.7875	0.97031	1.1078	1.3375	1.6297	2.6641	5.5219	52.545

Table 7.21: Training time(s) for ELM and SVM algorithms for NCI60 dataset

genes used are big, ELM always performs better than SVM-OVO. Even when all the gene are used, ELM still can perform quite well in terms of classification accuracy.

The choice the gene selection methods influence the classification accuracy of ELM and SVM algorithms. Among the three algorithms, BSS/WSS method performs the best among the three gene selection methods for most of the datasets. Therefore, selecting the appropriate method for gene selection is also important for the classification results regardless of the classifier used.

For the sake of comparison, we have also performed the external validation for the GCM dataset, Lung dataset and Lymphoma datasets. The performance results using external validation method are presented in Appendix A. The results also point out the same conclusion about the comparison of ELM and SVM-OVO, as were obtained using internal validation method.

In the next chapter, the conclusions from this thesis are given and future work is indicated .

Chapter 8

Conclusions and Future Work

8.1 Conclusions

In this thesis, we have described an in-depth investigation of the development and application of efficient sequential and batch learning artificial neural network methods for classification problems with large input dimensions and sparse data with an emphasis in bio-informatics applications.

In the first part of the thesis, sequential learning algorithms based on RBF neural networks are investigated.

- The newly developed sequential learning algorithm for RBF networks, viz. the GAP-RBF algorithm was investigated for classification problems and compared with other well-known sequential learning algorithm—Minimal Resource Allocation Network (MRAN) as well as the conventional Multilayer Feed forward Networks (MFNs) on three benchmark classification problems from the PROBEN1 database. The results indicate better performance of the GAP-RBF algorithm in terms of generalization, network size as well as training speed for problems with low input dimensions.

- Two limitations of GAP-RBF algorithm leading to inferior performance for classification applications with non-uniform distributed high dimensional input data were identified.
 - The calculation of the *significance* of a neuron is based on the assumption that the input data is uniformly distributed. If the input distribution is not uniform, especially when the input attributes are discrete, then the performance of GAP-RBF will be degraded. This may be a serious problem for classification applications.
 - Another difficulty arises when the input dimensionality is high. In the GAP-RBF algorithm, the parameters of the network are updated using an Extended Kalman Filter (EKF) method. When the number of inputs is large and as the number of neurons grows, the intensive computation of the covariance matrix of EKF leads to inefficient training and causes a computational overload.
- A new sequential learning algorithm for RBF networks referred to as Fast GAP-RBF (FGAP-RBF) algorithm was developed to overcome these drawbacks of GAP-RBF algorithm.
 - A new scheme of calculating the significance of a neuron based on the most recently received M input data points is presented in this thesis. As the input samples are fed into the network in a random order, these M samples are considered as a representation of the distribution in the whole input space. The new scheme offers a faster and more accurate way of calculating the neuron significance. It also eliminates the requirements of knowing the distribution of the input data *a priori*.
 - The decoupled EKF (DEKF) was introduced into the GAP-RBF algorithm to overcome the problem of computational overload [106] by reducing the size of the covariance matrix.

- The proposed FGAP-RBF algorithm is evaluated together with other sequential algorithms, such as GAP-RBF and MRAN as well as a batch learning algorithm—SVM on four real-world benchmark classification problems. The results show that the FGAP-RBF algorithm achieves a higher classification accuracy than the GAP-RBF and MRAN algorithms for all the four problems and the FGAP-RBF achieves comparable classification accuracies to SVM (batch algorithm) on these problems. The influences of the parameters of the FGAP-RBF algorithm were explored and guidelines on how to select these parameters were given.

In the second part, this thesis makes significant contributions to the multi-category classification problems in the bioinformatics area, especially for the applications of microarray gene expression-based cancer diagnosis. Gene expression-based classification problems are generally considered as difficult tasks because for such problems the input dimension is very high and the number of the samples is very small. Gene expression-based multi-category classification is even more difficult compared with its binary classification counterpart. In the gene expression profiling based classification area, multi-category classifications are mostly done by modifying binary classification methods on a one-versus-all (OVA) or one-versus-one (OVO) comparison basis. This modification inevitably involves many classifiers and increases system complexities. Conventional neural network based classification methods have been attempted for direct multi-category classification. However, it is found that these conventional neural network methods usually produce poor performance in terms of classification accuracy and training time.

- The sequential learning algorithm—FGAP-RBF was applied to a microarray gene expression-based multi-category classification problem. This is the first application of any sequential learning algorithm to such problems. The results show that the FGAP-RBF algorithm can achieve better performance in terms

of accuracy than the best results described in the literature. However, when the number of genes used for classification approaches a thousand, FGAP-RBF algorithm always causes the PC to crash after only a few hidden neurons are added. Therefore, we looked into a fast and efficient batch learning algorithm called Extreme Learning Machine (ELM) that can handle problems of very large input dimensions.

- Extreme Learning Machine (ELM) batch learning algorithm, developed by Huang *et al* [52], was used for six multi-category microarray gene expression cancer diagnosis problems. Its performance has been compared with other methods such as ANN, SANN and SVM algorithms. Study results show that when the number of categories for the classification task are large, ELM algorithm achieves a higher classification accuracy than the other algorithms with a lesser training time and a smaller network structure. It can also be seen that ELM achieves a better and more balanced classification for individual categories as well. For applications with smaller number of categories, ELM achieves a similar accuracy to that of SVM-OVO with a lower training time and a compact structure.

8.2 Recommendations for Future work

Possible areas of future work that emerge from this study are:

- Design of sequential learning algorithms based on combining the RBF networks with logistic and softmax outputs appropriate to classification problems. This combination is beneficial since the network outputs can then be interpreted as class posterior probabilities, which allows for richer analysis. Some recent work [93] shows this can be done without compromising the efficiency of RBF

training for batch learning algorithms. Future work will be carried out to improve sequential methods in order to have the advantages of both interpreting outputs as posterior probabilities and achieving a fast training speed.

- Improve the sequential learning algorithm further for problems of very large input dimensions. Compared with GAP-RBF, the FGAP-RBF algorithm has improved greatly in the capability of handling the problems of high dimensionality. The upper limit of the input dimension has improved from less than 50 to a few hundred. The training speed is also improved by at least two order of magnitudes for problems with dimensions around 50. However, when the classification with more than a thousand inputs is performed using FGAP-RBF algorithm, it always causes a Pentium 4, 2.4GHZ PC with 512MB memory to crash after only a few hidden neurons are added. Therefore, further improvements need to be done to make sequential learning algorithms work for even larger dimensions.
- The significance of the neuron is defined based on the quantitative values of the output of a neuron. Its physical meaning is very clear in terms of function approximation. But in the case of classification, the meaning is not clear. Therefore, to find a suitable definition of the significance of a neuron for the classification application and an improved learning algorithm specially designed for classification is a very important topic for future work.
- The FGAP-RBF algorithm can be used for bio-medical applications where times series are involved. So far the classification problems we solved are all of static data. The sequential learning ability of FGAP-RBF algorithm can catch the dynamic changes of the status of these signals and make online predictions. Future work on these signals have to be undertaken.
- There have been a number of versions of on-line SVM methods proposed recently. Although some of them are not truly sequential algorithm (they require

all the training data to be available before learning starts), we also feel it is worth investigating them in the future.

- In chapter 7, in order to compare the batch learning classifiers ELM and SVM-OVO, we only used the basic gene selection methods that provide a good foundation for the comparison of different classifiers. It is worth studying advanced gene selection methods in the future work.

Author's Publications

Journal Papers

- Runxuan Zhang, Guang-Bin Huang, N. Sundararajan, and P. Saratchandran, “Improved GAP-RBF Network for Classification Problems”, accepted by Neurocomputing, August 2006.
- Runxuan Zhang, Guang-Bin Huang, N. Sundararajan and P. Saratchandran, “Multi-Category Classification Using Extreme Learning Machine for Microarray Gene Expression Cancer Diagnosis”, accepted by IEEE transactions on Computational Biology and Bioinformatics, May 2006.

Conference Papers

- Runxuan Zhang, N. Sundararajan, Guang-Bin Huang and P. Saratchandran, “An Efficient Sequential Learning Neural Network Algorithm for Bio-medical Classification Problems”, pp. 2477-2482, vol.3, Proc of International Joint Conference on Neural Networks (IJCNN04), Budapest, Hungary, July 2004.
- Runxuan Zhang, N. Sundararajan, Guang-Bin Huang and P. Saratchandran, “An Efficient Sequential RBF Network for Gene Expression-Based Multi-category Classification”, IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB05), San Diego, USA, November 2005.

Bibliography

- [1] R. Alexandridis, S. Lin, and M. Irwin. Class discovery and classification of tumor samples using mixture modeling of gene expression data: a unified approach. *Bioinformatics*, 20(16):2545–2552, 2004.
- [2] A. A. Alizadeh, M. B. Eisen, R. E. Davis, C. Ma, I. S. Lossos, A. Rosenwald, J. C. Boldrick, H. Sabet, T. Tran, X. Yu, J. I. Powell, L. Yang, G. E. Marti, T. Moore, J. H. Jr, L. Lu, D. B. Lewis, R. Tibshirani, G. Sherlock, W. C. Chan, T. C. Greiner, D. D. Weisenburger, J. O. Armitage, R. Warnke, R. Levy, W. Wilson, M. R. Grever, J. C. Byrd, D. Botstein, P. O. Brown, and L. M. Staudt. Distinct types of diffuse large b-cell lymphoma identified by gene expression profiling. *Nature*, 403(6769):503–511, 2000.
- [3] U. Alon, N. Barkai, D. A. Notterman, K. Gish, D. M. S. Ybarra, and A. J. Levine. Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. *Proceedings of National Academy Sciences, USA*, 96(12):6745–6750, 1999.
- [4] A. Antoniadis, S. Lambert-Lacroix, and F. Leblanc. Effective dimension reduction methods for tumor classification using gene expression data. *Bioinformatics*, 19(5):563–570, 2003.
- [5] S. A. Armstrong, J. E. Staunton, L. B. Silverman, R. Pieters, M. L. den Boer, M. D. Minden, S. E. Sallan, E. S. Lander, T. R. Golub, and S. J. Korsmeyer. Mll translocations specify a distinct gene expression profile that distinguishes a unique leukemia. *Bioinformatics*, 30(16):41–47, 2002.
- [6] C. Aviles-Cruz, A. Guerin-Dugue, J. L. Voz, and D. V. Cappel. Enhanced learning for evolutive neural architecture. In *ESPRIT Basic Research Project Number 6891*, 1995.

- [7] A. M. Bagirov, B. Ferguson, S. Ivkovic, G. Saunders, and J. Yearwood. New algorithms for multi-class cancer diagnosis using tumor gene expression signatures. *Bioinformatics*, 19(14):1800–1807, 2003.
- [8] E. B. Baum and F. Wilczek. Supervised learning of probability distributions by neural networks. *Neural information processing systems, Proceedings of the First IEEE Conference*, pages 52–61, 1988.
- [9] D. A. Benson, M. Boguski, D. J. Lipman, and J. Ostell. Genbank. *Nucleic Acids Research*, 28(1):15–18, 2000.
- [10] J. O. Berger. *Statistical Decision Theory and Bayesian Analysis*, Springer Series in Statistics (2nd edition). Springer, 1985.
- [11] J. M. Bernardo and A. F. M. Smith. *Bayesian theory*. Wiley, 1994.
- [12] D. P. Berrar, W. Dubitzky, and M. Granzow. *A Practical Approach to Microarray Data Analysis*. Springer, 2002.
- [13] A. Bhattacharjee, W. G. Richards, J. Staunton, C. Li, S. Monti, P. Vasa, C. Ladd, J. Beheshti, R. Bueno, M. Gillette, M. Loda, G. Weber, E. J. Mark, E. S. Lander, W. Wong, B. E. Johnson, T. R. Golub, D. J. Sugarbaker, and M. Meyerson. Classification of human lung carcinomas by mrna expression profiling reveals distinct adenocarcinoma subclasses. *Proceedings of National Academy Sciences, USA*, 98(24):13790–13795, 2001.
- [14] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.
- [15] C. L. Blake and C. J. Merz. UCI repository of machine learning databases. In <http://www.ics.uci.edu/~mllearn/MLRepository.html>. Department of Information and Computer Sciences, University of California, Irvine, USA, 1998.
- [16] R. Bolle, J. Connell, S. Pankanti, N. Ratha, and A. Senior. *Guide to Biometrics*. Springer, 2003.
- [17] P. E. Bourne and H. Weissig, editors. *Structural Bioinformatics*. Wiley, 2003.
- [18] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and regression trees*. Chapman Hall, New York, 1993.
- [19] P. O. Brown and D. Botstein. Exploring the new world of the genome with dna microarrays. *Nature Genetics*, 21:33–37, 1999.

- [20] W. L. Buntine and A. S. . Weigend. Computing second derivatives in feed-forward networks: A review. *IEEE Transactions on Neural Networks*, 5(3), 1991.
- [21] D. Casasent and X. wen Chen. Radial basis function neural networks for nonlinear fisher discrimination and neyman=pearson classification. *Neural networks*, 16:529–535, 2003.
- [22] S. Chen, C. Cowan, and P. Grant. Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Transactions on Neural Networks*, 2(2):302–309, 1991.
- [23] C. K. Chow. An optimum character recognition system using decision functions. *IRE transactions*, pages 247–254, 1957.
- [24] P. Degoulet. *Introduction to Clinical Informatics*. Springer, 1996.
- [25] M. Dettling and P. Buhlmann. Boosting for tumor classification with gene expression data. *Bioinformatics*, 19(9):1061–1069, 2003.
- [26] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley, 2000.
- [27] S. Dudoit, J. Fridlyand, and T. P. Speed. Comparison of discrimination methods for classification of tumors using gene expression data. *Journal of the American Statistical Association*, 97(457):77–87, 2002.
- [28] M. B. Eisen, P. T. Spellman, P. O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proceedings of National Academy Sciences, USA*, 95(25):14863–14868, 1998.
- [29] A. Esposito, M. Marinaro, D. Oricchio, and S. Scarpetta. Approximation of continuous and discontinuous mappings by a growing neural rbf-based algorithm. *Neural Networks*, 13(6):651–665, 2000.
- [30] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936.
- [31] E. Fix and J. Joseph L. Hodges. Discriminatory analysis: Nonparametric discrimination: consistancy properties. *USAF School of Aviation Medicine*, 4:261–279, 1951.

- [32] E. Fix and J. Joseph L. Hodges. Discriminatory analysis: Nonparametric discrimination: small sample performance. *USAF School of Aviation Medicine*, 11:280–322, 1952.
- [33] W. J. Fu, E. R. Dougherty, B. Mallick, and R. J. Carroll. How many samples are needed to build a classifier: a general sequential approach. *Bioinformatics*, 21(1):63–70, 2005.
- [34] X. Fu and L. Wang. Data dimensionality reduction with application to simplifying rbf network structure and improving classification performance. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 33(3):399–409, 2003.
- [35] T. S. Furey, N. Cristianini, N. Duffy, D. W. Bednarski, M. Schummer, and D. Haussler. Support vector machine classification and validation of cancer tissue samples using microarray expression data. *Bioinformatics*, 16(10):906–914, 2000.
- [36] L. Gang, Y. Wenyu, L. Ling, Y. Qilian, and Y. Xuemin. An artificial-intelligence approach to ECG analysis. *IEEE Engineering in Medicine and Biology Magazine*, 19(2):95–100, 2000.
- [37] M. E. Garber, O. G. Troyanskaya, K. Schluens, S. Petersen, Z. Thaesler, M. Pacyna-Gengelbach, M. van de Rijn, G. D. Rosen, C. M. Perou, R. I. Whyte, R. B. Altman, P. O. Brown, D. Botstein, and I. Petersen. Diversity of gene expression in adenocarcinoma of the lung. *Proceedings of National Academy Sciences, USA*, 98(24):13784–13789, 2001.
- [38] A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin. *Bayesian Data Analysis*. Chapman Hall, 1995.
- [39] T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomfield, and E. S. Lander. Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *Science*, 286(5439):531–537, 1999.
- [40] G. H. Gonnet, C. Korostensky, and S. Benner. Evaluation measures of multiple sequence alignments. *Journal of Computational Biology*, 7(1), 2000.
- [41] J. Gonzalez, I. Rojas, J. Ortega, H. Pomares, F. Fernandez, and A. Diaz. Multiobjective evolutionary optimization of the size, shape, and position pa-

- rameters of radial basis function networks for function approximation. *IEEE Transactions on Neural Networks*, 14(6):1478–1495, 2003.
- [42] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1), 2002.
- [43] M. Hagiwara. Theoretical derivation of momentum term for backpropagation. In *International Joint Conference on Neural Networks*, volume 1, pages 682–686, Baltimore, 1992.
- [44] J. B. Hampshire and A. Waibel. Connectionist architectures for multi-speaker phoneme recognition. In *Advances in Neural Information Processing Systems*, volume 2, pages 203–210, San Mateo, 1990.
- [45] J. M. Hancock and M. J. Zvelebil, editors. *Dictionary of Bioinformatics and Computational Biology*. Wiley, 2004.
- [46] S. Haykin. *Neural networks: A comprehensive foundation*. New Jersey: Prentice Hall, 1999.
- [47] I. Hedenfalk, D. Duggan, Y. Chen, M. Radmacher, M. Bittner, R. Simon, P. Meltzer, B. Gusterson, M. Esteller, M. Raffeld, Z. Yakhini, A. Ben-Dor, E. Dougherty, J. Kononen, L. Bubendorf, W. Fehrle, S. Pittaluga, S. Gruvberger, N. Loman, O. Johannsson, H. Olsson, B. Wilfond, G. Sauter, O.-P. Kallioniemi, A. Borg, and J. Trent. Gene-expression profiles in hereditary breast cancer. *Circulation*, 344:539–548, 2001.
- [48] U. Hoppe, S. Weiss, R. W. Stewart, and U. Eysholdt. An automatic sequential recognition method for cortical auditory evoked potentials. *IEEE Transactions Biomedical Engineering*, 48(2):154–164, 2001.
- [49] D. Hoyer, R. Bauer, K. Conrad, M. Galicki, A. Doring, H. Hoyer, B. Walter, H. Witte, and U. Zwiener. Specific monitoring of neonatal brain function with optimized frequency bands. *IEEE Engineering in Medicine and Biology Magazine*, 20(5):40–46, 2001.
- [50] G.-B. Huang. Learning capability and storage capacity of two-hidden-layer feedforward networks. *IEEE Transactions on Neural Networks*, 14(2):274–281, 2003.
- [51] G.-B. Huang, L. Chen, and C.-K. Siew. Universal approximation using incremental constructive feedforward networks with random hidden neurons. *accept by IEEE Transactions on Neural Networks*, 17(3), 2006.

- [52] G.-B. Huang, P. Saratchandran, and N. Sundararajan. An efficient sequential learning algorithm for growing and pruning RBF (GAP-RBF) networks. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 34(6):2284–2292, 2004.
- [53] G.-B. Huang, P. Saratchandran, and N. Sundararajan. A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation. *IEEE Transactions on Neural Networks*, 16(1):57–67, 2005.
- [54] G.-B. Huang and C.-K. Siew. Extreme learning machine with randomly assigned RBF kernels. *International Journal of Information Technology*, 11(1), 2005.
- [55] G.-B. Huang and C.-K. Siew. Extreme learning machine: RBF network case. In *Proceedings of the Eighth International Conference on Control, Automation, Robotics and Vision (ICARCV 2004)*, Kunming, China, 6-9 Dec, 2004.
- [56] G.-B. Huang, Q.-Y. Zhu, K. Z. Mao, C.-K. Siew, P. Saratchandran, and N. Sundararajan. Can threshold networks be trained directly? *IEEE Transactions on Circuits and Systems II*, 53(3):187–194, 2006.
- [57] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. Extreme learning machine: theory and applications. *accepted by Neurocomputing*, 2005.
- [58] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. Extreme learning machine: A new learning scheme of feedforward neural networks. In *Proceedings of International Joint Conference on Neural Networks (IJCNN2004)*, Budapest, Hungary, 25-29 July, 2004.
- [59] X. Huang and W. Pan. Linear regression and two-class classification with gene expression data. *Bioinformatics*, 19(16):2072–2078, 2003.
- [60] Y.-S. Hwang and sung Yang Bang. An efficient method to construct a radial basis function neural network classifier. *Neural networks*, 18(8):1495–1503, 1997.
- [61] A. K. Jain, R. P. Duin, and J. Mao. Statistical pattern recognition: a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):4–37, 2000.
- [62] V. Kadiramanathan and M. Niranjana. A function estimation approach to sequential learning with neural networks. *Neural Computation*, 5(6):954–975, 1993.

- [63] N. B. Karayiannis. Reformulated radial basis neural networks trained by gradient descent. *IEEE Transactions on Neural Networks*, 10(3):657–671, 1999.
- [64] N. B. Karayiannis and G. W. Mi. Growing radial basis neural networks: merging supervised and unsupervised learning with network growth techniques. *IEEE Transactions on Neural Networks*, 8(6):1492–1506, 1997.
- [65] N. B. Karayiannis and M. M. Randolph-Gips. On the construction and training of reformulated radial basis function neural networks. *IEEE Transactions on Neural Networks*, 14(4):835–846, 2003.
- [66] R. Kasturi and M. M. Trivedi. *Image Analysis Applications*. Marcel Dekker, 1990.
- [67] A. D. Keller, M. Schummer, L. Hood, and W. L. Ruzzo. Bayesian classification of dna array expression data. *Technical Report, University of Washington*, 2000.
- [68] J. Khan, J. S. Wei, M. Ringner, L. H. Saal, M. Ladanyi, F. Westermann, F. Berthold, M. Schwab, C. R. Antonescu, C. Peterson, and P. S. Meltzer. Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks. *Nature Medicine*, 7(6):673–679, 2001.
- [69] D. M. Kline and V. L. Berardi. Evaluating squared-error and cross-entropy functions for training neural networks classifiers. *Decision Sciences Institute 2002 Annual Meeting Proceedings*, pages 169–174, 2002.
- [70] M. Lagerholm, C. Peterson, G. Braccini, L. Edenbrandt, and L. Sornmo. Clustering ECG complexes using hermite functions and self-organizing maps. *IEEE Transactions on Biomedical Engineering*, 47(7):838–848, 2000.
- [71] E. S. Lander. Array of hope. *Nature Genetics*, 21:3–4, 1999.
- [72] J. W. Lee, J. B. Lee, M. Park, and S. H. Song. An extensive comparison of recent classification tools applied to microarray data. *Computational Statistics and Data Analysis*, 48:869–885, 2005.
- [73] K.-M. Lee and W. N. Street. An adaptive resource-allocating network for automated detection, segmentation, and classification of breast cancer nuclei topic area: image processing and recognition. *IEEE transactions on neural networks*, 14(3):680–687, 2003.

- [74] S.-W. Lee. *Advances in Handwriting Recognition*. World Scientific, 1999.
- [75] Y. Lee and C.-K. Lee. Classification of multiple cancer types by multicategory support vector machines using gene expression data. *Bioinformatics*, 19(9):1132–1139, 2003.
- [76] A. M. Lesk. *Introduction to Bioinformatics*. Oxford University Press, 2002.
- [77] L. Li, C. R. Weinberg, T. A. Darden, and L. G. Pedersen. Gene selection for sample classification based on gene expression data: study of sensitivity to choice of parameters of the ga/knn method. *Bioinformatics*, 17(12):1131–1142, 2001.
- [78] M.-B. Li, G.-B. Huang, P. Saratchandran, and N. Sundararajan. Fully complex extreme learning machine. *Neurocomputing (in press)*, 2005.
- [79] T. Li, C. Zhang, , and M. Ogihara. A comparative study of feature selection and multiclass classification methods for tissue classification based on gene expression. *Bioinformatics*, 20(15):2429–2437, 2004.
- [80] Y. Liao. *Neural networks for pattern classification and universal approximation*. PhD thesis, North Carolina State University, 2002.
- [81] R. J. Light and B. H. Margolin. An analysis of variance for categorical data. *Journal of the American Statistical Association*, 66(335):534–544, 1971.
- [82] R. Linder, D. Dew, H. Sudhoff, D. Theegarten, K. Remberger, S. J. Poppl, , and M. Wagner. The ‘subsequent artificial neural network’ (sann) approach might bring more classificatory power to ann-based dna microarray analyses. *Bioinformatics*, 20(18):3544–3552, 2004.
- [83] D. Lowe. Characterising complexity by the degrees of freedom in a radial basis function network. *Neurocomputing*, 19(1).
- [84] Y. Lu and J. Han. Cancer classification using gene expression data. *Information Systems*, 28(4):243–268, 2003.
- [85] N. M. Luscombe, D. Greenbaum, and M. Gerstein. What is bioinformatics? a proposed definition and overview of the field. *Intl. Medical Informatics Association*, pages Yearbook, Pages 83–99, 2001.
- [86] D. Michie, D. Spiegelhalter, and C. Taylor, editors. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, 1994.

- [87] C. Miller, J. Gurd, and A. Brass. A rapid algorithm for sequence database comparisons: application to the identification of vector contamination in the embl databases. *Bioinformatics*, 15(2):111–121, 1999.
- [88] M. L. Minsky and S. A. Pappert. *Perceptrons: an introduction to computational geometry*. MIT press, 1969.
- [89] T. M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.
- [90] J. Moody and C. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:289–303, 1989.
- [91] L. Moreno, J. D. Pineiro, J. L. Sanchez, S. Manas, J. Merino, L. Acosta, and A. Hamilton. Brain maturation estimation using neural classifier. *IEEE Transactions on Biomedical Engineering*, 42(4):428–432, 1995.
- [92] N. Morgan and H. Bourlard. Continuous speech recognition using multi-layer perceptrons with hidden markov models. In *International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 413–416, Albuquerque, 1990.
- [93] I. T. Nabney. Efficient training of rbf networks for classification. *International Journal of Neural Systems*, 14(3):201–208, 2004.
- [94] D. V. Nguyen and D. M. Rocke. Multi-class classification via partial least squares with gene expression profiles. *Bioinformatics*, 18(9):1216–1226, 2002.
- [95] C. L. Nutt, D. R. Mani, R. A. Betensky, P. Tamayo, J. G. Cairncross, C. Ladd, U. Pohl, C. Hartmann, M. E. McLaughlin, T. T. Batchelor, P. M. B. A. von Deimling, S. L. Pomeroy, T. R. Golub, and D. N. Louis. Gene expression-based classification of malignant gliomas correlates better with survival than histological classification. *Cancer Research*, 63:1602–1607, 2003.
- [96] C. A. Orengo and W. R. Taylor. Ssap: sequential structure alignment program for protein structure comparison. *Methods Enzymol*, 266:617–635, 1996.
- [97] S. Osowski and T. H. Linh. ECG beat recognition using fuzzy hybrid neural network. *IEEE Transactions on Biomedical Engineering*, 48(11):1265–1271, 2001.
- [98] A. Papoulis. *Probability, Random Variables and Stochastic Processes*. McGraw-Hill, 1991.

- [99] E. Parzen. On estimation of a probability density function and mode. *Annals of Mathematical Statistics*, 33(3):1065–1076, 1962.
- [100] B. O. Peters, G. Pfurtscheller, and H. Flyvbjerg. Automatic differentiation of multichannel EEG signals. *IEEE Transactions on Biomedical Engineering*, 48(1):111–116, 2001.
- [101] J. Platt. A resource-allocating network for function interpolation. *Neural Computation*, 3(2):213–225, 1991.
- [102] N. Pochet, F. D. Smet, J. A. K. Suykens, and B. L. R. D. Moor. Systematic benchmarking of microarray data classification: assessing the role of nonlinearity and dimensionality reduction. *Bioinformatics*, 20(17):3185–3195, 2004.
- [103] S. L. Pomeroy, P. Tamayo, M. Gaasenbeek, M. A. Lisa M. Sturla and, M. E. McLaughlin, J. Y. H. Kim, L. C. Goumnerova, P. M. Black, C. Lau, J. C. Allen, D. Zagzag, J. M. Olson, T. Curran, C. Wetmore, J. A. Biegel, T. Poggio, S. Mukherjee, R. Rifkin, A. Califano, G. Stolovitzky, D. N. Louis, J. P. Mesirov, E. S. Lander, and T. R. Golub. Prediction of central nervous system embryonal tumour outcome based on gene expression. *Nature*, 415:436–442, 2002.
- [104] M. J. D. Powell. Radial basis functions for multivariable interpolation: A review. In *Clarendon Press Institute Of Mathematics And Its Applications Conference Series, Algorithms for approximation*, pages 143–167, 1987.
- [105] L. Prechelt. PROBEN1-a set of neural network benchmark problems and benchmarking rules. In *Technical Report 21/94*, Fakultat fur Informatik, University Karlsruhe, Germany, 1994.
- [106] G. V. Puskorius and L. A. Feldkamp. Neurocontrol of nonlinear dynamics systems with kalman filter trained recurrent networks. *IEEE Transactions on Neural Networks*, 5(2):279–297, 1994.
- [107] G. V. Puskorius and L. A. Feldkamp. Decoupled extended kalman filter training of feedforward layered networks. In *Proceedings of the International Joint Conference on Neural Networks*, pages 771–777, Seattle, 8-14 July 1991.
- [108] J. R. Quinlan. *Learning efficient classification procedures and their application to chess end games*. R. S. Michalski, J. G. Carbonell and T. M. Mitchell, editors, *Machine learning: An artificial intelligence approach*. Morgan Kaufmann, San Francisco, CA, 1983.

- [109] J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann, 1993.
- [110] S. Ramaswamy, P. Tamayo, R. Rifkin, S. Mukherjee, C.-H. Yeang, M. Angelo, C. Ladd, M. Reich, E. Latulippe, J. P. Mesirov, T. Poggio, W. Gerald, M. Loda, E. S. Lander, and T. R. Golub. Multiclass cancer diagnosis using tumor gene expression signatures. *Proceedings of National Academy Sciences, USA*, 98(26):15149–15154, 2002.
- [111] M. D. Richard and R. P. Lippmann. Neural network classifiers estimate bayesian a posteriori probabilities. *Neural Computation*, 3(4):461–483, 1991.
- [112] B. D. Ripley. *Pattern recognition and neural networks*. Cambridge University Press, 1996.
- [113] I. Rojas, H. Pomares, J. Bernier, J. Ortega, B. Pino, F. Pelayo, and A. Prieto. Time series analysis using normalized pg-rbf network with regression weights. *Neurocomputing*, 42:267–285, 2002.
- [114] D. T. Ross, U. Scherf, M. B. Eisen, C. M. Perou, C. Rees, P. Spellman, V. Iyer, S. S. Jeffrey, M. V. de Rijn, M. Waltham, A. Pergamenschikov, J. C. Lee, D. Lashkari, D. Shalon, T. G. Myers, J. N. Weinstein, D. Botstein, and P. O. Brown. Systematic variation in gene expression patterns in human cancer cell lines. *Pharmacogenomics*, 24(3):227–235, 2000.
- [115] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by propagating errors. *Nature*, 323(99):533–536, 1986.
- [116] D. E. Rumelhart and J. L. McClelland, editors. *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations*. MIT Press, 1986.
- [117] M. Salmeron, J. Ortega, C. G. Puntonet, and A. Prieto. Improved ran sequential prediction using orthogonal techniques. *Neurocomputing*, 41:153–172, 2001.
- [118] B. Scholkopf, C. J. C. Burges, and A. J. Smola, editors. *Advances in kernel methods: support vector learning*. MIT Press, 1999.
- [119] R. Schowengerdt, editor. *Remote Sensing: Models and Methods for Image Processing*. Elsevier, 1997.

- [120] F. Schwenker, H. A. Kestler, and G. Palm. Three learning phases for radial-basis-function networks. *Neural Networks*, 14:439–458, 2001.
- [121] D. Serre. *Matrices: Theory and Applications*. Springer-Verlag New York, Inc, 2002.
- [122] M. A. Shipp, K. N. Ross, P. Tamayo, A. P. Weng, J. L. Kutok, R. C. Aguiar¹, M. Gaasenbeek, M. Angelo, M. Reich, G. S. Pinkus, T. S. Ray, M. A. Koval¹, K. W. Last, A. Norton, T. A. Lister, J. Mesirov, D. S. Neuberg, E. S. Lander, J. C. Aster, and T. R. Golub. Gene expression correlates of clinical prostate cancer behavior. *Nature Medicine*, 8:68–74, 2002.
- [123] D. Singh, P. G. Febbo, K. Ross, D. G. Jackson, J. Manola, C. Ladd, P. Tamayo, A. A. Renshaw, A. V. D’Amico, J. P. Richie, E. S. Lander, M. Loda, P. W. Kantoff, T. R. Golub, and W. R. Sellers. Diffuse large b-cell lymphoma outcome prediction by gene-expression profiling and supervised machine learning. *Cancer Cell*, 1(2), 2002.
- [124] W. E. Snyder and H. Qi. *Machine Vision*. Cambridge University Press, 2004.
- [125] D. F. Specht. Generation of polynomial discriminant functions for pattern recognition. *IEEE Transactions on Electronic Computers*, 16(3):308–319, 1967.
- [126] A. Statnikov, C. F. Aliferis, I. Tsamardinos, D. Hardin, and S. Levy. A comprehensive evaluation of multicategory classification methods for microarray gene expression cancer diagnosis. *Bioinformatics*, 21(5):631–643, 2005.
- [127] J. E. Staunton, D. K. Slonim, H. A. Collier, P. Tamayo, M. J. Angelo, J. Park, U. Scherf, J. K. Lee, W. O. Reinhold, J. N. Weinstein, J. P. Mesirov, E. S. Lander, and T. R. Golub. Chemosensitivity prediction by transcriptional profiling. *Proceedings of National Academy Sciences, USA*, 98(19):10787–10792, 2001.
- [128] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society B*, 36:111–147, 1974.
- [129] A. I. Su, J. B. Welsh, L. M. Sapinoso, S. G. Kern, P. Dimitrov, H. Lapp, P. G. Schultz, S. M. Powell, C. A. Moskaluk, J. Henry F. Frierson, and G. M. Hampton. Molecular classification of human carcinomas by use of gene expression signatures. *Cancer Research*, 61:7388–7393, 2001.

- [130] Y. Su, T. M. Murali, V. Pavlovic, M. Schaffer, and S. Kasif. Rankgene: identification of diagnostic genes based on expression data. *Bioinformatics*, 19(12):1578–1579, 2003.
- [131] A. L. Tarca and J. E. K. Cooke. A robust neural networks approach for spatial and intensity-dependent normalization of cdna microarray data. *Bioinformatics*, 21(11):2674–2683, 2005.
- [132] S. Theodoridis and K. Koutroumbas. *Pattern Recognition*. Academic Press, 2003.
- [133] M. I. Torres and A. Sanfeliu, editors. *Pattern Recognition and Applications*. IOS Press, 2002.
- [134] O. Troyanskaya, M. Cantor, G. Sherlock, P. Brown, T. Hastie, R. Tibshirani, D. Botstein, and R. B. Altman. Missing value estimation methods for dna microarrays. *Bioinformatics*, 17(6):520–525, 2001.
- [135] C. H. Wu and J. W. McLarty. *Neural Networks and Genome Informatics*. Elsevier Science, 2000.
- [136] S. Wu, Y. Qian, Z. Gao, and J. Lin. A novel method for beat-to-beat detection of ventricular late potentials. *IEEE Transactions on Biomedical Engineering*, 48(8):931–935, 2001.
- [137] M. Xiong, W. Li, J. Zhao, L. Jin, and E. Boerwinkle. Feature (gene) selection in gene expression-based tumor classification. *Molecular Genetics and Metabolism*, 73(3):239–247, 2001.
- [138] Z. R. Yang. Mining gene expression data based on template theory. *Bioinformatics*, 20(16):2759–2766, 2004.
- [139] C.-H. Yeang, S. Ramaswamy, P. Tamayo, S. Mukherjee, R. M. Rifkin, M. Angelo, M. Reich, E. Lander, J. Mesirov, and T. Golub. Molecular classification of multiple tumor types. *Bioinformatics*, 17:S316–S322, 2001.
- [140] E.-J. Yeoh, M. E. Ross, S. A. Shurtleff, W. K. Williams, D. Patel, R. Mahfouz, F. G. Behm, S. C. Raimondi, M. V. Relling, A. Patel, C. Cheng, D. Campana, D. Wilkins, X. Zhou, J. Li, H. Liu, C.-H. Pui, W. E. Evans, C. Naeve, L. Wong, and J. R. Downing. Classification, subtype discovery, and prediction of outcome in pediatric lymphoblastic leukemia by gene expression profiling. *Cancer Cell*, 1(2):133–143, 2002.

- [141] L. Yingwei, N. Sundararajan, and P. Saratchandran. A sequential learning scheme for function approximation using minimal radial basis function (RBF) neural networks. *Neural Computation*, 9:461–478, 1997.
- [142] L. Yingwei, N. Sundararajan, and P. Saratchandran. Performance evaluation of a sequential minimal radial basis function (RBF) neural network learning algorithm. *IEEE Transactions on Neural Networks*, 9(2):308–318, 1998.
- [143] X. Zhou and K. Z. Mao. **LS** bound based gene selection for dna microarray data. *Bioinformatics*, 21(8):1559–1564, 2005.
- [144] Q. Zhu and S. Billings. Fast orthogonal identification of nonlinear stochastic models and radial basis function neural networks. *International Journal of Control*, 64(5):871–886, 1996.

Appendix A

This appendix presents the external cross validation results for GCM dataset, Lung dataset and Lymphoma datasets.

Experiment 1 - GCM Dataset

In [110], 144 training samples and 46 primary testing samples in GCM dataset, are combined and then these 190 samples are randomly split into 100 splits of training and testing sets of 144 and 46 samples in a class proportional manner. In order to have a direct comparison with [110], for ELM and SVM-OVO method, we perform the classification task using the same data and experimental methods as described by [110]. The most significant 14, 28, 42, 56, 70, 84 and 98 genes selected by the recursive feature elimination method as described in [110]. The average testing accuracy of the 100 splits of training and test set of ELM and SVM-OVO together with the results of SVM-OVA from [110] are shown in Table 8.1 and Figure 8.1.

#Genes	ELM		SVM-OVO		SVM-OVA
	Accuracy	Std. Deviation	Accuracy	Std. Deviation	Accuracy
14	68.85	5.1	63	5.0	69
28	73.59	4.9	65.26	6.1	70.5
42	73.13	5.9	68	5.2	70.5
56	73.96	5.2	69.85	5.8	71.5
70	74.46	4.9	71.26	5.2	71.5
84	75.24	5.1	72.44	4.9	71.75
98	76.5	5.3	73.78	5.1	72

Table 8.1: Testing accuracy (%) of different algorithms on GCM dataset

For the other two datasets, viz., Lung and Lymphoma , evaluations are carried out using both ELM and SVM-OVO algorithms. Only BSS/WSS method [27] is used for gene selection, which sorts the genes in a descending order by the ratio of 'between

Figure 8.1: Comparison of the testing accuracies of ELM, SVM-OVO, SVM-OVA.

group to within group' sum of squares. Ten choices of number of genes, from 10 to 100 in an increment of 10, are selected and used in the simulation of both ELM and SVM-OVO algorithms. The average testing accuracy over 100 trials for ELM and SVM-OVO are shown in Table 8.2 to 8.3.

Experiment 2 - Lung Dataset

#Genes	ELM		SVM-OVO	
	Accuracy	Std. Deviation	Accuracy	Std. Deviation
10	85.40	7.0	85.93	7.1
20	89.64	5.4	87.07	6.1
30	92.00	6.9	92.50	5.7
40	93.43	5.7	92.64	5.3
50	92.36	6.8	93.24	6.4
60	91.86	6.4	93.64	6.3
70	92.35	6.8	92.93	6.0
80	91.00	6.4	92.14	6.7
90	90.36	7.1	91.50	6.6
100	90.21	6.9	90.71	6.6

Table 8.2: Testing accuracy (%) of ELM and SVM-OVO algorithms on Lung dataset

Experiment 3 - Lymphoma Dataset

Remarks

#Genes	ELM		SVM-OVO	
	Accuracy	Std. Deviation	Accuracy	Std. Deviation
10	97.33	4.3	98.33	3.6
20	96.83	5.5	99.67	2.0
30	95.92	6.1	99.25	2.7
40	95.00	6.5	99.58	2.2
50	97.00	4.8	100	0
60	97.33	4.1	100	0
70	96.50	5.1	99.33	2.3
80	97.83	4.5	99.92	0.8
90	96.57	5.1	99.92	0.8
100	96.33	5.8	100	0

Table 8.3: Testing accuracy (%) for ELM and SVM-OVO algorithms on Lymphoma dataset

From the results on the three datasets, we find that ELM achieves a better performance than SVM-OVO on GCM dataset and a similar performance on the Lung dataset and Lymphoma dataset as compared to SVM-OVO. Similar results can be observed in Chapter 7.