Aston University

Some pages of this thesis may have been removed for copyright restrictions.

If you have discovered material in AURA which is unlawful e.g. breaches copyright, (either yours or that of a third party) or any other law, including but not limited to those relating to patent, trademark, confidentiality, data protection, obscenity, defamation, libel, then please read our Takedown Policy and contact the service immediately

The Impact of Architecture on the Performance of Artificial Neural Networks

Richard Thomas John Bostock

Doctor of Philosophy

The University of Aston in Birmingham

August 1994

© This copy of the thesis has been supplied on the condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the thesis and no information derived from it may be published without proper acknowledgement.

The University of Aston in Birmingham

The Impact of Architecture on the Performance of Artificial Neural Networks

Richard Thomas John Bostock

Doctor of Philosophy

September 1995

Summary

A number of researchers have investigated the impact of network architecture on the performance of artificial neural networks. Particular attention has been paid to the impact on the performance of the multi-layer perceptron of architectural issues, and the use of various strategies to attain an optimal network structure. However, there are still perceived limitations with the multi-layer perceptron and networks that employ a different architecture to the multi-layer perceptron have gained in popularity in recent years. Particularly, networks that implement a more localised solution, where the solution in one area of the problem space does not impact, or has a minimal impact, on other areas of the space. In this study, we discuss the major architectural issues effecting the performance of a multi-layer perceptron, before moving on to examine in detail the performance of a new localised network, namely the bumptree.

The work presented here examines the impact on the performance of artificial neural networks of employing alternative networks to the long established multi-layer perceptron. In particular, networks that impose a solution where the impact of each parameter in the final network architecture has a localised impact on the problem space being modelled are examined. The alternatives examined are the radial basis function and bumptree neural networks, and the impact of architectural issues on the performance of these networks is examined. Particular attention is paid to the bumptree, with new techniques for both developing the bumptree structure and employing this structure to classify patterns being examined.

Keywords: Neural Networks, Multi-Layer Perceptron, Radial Basis Function, Bumptree, Localised Impact, Optimal Network Structure.

To my parents,
without your help and support this work would not have been possible.

Acknowledgements

I wish to express my thanks to the following people:

Dr. Alan Harget for his support and guidance.

Professor David Bounds for his invaluable assistance.

Dr. Richard Rohwer for his guidance in some of the more complex parts of my research.

Dr. Ela Claridge for her input to our work on the use of neural networks to diagnose skin cancer.

Bryn Williams for his input to our work into the development of the genetic bumptree.

Lynn Bostock for her unfailing support.

To my friends and colleagues in the department.

And finally to Serc for providing the necessary funding.

List of Contents

Title Page 1

Summary 2

Dedication 3

Acknowledgements 4

List of Contents 5

List of Figures 8

List of Tables 11

1. Introduction 14

2. An Examination Of Architectural Issues Associated With The Multi-Layer Perceptron 20

2.1 Introduction 20

2.2 The Limitations of the Standard Multi-layer Perceptron 20

2.3 Dynamic Network Design Strategies 24

  2.3.1 Pruning Algorithms 24

  2.3.2 Constructive Algorithms 35

  2.3.3 Combined Algorithms 41

  2.3.4 Genetic Algorithms Applied to Multi-layer Perceptrons 43

2.4 Summary 45

3. Alternate Classification Systems That Implement Local Solutions 47

3.1 Introduction 47

3.2 The Radial Basis Function Network 47

3.3 Decision Trees (Machine Learning Algorithms) 54

3.4 Neural Networks that Utilise a Tree Structure 61

  3.4.1 The Bumptree 62

  3.4.2 The Constructive Tree RBF 64

3.5 Summary 66

4. Architectural Issues in Developing a Bumptree Neural Network 67

4.1 Introduction 67

4.2 Introduction to the Bumptree Neural Network 68

4.3 Placement and Dimensions of Functions: An Introduction 70

4.4 Omohundro's Approach 73
4.5 Multiple Initial Functions (MIF) 73
4.6 The N-Function Bumptree 77
4.7 The Use of Non-Hierarchical Clustering Techniques 79
4.8 Comparative Results for Function Centring and Constraint Techniques 82
4.8.1 Training and Generalisation Performance 84
4.8.2 Training Time 102
4.9 Summary 114

5. Further Architectural Issues in Developing a Bumptree Neural Network 117
5.1 Introduction 117
5.2 The Learning Algorithm 118
5.3 The Use of the Bumptree Structure to Calculate the Output of the Network 125
5.3.1 Comparative Results for Different Output Calculation Techniques 128
5.4 Additional Training Mechanisms Included in the Training Algorithm 131
5.5 The Genetic Bumptree 134
5.6 Summary 138

6. Comparative Study of the Bumptree, RBF and MLP Networks 141
6.1 Introduction 141
6.2 Highly Non-Linear Problems Without a Test of Generalisation Performance 144
6.2.1 The Radial Basis Function Network 145
6.2.2 The Multi-Layer Perceptron 148
6.2.3 Classification Performance of the MLP, RBF and Bumptree Neural Networks 151
6.3 The Diabetes Diagnosis Data Set 154
6.3.1 The Radial Basis Function Network 155
6.3.2 The Multi-Layer Perceptron 157
6.3.3 Classification Performance of the MLP, RBF and Bumptree Neural Networks 158
6.4 The Skin Cancer Diagnosis Data Set 159
6.4.1 The Radial Basis Function Network 160
6.4.2 The Multi-Layer Perceptron 163
6.4.3 Classification Performance of the MLP, RBF and Bumptree Neural Networks 164

6.5 The Iris Data Set 166
6.5.1 The Radial Basis Function Network 166
6.5.2 The Multi-Layer Perceptron 169
6.5.3 Classification Performance of the MLP, RBF and Bumptree Neural Networks 170
6.6 The Petersen and Barney Vowel Data 171
6.6.1 The Radial Basis Function Network 172
6.6.2 The Multi-Layer Perceptron 175
6.6.3 Classification Performance of the MLP, RBF and Bumptree Neural Networks 176
6.7 Computational Complexity of the MLP, RBF and MIF Bumptree Networks 178
6.8 Summary 193

7. Conclusion 195

References 199

Appendix A 206
Appendix B 210
Appendix C 213
Appendix D 219

List of Figures

Figure 1.1 - The long-range effects of hyperplanes on classification with the standard MLP. 16

Figure 1.2 - Improved classification performance on the training set for the MLP through the use of additional hyperplanes. 16

Figure 2.1 - The OBD procedure. 34

Figure 2.2 - Percentage of training patterns correct versus the number of iterations for a network that adds in hidden units at flat spots in the learning process. 37

Figure 2.3 - Percentage of test patterns correct versus the number of iterations for a network with insufficient hidden units. 37

Figure 3.1 - The RBF network structure. 48

Figure 3.2 - A placement of basis functions encouraging good performance. 51

Figure 3.3 - A placement of basis functions giving poor performance. 52

Figure 3.4 - A two level bucket structure. 58

Figure 3.5 - A binary trie structure. 59

Figure 3.6 - A binary tree structure. 59

Figure 4.1 - The bumptree structure relating to the problem described in the text. 71

Figure 4.2 - A problem space divided by the functions in the neural network described in the text. 72

Figure 4.3 - Bump functions adhering to Omohundro's constraints. 73

Figure 4.4 - The MIF approach for adding functions to the network. 75

Figure 4.5 - An n-function bumptree to solve the iris problem. There are four functions at each splitting point because the problem has four input dimensions. 78

Figure 4.6 - The Forgy non-hierarchical clustering technique. 81

Figure 4.7 - The MacQueen's k-means non-hierarchical clustering technique. 82

Figure 4.8 - Average percentage performance on the training set for the various function centering and constraint techniques. 95

Figure 4.9 - Average percentage performance on the generalisation set for the various function centering and constraint techniques. 96

Figure 4.10 - The average performance of various bumptree versions on the training set of the iris, diabetes, vowel recognition and skin cancer tasks. 98

Figure 4.11 - The average performance of various bumptree versions on the generalisation set of the iris, diabetes, vowel recognition and skin cancer tasks. 99

Figure 4.12 - Pseudo code detailing steps involved in calculating the activation of the 10 candidate functions with the MIF approach. 107
Figure 4.13 - Pseudo code detailing steps involved in calculating the error of the 10 candidate functions with the MIF approach. 110
Figure 4.14 - Calculations involved in calculating the error of the 10 candidate functions with the MIF approach. 111
Figure 4.15 - The main steps employed to construct the MIF bumptree. 112
Figure 51 - Matrix 1, which is derived from equations 5.7 and 5.8. 121
Figure 5.2 - Matrix 2, which is the Alpha and Beta matrix, derived from equations 6.7 and 6.8. 121
Figure 5.3 - Matrix 3, which is the result matrix derived from the equations given in 6.7 and 6.8. 121
Figure 5.4 - Functions at the various levels of the tree are active on any individual pattern. This figure demonstrates how 4 of the bumptrees functions are active on the pattern that has just been presented to the network. Here functions 1,3, 6 and 9 are active. 126
Figure 6.1 - The average percentage performance of RBF networks employing the various function types with the optimum number of functions. 147
Figure 6.2 - The average percentage performance of RBF networks employing various numbers and types of functions on the Encoder (8) problem. 148
Figure 6.3 - The average percentage performance of the MLP, RBF and MIF bumptree neural networks on the Parity (6), Encoder (8) and XOR problems. 152
Figure 6.4 - The eight attributes for the diabetes data set. 154
Figure 6.5 - The average percentage performance of the RBF network that employed thin plate splines on the diabetes data set. 156
Figure 6.6 - The average percentage performance of the MLP on the iris pattern classification problem. 169
Figure 6.7 - The average percentage performance of the MLP, RBF and MIF bumptree neural networks on the iris pattern classification problem. 170
Figure 6.8 - The average percentage performance of RBF networks employing gaussian functions on the Petersen and Barney vowel data. 173
Figure 6.9 - The average percentage performance of RBF networks employing thin plate splines on the Petersen and Barney vowel data. 174
Figure 6.10 - The average percentage performance of RBF networks employing multi-quadratic functions on the Petersen and Barney vowel data. 174
Figure 6.11 - The average percentage performance of RBF networks employing inverse multi-quadratic functions on the Petersen and Barney vowel data. 175
Figure 6.12 - The average percentage performance of the MLP, RBF and MIF bumptree neural networks on the Petersen and Barney vowel recognition task. 177
Figure 6.13 - The main steps employed in training the MLP. 180
Figure 6.14 - The main steps employed to construct the RBF network. 181
Figure 6.15 - The main steps employed to construct the MIF bumptree. 182
Figure 6.16 - Pseudo code detailing the steps involved in calculating the activation of the functions in the RBF network. 185

Figure 6.17 - The procedure required to produce the output to a query of the MLP 190
Figure 6.18 - Routine to calculate the activation of gaussian functions. 191
Figure 6.19 - Routine for calculating the response to a query of the MIF bumptree 191

List of Tables

Table 4.1 - The data sets employed in the study. 83

Table 4.2 - The average percentage performance of the Omohundro approach to function centering and constraining over ten runs. 85

Table 4.3 - The average percentage performance of the MIF approach to function centering combined with Omohundro's approach to function constraining 87

Table 4.4 - The average percentage performance of the MIF approach to function centering combined with reduced constraints on the dimensions of the functions at the lower levels of the tree. The functions below the top level are restricted to the area covered by the patterns upon which their parent was active. 88

Table 4.5 - The average percentage performance of the MIF approach to function centering combined with reduced constraints on the dimensions of all the functions in the tree. The functions below the top level are restricted to the area covered by the patterns upon which their parent was active, and the top level functions have a radius of 1 in each dimension. 90

Table 4.6 - The average percentage performance of the MIF approach to function centering combined with reduced constraints on the dimensions of all the functions in the tree. All the functions have a radius of 1 in each dimension. 92

Table 4.7 - The average percentage performance for the MIF centre placement and confining techniques and Omohundro's approach on the training sets examined in this study. Each technique is identified by a number which corresponds to a key given in figure 4.15. 94

Table 4.8 - The average percentage performance for the MIF centre placement and confining techniques and Omohundro's approach on the generalisation sets examined in this study. Each technique is identified by a number which corresponds to a key given in figure 4.15. 95

Table 4.9 - The average percentage performance for the n-function bumptree, combined with radii of 1 for each function in each dimension. 97

Table 4.10 - The average percentage performance for the bumptree employing the Forgy non-hierarchical clustering technique, combined with MIF with all radii set to 1. 102

Table 4.11 - The average percentage performance of the various bumptree versions in terms of the number of functions required to reach the level of generalisation performance discussed in section 4.8.1. 103

Table 4.12 - Calculations required in order to assign the patterns to the functions depending on their activation, to optimise the Alpha and Beta values, and to calculate the error of the functions. 113

Table 5.1 - The average percentage performance of the MIF bumptree with all radii set to 1 and the output calculated using the LAF approach for different settings of *SMALL* on the iris problem. 123

Table 5.2 - The average percentage performance of the MIF bumptree with all radii set to 1 and the output calculated using the LAF approach for different settings of *SMALL* on the skin cancer diagnosis problem. 124

Table 5.3 - The average percentage performance of the MIF bumptree with all radii set to 1 and the output calculated using the LAF approach for different settings of *SMALL* on the Parity (6) problem. 124

Table 5.4 - The average percentage performance of the MIF bumptree with all radii set to 1 and the output calculated using the LAF output calculation technique. 128

Table 5.5 - The average percentage performance of the MIF bumptree with all radii set to 1 and the output calculated using the AAF technique. 130

Table 5.6 - The average number of functions required by the LAF and AAF output calculation techniques. 130

Table 5.7 - The average percentage performance of the genetic bumptree that employs the genetic algorithm to position the functions and the one-shot learning algorithm to optimise
the weight and bias values. 138

Table 6.1 - The average percentage performance of RBF networks employing the different
function types. 146

Table 6.2 - The average percentage performance of the MLP on the XOR problem. 149

Table 6.3 - The average percentage performance of the MLP on the Parity (6) problem. 150

Table 6.4 - The average percentage performance of RBF networks using thin plate splines on the diabetes data set. 155

Table 6.5 - The average percentage performance of RBF networks using gaussian functions on the diabetes data set. 157

Table 6.6 - The average percentage performance of the MLP on the diabetes data set. 158

Table 6.7 - The average percentage performance of the MLP, RBF and MIF bumptree neural networks on the diabetes data set for the networks with the best average

generalisation performance. 159
Table 6.8 - The average percentage performance of RBF networks employing thin plate splines on the skin cancer data set. 161
Table 6.9 - The average percentage performance of RBF networks employing gaussian functions on the skin cancer data set. 162
Table 6.10 - The average percentage performance of RBF networks employing multi -quadratic functions on the skin cancer data set. 162
Table 6.11 - The average percentage performance of RBF networks employing inverse multi-quadratic functions on the skin cancer data set. 163
Table 6.12 - The average percentage performance of the MLP on the skin cancer data set. 164
Table 6.13 - The average percentage performance of the MLP, RBF and MIF bumptree neural networks on the skin cancer data set. 165
Table 6.14 - The average percentage performance of RBF networks employing thin plate splines on the iris data set. 167
Table 6.15 - The average percentage performance of RBF networks employing multi-quadratic functions on the iris problem. 167
Table 6.16 - The average percentage performance of RBF networks employing inverse multi-quadratic functions on the iris data set. 168
Table 617 - The average percentage performance of the MLP on the Petersen and Barney vowel data. 176
Table 6.18 - The size of networks to produce the best generalisation performance on the problems with a valid test of generalisation. 179
Table 6.19 - Total Calculations required to train the MIF bumptree to a solution. 183
Table 6.20 - Total calculations (additions, subtractuions, multiplications, divisions) required by the MLP to achieve the desired level of performance. 184
Table 6.21 - The total calculations required to train the RBF network to an acceptable solution. 188
Table 6.22 - The calculations required to respond to a query of a trained MLP network 190
Table 6.23 - The calculations required by the MIF bumptree and RBF networks to respond to a query. 192

Chapter 1

Introduction

The development of computers able to display intelligence comparable to that displayed by humans has always been an appealing idea for computer scientists and the principal objective for researchers in artificial intelligence. This has been a very active area of research, with some limited success being achieved in recent years. Initial attention focused on the use of machine-learning algorithms, including rule-based or expert systems, and these systems have enjoyed a degree of success and commercial acceptance. Expert systems approach the task of creating "intelligent" computers through explicitly embodying the knowledge of experts in the system through the careful hand crafting of the knowledge base. In practice the knowledge acquisition phase of expert system development has proved to be time consuming and difficult. An alternative approach is to allow the system to develop its own representation of the problem area through the presentation of training examples. This approach is able to overcome many of the problems found in the development of expert systems, and is the one employed by neural networks in pattern classification.

Attention has turned increasingly towards the use of neural networks in an attempt to create "intelligent" computer systems for pattern classification tasks. The ability of neural networks to develop their own "knowledge" of the problem to be classified should allow the development of computer systems with the ability to learn to classify patterns. A wide range of neural networks employing different approaches have been developed to classify patterns. The neural network that has been most widely used for pattern classification is the multi-layer perceptron (MLP) trained by the standard back-propagation learning algorithm (Rumelhart, Hinton and Williams 1988). One of the major alternative systems to the MLP approach for pattern classification is the Radial Basis Function (RBF) network

(Powell 1985, Broomhead and Lowe 1988). Recently attention has also focused on the development of neural networks employing tree based structures; these form the main basis of this study.

The MLP, RBF and tree based neural networks examined in this study employ the supervised learning approach, in which patterns are presented to the network and the resulting output compared against the desired output with the network weights adjusted to minimise any difference. Unsupervised learning algorithms offer an alternative approach to the task of training the network by allowing it to self-organise, adjusting the weights according to a well-defined algorithm to produce the desired changes. The unsupervised learning algorithms do not use a straightforward error minimisation technique as the main component of the training algorithm. Networks that adopt an unsupervised approach to learning, such as the Kohonen feature map (Kohonen 1988) do not fall within the scope of this thesis. The standard MLP employs a supervised learning algorithm, and trains to a solution through the explicit minimisation of a given error. It includes no unsupervised component. In contrast, both the RBF and the tree based network examined in this study embody an unsupervised element alongside a supervised one in the training process.

A fundamental difference between the neural networks examined in this study is the approach adopted during the training phase. This difference will be further examined in later chapters. Another fundamental difference between the three neural networks examined in this study is the use of local and non-local (global) partitioning of the problem space. The MLP uses hyperplanes to partition the problem space, and these suffer from long range effects, since moving the hyperplanes to accommodate data in one area of the problem space can have an adverse effect on data elsewhere. This is shown in figure 1.1, where the hyperplane has been moved from position 1 to position 2 to enhance classification of patterns in class 1. However, this has the effect of degrading the

classification of patterns in class 2. Figure 1.2 shows how increasing the number of hyperplanes will improve classification performance on the training set. The RBF network and the tree based network employ a more local solution than the MLP, and it is hoped this might offer improved classification performance.

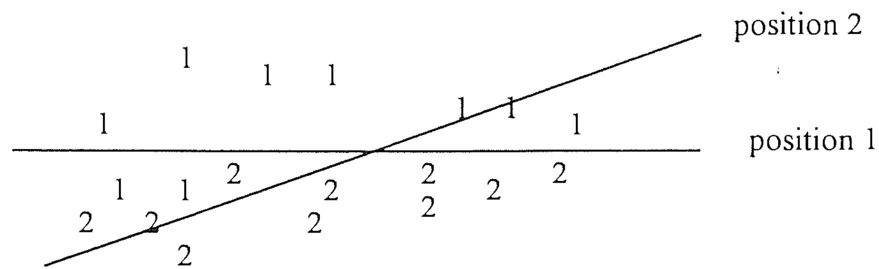

Figure 1.1 - The long-range effects of hyperplanes on classification with the standard MLP.

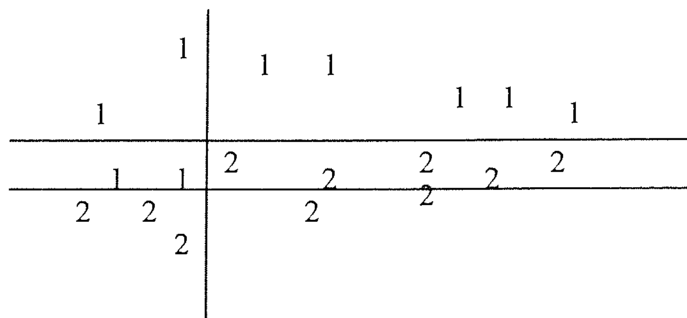

Figure 1.2 - Improved classification performance on the training set for the standard MLP through the use of additional hyperplanes.

Another aspect of the MLP trained with the standard back propagation learning algorithm that needs considering is the number of arbitrary parameters that need to be determined. Values need to be assigned to the learning rate, the momentum term, the initial weight and bias values, and most significantly the number of units in the hidden layer. If there are too few hidden units then the network will be unable to adequately partition the problem space, and if there are too many units then it will simply memorise the patterns of the training set and consequently give poor performance for patterns previously unseen. Thus,

the number of units assigned to the hidden layer is crucial to the network achieving optimal performance; it is not surprising therefore that much research has been concerned with investigating network architecture.

As a consequence of the limitations of the MLP, studies have also been undertaken into the development of alternative neural networks. The rationale for studying such networks is that the MLP, even with the optimum set of parameters and architecture is still susceptible to the "hyperplane effect" mentioned above. This has led to the development of neural networks that implement a local rather than global solution in the hope that they will offer an improvement in generalisation performance and training time. To date, the most popular of such neural networks are those that employ radial basis functions and these like the MLP are, in theory, able to form an arbitrarily close approximation to any continuous non-linear mapping.

The RBF network is, however, not the only alternative. Recently attention has begun to focus on the use of tree based structures to create neural networks that implement a local solution; such structures have their origins in the decision trees of machine learning. It is hoped that by employing a tree structure it will be possible to develop a neural network that will learn faster and perform better in terms of generalisation than the MLP or RBF neural networks. However, the work on tree based neural networks is in its early stages and the plausibility of these aims remains to be discovered.

The work presented in this thesis is concerned with investigating the impact of architecture on the performance of a neural network, with particular reference to the MLP, RBF, and a tree based network. The latter is based on the bumptree geometric data structure developed by Omohundro (1991).

The main aims of this work are as follows:

(1) To examine architectural issues that exist with neural networks. Attention will be paid to the MLP, the RBF and tree based neural networks.

(2) To develop a tree based localised neural network based on the bumptree geometric data structure introduced by Omohundro (1991), and to ascertain the impact that various architectural issues have on the performance of this localised network.

(3) To conduct a comparative study of the performance of the MLP, RBF and the bumptree neural network on a wide range of natural and artificial problems. Particular attention will be given not only to performance but also to the computational requirements of each algorithm, for example, the computational time required for training.

The thesis is organised as follows:

Chapter 2 presents a general review of architectural issues that impact on the performance of the MLP. Particular attention is paid to reviewing the current literature concerning the development of dynamic network design strategies for attaining the optimal architecture for the MLP. This focuses primarily on the pruning and constructive algorithms that have been introduced. In addition, combined algorithms including both a constructive and a pruning component, and the use of genetic algorithms to optimise the structure of the MLP are examined.

Chapter 3 is concerned with examining neural networks that implement a more local solution than the MLP. It presents a general overview of the approach adopted by the RBF neural network, in addition to which it introduces the idea of employing the tree based structures introduced in the decision trees of machine learning to develop neural networks that implement a local solution. This chapter also introduces the bumptree neural network, a neural network that employs a tree based structure.

Chapter 4 discusses the architectural issues that need to be resolved if the bumptree neural network is to be employed satisfactorily. Consideration is given to how the functions employed to partition the problem space should be defined; a number of alternate techniques are examined.

Chapter 5 examines further architectural issues that exist with the bumptree neural network. For example, the nature of the learning algorithm to be applied to the partitioned areas, and the use to which the tree structure is to be put when calculating the output of the network.

Chapter 6 presents a comparative study of the performance of the MLP, RBF and bumptree neural networks on a wide range of problems. The performance of these different networks is compared in terms of their generalisation performance, the time taken to train to a solution, and the time taken by the trained system to produce a response to a query.

Chapter 7 summarises the work conducted and discusses the major achievements of this work and possible future extensions.

Chapter 2

An Examination of Architectural Issues Associated With Multi-Layer Perceptrons

2.1 Introduction

Multi-Layer Perceptrons (MLP's) that have been trained with the back-propagation learning algorithm (Rumelhart, Hinton and Williams 1988) have been widely used in the development of artificial neural networks. They have been employed in such diverse fields as medical diagnosis (Astion and Wilding 1992), hand-written text recognition (Le Cun *et al.* 1990) and phonetic classification and recognition (Leung *et al.* 1989, 1991). A major reason for their popularity is that whilst some problems are more efficiently modelled by other more specialised networks, such as radial basis function networks or binary tree structures, the multi-layer perceptron is a good general learning tool for a wide range of applications.

2.2 The Limitations of the Standard Multi-layer Perceptron

The MLP trained with the standard back-propagation learning algorithm, whilst being a good general learning tool, possesses some inherent limitations. Firstly the learning algorithm is computationally demanding and slow since it employs an iterative gradient descent method. Secondly it is not guaranteed that the MLP will converge to an adequate solution even when one exists, and thirdly the performance of the MLP is dependent upon a number of arbitrary parameters. These arbitrary parameters need to be adequately resolved if optimal performance is to be produced by the network. In particular, the number of hidden units required to produce optimal generalisation performance needs to be determined. If too few hidden units are employed then the

network will fail to achieve satisfactory performance, whilst if too many hidden units are used then the network will tend to "memorise" the patterns in the training set and consequently give poor performance for patterns not included in the training set (Mozer and Smolensky 1989). The best generalisation performance is obtained by trading the training error against network complexity (Le Cun, Denker and Solla 1990). That is, whilst it may be the case that a network with a large number of hidden units may be able to reach a low error on the training set, the network complexity required to reach this level of error prohibits good generalisation performance. Baum and Hausler (1989) claim that the best generalisation performance is attained from a network containing a minimal number of hidden units. The main principle here is that a smaller network is more likely to generalise well because it has extracted the essential and significant features of the data.

The issues of computational expense and learning time are also linked to the number of hidden units employed by the MLP. Learning time is generally quicker in terms of the number of iterations taken to reach a solution when more hidden units are employed (Mozer and Smolensky 1989), although the computational expense of each iteration in the learning process is increased because of the larger number of units; the saving in computational effort is, therefore, reduced by the increased computational cost of each iteration. Hence, not only is it important to attain a minimal, or near minimal, sized network in order to attain good generalisation, it is also necessary if the computational cost of the learning process is to be optimised.

A study on network architecture by Baum and Hassler (1989) indicated that the number of training patterns required to obtain good generalisation increased when the number of hidden units increased. Widrow (1987) suggested that the number of patterns required to reach a 90% accuracy level was about ten times the number of weights in the network.

Hence according to this a small 3-n-1 network that employs five hidden units (20 connections) requires 200 training patterns if a 90% level of accuracy is to be achieved. If fifty hidden units are employed (300 connections) then 3000 patterns are required to train to the same level of accuracy, an increase of 1500%. Increasing the number of patterns upon which the network is to be trained clearly imposes further computational demands. Although Widrow's ten percent rule may not have universal application, clearly large networks impose considerable demands on the size of training set required to achieve a meaningful network performance.

Therefore, the number of hidden units employed has a significant impact on the performance of the MLP. However, in order to attain optimum performance there are other issues that need to be considered. In particular, the number of layers of hidden units needs to be determined. The standard MLP employs a single layer of hidden units, and Lippmann (1987) demonstrated that an MLP with a single hidden layer can implement arbitrary convex decision boundaries. Further, Cybenko (1989) has shown that a network with a single hidden layer can form an arbitrarily close approximation to any continuous non-linear mapping. These results do not, however, imply that there is no benefit to having more than a single hidden layer. For some problems a small 2 hidden layer network can be used where a single hidden layer network would require an infinite number of nodes (Chester 1990). It has also been shown that there are problems which require an exponential number of nodes in a single hidden layer network that can be implemented with a polynomial number of nodes in a 2 hidden layer network (Hajnal *et al.* 1987). The use of multiple layers of hidden units, while offering some potential benefits, does not, however, diminish the problem of determining the appropriate number of hidden units. It simply extends this problem from one to multiple layers. The use of multiple layers of hidden units and its impact on network performance needs close examination. In addition, approaches for determining the number of units to be employed

in each of the hidden layers need examining. However, these issues do not fall within the scope of this study. Instead this study uses MLP's with a single hidden layer, and the focus of attention is with determining the number of units to be employed in the single hidden layer. The impact on network performance of varying the number of hidden units is examined in later chapters.

Another additional parameter that needs to be altered to suit the problem being modelled is the learning rate. This is a parameter in the learning algorithm that plays a role in determining the size of the alterations made to the weights in the network at each weight change. In addition to the learning rate, the momentum term needs to be altered to suit the problem being modelled. This parameter in the learning algorithm plays a role in determining the degree to which the present weight change is effected by the previous weight change. The starting weight and bias values for the connections within the network also need to be determined, and these can be set to any value deemed appropriate. The standard MLP employs feed forward connections that do not by-pass layers. However, it is possible to employ connections within the network that feedback rather than forward, and feed forward connections that by-pass layers. The use of these alternate connections and their impact on network performance is not examined in this study.

Therefore, in order for the network to achieve optimal performance it is necessary not only to adjust the number of hidden unit's employed to solve each problem, it is also necessary to examine the parameters identified above. The impact of each of these parameters cannot be determined without further examination, and this further examination does not fall within the scope of this study. The architectural issue concerning the MLP that this work concentrates on is the impact on network

performance of the number of hidden units, and approaches for achieving the optimal number will be examined.

2.3 Dynamic Network Design Strategies

It is not possible to determine the optimal architecture of a network unless long and protracted empirical studies are conducted in which network performance is related to architecture. The number of hidden units providing optimal performance on a problem cannot be determined prior to the commencement of the training process. However, some attempts have been made to determine the optimal architecture during the learning process. The aim of the dynamic network design strategies is to define a network architecture capable of providing good generalisation performance, and it is commonly held that this in turn implies the attainment of a minimal sized network. Four different approaches have been proposed that attempt to dynamically determine the optimal number of hidden units in the multi-layer perceptron. These four approaches are constructive strategies, pruning strategies, combined constructive and pruning strategies, and genetic algorithm based strategies. Of these approaches, the constructive and pruning approaches have received the most attention to date.

2.3.1 Pruning Algorithms

The underlying principle of the pruning approach is to commence the training process with a large number of hidden units and then systematically reduce the number of hidden units and/or connections either during training or at the conclusion of training. A number of different algorithms have been studied, and the majority of these concentrate on the

removal of hidden units from the network. However, algorithms have also been studied that prune connections, in addition to some that prune both connections and hidden units. Pruning can either be carried out as an integral part of the training process or can be implemented at the completion of the training process. In the later case the algorithms that carry out the pruning operation employ an external monitoring system, whilst those that carry out the pruning during the training process modify the standard learning algorithm accordingly.

Algorithms that employ a global monitoring system carry out the removal of units and/or connections from the network through the use of a monitoring system which is applied at the termination of the learning process, when the network has been trained to the desired level of performance. In some cases (Sietsma 1990) a certain degree of retraining is necessary once the pruning operation has been completed, if an acceptable performance of the reduced network is to be achieved. In general when a pruning algorithm employs a global monitoring system the back-propagation learning algorithm is unchanged.

Sietsma (1990) investigated a pruning algorithm that used an external monitoring system to remove redundant units together with a modified version of the back-propagation learning algorithm to train the initial network. Sietsma was concerned particularly with investigating the effect on generalisation performance of removing redundant units from the network. She defined two different classes of redundant units that could be profitably removed from the network. Firstly, non-contributory units were identified: these had an approximately constant output across the training set or had their output mimicked across the output of another unit. Secondly, unnecessary information units were identified: these provided information to the next layer which was redundant as far as the classification process was concerned. Sietsma applied a pruning algorithm based on the removal of these two types of redundant unit from a network that had been trained with

a slightly modified back-propagation learning algorithm. Sietsma modified the learning algorithm by using a decay term which she claimed did not affect convergence but did reduce the average size of the weights. The decay would also appear to limit the further reduction of the error level when the required weight changes became sufficiently small, since these small changes were eradicated by the decay term. In addition, Sietsma also induced noise into the training process, which she claimed led to better generalisation performance. The data set she employed to test this pruning algorithm consisted of a series of sine waves of different frequencies.

Sietsma investigated the effect on network performance of removing redundant units (Sietsma 1990, Sietsma and Dow 1991) using the two stage pruning process discussed above. The first stage removed the non-contributory units whilst the second stage removed the unnecessary information units. In a typical set of results Sietsma found that the two stage pruning process gave a reduced network structure of (64)-9-3 with a generalisation performance which was superior to that given by the original network (64)-20-8-3. However, she found that the smallest networks determined by the pruning algorithm did not always give the best generalisation performance. In one particular study a (64)-8-3 network gave a poorer performance on the generalisation set than a (64)-9-3 network. It would seem that the pruning algorithm occasionally removed meaningful hidden units from the network thus leading to a reduction in network performance. Sietsma argues that the slightly larger networks performed better because of the addition of noise to the data set during the training process. In Sietsma's approach the network was retrained after removing the unnecessary information units, so that the remaining hidden units could then be retrained to perform the task previously carried out by the units removed. This additional training took far less time than the original training process and the weights did not usually change greatly. Sietsma found that when the optimal architecture was taken as the initial architecture, (64)-9-3, the network failed to

converge; she concluded from this that the number of initial hidden units should exceed the optimal number for convergence to be attained.

Pruning algorithms that employ an external monitoring system have the attraction that there is no need to alter the learning algorithm. Training proceeds with a prescribed number of hidden units until it terminates, when any extraneous units and/or connections are removed from the network. The main problem with this approach, as with any other type of pruning, is to decide which units and/or connections should be removed from the network, whether any additional training is required by the reduced network, and what is to happen to the weighted values of a unit when it is removed from the network. They can simply be discarded or can somehow be redistributed around the remaining units. In Sietsma's study (1990) the output weights from the units that were removed from the network after stage one pruning were redistributed evenly to the remaining units, whilst the weights from the input to hidden units were discarded. In addition, all the weights from the units deleted by the stage two pruning process were discarded. Another important issue that needs to be considered is whether retraining should be carried out once the external monitoring system has removed superfluous units. The use of an external monitoring system clearly increases the computational cost of training a network, but offers the possibility of improved generalisation performance.

For pruning algorithms invoked during the training process Wynne-Jones (1991) has distinguished two different approaches. In the first approach the final architecture is attained through the minimisation of a biased cost function in the learning process. In the second approach the final architecture is determined by the removal of units and/or connections during training according to the relevance of the unit or weight. The relevance concerns the degree to which the unit or weight contributes to the reduction of

the error level in the network. The aim of both of these approaches is the same - the removal of non-essential hidden units.

Chauvin (1989) developed an algorithm to arrive at the optimum network architecture through the minimisation of a biased cost function. Optimality was defined by Chauvin as the minimisation of a function of the "energy" spent by the hidden units in the network to solve the given problem. Hidden units that did not significantly contribute towards the minimisation of the error were removed from the network. Chauvin considered a hidden unit to be unused and available for pruning

> '. . . . when its activation over the entire range of patterns contributes little to the activation's of the output units.'
>
> (Chauvin, 1989, P524)

The technique employed by Chauvin caused redundant hidden units in the network to decay, by decreasing an energy term written as a function of the sum of the activation squared of each hidden unit. The standard back-propagation learning algorithm is a gradient descent on the cost function shown in equation (2.1):

$$C = \sum_{j}^{p} \sum_{i}^{o} (t_{ij} - O_{ij})^2 \qquad (2.1)$$

where t is the desired output of an output unit, O the actual output, and the sum is taken over the set of output units, o, for the set of training patterns, p. Chauvin modified this cost function to that shown in equation (2.2):

$$C = \mu_{er} \sum_{j}^{p} \sum_{i}^{o} (t_{ij} - O_{ij})^2 + \mu_{en} \sum_{j}^{p} \sum_{i}^{H} e(O_{ij}^2) \qquad (2.2)$$

where the sum of the second term is taken over a set or subset of the hidden units H and e is a monotonic function. The first term in the cost function is the error term; the second the energy term. The theoretical minimum of this function is found when the desired activation is equal to the observed activation for all output units and all presented patterns, and when the hidden units do not, in Chauvin's terminology, "spend any energy" in achieving this. Hence, Chauvin's approach aimed to remove any hidden units that did not make a significant contribution to the activation of the output units. These units, whilst present in the network, still have to be trained and their impact on the output units considered even though their contribution may be insignificant. The energy to which Chauvin referred could be regarded simply as the extent to which the activity of the hidden units fails to contribute to the minimisation of the error. The minimum of the function cannot be reached in practice, since the hidden units have to spend some energy to solve a given problem. The quantity of energy spent will in part be determined by the relative importance given to the error and energy terms during training. In principle, if a hidden unit has a constant activation for all patterns presented to the network then it contributes only to the energy term and will be suppressed by the algorithm.

The algorithm that Chauvin employed to arrive at the final network topology was simply the standard back-propagation algorithm with a different back-propagated term. The signal used to update the weights was the back-propagated signal from the previous layer augmented by the energy of the current hidden layer. A vital component of this algorithm proved to be the energy function employed, since Chauvin found the algorithm's performance to be very sensitive to the choice of function. The derivative of the energy function with respect to the squared activation/energy of the units is shown in equation (2.3):

$$e' = \frac{\partial e(o^2)}{\partial o_i^2} = \frac{1}{(1+o^2)^n} \tag{2.3}$$

where n is an integer that determines the precise shape of the energy function. When n=0, e=1 and high and low energy units are penalised equally. When n=1, e is a logarithmic function that penalises low energy units most severely. When n=2, the energy penalty reaches an asymptote, which Chauvin does not identify, and high and middle energy units receive the same penalty.

Chauvin tested the algorithm on the XOR problem, the 3 bit parity problem, the symmetry problem (Rumelhart, Hinton and Williams 1988), and on phonetic labelling tasks, where the input patterns consist of spectrograms corresponding to nine syllables. The results show the algorithm to be capable of producing an optimal or nearly optimal architecture for this wide variety of tasks. Chauvin also argues that because the algorithm imposes a constraint on the solution space, by constraining the energy spent by the units in the hidden layers of the network, the generalisation properties of the network are enhanced. Unfortunately, he has provided no theoretical or practical evidence to support this claim.

Hanson and Pratt (1989) also carried out work into the development of an algorithm to reduce the number of hidden units in the network through the minimisation of a biased cost function. The algorithm they developed caused non-essential hidden units to decay away through the use of an additional expression. There are a number of alternative ways in which the weights can be caused to decay, including the use of a cost function across the entire network, such as that adopted by Chauvin. The approach adopted by Hanson and Pratt considered each hidden unit's weight group separately. This has the potentially desirable effect of isolating weight changes to the weight group of each hidden unit which could then be used to eliminate hidden units from the network. Hanson and Pratt employed either the hyperbolic bias or the exponential bias in their modified learning algorithm, but there exist many other expressions that could have been employed by the

learning algorithm. Hanson and Pratt tested the performance of their pruning algorithm on a speech recognition problem, XOR and 4-bit parity. On the parity and XOR problems the number of hidden units tended to decrease towards the minimum required to solve the problem, although there was a decrease in the rate of convergence. In the speech recognition problem the number of hidden units tended to decrease, while the performance improved when the pruning algorithm was applied. Hanson and Pratt introduced an interesting concept in the way that they focus attention on the weights for particular hidden units as opposed to focusing attention on the network as a whole in their modified learning algorithm. They provide encouraging results, but further examination on problems that allow a more in-depth analysis of generalisation performance is required before the technique can be fully evaluated.

Mozer and Smolensky (1989) developed an algorithm to arrive at the final network architecture by removing units and/or weights from the network based on a measure of the relevance of a unit or weight. This algorithm typifies the general approach whereby an attempt is made to define the significance of each unit and/or connection so that meaningful pruning can occur. The prime difference between this approach and that of minimising a biased cost function is that here a metric directly calculates the relevance of the units and/or connections at various times during the training and simply removes the irrelevant ones from the network. This can be contrasted to the use of a biased cost function which simply causes weights to decay during training.

The most important aspect of Mozer and Smolensky's algorithm is the relevance metric that they employed. They regard the ideal relevance metric as being that shown in equation (2.4):

$$\text{Relevance}_i = E_{\text{without unit } i} - E_{\text{with unit } i} \tag{2.4}$$

However, they argued that it was not feasible to employ this relevance metric because of the impact that it would have on the training time of the network, and they attempted instead to approximate this figure by carrying out a single pass through the network with the linear error function shown in equation (2.5):

$$E = \sum_{p} \sum_{j} |t_{pj} - O_{pj}| \tag{2.5}$$

Since the derivative of this function is independent of the difference between the output activity and the target output it does not reach zero as the error decreases; this issue caused a problem for Mozer and Smolensky when they employed the standard quadratic error function.

Mozer and Smolensky tested the performance of their algorithm on a variety of problems: the cue salience problem, the rule-plus-exception problem, the train problem, the four bit multiplexor problem, and the random mapping problem. The results obtained show that the algorithm could calculate the relevance of units with some degree of accuracy. For example, the performance of the network was improved from the 41% achieved by a random removal of units to 81% with the removal of units based on their relevance metric.

Le Cun, Denker and Solla (1990) also developed an algorithm for removing units from the network. This algorithm, entitled optimal brain damage (OBD), attempted to improve the performance of the network through the removal of unimportant weights. Like Mozer and Smolensky (1989) they employed a relevance measure to determine which units should be removed. OBD deletes weights with what Le Cun, Denker and Solla termed small "saliency". That is, the weights considered to have the least effect on the training error. The assumption made was that small-magnitude parameters would

have the least saliency and so could be sequentially removed. Once these small-magnitude parameters had been removed a degree of retraining then took place. This process was then repeated. They proposed a theoretically justified saliency measure rather than relating the magnitude of a weight to its saliency. The technique to achieve this made use of the second derivative of the error function with respect to the parameters. The assumption here was that

> '. . . . objective functions play a central role . . . .; therefore it is more than reasonable to define the saliency of a parameter to be the change in the objective function caused by deleting that parameter'
>
> (Le Cun, Denker & Solla, 1990, P600)

Le Cun, Denker and Solla (1990) showed that it was possible to construct a local model of the error function and analytically predict the effect of removing a parameter without actually doing so. This local model of the error function was constructed by using the second derivative for each parameter with respect to the error; the second derivative was calculated by the procedure defined by Le Cun (1987).

The OBD procedure was a six stage procedure that made use of the second derivative for each parameter in relation to the error function in order to determine which units should be removed from the network. The six steps are shown in figure 2.1. When a parameter was deleted its value was set to zero.

1. Choose a reasonable network architecture.
2. Train the network until a reasonable solution is obtained.
3. Compute the second derivatives for each parameter.
4. Compute the saliency's for each parameter.
5. Sort the parameters by saliency and delete some low-saliency parameters.
6. Iterate to step 2.

Figure 2.1: The OBD procedure (Le Cun, Denker & Solla 1990).

The OBD algorithm was tested on a hand-written text recognition problem (Le Cun *et al.* 1990), and the early results were promising with the number of parameters in a practical network being reduced by a factor of four. This led to a significant increase in the speed of the network, and a slight increase in recognition accuracy. However, a point that needs to be reinforced about the OBD procedure is that it was applied to a network already considered to be optimal for the problem in question. OBD was, therefore, able to make significant improvements to a network that already worked well, which is perhaps a more significant achievement than improving the performance of a sub optimal network. Whilst the use of the second derivative as the basis for pruning units gave good results in this limited study, it would be interesting to determine the performance of the technique for different problems and the computational cost involved.

In pruning algorithm's therefore, the general approach is that training commences with a large network and some measure of relevance is applied to the units and/or connections in order to arrive at a reduced network topology that might be expected to be optimal or near optimal for the problem in question. There are, however, a number of issues that need to be examined if a pruning algorithm is to be applied. The first and obviously most important one concerns the nature of the metric used to decide which units and/or connections should be removed from the network. Whether an external monitoring

system is used or a biased cost function is minimised the aim is the same, to reduce the architecture of the network to the optimal size. In order to achieve this it is necessary to remove units and/or connections which are redundant. Mozer and Smolensky (1989) showed how badly a network performed when random units were removed, thus illustrating the possibility of pruning giving a degraded network performance. In addition, there are a number of associated issues that need to be resolved. For example, when a unit is removed from a network, should its connection weights be redistributed to the remaining connections ? Should any degree of retraining be carried out once the pruning operation has been implemented ? These questions need to be resolved in order to implement a pruning algorithm. In addition, the computational complexity and cost involved in training a network is also of considerable importance. Clearly the number of calculations required at each stage of the learning process will be dictated by the size of the given network. In addition, the use of a pruning algorithm adds to the training time since it is additional to the standard learning algorithm.

2.3.2 Constructive Algorithms

The underlying principle of the constructive approach is to commence the training process with a network that is too small for the given task and add hidden units until the network's performance is satisfactory. Important work has been carried out in this area by Honovar and Uhr (1988), Nadal (1989), Fahlman and Lebierre (1990), Frean (1990), Wynne-Jones (1992) and Ash (1989). The usual procedure for constructive algorithms is to add units into the single hidden layer of a multi-layer perceptron rather than creating multi-layered networks. An exception to this is Fahlman and Lebierre's algorithm, where each unit added to the network forms a single layer. There have been a number of

constructive algorithms described in the literature, but only the more important of these will now be described.

Dynamic node creation was the technique adopted by Ash (1989) to develop a network that was large enough to learn the mapping and as small as possible to generalise well. The constructive algorithm developed by Ash added new hidden units into a single hidden layer. When a new unit was added to the network, the weights of the new unit and those of the existing units were all subjected to the learning algorithm. Ash claimed that if only the new weights were adjusted then this would impose constraints on the network's performance. To determine when a new hidden unit was to be added to the network he made use of the fact that for a given learning rate with a given network size, the error tends to reach a plateau. Ash adopted the approach that when the error level was stationary, or decreasing very slowly, and the level of performance of the network was still poor, then a new unit should be added to the network. The flattening of the error curve such as that shown in figure 2.2 was detected by examining the ratio of the reduction in the squared error over the last x trials to the squared error when the last node was added. When the ratio fell below some threshold, that was problem dependent and user defined, then a new hidden unit was added to the network. This process continued until the error level on the training set was considered satisfactory. Figure 2.3 depicts the performance of a network containing an insufficient number of hidden units. The network's performance stagnates at a poor level since the network does not have sufficient degrees of freedom to model the problem.

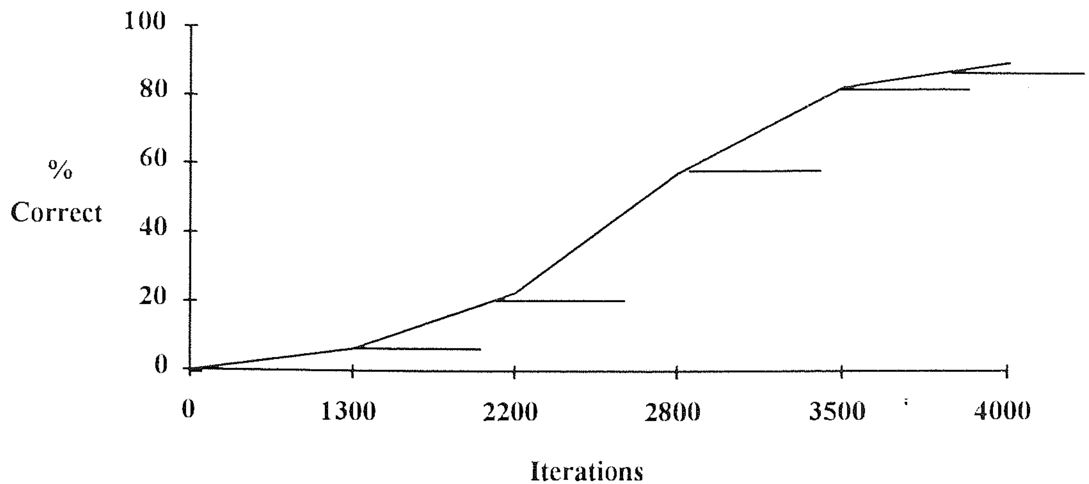

Figure 2.2: Percentage of training patterns correct versus number of iterations for a network that adds in hidden units at flat spots in the learning process (the addition of units is marked on the graph).

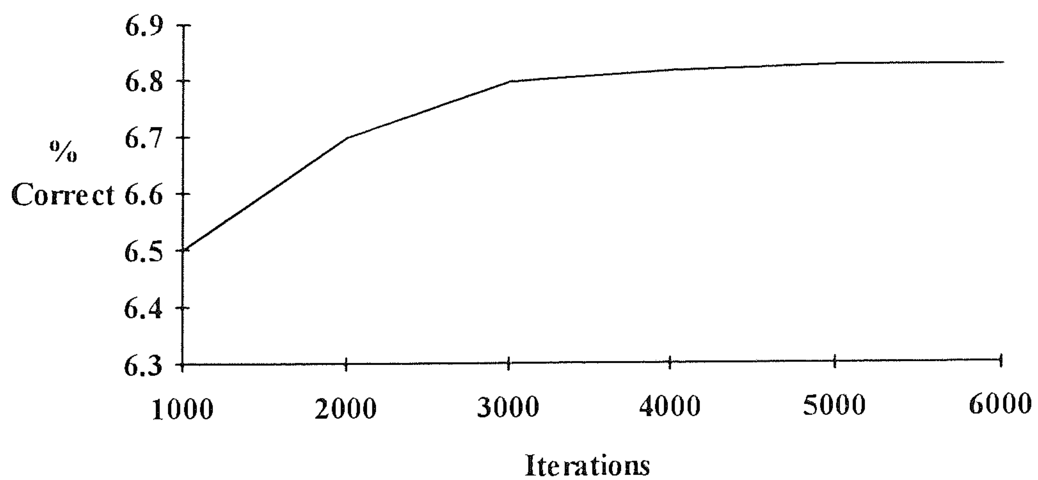

Figure 2.3: Percentage of test patterns correct versus the number of iterations for a network with insufficient hidden units.

Ash tested the dynamic node creation technique on a number of differing problems that included XOR, Symmetry (4) and Encoder (16). In most cases dynamic node creation was able to find a near-minimal solution with a computational expense that was

competitive with the standard back-propagation algorithm. The results reported by Ash are encouraging, but the algorithm has only been tested on small-scale problems which did not allow generalisation performance to be measured.

Wynne-Jones (1992) proposed a constructive algorithm that arrived at the optimal network structure through the application of a technique that split the units in the hidden layer when performance was deemed to be unsatisfactory. Training commenced with a small network and proceeded until there was no further improvement in performance on the training set. Once this point had been reached the network was increased in size by splitting the units in the hidden layer. The rationale being that the network possessed insufficient degrees of freedom to describe the given problem, and splitting the existing hidden units would increase these, hopefully leading to an improved performance. The units to be split were identified through the use of a principal component analysis on the oscillating weight vectors and also by the examination of the Hessian matrix of second derivatives of the error with respect to the weights. According to Wynne-Jones it is also possible, to apply the second derivative method to the input layer, where it provides a useful indication of the relative importance of the various inputs for the given classification task.

Applying the node splitting technique to a standard multi-layer perceptron is equivalent to introducing a hinge in the decision boundary so that recognition of higher dimensionality can be attained. However, the long range effects of decision boundaries can cause the new nodes to slip back to the original position of the old "parent" node, thus nullifying any performance gains. Wynne-Jones feels that the node-splitting technique whilst reasonably unsuccessful with the multi-layer perceptron may have greater success with networks using localised receptive fields such as radial basis functions.

Fahlman and Lebierre (1990) also developed a constructive algorithm, which, in contrast to the above studies, commenced the training process with no hidden units. Each unit when added to the network became a permanent feature-detector, available for producing outputs or for creating other feature detectors. Another difference in their approach was that the network was comprised of many layers where each layer consisted of one unit. The cascade-correlation learning algorithm arrived at the ultimate network size through a constructive procedure, and the use of the Quickprop algorithm (Fahlman 1988) to train the network allowed the training time to be substantially reduced from that attained with the standard back-propagation learning algorithm. Fahlman and Lebierre claim that any training algorithm for the multi-layer perceptron that allows all the weights to be updated concurrently will suffer from the moving target problem, in which each hidden unit tries to evolve simultaneously into a feature detector. The fact that all the units are changing together greatly complicates the task, since each unit is having to respond at each stage of the training process to differing contributions from the other units. Thus Fahlman and Lebierre claim that the time taken for a unit to acquire its proper role is increased by the varying contributions of the other units.

Cascade-correlation combated this problem by allowing only a few of the weights in the network to alter at any one time. More specifically, Cascade-correlation attempted to attain a suitable architecture through the addition of single hidden units to the network and the modification of a limited number of weights upon the introduction of each new unit. This had the effect of making each of the hidden units act as a particular feature detector. Once a unit had been added to the network and its training completed, the manner in which the weights of the units were fixed meant that the unit would always recognise the particular feature for which it had been trained. The view of Fahlman and Lebierre differed from that of Ash (1989) who argued that freezing existing weights would seriously limit the performance of the network.

To create a new unit in Cascade-correlation a candidate was first identified that received trainable input connections from all the external inputs to the network and from all the pre-existing hidden units. The output of this candidate unit was not initially connected to the rest of the network and hence did not affect network performance. A number of training passes were then made, which had the effect of allowing the candidate unit to reduce the error level of the existing network without disrupting it by the introduction of new untrained weights. When a candidate unit was introduced into the network its input weights were fixed so that the perturbation to the rest of the network caused by the unit's introduction would be minimal. An alternative approach to this was to employ a pool of candidate units and only accept the best into the network structure.

Fahlman and Lebierre make a number of claims for the cascade-correlation algorithm, the most significant of these being that it is able to build a small, though not necessarily optimal, network to solve a variety of problems. In addition, they claim that the algorithm is able to build deep networks, consisting of a number of hidden layers, without the dramatic increase in computational expense that is witnessed in back-propagation networks that contain multiple hidden layers. Thirdly, cascade-correlation is useful for incremental learning since the feature detection capability of each unit is not greatly altered when a new unit is introduced into the network. Fahlman and Lebierre's study is limited since they failed to consider the generalisation performance of the resulting network.

Therefore, constructive algorithms that approach the issue of dynamically constructing network architecture's that are suitable for given problems have adopted a multitude of differing solutions to the basic issues that need to be addressed. The primary issue to be addressed concerns when a new unit should be added to the network, and when the addition of units should cease. Another issue concerns the value to be allocated to the

weights of new units. In addition, should the existing units be trained alongside the new unit or have their weights fixed at current levels. It is also necessary to decide what period of time a new hidden unit is to be given to impact on the performance of the network before additional hidden units should be added. Finally, a decision when to terminate the learning process is required. Constructive algorithms must provide good answers to these questions if they are to attain an optimal network structure.

2.3.3 Combined Algorithms

Another approach to the problem of obtaining an optimal size network is to try and combine the constructive and pruning methods so that they operate in a co-operative manner. A combined algorithm will allow a reduced network to commence training with its size increasing until an acceptable performance is attained. Once this has been reached a pruning algorithm can then be applied in order to remove any redundant hidden units. In addition, the application of a pruning algorithm on the completion of training will allow the removal of any further redundant connections, thus reducing any extraneous degrees of freedom in the network, leading to an improvement in generalisation performance.

Chiu and Hines (1991) presented a combined algorithm that commenced training with a single hidden unit and added further units until an acceptable solution was reached, when the network was then reduced in size through pruning. The insertion of units into the network was fairly rudimentary, with a new hidden unit added after every two hundred and fifty iterations until convergence was attained. The pruning technique was based on the premise that connections of a small magnitude could be pruned from the network

since they contributed little to the forward units in the network. Chiu and Hines also removed units in the hidden layer when they had lost more than half their connections.

The approach taken by Chiu and Hines has a number of shortcomings that limit the value of the study. The general utility of their approach cannot be determined, nor can the generalisation performance of the network be evaluated since they restricted their study to a single problem, the well known XOR problem. In addition, the rationale for pruning the hidden units has little theoretical foundation; it is questionable whether a unit that has lost fifty percent of its connections is no longer relevant to the network simply because of this.

Hirose *et al* (1991) also developed a combined algorithm to determine network architecture. Training commenced with one hidden unit and units were added to the network whenever a local minima, which was identified by testing the reduction in error that occurred after every hundred iterations, was encountered. If the reduction in the error was less than one percent of the previous error, and the error was higher than some user defined value, then the network was considered to be trapped in a local minima and a further hidden unit added. Once the algorithm had converged below a specified error level then the reduction phase of the algorithm was entered. This consisted of removing the hidden units one at a time and testing whether with a little retraining the network could reach a solution. If a solution could not be achieved then the network was restored to its previous state and training was deemed complete.

The work by Hirose *et al* once again failed to address the generalisation performance of the final network. In addition, in their algorithm units were removed in the reverse order to their insertion into the network and this last-in-first-out ordering may not be the best

sequence for removal, since the utility of a unit may not be determined by its position in the insertion sequence.

Combined algorithms attempt to attain an optimum network architecture through the application of both the constructive and pruning approaches in a single algorithm. A combined approach allows the system to reverse the expansion or contraction of a network thus offering the possibility that an optimal architecture can be attained. The main drawback with the combined approach is that training time and computational complexity will be considerable since time will have to be spent in building the network and then pruning it. The development of these combined algorithms is still in its early stages and so further work needs to be carried out before any performance comparison can be made between the combined and separate approaches.

2.3.4 Genetic Algorithms Applied to Multi-layer Perceptrons

Genetic Algorithms are an optimisation technique, based on the mechanics of Darwinian evolution, first introduced by Holland (1975). They operate by searching multiple areas of the state space simultaneously. The parameters to be optimised are usually coded as a binary string, referred to as a chromosome, and initially the population consists of a predetermined number of members with randomly designated chromosomes. From these initial members evolve the later more "successful" members whose chromosomes are more suited to solving the given problem. The evolution process is comprised of four major stages which are as follows. First, the fitness of each member of the population is calculated by some measure that determines how close the member is to the solution. Second, a number of members of the current population are selected, with the fitter members having a better chance of being selected. Third, a new generation is formed

from the selected parents by the application of the genetic operators. The two main operators are crossover, which swaps sections of the binary strings of two parents selected from the population, and mutation which randomly inverts one or more bits of the binary string. Fourth, all of the offspring just created are introduced into the population, usually replacing the least-fit of the existing individuals to form the next generation. This four stage process is repeated for a predetermined number of generations or until a satisfactory solution is obtained.

Genetic algorithms have been employed to optimise multi-layer perceptrons both in terms of connection weights and biases and in terms of architecture. The use of a genetic algorithm to optimise the weights and biases of a Multi-layer perceptron (Whitley & Hanson 1989, Marshall & Harrison 1991) has not proved advantageous. Indeed the relative training speed of genetic algorithms means that they are an inherently inefficient method of training a multi-layer perceptron. There seems little or no benefit to be gained by using a genetic algorithm for this task (Williams 1992). Genetic algorithms have also been employed in relation to multi-layer perceptrons to optimise their architecture in terms of the network topology. The application of genetic algorithms to determine network architecture has been relatively successful where simple problems like XOR are concerned (Miller *et al.* 1989) and gave encouragement that for some problems genetic algorithms could be used profitably. However, the experiments on the simple problems have tended to use simple representations and these have failed to scale up to larger "real" problems. Attempts have been made to deal with real-life problems by employing complex, weakly specified, representations (Harp *et al.* 1989, Harp & Samad 1991). However, to date an ideal representation does not appear to have been found and the ability of genetic algorithms to produce adequate architecture's for multi-layer perceptrons remains to be demonstrated (Radcliffe 1991).

2.4 Summary

This chapter has examined the architectural issues that need to be considered in order to obtain optimal performance from an MLP. The issue that has been focused on in this study is that of identifying the number of hidden units to be employed to provide optimal performance on any given problem. This chapter has examined a number of alternative solutions that have been adopted to dynamically determine the size of the hidden layer of the network. It has reviewed work that has been carried out in this area, and whilst this review is not exhaustive it does discuss in some depth the major approaches that have been adopted to date. The majority of these have focused mainly on the development of constructive and pruning algorithms, but recently there has been increased research interest in combined algorithms. Genetic algorithms have also been employed to optimise network architecture, but to date with variable results. The aim of all these algorithms is to attain an architecture that is optimal, or near optimal, in terms of network performance. The work has focused largely on reducing the size of a network because it is a widely held belief that the generalisation performance of the MLP is thought to improve with small networks.

The main alternative to employing a dynamic network design strategy is to determine the optimal network size through long and protracted empirical studies. In these studies the number of hidden units are varied and the effect on performance recorded. Since this study is not aiming to provide an exhaustive study of the various dynamic network design strategies, it has been decided to address the issue of determining the number of hidden units empirically. Whilst this process is likely to take longer than using a dynamic network design strategy, it will nonetheless allow a standard MLP to be used in the comparative studies. The dynamic network design strategies discussed in this chapter reveal that the problem with determining the number of hidden units required is being

addressed with some success. However, the problem still exists. In addition, there are the other arbitrary parameters described above, and alternative network structures may be able to provide a solution that requires is less dependent on arbitrary parameters. Two alternatives, namely the RBF and tree based networks will be examined in future chapters.

In addition to the dynamic network design strategies, alternative approaches to dynamically altering the topology of the network to improve the generalisation of a multi-layer perceptron have been examined (Yu and Simmons 1990). However, the problem with these approaches is that the size of the network still needs to be defined at the commencement of training, a problem which exists whether standard back-propagation is employed or some variation of it (Yu and Simmons 1990, Fahlman 1988). Therefore, although techniques like that employed by Yu and Simmons produce promising results in terms of generalisation performance they do not solve the problem of determining the optimal network structure prior to the commencement of the training process.

Chapter 3

Alternate Classification Systems That Implement Local Solutions

3.1 Introduction

The development of alternative neural network structures to the MLP has been actively studied for a number of years. Most of the work has concentrated on developing networks which employ a more local partitioning of the problem space than the MLP, which adopts a "long range" approach using hyperplanes. Long range and local partitioning refers to the area of the problem space upon which each neuron impacts. When a local partitioning technique is employed each neuron impacts only on a subset of the patterns. Networks which impose a local partitioning of the problem space will, it is believed, through the emergence of "local experts" offer improved generalisation performance and a decreased training time.

A number of alternative classification systems to the MLP have been developed, but attention will focus in this study on the Radial Basis Function (RBF) network (Powell 1985; Moody and Darken 1988, 1989; Broomhead and Lowe 1988), and neural network systems based on ideas developed in decision trees (Brieman *et al.* 1984, Buntine 1991). Particular attention will be paid to the Bumptree structure that was first introduced by Omohundro (1991).

3.2 The Radial Basis Function Network

The RBF approach can develop networks for classification and function approximation and just like the MLP, in theory at least, can produce networks capable of forming an

arbitrarily close approximation to any continuous non-linear mapping (Poggio and Girosi 1989). RBF networks have been successfully applied to a variety of areas including speech recognition (Renals and Rohwer 1989) and financial forecasting (Lowe & Webb 1991). The results from these applications suggest that RBF's, or other techniques for implementing a more local solution, are worth further investigation as a possible alternative to the MLP.

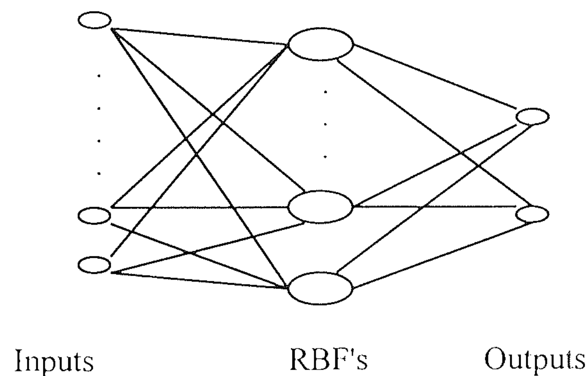

Figure 3.1: The RBF Network Structure.

An RBF network is a two layer network, depicted in figure 3.1, whose output nodes form a linear combination of the basis functions computed by the hidden layer nodes. The basis functions produce a localised response to input stimulus. That is, they produce a significant non-zero response only when the input falls within a small localised region of the input space. For this reason RBF networks are sometimes referred to as localised receptive fields. The hidden layer is composed of a series of basis functions, and although implementations vary, the most common basis function for this task is the gaussian response function, given in (3.1).

$$U_{1j} = \exp\left[- ((x-w_{1j})^T * (x-w_{1j}))/2\sigma^2_j \right] \quad j=1,2 \ldots N_1 \qquad (3.1)$$

where U_{1j} is the output of the jth node in the first layer, x is the input pattern, w_{1j} is the weight vector for the jth node in the first layer, that is, the centre of the Gaussian for node j, σ^2_j is the normalisation parameter for the jth node, and N_1 is the number of nodes in the first layer.

The node outputs range from 0 to 1, and the closer the input to the centre of the gaussian, the larger the response of the node. In addition to the gaussian response function, there are other functions that can be employed in the hidden layer of an RBF network. The main alternatives are the inverse multi-quadratic function, the multi-quadratic function, and the thin plate spline function. The inverse multi-quadratic function, like the gaussian, generates the highest outputs for patterns closest to the function centre. However, the multi-quadratic and the thin plate spline are non-localised functions, and as such their output increases as the input patterns move further away from their centre. Surprisingly localised networks formed using non-localised functions, are often found to perform better than those employing the more local functions; a point that will be demonstrated in chapter 6.

The output layer produces its output based on a weighted linear combination of the outputs from the basis functions in the hidden layer. Only the output of those functions that give a response to a pattern are taken into account at the output layer, with the impact on the network being proportional to the response of the function. Hence, if an input vector lies between the centre of two gaussian functions then the corresponding output of the network will be a weighted average of the output of the two active gaussian functions. The overall network therefore performs a non-linear transformation by forming a linear combination of the non-linear basis functions (Powell 1985).

There are a variety of approaches to learning in the RBF network, and most of these approach the problem in two stages. The first stage is involved with learning in the hidden layer, and this generally employs an unsupervised learning algorithm. The second stage is concerned with learning in the output layer, and this generally uses a supervised learning algorithm that attempts to minimise the mean squared error. Once an initial solution is found using this two stage approach, a supervised learning algorithm is sometimes applied to both layers simultaneously to fine tune the parameters of the network.

The unsupervised part of the learning process attempts to determine the centres of the basis functions and their radii. A number of methods can be employed, but the chosen method must ensure that the distribution of centres in the problem space is similar to that obtained in the training data. Hence, it must be assumed that the training data is representative of the problem, if good generalisation performance is to be attained. One technique for choosing centres is to place them at various points across the space covered by the data. An alternative to this is to select actual points from the data set as the centres. Another approach is to employ a clustering technique, such as the K-means clustering algorithm, to position the centres. Whichever of these techniques is chosen the number of basis functions required and their radii need to be determined.

It is reasonably straightforward to define a gaussian function so that each function peaks when a particular training pattern or similar patterns are presented. However, this approach becomes cumbersome and not practically feasible when there is a large training set. Attention has therefore turned to techniques, such as K-means, which are capable of positioning functions without requiring one function per data point. For example, Moody and Darken (1989) used a modification of the K-means clustering algorithm to centre the

basis functions in a network. Positioning the basis functions is important in classification problems since:

> '. . . . class membership proves to be important in any clustering algorithm dealing with classification, because a sample of a specific class in the envelope of another class yields erroneous results'
>
> (P596, Musavi, 1992)

Therefore, the outcome of the clustering algorithm in separating the classes when positioning the basis functions influences the accuracy of the network and the extent to which the number of basis functions can be reduced, and consequently the computational expense involved in training the network.

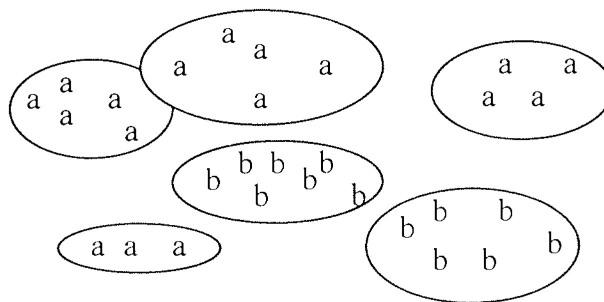

Figure 3.2 : A placement of basis functions that will encourage good performance. Patterns are classed as either a or b.

In positioning a basis function consideration must also be given to the width of the function. While it is not a problem when two functions concerned with the same class overlap, as they do in figure 3.2, it is imperative that functions concerned with two differing classes should be separated, since any overlapping will effect the accuracy of the network, minimising generalisation performance and degrading the recognition of local properties. Musavi (1992) has presented an iterative clustering algorithm that takes class membership into account. The aim is to achieve clusters that only contain points of the same class. The technique determines the radii of the functions by minimising the

overlap between the nearest neighbours of different classes, but it often fails to completely separate local densities of different classes. Figure 3.2 demonstrates a placement of centres that should permit a good level of performance, whilst the situation shown in figure 3.3 would give poor performance.

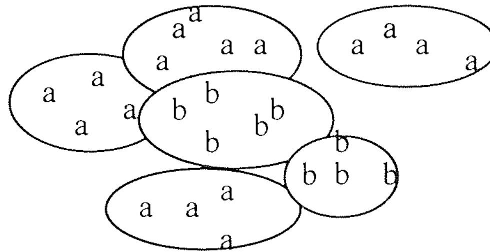

Figure 3.3 : A placement of basis functions giving poor performance. Patterns are classed as either a or b.

In addition to positioning the basis functions on the problem space it is necessary to train the output layer. This layer is usually trained after the centre and radii parameters of the basis functions in the hidden layer have been determined. The parameters of the basis functions are sometimes changed once the output layer has been trained and the performance of the network examined, although in these cases the output layer is still retrained after the hidden layer has been finalised. The output layer of the RBF network is trained to minimise the mean squared error of the network where the output of node (i) is calculated as (3.2),

$$y_{ip}^{(T)} = \sum_j W_{ij} y_{jp}^{(H)} \qquad (3.2)$$

where $y_{ip}(H)$ is the output of the jth radial basis function on the pth example, and wij is the weight from basis function j to output node i. The error of the network can be written as (3.3),

$$E(w) = 0.5 \sum_{ip} (\sum_j w_{ij} y_{jp}^{(H)} - D_{ip}^{(T)}) \qquad (3.3)$$

where $D_{ip}(^{T})$ is the desired output for output node i on pattern p. This error has its minimum at the point when the derivative given in (3.4) vanishes.

$$\frac{dE}{dw_{kl}} = \sum_{j}\sum_{p} y_{jp}^{(H)} y_{ip}^{(H)} - \sum_{p} D_{kp}^{(T)} y_{ip}^{H} \tag{3.4}$$

Let R be the correlation matrix of the basis function outputs (3.5).

$$R_{ij} = \sum_{p} y_{jp}^{(H)} y_{ip}^{(H)} \tag{3.5}$$

The weight matrix w* which minimises E lies where the gradient vanishes (3.6),

$$w_{ij} = \sum_{k}\sum_{p} y_{ip}^{(T)} y_{kp}^{(H)} (R^{-1})_{kj} \tag{3.6}$$

Thus, the problem is solved by inverting the square H*H matrix R, where H is the number of basis functions. The matrix inversion can be accomplished by standard techniques, or by a singular value decomposition method, such as the Gauss Jordan technique, if there is a danger of encountering a singular matrix. An alternative approach to that given above has been adopted by Broomhead and Lowe (1988) which focuses on the linear system embedded in the error formula itself. The procedure for training the output layer of the RBF is, therefore, not an iterative procedure in the tradition of back propagation, and consequently for large data sets training can be several orders of magnitude quicker than back propagation.

Therefore, one of the major advantages of the RBF network is that learning tends to be much faster than with an MLP. One of the main reasons for this is that the training algorithm is not an iterative procedure like the backpropagation learning algorithm. Another main reason for the faster training times of the RBF is that the learning algorithm is broken into two stages, and the algorithms used in both stages can be made relatively

efficient. The first stage, involves finding an optimal number of functions and positioning them on the problem space. Once the basis functions are in position, and the hidden layer parameters fixed, learning in the output layer takes place. In addition to the advantage in terms of training speed, the generalisation performance of RBF networks is comparable to that of MLP's (Musavi 1992; Chen, Cowan and Grant 1991). However, the response time to queries is slower. In addition to the RBF, attention has also turned to alternative techniques that produce a localised solution. In particular the decision tree algorithms of machine learning (Omohundro 1987, 1990, 1991; Gentric and Withagen 1993) have come to the fore. These algorithms use splitting techniques in an attempt to build tree structures that accurately model the data.

3.3 Decision Trees (Machine Learning Algorithms)

Machine learning has been an active area of research in artificial intelligence for many years. The majority of this work has been carried out into classification of patterns, and this is the field in which most success has been achieved. The basic problem with machine learning algorithms is to derive a function from samples of data, described by attributes and a class, that is able to predict the class of other samples from the same domain. The general scheme for studying the predictions made by any particular method is to present it with training samples to allow it to learn to map a problem. Subsequently some unseen test data can be presented to assess the ability of the method to classify unseen patterns. A simplistic method for comparing the performance of the methods is to examine the number of correct classifications each is able to make both on the training data and on the unseen test data.

Decision trees are part of a group of learning algorithms that have been developed in artificial intelligence which together with systems that employ if-then rules, have become known as machine learning algorithms. Decision trees have been an active area of research for many years, and have enjoyed a measure of success when applied to classification tasks. The early work on decision trees continued in parallel with work on the if-then decision rules that were embodied in the expert systems developed in the early 1970's. Indeed, it was found that decision trees could be used just as effectively as decision rules in expert systems. It was also recognised that the ability of decision trees to partition an input space into small areas of similar characteristics could be of use to the neural network community. Decision trees are not neural networks, they are based on the hierarchical data structures from computational geometry and employ a more direct representation of information. As with neural networks, the decision trees "learn" and can adapt themselves to different statistical distributions of inputs.

Decision trees can be used to represent a multiple-choice decision procedure which starts from the root of the tree and proceeds to one of the terminal nodes or leaves, with each choice corresponding to the value of an attribute and each leaf determining a class. Decision trees can be grown by a data driven approach, whereby a sample of the data is selected and a tree built to correctly classify this data. That is, the tree is partitioned into subsets determined by some particular attribute of the training data, and this process continues until the examples in each set have the same class or until there are no further attributes to consider. Each of the decision tree algorithms has its own approach to determine when the presentation of data should be terminated, and how the multiple-choice conditions are consequently altered. There exist a number of such algorithms, with the classification and regression tree (CART) (Brieman *et al.* 1984) probably the best known. Omohundro (1987) has introduced a number of alternative techniques that can be used to partition the input space, and which could be used to develop neural networks.

Decision trees are required to make a decision about which attribute of the data should provide the splitting criterion at each point where a new branch is to be grown. The aim is to arrive at the smallest possible tree consistent with correct classification. However, this is computationally expensive to achieve so most of the decision trees employ a heuristic to guide the search process. For example, a number of new splitting criterion can be applied to arrive at the best single split, classified as the one that causes the most patterns to be correctly classified. Most of the existing decision trees perform splits on the basis of a single attribute, and they divide the space up so that those items which are geometrically close to each other are placed in the same class. This is a local criterion for splitting, so it does not guarantee to produce an optimal tree. As a consequence of this, it has been necessary for decision tree algorithms to incorporate a technique for pruning the network to a more acceptable size. The two pruning approaches that have been adopted are: pre-pruning and post-pruning. Pre-pruning is a process depending upon a series of conditions that decide whether a node should be split, or whether it should be regarded as complete during training. Post-pruning removes various layers of the tree after the complete tree has been built and the data correctly classified. At the culmination of the building process various layers of the tree are removed.

CART is a binary decision tree algorithm developed by Brieman *et al.* (1984). By definition CART has exactly two branches at each internal node. The basic principle behind the construction approach adopted by CART, which has gained wide acceptance, is to select each split so that the descendant subsets of the data are "purer" than those of the parent. The term purity can be illustrated by the following example. Consider an initial set of 100 patterns, 50 from each class. If the patterns are split into groups of 40:5 and 10:45 then the purity is greatly increased. The ideal target is splits of 50:0 and 0:50. The pruning algorithm that CART employs (Brieman *et al.* 1984) is cost-complexity

pruning and is one of the most sophisticated adopted by a decision tree algorithm. Basically this algorithm approaches the task by dividing the data set into two groups, one for training and one for validation. The tree is built using only the training set, taking into account a measure that is concerned with balancing the likely error of the tree against its size. The ultimate tree structure is determined by building a number of different trees on the training set and then calculating the error for both the training set and the validation set. The tree that performs the best on both data sets is retained.

The idea of dividing up the problem space for the purpose of implementing a local solution based on the input of the patterns is central to decision tree algorithms, but is also embodied in the RBF network. As is the case with decision trees, the RBF network splits up the data based on its input, and once this has been done produces further splits based on the desired output of the patterns, until all patterns have been correctly classified. However, the RBF and decision tree systems use different techniques to classify the patterns once the splitting has been achieved. The techniques employed by the decision tree algorithms to divide up the problem space could provide very useful information for the development of local neural network solutions.

Omohundro (1987) has carried out a reasonably extensive study of the various techniques employed by decision tree algorithms for classifying data into appropriate groups. He concentrates on the types of splitting that can be employed in a decision tree, rather than focusing on the nature of the pruning algorithms or the actual criterion for determining when splitting should take place. Omohundro examined a number of alternative techniques for classifying data into appropriate groups. These are based initially on the input of the patterns and subsequently their desired output. Initially, Omohundro concentrated on one-dimensional data sets, and the most important structures found for classifying data into groups were binary tries, binary trees, and two level buckets. All

these techniques stored the data in what Omohundro termed bins, although they differed in the way they used the bins to partition the problem space. A two level bucket approaches the issue of partitioning the problem space by initially commencing with a single level of bins. The bins at the first level of the bucket are all of the same size. The problem space is then further partitioned by the addition of a second level of bins. These second level bins may vary in size depending on the density of data in the region. The bins are smaller in regions where there is a lot of data, meaning that the size of a bin inversely reflects the amount of data available in the area it is concerned with. Each bin produced from the same parent is of the same size. Figure 3.4 shows a possible structure of a two level bucket.

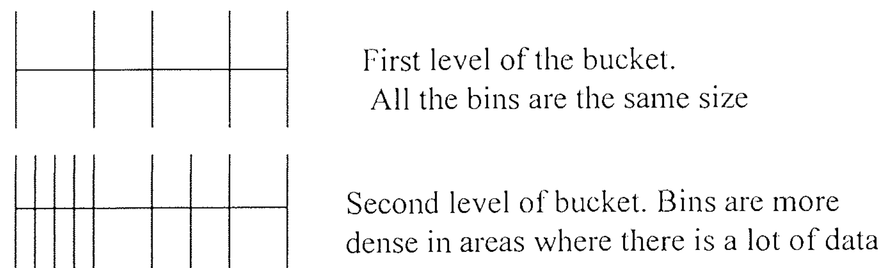

Figure 3.4: A two level bucket structure.

To adapt the bucket structure more closely to the input distribution, it is necessary to employ structures that can consist of more than two layers. Tries are one such structure, being a modified two layer bucket structure that, by decomposing the structure into an arbitrary number of bins at each level, can adapt the bins more closely to the requirements of the data . At each level the number of bins is the same. That is, if two bins are employed at the top level of the trie, as in figure 3.5, whenever further bins are added they will always be added in pairs. Densely populated regions may be split into many levels, whilst conversely, sparse regions may not be split beyond the first level. A

particular type of trie is the binary trie, shown in figure 3.5, where at each splitting point the area is divided into half. Any bin that is further split can be viewed as a parent of the nodes used to partition the area further. With a binary trie each parent bin that requires splitting will have two children, and both the children will be of the same size.

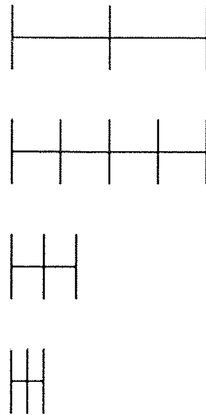

Figure 3.5: A binary trie structure.

Trees are another important structure for classifying data into appropriate groups that Omohundro examined. These allow the bins to be even more closely modelled to the data, by allowing each region to be split into different sizes. That is, the bins at any given point in the tree do not have to be of the same size, thus allowing greater flexibility for splitting the data. A particularly important type of tree is the binary tree shown in figure 3.6. The binary tree splits the problem space, or that part of it which the parent bin was concerned with, into two each time further bins are added to the structure. There is no limit to the number of times further bins can be added to the tree structure.

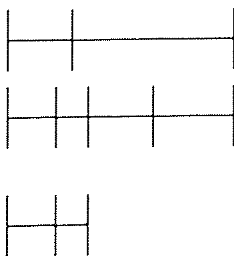

Figure 3.6: A binary tree structure.

Therefore, the techniques employed by the two layer bucket, binary trie, and binary tree differ, with each approach allowing different levels of freedom in partitioning the problem space. The two layer bucket is the most restricted of these approaches, only allowing two attempts at partitioning the problem space, and forcing the bins from any parent bin to all be of the same size. The trie allows more freedom by allowing more than two layers, thereby allowing areas where data is dense to be split more times than areas where the data is sparse. However, it is clear that the tree approach has the greatest freedom to model the data. Consequently it might be expected to be the preferred approach for decision trees. As a significant part of the RBF approach consisted in positioning functions on the problem space in such a way as to separate the data into discrete output classes, the approaches adopted by decision trees may also have a use with neural networks implementing a local solution.

In addition to the techniques for splitting one-dimensional data sets, Omohundro has discussed a number of techniques for use with multi-dimensional data. These are all basically extensions of the one-dimensional techniques, with the most important of these being grids, k-d tries and k-d trees. Grids are expansions of the two-layer buckets, and they divide the space into areas of constant size across the entire problem space. They uniformly partition each dimension exactly as buckets did with a one-dimensional problem space. In addition, it is possible to partition the different dimensions into a different number of areas. The k-d trie adopts the same policy for partitioning the data in multi-dimensional space as that adopted by tries operating in a single dimension. Indeed, at each split only one dimension is considered. The most important of the k-d tries is the binary k-d trie which splits each bin in half along a particular dimension. Perhaps the most important of the multi-dimensional structures, as was the case with the one-dimensional splitting techniques, is that employed by trees. Once again the binary tree is

of particular significance. The k-d tree embodies the approach outlined in the discussion of the one-dimensional trees, and again each split considers only a single dimension.

This discussion of the decision tree algorithms has revealed that a great deal of work has been involved in developing techniques for splitting data. The primary aim is to split the data into subgroups that are of the same class, based on the input values of the patterns, so that only patterns in the same class are grouped together in a node of the tree where an output value will be calculated; this is similar to the requirements of the RBF network. Hence, it appears that it may be possible to apply some of the techniques used in decision trees for building neural networks that implement a local solution. The fundamental idea that underlies the partitioning approach appears to be equally applicable to positioning RBF functions, and consequently justifies further investigation.

3.4 Neural Networks that Utilise a Tree Structure

The use of tree based structures in the field of neural networks has in recent years become an area of considerable interest. It is hoped that by utilising a tree structure it will be possible to develop neural networks that will be more local in nature, will require less time to construct, and will perform better than existing networks in terms of generalisation. Important work in this area has been carried out by Omohundro (1991), and Gentric and Withagen (1993), and these studies suggest that neural network systems employing tree structures offer important potential benefits.

3.4.1 The Bumptree

Omohundro (1991) has described a new class of geometric data structure termed the bumptree, which can be used for learning, representing and evaluating geometric relationships in a wide variety of contexts. Bumptrees are able to provide efficient access to a collection of functions on a Euclidean space of interest. They are a natural generalisation of several other geometric data structures, including oct-trees, k-d trees, balltrees (Omohundro 1987) and boxtrees (Omohundro 1989). The bumptree is able to partition the data based on multi-dimensions at each splitting point, unlike the k-d tree, k-d trie and other techniques discussed above which split the data on only a single dimension at each split.

The bumptree developed by Omohundro is a complete binary tree in which each leaf corresponds to a function of interest. Functions are associated with each internal node, subject to the constraint that the function for each interior node must be everywhere larger than each of the functions associated with the leaves beneath it. An important type of bumptree, and one which appears to offer considerable promise as the basis for a neural network classifier employs collections of gaussians to represent a multi-dimensional space (Bostock and Harget 1994; Williams *et al.* 1993, 1994). Omohundro has discussed a number of approaches for building a balltree structure (Omohundro 1989), and claims that each of these can be applied to building a bumptree. The most time efficient approach recursively splits the functions into two sets of almost the same size in a top down manner and is analogous to the basic k-d construction technique. The approach that Omohundro claims will give the best level of performance, but which is computationally slow, builds the tree bottom up, greedily deciding on the best pair of functions to join under a single node. Bumptrees may be used to efficiently support many

important queries, the simplest of which involves the presentation of a point in order to ascertain which functions are active.

The Bumptree approaches the task of mapping a problem space by building local models of the mapping in each region of the space using only data associated with the training samples that are nearest to that region. Each of the local models is based around a function, with each of the functions peaking in a particular region and steeply diminishing to zero outside the particular area of influence. The bumptree developed by Omohundro organises the local models so that only those that have a great influence on a query sample need to be evaluated. Omohundro makes use of the concept of a partition of unity in order to evaluate the performance of the bumptree when more than one function is active. The partition of unity involves normalising the output of each of the active gaussians so that their sum is equal to one. Therefore, the error of the full model is bounded by the errors of the local models and yet the full approximation is as smooth as the local bump functions.

Omohundro has applied the bumptree structure to the problem of learning to map the motion of a robot arm. He has compared its performance to that of an RBF network, and found that the RBF network achieves a smaller error than the bumptree but is more computationally demanding, being much slower than the bumptree to achieve a given error. In addition, retrieval time with an RBF network requires that the value of each basis function be computed on each query input and that these results be combined according to the best fit weight matrix. In contrast, bumptrees are not required to test the performance of every function, since decisions taken at the top level automatically prune the lower level functions. Hence, the bumptree has a quicker retrieval time than RBF networks. The performance of the bumptree approach on the robot arm movement mapping task has suggested that further analysis of the technique would be advantageous,

and this has been carried out in the present study and will be described in chapters 4,5 and 6. Thus, the objective is to develop a bumptree which gives a generalisation performance that is comparable to that attained by the RBF and MLP networks, whilst training to a solution faster than both the RBF and MLP, with a retrieval time at least comparable to the other networks.

3.4.2 The Constructive Tree RBF

In addition to the bumptree, a study has been made into combining a tree structure with a standard radial basis function network. Gentric and Withagen (1993) developed a constructive tree RBF (CTRBF) in an attempt to speed up the retrieval time of an RBF network. It was based on the premise that not all the basis functions are active at the same time. Indeed, in order to calculate the output of an RBF network it is necessary only to calculate the activity of the basis functions which are active for the given pattern. The aim of Gentric and Withagen was to employ a tree structure that would enable the active basis functions to be located with minimal calculation and computation time. The emphasis here was not to enhance the classification performance of the RBF, but rather to speed up the retrieval time.

The tree structure developed by Gentric and Withagen was a multi-level tree, consisting of a parent, or root node, connected to a number of nodes referred to as sons - each son node having a radius smaller than that of the parent node. If a node has more than one son then the son nodes are referred to as brothers, and each brother has an equal radius. The depth of the tree could be unlimited, but the criterion outlined above had to be adhered to each time a new level was added. If a node had no son then it was termed a leaf node, and if a node had no brothers it was referred to as a terminal node.

Gentric and Withagen discussed both a supervised and an unsupervised approach to building a CTRBF network, and both of these approaches commenced by having a single root node connected to a leaf for every basis function in the network. This structure was then developed into a tree through the use of either unsupervised or supervised training procedures. Both these procedures attempted to assign the centre of the basis functions, so that they could be enclosed by one of the nodes at a higher level in the tree. The centre in this instance actually related to the pattern of the data set that the function was created around. Hence, the design that Gentric and Withagen discussed commenced by having a node for every basis function in the network, and the original RBF network was constructed through the use of one of the standard methods discussed earlier. The building of an RBF network and the subsequent forming of the tree structure will mean that this procedure will increase the learning time of the network, although it is hoped with an accompanying reduction in retrieval time.

Once the tree has been built, the network's response to a given input was evaluated commencing at the root node. The initial step in the evaluation process was to compute the distance between the input pattern and the centre of the current node. If the distance was larger than the radius of the node then the younger brother was examined, otherwise the son was examined. When the search led to a leaf node then the corresponding basis function of the RBF network had its value calculated in order to produce an output from the network. The search could be extended so that more than one of the basis functions was used to calculate the final output of the network - and it was found that this led to an improvement in the performance level attained.

CTRBF has been applied to a hand-written character recognition task, and the results demonstrate that the performance in terms of correct classifications is comparable to that attained by the MLP and by a standard RBF network. In addition, the evaluation time of

CTRBF is comparable to that of the MLP but far superior to that of the RBF, whilst the training time of the CTRBF is quicker than the MLP, but slower than the standard RBF because the network is built prior to the development of the tree structure.

The technique developed by Gentric and Withagen is another approach that has embodied a tree based structure into a neural network classifier in the hope of improving the performance of the network. In this instance there appear to be a number of advantages over the MLP and RBF, but at the same time a number of drawbacks that need to be resolved in order to realise the full potential of the tree based structure. In particular, it seems limiting to build an RBF prior to constructing the tree structure. If the training time of the tree based approaches is to be comparable to that of the RBF then the tree structure will have to be created as an integral part of the training process. The approach by Omohundro allows a complete network to be built as a tree in a single process that is far quicker than either the MLP or the RBF, resulting in reduced training times. Thus, CTRBF is quicker than the MLP to train, and quicker than the RBF to evaluate a pattern.

3.5 Summary

This chapter has examined the work that has been carried out into the development of a neural network able to implement a solution in a more local manner than the MLP. It is hoped that such networks will provide better generalisation performance, faster learning times, and quicker evaluation times. The RBF, an example of such a network, has been shown to perform adequately in terms of generalisation and learning time. In our studies we have been inspired by the work of Omohundro to develop a network based on a local solution in an attempt to obtain a level of performance that is superior to that attained with RBF and MLP networks.

Chapter 4

Architectural Issues In Developing A Bumptree Neural Network

4.1 Introduction

The development of the MLP and the back propagation learning algorithm (Rumelhart, Hinton and Williams 1988) led to a resurgence of interest in the field of neural computing, after Minsky and Papert (1969) had, in their critique of Rosenblatt and his perceptron (Rosenblatt 1969), caused interest in the field to decline. The MLP and the back propagation learning algorithm together provide an effective neural network that is able to perform adequately across a wide range of problems. Whilst its performance has been adequate, much recent work has been devoted to developing artificial neural networks that are able to produce better performance. Work has been carried out in an attempt to achieve this through adjusting the MLP and its attendant learning algorithm. This work is examined in chapter 2. Work has also been carried out into the development of alternative networks to the MLP, specifically networks that adopt a differing approach to mapping the problem space. Alternative approaches that attempt to improve the performance attainable by artificial neural networks by employing a local solution were examined in chapter 3. This chapter will focus on the development of the bumptree neural network. In particular, techniques for partitioning the problem space and the effect these have on network performance will be examined.

Whilst the most widely examined of the approaches that employ a more local solution has been the RBF network, interest has increased in networks that explicitly employ some of the ideas embodied in decision trees. Omohundro (1991) introduced a structure that can be employed as a neural network architecture, whose origins lie in decision tree algorithms. Omohundro refers to this structure as a bumptree, and the combination of ideas from decision trees and neural networks that this structure represents is intriguing. However, in order to employ the bumptree structure as a

neural network there are a number of design issues that need to be resolved. This chapter will examine the issues that relate to the partitioning of the problem space, and their effect on the performance of the bumptree neural network. Performance on a number of problems ranging from traditional problems such as XOR and Parity (6) to skin cancer diagnosis, and vowel recognition will be considered.

4.2 Introduction to the Bumptree Neural Network

The bumptree is a new class of geometric data structure that was devised originally by Omohundro (1991) and shown to provide the basis for a neural network. It is a natural generalisation of several hierarchical data structures including oct-trees, k-d trees balltrees (Omohundro 1987) and boxtrees (Omohundro 1989). The bumptree neural network uses the bumptree data structure to partition the data, and then applies a learning algorithm to this partitioned data. The learning algorithm deals with each of the partitioned areas separately. Hence, the network approaches the task of mapping a problem space by building local models of the mapping in each region of the space, with each local model using only the data in the training set within the region in question. The data that is mapped in each of the local models is determined by the manner in which the bumptree data structure has partitioned the problem space.

The bumptree structure used to partition the problem space into local areas was examined in some detail in chapter 3 and shown to partition the data based on multiple dimensions at each splitting point. The bumptree structure developed by Omohundro is a complete binary tree in which each leaf corresponds to a function of interest, with each function in the tree being responsible for a larger area of the data set than each of the functions beneath it in the subtree. The type of function employed by the bumptree will have an impact on the manner in which the problem space is partitioned, and, as is

the case with RBF networks, a number of function types can be employed. For the work described here the author has used gaussian functions.

Developing a bumptree neural network involves three major issues. The first is the manner in which the functions are to be placed on the problem space, the dimensions of these functions and other issues that relate to the building of the bumptree. The second is concerned with calculating the output of the network. The third is the learning algorithm to be employed to model the data in each of the local areas once the partitioning of the problem space has been completed. The learning algorithm is invoked every time new functions are added to the network, and is executed once for each function added. The problem space is split into areas during the construction of the bumptree, and each of these areas is mapped individually by the learning algorithm. That is, each area is mapped independently of the other areas that exist. The solution reached in one area of the problem space will not impact on other areas of the problem space.

The issues of partitioning the problem space and employing a learning algorithm to model it are very closely interlinked in the training process. Initialisation of the training process involves placing two functions on the problem space and assigning each training pattern to one of them. The learning algorithm is then applied separately to each of these areas prior to the use of a performance measure to ascertain if the functions have been able to map the problem space adequately. If they have achieved this then training terminates, otherwise two further functions are added to the area of the data that has not been adequately mapped. These two functions further partition this area of the problem space before the learning algorithm is applied to the area again. This training process continues until an adequate mapping of the problem space is achieved.

The remainder of this chapter will be concerned with examining the various approaches that have been employed to partition the problem space. Chapter 5 will be concerned with examining other major issues in the development of a bumptree neural network, including the learning algorithm to be employed and the manner in which the output of the network is to be calculated.

4.3 Placement and Dimensions of Functions: An Introduction

The technique used to partition the problem space has a significant impact on the level of performance which can be achieved with the bumptree neural network. The task of partitioning the problem space is carried out initially by two functions that map the entire problem space. If these functions are unable to reach an adequate solution further functions are added to the network. The functions that have been employed by the author are gaussian functions, whose activation on each input dimension of a pattern is given by equation 4.1:

$$A_{fi} = \exp^{-(0.5*(((In_{pi}*c_{fi})/a_{fi})2))} * 1/(a_{fi} * \sqrt{3.14159*2}) \qquad 4.1$$

where A_{fi} = the activation of the function f on the pattern p for the ith input dimension, a_{fi} is the radius of the function f in dimension i, C_{fi} = the centre of the function f in dimension i, and In_{pi} = the ith dimension of the pth input vector. The activation of each of the functions on an entire input pattern is the product over all the input dimensions.

Two functions are placed on the problem space, and the patterns assigned to the function on which they are most active. A learning algorithm is then applied separately to the two functions, with each being trained on the training patterns assigned to them. A performance measure, based on the performance of the training set and on a

generalisation set is calculated to ascertain whether these functions have adequately modelled the problem space. If one, or both, of the functions fails to reach the required level of performance then further functions are added to the network and the above procedure repeated.

The bumptree employs a tree structure to add functions when the existing functions are unable to reach an acceptable level of performance. For instance, if function f1 models its part of the problem space satisfactorily then further function expansion in this part of the problem space is not required. However, if function f2 fails to model its part of the problem space satisfactorily then further expansion is necessary to adequately partition this part of the problem space. In this case two further functions f3 and f4 are derived from f2. Patterns will then be assigned to function f3 or f4 depending upon which gives the greater activity as calculated by equation 4.1. This process continues until effective partitioning of the problem space is achieved. The process leads to the type of tree structure shown in figure 4.1, where f1 is satisfactory but f2 is not and has consequently been expanded, as has f4. Thus a solution has been reached with functions f1, f3, f5 and f6. Figure 4.2 shows the division of the problem space given this sequence of events. Only functions f1, f3, f5 and f6 are shown, since functions f2 and f4 were found to be unsatisfactory.

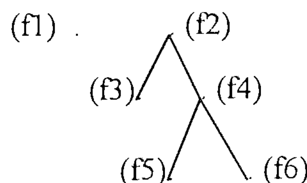

Figure 4.1 - The bumptree structure relating to the problem described in the text.

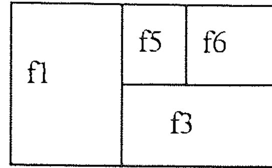

Figure 4.2 - A problem space divided by the functions in the bumptree neural network described in the text.

The bumptree neural network, therefore, approaches the task of modelling the problem space by dividing it up into smaller areas and solving each of these areas separately. Experimentation has revealed that the technique employed to split up the problem space has an impact on how well the bumptree neural network performs. This impact is to a degree problem dependent. The main issues to be considered when partitioning the problem space are how to determine the dimensions of the functions and when to terminate the addition of functions; this chapter will focus on the former issue and chapter 5 on the latter. To determine the dimensions of a function it is necessary to determine its radius in each dimension and its centre, which is described by one centre point in each input dimension.

Omohundro stipulated that every function, except the first level ones, must be wholly enclosed by its parent function. That is, the centre of the function and its radius in each of the dimensions must be totally enclosed by the parent function, as shown in figure 4.3. However, the author regards this criterion as being too restrictive, and has developed an alternative approach to constraining the dimensions of the functions within the bumptree neural network. The bumptree neural network is concerned with partitioning the problem space so that each level of the tree subdivides the area described by the level above it. As long as this constraint is adhered to, lower level functions can be placed anywhere and need not have their radii constrained, since the important thing is to allow the functions to be positioned at the point in the problem space that allows the best partitioning of the data. A number of different approaches for defining the functions have been studied and these will now be examined.

4.4 Omohundro's Approach

The first centre and radii defining technique implemented in this study adhered to Omohundro's criterion. At each splitting point within the tree the child function had its dimensions wholly enclosed within those of its parent function, as shown in figure 4.3. The technique adopted for positioning the initial parent functions on the problem space was to select random points within the area covered by the data, and to set the radii so that for every dimension the parent function fell within the area covered by the training set. Likewise, the child functions had their centres and radii chosen so that for every dimension they were constrained to the area of their parent function. This approach simply placed two functions at random points within the area of the parent function each time a function had to be split; in contrast to some of the alternative techniques examined below. The comparative results relating to the performance of this function centering and constraint approach, along with the other techniques that the author has developed are given in section 4.8.

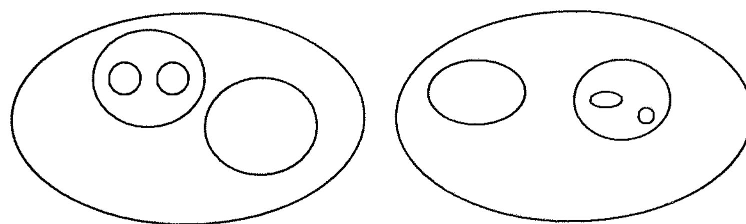

Figure 4.3 - Bump functions adhering to Omohundro's constraint.

4.5 Multiple Initial Functions (MIF)

What has been termed Omohundro's approach to function centering and constraint employed two functions when further functions were required. These were simply

placed at random points on the problem space. With Omohundro's approach the top level functions had their dimensions confined to within the area of the data set. In addition, the dimensions of each function below the top level were confined to the area of the associated parent function. It was felt that both the function centering and function constraint techniques employed by this approach could be improved upon, so alternative approaches were investigated. In a departure from Omohundro's approach a number of functions were defined initially so that the network might locate the new functions in more advantageous positions. The author decided to employ ten functions at each point where new functions were required by the network. The MIF approach could have employed any number of functions, and ten was a figure selected arbitrarily. Figure 4.4 further describes the MIF algorithm.

Ten sets of function centres in each dimension were selected, and these functions had their radii constrained in an appropriate manner. Once the functions dimensions had been determined the patterns were assigned to functions by equation 4.1. The learning algorithm was then applied to each of the functions active on any patterns, and an error level for the training set calculated. The next step involved the calculation of a "goodness" measure for each of the functions; calculated as shown in 4.2:

Calculated Error for Function / Number of patterns function active for (4.2)

The function with the best value was retained and the centres and radii of the other nine functions, or at least those that were active on any patterns, averaged out to create the second function. The patterns were then reassigned to the two remaining functions and the learning algorithm reapplied. A performance measure was applied in order to ascertain whether further functions were required. If they were the process was repeated. This procedure of employing multiple initial functions was combined with a number of different approaches for constraining the centre and radii of the functions.

```
For Functions = 1 to 10
        Assign function (Functions) a centerpoint in each dimension.
End For
For Patterns = 1 to NumPatterns
        For Functions = 1 to 10
                Calculate activity of function (Functions) on pattern (Patterns) using
                equation 4.1.
        End For
        Assign pattern  (Patterns) to the most active function.
End For
Apply the learning algorithm discussed in detail in chapter 5 to each function active on
any pattern, and calculate an error level for these. The aim of this learning algorithm is
to minimise the error of the functions on the patterns upon which they are active.
Calculate a goodness measure for each active function using equation 4.2.
Retain the function with the best goodness value and average out the remaining active
functions.
For Patterns = 1 to NumPatterns
        For Functions = 1 to 2
                Calculate activity of function (Functions) on pattern (Patterns) using
                equation 4.1.
        End For
        Assign pattern  (Patterns) to the most active function.
End For
Apply the learning algorithm discussed in detail in chapter 5 to each function active on
any pattern, and calculate an error level for these. The aim of this learning algorithm is
to minimise the error of the functions on the patterns upon which they are active.
```

Figure 4.4 - The MIF approach for adding functions to the network.

The first function centering and constraint technique that was developed using MIF was one that adhered quite closely to the technique discussed in section 4.4, except that it utilised the MIF approach when adding functions to the network. It was hoped that the addition of MIF to the technique discussed in section 4.4 would facilitate better placement of the functions on the problem space and lead to an improvement in results. Section 4.8 will discuss the performance attained.

The MIF approach for positioning functions on the problem space, outlined in figure 4.4, did not consider how the radii of functions should be determined. The constraint technique described in section 4.4 could be employed in conjunction with the MIF approach to provide a full function centering and constraint technique. However, the point was made above that the function constraint technique employed in Omohundro's approach could adversely affect the performance of the bumptree neural network, and that an alternative technique might produce better results. Constraining the lower level functions within the area of their parent function might be unnecessary. The important issue is that the lower level functions are constrained so that they are only ever active on patterns which their immediate parent dealt with. In addition, constraining the functions wholly inside their parent may cause problems at the lower levels of the tree. A function with a particularly small radius in one dimension in comparison to other functions is unlikely to attract patterns, since pattern assignment is calculated as a product across all dimensions.

The constraints on the radii of the functions were relaxed in stages, so that the effect of each relaxation could be determined. The first approach introduced a modified constraint rule in which the original top level functions were still constrained in each dimension to the area covered by the data set. However, the lower level functions were now constrained within each dimension so that they described the area of the problem space covered by those patterns for which their parent function was active, rather than the total area covered by their parent function. It was felt that introducing this technique would allow the lower level functions greater freedom in where they could be centred and in the value to be assigned to the radii in each dimension. The centre of each of the lower level functions had to fall for each dimension within the area of the problem space containing the patterns for which the parent function was active. The results of applying this constraint technique along with the MIF approach will be considered in section 4.8.

In the second approach the constraints employed by Omohundro on the size of the original parent functions were relaxed. The techniques outlined above constrain the initial functions so that their radius in each dimension falls within the area of the training set from where the centre is positioned. This constraint was relaxed by setting the radii of all the original functions to 1. The value of one was selected since the data presented to the bumptree neural network was always normalised to values between 0 and 1, and so a radius of 1 in each dimension meant that wherever a function was placed it could cover the entire problem space. The rationale here was to ensure that a function had a proper scope of responsibility in each dimension of the problem space. The functions below the first level still had their centre selected by the technique described above, and were still constrained so that each dimension fell within the area of the problem space assigned to their parent. The results of applying this technique along with the MIF approach will be examined in section 4.8.

The third approach relaxed the only remaining constraint of Omohundro's approach. That is, the restriction on the radii of the lower level functions was relaxed in order to truly test whether an unconfined approach to function centering and constraint would be beneficial. The approach assigned a value of 1 to each radius in each dimension for all the functions that comprised the network, and then centred the functions within the area covered by those training patterns upon which the parent function was active. The results from applying this technique, alongside the MIF approach will be examined in section 4.8.

4.6 The N-Function Bumptree

The bumptree that Omohundro discussed was a complete binary tree. However, he gave no theoretical basis for its binary nature, and the addition of multiple functions to the network every time a function was split offered a viable alternative to adding only

two functions at every split. Since the bumptree neural network has been employed to deal with problems containing multiple inputs, it was felt that the addition of more than 2 functions to the network at every splitting point might improve the performance of the network.

The n-function bumptree requires a means of determining the number of functions to be added to the network when function expansion is required. It was decided to relate the number of functions to the input dimensionality of the problem. That is, the number of functions to be added to the network should be equal to the number of input dimensions of the problem. For instance, the Iris problem has 4 input dimensions, and so 4 functions were added to the network each time function expansion was required. A possible bumptree that could be obtained is shown in figure 4.5. This criterion for determining the number of functions to be added to the network was selected because the assignment of patterns to functions is based on the input dimensions of the patterns. It was felt, therefore, that because the input dimensions of the problem play such a major role in assigning patterns to functions, allowing them to also determine the number of functions added at each expansion might result in improved performance.

It was hoped that this approach would enable a much finer partitioning of the data at each level of the tree giving improved performance with little increase in the retrieval time to answer a query. An alternative would be to base the number of functions added at each split on the number of output dimensions of the problem. This was rejected, however, because splitting is considered to be explicitly input driven.

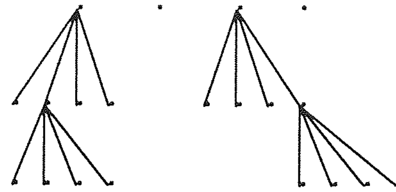

Figure 4.5 - An n-function bumptree to solve the Iris problem. There are four functions at each splitting point because the problem has four input dimensions.

The n-function bumptree has been implemented on a wide range of problems, and the results of applying this approach will be set out in section 4.8, where its performance will be compared to the other function centering and constraint techniques that have been implemented. In order to implement the n-function bumptree approach it is necessary to have a technique for centering and constraining the size of the functions. That is, the n-function approach to constructing a bumptree needs to be combined with an approach for determining the dimensions of the functions. The approach adopted for constraining the functions was to set the radii of all functions to one. In addition, the centre of each function in each dimension was constrained to fall within the area of the data set covered by their parent.

4.7 The Use of Non-Hierarchical Clustering Techniques

The approaches to function centering discussed in sections 4.4 and 4.6 are static techniques. That is, once the functions have been placed on the problem space they remain fixed in this position throughout the remainder of the training process. In addition, in the MIF technique the 10 initial functions once placed on the problem space remain fixed in position. The only movement of functions that is allowed to occur is when new functions are added to the network. When this occurs one of the new functions is obtained from the existing active functions by averaging the radii and centres of those functions considered to have inferior goodness measures. Once a function has been added to the network it is not allowed to be reassigned to a different

position in the problem space. This was considered to be a possible limitation on performance, and so a function centering technique that utilised a non-hierarchical clustering technique was developed. The effect of this on the performance of the bumptree was determined particularly with regard to its recognition capability, and time to train.

The central idea of the non-hierarchical clustering techniques is to initially partition the data into clusters, and then alter cluster membership so as to attain a better partitioning of the data according to some defined metric. The techniques begin with a series of initial points that divide up the problem space and then a sequence of moves is generated for each of these points until a satisfactory solution is reached. The principal aim of the non-hierarchical clustering technique is to position the functions so that the best partitioning of the problem space is achieved.

A number of different non-hierarchical clustering techniques have been described in the literature, with two of the most important being the Forgy method (Forgy 1965) and MacQueen's k-means method (MacQueen 1967); both of these methods were considered in the present study. Forgy suggests a very simple algorithm to partition a problem space, and the steps of this algorithm are set out in figure 4.6. When applied to the bumptree neural network, the algorithm first of all partitions the data by selecting a series of original functions and assigning each data point to the function with the highest level of activation (given by equation 4.1). Once all the patterns have been assigned to the functions, then those functions active on some patterns have their centre point in each dimension recalculated based on the current active patterns and their old centre points. The patterns are then reassigned to the function with the highest activation level and the process repeated. This iterative procedure continues until the movement of patterns between functions is below a defined threshold. After the functions have been centred, patterns are assigned and the learning algorithm applied as described above.

Step 1: Commence with any desired initial configuration, or partitioning, of the data.

Step 2: Allocate each data unit to the cluster with the nearest centre. The centre points are fixed for a full cycle through the entire data set.

Step 3: Compute new centre points for each of the clusters based on the data within the cluster and the old centre points.

Step 4: Execute steps 2 and 3 until the process converges; that is, continue until no data units change their cluster membership at step 2.

Figure 4.6 - The Forgy non-hierarchical clustering technique.

Another important non-hierarchical clustering technique is the k-means method developed by MacQueen (1967). MacQueen uses the term "k-means" to denote the process of assigning each data unit to the nearest cluster, out of k clusters. The steps of the k-means approach are set out in figure 4.7. There are several differences between this technique and that devised by Forgy, one of the main differences being that the centre of a cluster is computed on the basis of the cluster's current membership rather than its membership at the termination of clustering. Again in contrast to the Forgy method the centres of the initial functions are given by actual points from the data set rather than being randomly assigned. Also, the centre points of each of the functions is updated after each pattern is assigned, rather than after all the patterns in the training set have been assigned to one of the functions. In addition, the MacQueen's k-means technique takes only one pass through the data set to arrive at the final centres, whilst the Forgy technique may take numerous passes before the new function centres have been determined.

Step 1: Take the first k data units as clusters of one unit each.
Step 2: Assign each of the remaining data units to the cluster with the nearest centre. After each assignment recompute the centre of the cluster that received the unit.
Step 3: After all the data units have been assigned in step 2, take the existing cluster centres as fixed and make one more pass through the data set assigning each data unit to the nearest centre point.

Figure 4.7 - The MacQueen's k-means non-hierarchical clustering technique.

The function constraint technique selected for use with the non-hierarchical function centering technique was identified in section 4.5. That is, each function has its radii set to a value of 1, and its centre point constrained in each dimension to fall within the area of the data set the function is concerned with. This technique was chosen partly because it was found to perform well in conjunction with the MIF centering technique developed in section 4.5. In addition, a fixed radius for all functions in all dimensions was used in an attempt to isolate the effect of changing a function's centerpoint. A modified version of the Forgy technique was used in the present study, with a limitation on the number of iterations allowed before centering was considered to be complete. The relative performance of this technique will be discussed in section 4.8.

4.8 Comparative Results for Function Centering and Constraining Techniques

This section is concerned with examining the results obtained by the different function centering and constraint techniques identified in sections 4.4-4.7. The performance of these techniques has been examined for differing problems. The Parity (6), Encoder (8) and XOR problems which do not allow a test of generalisation have been examined. In addition, the Iris, Skin Cancer, Diabetes and Vowel recognition data sets which do allow a test of generalisation performance have been examined. The characteristics of

these data sets are given in table 4.1. Several performance issues will be considered in this section. First, the performance of the network on both the training and generalisation sets will be examined. Second, the time taken to train the network will also be evaluated. This will focus attention on issues such as the number of functions in the final network, the number of function addition stages required, and any additional computational effort demanded by the technique. In addition, the time taken by the network to answer a query will be examined. This will focus attention on the depth of the tree to be traversed and the number of calculations required to produce an answer.

When the results for the various function centering and constraint techniques are provided, reference will be made to the learning algorithm that was employed and the technique for calculating the output of the network. These issues will be examined in more detail in chapter 5. All of the results given in this section calculate the output of the network as that of the last active function in the tree. In addition, all the results given in this section use the one-shot learning algorithm that employs the Gauss-Jordan singular value decomposition technique (VanDer Rest 1992) to train the network.

Data set	Net details	Training set (ts) size	Generalisation set (gs) size	% of Output classes in ts	% of Output classes in gs
Iris	**4-n-3**	75	75	**33.3 each**	**33.3 each**
Skin cancer	**3-n-2**	62	62	**40 | 60**	**50 | 50**
Encoder 8	**8-n-8**	8	-	**12.5 each**	-
Parity 6	**6-n-2**	64	-	**50 each**	-
XOR	**2-n-1**	4	-	**50 each**	-
Diabetes	**8-n-2**	400	368	**38 | 62**	**31 | 69**
Petersen & Barney vowel data	**2-n-10**	320	333	**6 - 16**	**6 - 16**

Table 4.1 - The data sets employed in the study.

4.8.1 Training and Generalisation Performance

The performance of the various function centering and constraint techniques will be assessed in terms of the accuracy of classification of the training and generalisation sets. In order to measure this performance the bumptree neural network was trained and the generalisation set subsequently presented to the network for classification. The performance of the network was determined at the point where the highest level of generalisation performance was attained when over 70% of the training set was correctly classified. If the network failed to achieve this level of performance on the training set then the performance was determined at the point of best generalisation performance.

The performance of the various approaches to function centering and constraint identified in sections 4.4-4.7 on the training and generalisation sets has been measured as the percentage of correct classifications recorded. Whilst this technique provides information on the performance of the various approaches, it has limitations. The main limitation is that the division of patterns from the various output classes amongst the training and generalisation data sets can influence the performance recorded on them. For example, the performance of Omohundro's approach on the skin cancer diagnosis problem, given in figure 4.2, reveals the influence that an uneven division of patterns between the training and generalisation sets can have on performance. The training set is biased towards a single class, whilst the generalisation set is not. The improved performance on the generalisation set compared to the training set may simply reflect the more balanced nature of the generalisation set. Future examinations of the function centering and constraint techniques would profit by at least recording the percentage of each output class correctly classified. This would enable the identification of any bias introduced by the split of patterns from the different classes between the training and generalisation sets.

An examination of the performance of each technique will now be made before concluding with a discussion of their comparative performance. Table 4.2 provides summary information of how Omohundro's approach performed on the problems studied, with the best, worst and average performances being given together with the standard deviation for each set of results.

The results given in table 4.2 for Omohundro's approach to function centering and constraint were averaged over 10 runs. The output of the network was calculated using the last active function (LAF) technique, and it was trained using the one-shot learning algorithm employing singular value decomposition. Both these techniques are examined in detail in chapter 5. The results are quite encouraging, with the approach able to satisfactorily solve the iris, skin cancer, diabetes, XOR and encoder (8) problems. The results on the skin cancer diagnosis problem reveal the problem identified earlier with employing generalisation and training sets where the ratio of the different output classes differs. It is possible that this is responsible for the difference in performance on the training and generalisation data sets.

Figure		Iris	Skin Cancer	Parity 6	Encoder 8	XOR	Diabetes	Vowel Data
Mean	ts	95.5	79.0	56.9	100	100	75.9	72.0
	gs	92.1	82.2	-	-	-	79.7	65.6
Best	ts	100	85.5	65.6	100	100	77.0	81.3
	gs	98.7	83.9	-	-	-	80.7	71.2
Worst	ts	89.3	74.2	46.9	100	100	75.0	56.3
	gs	80.0	80.7	-	-	-	78.5	51.4
Standard	ts	1.3	3.5	5.3	0	0	0.7	7.1
deviation	gs	2.3	1.3	-	-	-	0.6	6.0

Table 4.2 - The average percentage performance of the Omohundro approach to function centering and constraint over ten runs.

Performance was poorer for the Petersen and Barney vowel data (Petersen and Barney 1952) and Parity (6) problems, the most complex problems examined. These required larger bumptrees than the other problems to reach an acceptable solution. The constraints to the dimensions of the functions enforced by Omohundro's approach were identified as a possible reason for the poorer performance on these problems. With this approach the lower level functions were not able to adjust the partitioning of the problem space sufficiently to achieve improved performance on the more complex problems. It was felt that this was because the functions were restricted to the area of their parent so closely that they were unable to introduce further splits as required. Relaxation of these constraints might, therefore, lead to an improvement in performance; a prediction which has been supported by the results obtained with the MIF technique. Table 4.2 shows that the results for the Petersen and Barney vowel recognition and Parity (6) problems have greater statistical variation than those obtained on the other problems.

The poor performance of the bumptree employing Omohundro's approach on the Petersen and Barney vowel data set and on Parity (6) led to the development of alternative approaches to the task of function centering and constraint. A major alternative for placing functions on the problem space was developed in the MIF approach. In conjunction with this, the constraints on the dimensions of the functions in the network were relaxed. The aim of both of these modifications was to enable the bumptree to produce a better partitioning of the problem space through the use of multiple initial functions where each function was assigned a significant radius in each dimension. The results achieved by this approach will now be considered. As a basis for comparison the results obtained with the MIF approach combined with the function constraint rule employed in Omohundro's approach, are given in table 4.3. The results were averaged over 10 runs. The LAF approach to output calculation was employed, in addition to the one-shot learning algorithm employing the singular value decomposition technique.

Figure		Iris	Skin Cancer	Parity 6	Encoder 8	XOR	Diabetes	Vowel Data
Mean	ts	98.3	84.8	61.0	100	100	75.8	72.4
	gs	94.8	80.3	-	-	-	79.7	64.9
Best	ts	100	91.94	68.8	100	100	77.3	86.9
	gs	96	83.9	-	-	-	81.0	73.0
Worst	ts	94.7	80.7	53.1	100	100	74.0	55.9
	gs	89.3	77.4	-	-	-	78.5	52.3
Standard	ts	3.0	4.2	5.5	0	0	1.2	9.9
deviation	gs	2.3	2.3	-	-	-	0.7	8.0

Table 4.3 - The average percentage performance of the MIF approach to function centering combined with Omohundro's approach to function constraining.

The results summarised in table 4.3 reveal an encouraging trend with regards to training performance. Average performance on the training set was equal to or better than that achieved by Omohundro's approach on six of the seven problems. The exception was the diabetes data set, where Omohundro's method achieved slightly superior average performance, although a difference of 0.1% is not particularly significant when the level of standard deviation is considered. The trend in generalisation performance was not as encouraging. Indeed, it only outperformed Omohundro's approach in terms of average percentage of correct classifications on the Iris problem. However, it was never outperformed by Omohundro's approach by more than one standard deviation from the mean, being outperformed by 1% in the case of the vowel recognition and diabetes data sets, and 2% in the case of the skin cancer data. The standard deviation figures for this approach were very similar to those given in table 4.2. The trend towards a slight worsening in average performance on the generalisation set compared to that attained by Omohundro's approach was the main cause for concern from the results given in table 4.3. However, the difference was only marginal, and it was hoped that by relaxing the constraints on function dimensionality the MIF approach would be able to at least equal the performance of Omohundro's approach on generalisation whilst retaining its advantage on the training set.

Figure		Iris	Skin Cancer	Parity 6	Encoder 8	XOR	Diabetes	Vowel Data
Mean	ts	99.3	79.8	99.5	100	100	75.7	83.0
	gs	95.3	79.7	-	-	-	79.8	73.2
Best	ts	100	87.10	100	100	100	78.0	89.5
	gs	97.3	82.3	-	-	-	80.7	76.3
Worst	ts	96.0	69.4	96.9	100	100	74.0	77.5
	gs	92.0	75.8	-	-	-	78.5	69.1
Standard	ts	1.4	5.6	1.0	0	0	1.1	5.8
deviation	gs	1.6	2.1	-	-	-	0.7	2.0

Table 4.4 - The average percentage performance of the MIF approach to function centering combined with reduced constraints on the dimensions of the functions at the lower levels of the tree. The functions below the top level are restricted to the area covered by the patterns upon which their parent was active.

The changes made to these constraints were examined in detail in section 4.5, and so will not be discussed here. The first modification introduced was to alter the size restriction for functions below the top level, constraining them to the area containing the active patterns covered by their parent. Previously they had been restricted to within the area of their parent function. The results obtained with this approach are shown in table 4.4.

The results given in table 4.4 were averaged over 10 runs. The LAF approach to output calculation was employed, in addition to the one-shot learning algorithm employing the singular value decomposition technique. These results show that this approach performed well in comparison to both the approaches whose results are examined above. For six of the seven data sets the average performance on the training set was equal to or better than that achieved by the other two approaches. The most significant improvement was on the Parity (6) problem, where the average performance improved from around 60% to 99.5%. The main reason identified for this was the ability of the bumptree with reduced constraints to introduce further splits of the

problem space with the lower level functions. It was felt that the reduction in the constraints on the dimensions of the functions was a significant factor in this improvement. The exception to this improving trend was the skin cancer data set where a 5% deterioration in performance resulted compared to that attained by the combined approach whose results are given in table 4.3. On the generalisation set this approach outperformed both the other approaches on three of the four data sets with a valid test of generalisation; although performance again deteriorated for the skin cancer data. A statistical analysis of the results show that greater consistency was achieved with this approach.

The results given in table 4.4 reveal a trend towards improved performance on both the training and generalisation sets. However, when considering this trend towards improved classification performance the standard deviation figures need to be considered. The standard deviation figures reveal that the performance of this bumptree is not as sensitive to initial conditions as the other two approaches examined above. This has contributed to the improving average performance, since the results are more consistently clustered around the average.

This work has shown that the removal of the strict constraints imposed by Omohundro led to an improved average performance on both the training and generalisation sets for the majority of the problems studied. The single exception is the skin cancer diagnosis problem, in which performance degraded and the standard deviation increased. The results led us to believe that a further relaxation of the function size might lead to additional improvement in network performance. Thus, the next stage in allowing the functions more freedom consisted of introducing a fixed radius of one for each top level function in each dimension, in the hope that this would enable a better partitioning of the problem space. The results obtained with this approach are set out in table 4.5.

Figure		Iris	Skin Cancer	Parity 6	Encoder 8	XOR	Diabetes	Vowel Data
Mean	ts	98.7	79.5	99.2	100	100	76.5	86.9
	gs	95.7	80.8	-	-	-	79.8	73.9
Best	ts	100	91.9	100	100	100	78.3	91.9
	gs	97.3	83.9	-	-	-	80.4	75.1
Worst	ts	93.3	74.2	96.9	100	100	75.3	81.6
	gs	94.7	77.4	-	-	-	78.0	71.8
Standard	ts	2.2	4.6	1.3	0	0	1.1	3.3
deviation	gs	0.8	2.0	-	-	-	0.9	0.9

Table 4.5 - The average percentage performance of the MIF approach to function centering combined with reduced constraints on the dimensions of the functions. The functions below the top level are restricted to the area covered by the patterns upon which their parent was active, and the top level functions have a radius of 1 in each dimension.

The results given in table 4.5 were averaged over 10 runs. The LAF approach to output calculation was employed, alongside the one-shot learning algorithm employing the singular value decomposition technique. These results show that a slight improvement in network performance was obtained for the data sets considered. The improvement was not as significant as that achieved with the last method, but this was to be expected, since the only alteration was restricted to the two top level functions in the network, as opposed to all the functions below the top level. However, an improvement in performance was demonstrated with this approach removed any chance of a constricted radii in any dimension occurring. Average performance on the training set improved from the previous techniques on the diabetes and vowel data sets, but worsened slightly, on the iris, skin cancer and Parity (6) problems. Of the four problems with a valid test of generalisation improved average performance was attained on the iris, skin cancer and vowel recognition problems, and identical performance achieved on the diabetes diagnosis problem. Performance on the iris and vowel recognition problems was the best attained by the techniques examined in this

section so far. However, performance on the skin cancer diagnosis problem is still worse than that achieved by Omohundro's approach, although the increase in the standard deviation may contribute to this difference in the general trend. The standard deviation figures demonstrate that this approach is less dependent upon the initial placing of the functions than the other techniques considered so far.

The performance of the MIF approach with a further relaxation of the constraints on the functions dimensions is given in table 4.6. The only remaining constraint existing with the bumptree examined in table 4.5 concerns the constraint of the radius of the functions below the top level to the area covered by the patterns upon which the parent function was active. Even this was now relaxed. All functions had their radius in each dimension set to 1. It was hoped that the additional freedom now given to the functions would enable them to provide a better partitioning of the problem space. A uniform radius of 1 assigned to each function meant that the problem caused by small radii was totally eradicated. The results obtained with this are shown in table 4.6.

The results given in table 4.6 were averaged over 10 runs. The LAF approach to output calculation was employed, in addition to the one-shot learning algorithm employing the singular value decomposition technique. These results support the belief that the removal of constraints on the size of functions provides a better partitioning of the problem space. On the iris data set this unconstrained approach outperformed all the other approaches in terms of average performance on both the training set and the generalisation set. It correctly modelled all the training set on every run, and attained a level of 97.3% on 90% of the runs on the generalisation set. On the skin cancer data set this approach was found to have an average performance 5% worse on the training set than the best performance. On the generalisation it was found to have an average performance 1.4% worse than the best performance On the Parity (6) problem this approach was found to be slightly inferior, by 1% to the best performance attained. However, it attained an average performance level of 98.3% and correctly classified all

sixty four patterns 60% of the time. Once again the results are very closely clustered around the mean value. This approach like all the other approaches correctly classified all the patterns for the XOR and Encoder (8) problems. On the diabetes diagnosis data set this technique outperformed all the other approaches on both the training set and the generalisation set, although the difference on the training set was only 0.1 %. On the Petersen and Barney vowel recognition data set this approach was outperformed by the best approach by 0.4% on the training set and 0.3% on the generalisation set.

Figure		Iris	Skin Cancer	Parity 6	Encoder 8	XOR	Diabetes	Vowel Data
Mean	ts	100	79.8	98.3	100	100	76.5	86.5
	gs	97.5	80.8	-	-	-	79.9	73.6
Best	ts	100	93.6	100	100	100	79.5	94.7
	gs	98.7	83.9	-	-	-	81.0	77.2
Worst	ts	100	74.2	92.2	100	100	74.8	73.8
	gs	97.3	77.4	-	-	-	78.0	71.8
Standard	ts	0	5.2	4.0	0	0	1.1	3.0
deviation	gs	0.5	2.0	-	-	-	0.9	1.5

Table 4.6 - The average percentage performance of the MIF approach to function centering combined with reduced constraints on the dimensions of the functions at all levels of the tree. All the functions have a radius of 1 in each dimension.

The results achieved by the bumptree that employed a unit radius in each dimension revealed a trend towards improved performance, but the differences between this approach and the other approaches that significantly reduced the constraints on the dimensions of the functions was minimal. The general trend is for the results to be less dependent on the initial starting point, and this undoubtedly contributes to the improved average performance since there are no particularly poor results to force the average performance down. The trend towards improved average performance supports the belief that the reduction of the constraints on the function dimensions

allows the network to partition the problem space in a more satisfactory manner. The approach that performed at the highest level over most of the problems was the one that combined the MIF approach with the most relaxed function constraint technique of setting the radii of all the functions in all dimensions to 1. If a function has a small radius in one or more dimensions in comparison to other functions at its own level then this will adversely affect its chances of attracting patterns and will therefore lead to an inaccurate partitioning of the problem space. The setting of all the functions dimensions to 1 normalises the radii and removes this problem. The Parity (6) problem and the Petersen and Barney vowel recognition problem could only be mapped to a satisfactory degree by approaches that adopted the relaxed function constraints, because the reduced constraints allowed the lower level functions to further partition the problem space. Hence, the overall trend of results for the training set and where applicable, the generalisation set has been for the average number of correct classifications to increase with a relaxation of the constraints on function dimensions.

The improvement in average performance has been accompanied by a decrease in the standard deviation. This demonstrates that the ability of the final network to reach a good solution is less dependant on its original starting point. The standard deviation figures also reveal that except for the approaches that do not relax the constraints to any degree the differences in performance are small. The trends toward improved performance are apparent, but the initial reduction of the constraints by the approach whose results are given in table 4.4 produced the most significant improvement. The performance of the five function centering and constraint techniques examined to date are summarised in figures 4.8 and 4.9, and tables 4.7 and 4.8.

Bumptree version				DATA	SET		
	Iris	**Skin cancer**	**Encoder 8**	**XOR**	**Parity 6**	**Vowel data**	**Diabetes**
Version 1							
% correct	95.5	79.9	100	100	56.9	72.0	75.9
standard deviation	1.3	3.5	0	0	5.3	7.1	0.7
Version 2							
% correct	98.3	84.8	100	100	61.0	72.4	75.8
standard deviation	3.0	4.2	0	0	5.5	9.9	1.2
Version 3							
% correct	99.3	79.8	100	100	99.5	83.0	75.7
standard deviation	1.4	5.6	0	0	1.0	5.8	1.1
Version 4							
% correct	98.7	79.5	100	100	99.2	86.9	76.5
standard deviation	2.2	4.6	0	0	1.3	3.3	1.1
Version 5							
% correct	100	79.8	100	100	98.3	86.5	76.5
standard deviation	0	5.2	0	0	4.0	3.0	1.1

Table 4.7 - The average percentage performance for the MIF centre placement and confining techniques and Omohundro's approach on the training sets examined in this study. Each technique is identified by a number which corresponds to the key given in figure 4.8.

To conclude, the results for the approaches examined above have shown that the performance of the various function centering and constraint techniques is to a degree problem dependent. The approach examined above that gives the best overall average performance, on both the training and generalisation sets, is the approach that combines the MIF approach with unit radii in all dimensions. It is however necessary to consider that the results may have been influenced to a degree by the fact that the training and generalisation sets employ different ratios of the output classes. However, the general trend towards improved results once the functions are allowed increased freedom to partition the problem space is clear.

Bumptree version	DATA SET			
	Iris	Skin cancer	Vowel data	Diabetes
Version 1				
% correct	92.1	82.2	65.6	79.7
standard deviation	2.3	1.3	6.0	0.7
Version 2				
% correct	94.8	80.3	64.9	79.7
standard deviation	2.3	2.3	8.0	1.2
Version 3				
% correct	95.3	79.7	73.2	79.8
standard deviation	1.6	2.1	2.0	0.7
Version 4				
% correct	95.7	80.8	73.9	79.8
standard deviation	0.8	2.0	0.9	0.9
Version 5				
% correct	97.5	80.8	73.6	79.9
standard deviation	0.5	2.0	1.5	0.9

Table 4.8 - The average percentage performance for the MIF centre placement and confining techniques and Omohundro's approach on the generalisation sets examined in this study. Each technique is identified by a number which corresponds to the key given in figure 4.9.

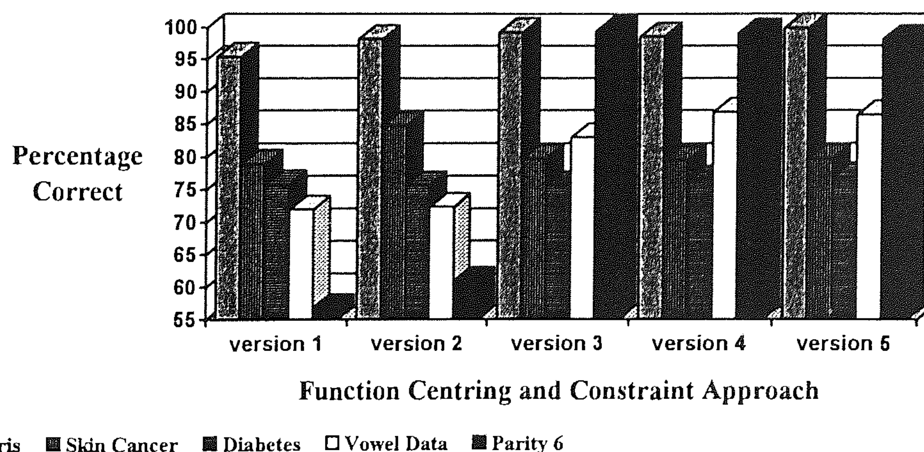

Figure 4.8 - Average percentage performance on the training set for the various function centering and constraint techniques.

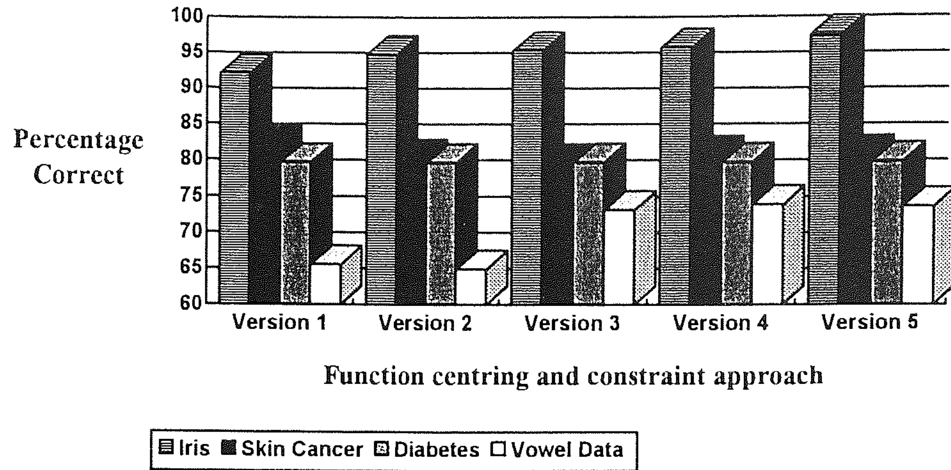

Figure 4.9 - Average percentage performance on the generalisation set for the various function centering and constraint techniques.

Key for figures 4.8, 4.9, and tables 4.7 and 4.8.
Version 1: Omohundro's approach.
Version 2: Omohundro's approach + MIF.
Version 3: MIF + lower level constrained within parents pattern area.
Version 4: MIF + top level radii =1 + lower level constrained within parents pattern area.
Version 5: MIF + all functions radii =1.

The techniques examined above ultimately add two functions to the network at every point where additional functions are required, but as section 4.6 pointed out, there is no inherent restriction which stipulates that the bumptree must be binary in nature. Indeed, the addition of multiple functions to the network at each splitting point might lead to a better partitioning of the problem space with a consequent improvement in performance. The results obtained by adding n functions to the network at each addition, referred to as the n-function bumptree are shown in table 4.9; n here is defined by the number of input dimensions of the problem.

Figure		Iris	Skin Cancer	Encoder 8	XOR	Diabetes	Vowel Data
Mean	ts	98.1	82.4	100	100	77.7	97.6
	gs	92.7	78.7	-	-	78.6	67.0
Best	ts	100	98.4	100	100	80.5	99.4
	gs	96.0	85.5	-	-	80.2	96.3
Worst	ts	94.7	75.8	100	100	75.8	62.5
	gs	82.7	67.7	-	-	77.2	72.7
Standard		2.1	5.8	0	0	1.4	0.8
deviation		3.2	6.7	-	-	0.9	3.4

Table 4.9 - The average percentage performance for the n-function bumptree, combined with radii of 1 for each function in each dimension.

The results shown in table 4.9 were averaged over 10 trials, and employed a bumptree trained using the one-shot learning algorithm employing the singular value decomposition technique. The output of the bumptree was calculated via the LAF technique for output calculation. The performance of this n-function bumptree was compared against those networks employing the techniques described earlier. The iris data set revealed that the n-function bumptree, whilst outperforming a network constructed with Omohundro's approach, was not able to match the average performance of any of the MIF approaches on either the training or generalisation set, although it attained comparable performance on the training set. The approach tended to produce results with greater variation than the other techniques examined to date, as illustrated by the standard deviation figures.

Figures 4.10 and 4.11 show the average performance of the various versions of the bumptree on the iris, diabetes, vowel recognition and skin cancer data sets for which generalisation could be measured. Only on the iris data set is the n-function bumptree outperformed on the training set, and then by the MIF bumptree which obtained complete correctness for all trials. On the diabetes, skin cancer and vowel recognition tasks the n-function bumptree performed at least as well as the other two techniques on the training set, and significantly better (by 10%) on the vowel recognition data.

However, in terms of average generalisation performance, the n-function bumptree is outperformed on all the problems examined, as shown in figure 4.18m, by the MIF approach with all radii set to 1. The n-function bumptree is able to outperform Omohundro's approach only on the iris and vowel recognition tasks. In addition, the standard deviation figures show that, particularly for the iris and skin cancer data sets, the performance attained by the n-function bumptree is very dependent on where the functions are placed on the problem space.

Therefore, whilst the n-function bumptree has performed well on the training sets of the problems in question, it has given a poorer performance on the generalisation sets. It has also been found to be sensitive to the initial placement of the functions on the problem space. However, the network gives a performance comparable to that attained by Omohundro's network. In addition to the results given in figures 4.10 and 4.11, the n-function bumptree was able to correctly classify 100% of the patterns for all the trials on the XOR and Encoder(8) problems. The divergence in the level of performance attained on the training and generalisation sets suggests that the n-function bumptree is overfitting the training set at the expense of performance on the generalisation set.

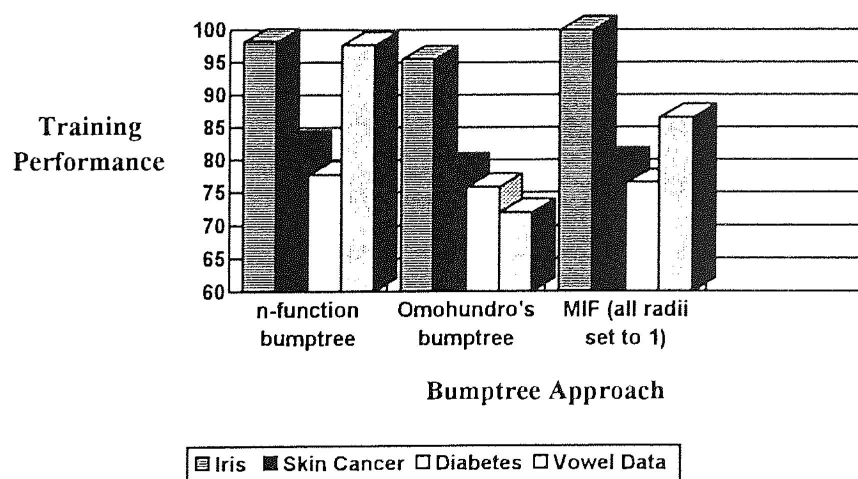

Figure 4.10 - The average percentage performance of various bumptree versions on the training sets of the iris, diabetes, vowel recognition and skin cancer tasks.

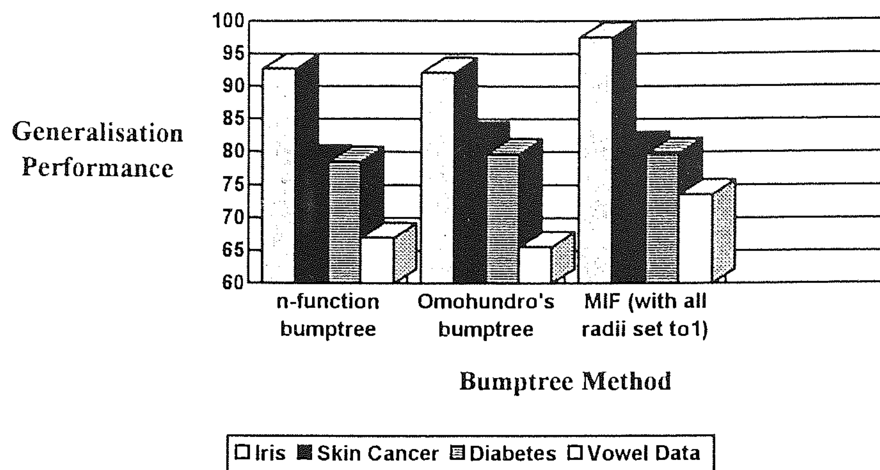

Figure 4.11 - The average percentage performance of various bumptree versions on the generalisation sets of the iris, diabetes, vowel recognition and skin cancer tasks.

None of the techniques examined so far has been able to produce the best average classification performance on all the data sets. However, the general trend is for techniques employing the MIF technique combined with reduced function constraints to produce the best average generalisation performance. The n-function bumptree displays evidence, particularly on the vowel recognition problem, of overfitting the training set at the expense of generalisation performance. In addition, the approaches that did not reduce constraints sufficiently, whose results are given in tables 4.2 and 4.3, are not able to model complex problems sufficiently. It is possible that this is due to their inability to achieve further splits in the problem space with the lower level functions. One possible technique for further improving the performance of the bumptree that was discussed in section 4.7 is the use of a non-hierarchical clustering technique to position the functions on the problem space. The Forgy technique, examined in section 4.7 was employed, and this technique was employed in conjunction with the MIF approach all radii for all functions set to 1.

Employing a non-hierarchical clustering technique in conjunction with the MIF approach raised the issue of where to apply the clustering technique. There exist two

points at which the clustering algorithm could be employed, with the first being concerned with the initial placement on the problem space of the ten functions. The second point is after the dimensions of the two functions to be added to the network from the ten initial functions have been decided upon. The author implemented the Forgy non-hierarchical clustering technique at these points both individually and simultaneously, and the results on the iris, skin cancer, diabetes and vowel recognition data sets are given in table 4.10.

The results displayed in table 4.10 were attained by bumptree structures trained with the one-shot learning algorithm employing the singular value decomposition technique. The output was calculated by the LAF technique. The results were averaged over 10 trials. The "best position slot" of table 4.10 refers to where the Forgy technique was applied to produce the result that is shown in the table. The results show that applying the clustering technique only after the final two functions have been allocated fails to produce better results than applying it in either of the other ways. The performance achieved on the iris data set, in terms of both training and generalisation was bettered by the MIF approaches that set the top level radii to 1, regardless of how the lower level functions radii were confined. The standard deviation on this data set showed that the approach is relatively independent of the placement of the functions on the problem space on three of the four problems. On the skin cancer diagnosis data set, the best average performance attained by any of the techniques examined in this chapter is achieved by this approach. However, on the training sets it is outperformed by all the other approaches. The standard deviation of 0 obtained shows that the results are invariant to the initial conditions. This can be viewed as a local minima, which in this instance happens to provide the best generalisation. Regardless of how many function splits are introduced the network always returns the same partition of the problem space as that achieved at the top level. The performance attained on the diabetes data set is similar to that attained by the other approaches, but is superior for generalisation. Finally, the performance of this approach on the vowel recognition data set was the

worst, by over 15% on both the training and generalisation sets of any of the approaches examined in this chapter. This approach has not been able to achieve an average performance level of better than 50% correctness on either the training or the generalisation sets. The standard deviation figure on the vowel recognition task reveals that this approach is very dependant on where the original functions are placed on the problem space. The functions added to the network at the earlier stages produced a partitioning that the later functions were unable to influence. This can be viewed as an inability to extract the network from local minima, a problem also encountered with the skin cancer diagnosis data.

The use of a non-hierarchical clustering technique in conjunction with the best MIF approach produced variable results. It produced excellent results on some problems, but was plagued by local minima on others from which it was unable to extricate itself. In particular, the performance on the skin cancer diagnosis data set revealed this characteristic. In addition, the use of a non-hierarchical clustering technique in conjunction with the best MIF approach had a drastic effect on the level of performance attained on the vowel recognition problem. It led to a decrease in performance of 35% on the training set and 23% on the generalisation set. The results on the vowel recognition problem were disappointing, and reveal limitations in the ability of the bumptree employing the Forgy non-hierarchical clustering approach to solve complex problems. These limitations relate primarily to the inability of the network to extract itself from local minima. The introduction of additional functions is unable to extricate the network from a minima once one has been encountered.

Figure		Iris	Skin Cancer	Diabetes	Vowel Data
Mean	ts	99.1	77.4	76.0	52.0
	gs	95.5	82.3	80.3	50.3
Best	ts	100	77.4	77.0	56.3
	gs	97.3	82.3	81.0	54.1
Worst	ts	96.0	77.4	75.0	45.3
	gs	90.7	82.3	78.8	40.8
Standard deviation	ts	1.8	0	0.9	8.3
	gs	2.7	0	0.9	8.7
Best Position		**10 functions**	**Both**	**10 functions**	**Both**

Table 4.10 - The average percentage performance for the bumptree employing the Forgy non-hierarchical clustering technique, combined with MIF with all radii set to 1.

This study has shown that the MIF technique combined with reduced function constraints attained a consistently high level of training and generalisation performance across all the data sets tested. It is therefore recommended as the best general purpose algorithm of those examined. However, in making this recommendation it is necessary to consider that the presentation of training and generalisation sets with differing ratios of the output classes could have influenced performance. In addition, some of the differences in performance are small when considered across all the approaches, and the differences can be seen to relate to the impact of starting conditions on the performance of the networks. Nevertheless, a trend towards improved performance when the MIF approach is combined with reduced function constraints is revealed.

4.8.2 Training Time

This section will examine the performance of the various function centering and constraint techniques introduced in this chapter in terms of the time taken to train to a solution. Consideration will be given to the number of functions needed and the computational complexity involved in the training process. The number of functions

required by each approach is summarised in table 4.11. Only those data sets that allow a valid test for generalisation are examined, since the aim is to analyse the time taken to reach the level of generalisation performance described in section 4.8.1.

	Version 1	Version 2	Version 3	Version 4	Version 5	Version 6	Version 7
Skin cancer	4	11	4	3	9	11	2
Diabetes	3	6	7	11	5	23	8
Vowel data	104	104	96	67	53	193	16

Table 4.11 - The average percentage performance of the various bumptree versions in terms of the number of functions required to reach the level of generalisation performance discussed in section 4.8.1.

Key:
Version 1: MIF + all functions radii = 1.
Version 2: MIF + top level radii =1 + lower level constrained within parents pattern area.
Version 3: MIF + top level functions constrained to within the area of the data set + lower level constrained within parents pattern area.
Version 4: Omohundro's approach + MIF
Version 5: Omohundro's approach.
Version 6: N-function bumptree.
Version 7: MIF bumptree employing the Forgy non-hierarchical clustering technique.

The results in table 4.11 for the various centering techniques were averaged over 10 trials. The bumptrees were trained with the one-shot learning algorithm employing the singular value decomposition technique, and the LAF approach to output calculation was employed. The results demonstrate that with the exception of the vowel recognition task the different bumptree structures tended to employ a similar number of functions to solve the particular problems. However, the vowel recognition task is a good guide to the performance level reported in section 4.8.1. Version 7 employs insufficient functions and is, therefore, unable to adequately map the problem space. This is because once the problem space has been partitioned 2 or 3 times, further

partitions simply replicate those that already exist. The same is true of versions 4 and 5, since they employ insufficient functions to map the problem space. They constrain the lower level functions to the same area as their parent, and eventually the problem space becomes saturated by the functions that have been created. Version 6 employs too many functions to model the vowel recognition data, and the results in section 4.8.1 reflect this. The approach outperforms all the others on the training set for almost all the problems, but cannot match the performance of the better MIF techniques in terms of generalisation. Hence, partitioning the problem space with multiple functions at each point where additional functions are required leads to the bumptree over-fitting the training set with a subsequent degradation in generalisation performance.

The computational effort required by each of the bumptrees identified in table 4.11 to reach a solution is linked to the number of functions employed. However, it is also linked to the technique employed for constructing the network. The main steps involved in the MIF bumptree with all radius set to 1 are given in Appendix A in the form of pseudo code. This provides an outline of the approach adopted by this bumptree against which the other approaches can be compared, and provides a basis for the analysis of the computational effort involved with the different bumptrees.

The MIF approach uses multiple candidate functions to determine the dimensions of the functions to be added to the network. This is potentially more computationally expensive in terms of adding functions to the network than some of the other approaches. Compared to Omohundro's approach there is the additional step of assigning the initial candidate functions to the problem space and their subsequent processing. The candidate functions have to be placed on the problem space, the patterns assigned between them, the learning algorithm applied to the active ones and a goodness measure calculated for each of them. The functions not chosen as the best are then averaged out to provide the second function to be added to the network. The

patterns are then assigned as appropriate. Each of the MIF approaches adopts a different approach to constraining the dimensions of the functions, and the least computationally expensive of these is to set all the radii to 1. The most computationally expensive of the approaches is that which employs a non-hierarchical clustering technique to obtain better partitioning of the problem space. This approach employs all the steps employed by the MIF technique, in addition to the Forgy technique described in figure 4.6.

Omohundro's approach was the simplest for positioning functions on the problem space. It simply placed two functions within the area covered by the training set, constrained their radius so that they fell within the area of the data set, and then assigned the patterns to these functions. Once the patterns had been assigned the learning algorithm was applied to them. Finally, a performance measure was applied to ascertain whether further functions were required. Further functions were added by repeating the process used to add the initial functions. These additional functions were positioned so that their centre fell within the area covered by their parent function. In addition, the radii of the functions were constrained to fall within the area of the parent. The n-function bumptree adopted a very similar approach to Omohundro's, except that the process had to be carried out for multiple functions instead of two at each addition. The actual number of functions was determined by the input dimensionality of the problem. This approach set the radii of every function to one. The addition of functions by the n-function bumptree is, therefore, computationally more expensive. However, gains are achieved by potentially reaching a solution with fewer addition of functions, although each addition is more computationally expensive.

The computational complexity of each of the bumptree approaches identified above will be analysed in terms of the mathematical operations required for adding functions to the network. The pseudo code given in Appendix A provides a basis for the analysis, and the focus of attention will be on identifying differences between the

various approaches. A major process involved in constructing the bumptree network concerns the addition of functions to the network. This process involves a number of procedures, and commences with the top level, or parent functions, being positioned on the problem space. If these cannot adequately model the problem space then further functions are added. The addition of functions by the MIF bumptree involves the initial step of determining centres for the 10 candidate functions to be considered for addition to the network. This utilises the calculation given in equation 4.2:

$$\sum_{f=1}^{f\max}\sum_{u=1}^{u\max} RandomNumber/((\pi^*(Max[u]-Min[u]))+Min[u]) \qquad (4.2)$$

where fmax is the number of functions; umax is the number of units; Max[u] is the maximum value for input dimension u; Min[u] is the minimum value for input dimension u; RandomNumber is the random number used to provide the initial figure to determine the centre of function f in each input dimension. The use of Min[u] and Max[u] is to ensure that the centre of the parent function dimension by dimension falls within the area covered by the data set for the initial parent functions. For functions added after the initial two functions Min[u] and Max[u] refer to the maximum and minimum values in each dimension of the patterns upon which the parent function was active.

The allocation of centre points to the ten candidate functions to be considered for addition to the network, whilst an integral part of the MIF approach, is not present in either the n-function bumptree or in Omohundro's approach. These other approaches do not employ candidate functions, they simply position the functions to be added on the problem space. For each input dimension, each of the ten functions requires a process that involves 1 division, 1 multiplication, 1 addition, and 1 subtraction. Hence, for the Encoder (8) problem this initial step of assigning centres to the candidate functions involves a total of 320 divisions, multiplication's, additions, and subtraction's that are not required by either the n-function bumptree or Omohundro's approach.

Once the dimensions of the candidate functions have been determined, it is necessary to assign the patterns in the training set to the function with the highest activation level. Figure 4.12 provides the pseudo code for this process, detailing the calculations that take place. Once again these calculations relate to the 10 candidate functions used to determine the dimensions of the two functions to be added to the bumptree structure and therefore the process is unique to the MIF bumptree.

```
For Patterns = 1 to Num Patterns in Training Set
  For Functions = 1 to 10
      Temp 5 =1
      For Units = 1 to Num Input Units
            Temp1 = 1/(1*(sqrt(3.141592654*2)))
            Temp2 = 0.5 *(Input [Patterns][Units]-Function centre[Units])/1)^2
            Temp3 = exp(-Temp2)
            Temp4 = Temp1*Temp3
            Temp5 = Temp5*Temp4
      End
      Function[Functions] Activation Level = Temp5
  End
End
```

Figure 4.12 - Pseudo Code detailing the steps involved in calculating the activation of the 10 candidate functions with the MIF approach.

The pseudo code in figure 4.12 demonstrates that for each pattern each function on each input dimension carries out 3 divisions, 6 multiplication's, 2 subtraction's, and 1 exponential calculation. Therefore, for the XOR data set that comprises two inputs and four training patterns there are 960 calculations required. The equation to determine the number of calculations required for a training set is given in equation 4.3. The equation given in 4.3 is applied to the candidate functions when the MIF bumptree is employed.

$$\text{NumFunctions*Num Input Units*Num Patterns*Num Calculations} \quad (4.3)$$

Once the activation of the 10 candidate functions has been calculated, it is necessary to assign each pattern in the training set to the function with the highest activation level. This is a simple comparative process, and following this each of the ten functions considered to be active on sufficient patterns has its weight and bias parameters optimised. The process of optimising the weight and bias, or Alpha and Beta parameters is detailed in Appendix B. The matrix containing the optimised Alpha and Beta values is arrived at by multiplying the result matrix (matrix3) by the pseudo inverse of matrix1. Matrix1 can be populated in two stages. The first stage populates an area equal to the square of the number of input dimensions. For each of the ten candidate functions, every slot is filled by a process using 1 multiplication and 1 addition for every pattern the function is active on in the training set. Hence, the number of calculations involved in this process is determined as the number of input dimensions squared multiplied by the number of patterns in the training set multiplied by the number of calculations. No reference is made to the number of functions the process is carried out for since regardless of this the operation is carried out for every training pattern. The outer loops in the pseudo code given in Appendix B have to be traversed more times when more functions are being examined, but the number of calculations involved in the actual process is the same. The first stage of populating Matrix1 for the Encoder (8) will require 8*8*8*2 calculations, a total of 1024 calculations. The second stage of populating Matrix1 involves every function for every input dimension having an addition carried out over every pattern upon which the function is active. This process is carried out twice. Each time the calculations required can be given as the number of input dimensions multiplied by the number of patterns in the training set. Using the example of the Encoder (8) problem this requires 8*8, or 64 calculations, a total of 128 calculations.

Matrix3 is populated through a very similar process to Matrix1. It is again populated in two stages. The first stage populates an area of the matrix equal to the number of input dimensions multiplied by the number of output dimensions. For each of the candidate

functions, every slot is filled by a process using 1 multiplication and 1 addition for every pattern the function is active on in the training set. Hence, the number of calculations involved in this process is determined as the number of input dimensions multiplied by the number of output dimensions multiplied by the number of patterns in the training set multiplied by the number of calculations. No reference is made to the number of functions the process is carried out for, since regardless of this the operation is carried out for every training pattern. Once again, the outer loops in the pseudo code given in Appendix B have to be traversed more times when more functions are being examined, but the number of calculations involved in the actual process is the same. The second stage of populating Matrix3 involves every function for every output dimension having an addition carried out over every pattern upon which the function is active. This process is carried out twice. The calculations required can be given as the number of output dimensions multiplied by the number of patterns in the training set.

Once the two matrices have been populated it is necessary to produce the pseudo inversion of matrix1, and this is done using the Gauss Jordan method. This involves identifying the pivot row and carrying out the elimination process, but this process is not computationally intensive. Once the pseudo inverse of Matrix1 has been calculated, it is necessary to multiply matrices 1 and 3 for each function. For every row in the result matrix, a process involving a single multiplication is carried out. The number of calculations is given as Input dimensions +1 multiplied by the number of output dimensions. Using the example of the Encoder (8) problem, each function requires a process consisting of 9*8, or 72 calculations.

```
For F = 1 to 10
 Temp6 = 0
 For Pattern =1 to Num Patterns Function[F] Active On
  Temp5 = 0
  For U = 1 To Num Output Units
   Temp2 = 0
   For UI =1 to Num Input Units
    Temp = Function[F] ALPHA[UI][U]*Input[patterns][UI]
    Temp2=Temp2+Temp
   End
   Temp3 = (Temp2 + BETA[U])-Output[patterns][U]
   Temp4 = Temp4 * Temp3
   Temp5 = Temp5 + Temp4
  End
  Temp6 = Temp6 + Temp5
 End
 Function[F] Error =0.5*Temp6
End
```

Figure 4.13 - Pseudo Code detailing the steps involved in calculating the error of the 10 candidate functions with the MIF approach.

Once the 10 candidate functions have had their Alpha and Beta parameters optimised, it is necessary to calculate an error level for each candidate function considered to be active on sufficient patterns. Figure 4.13 provides pseudo code detailing the calculations involved in calculating the error of the candidate functions. Each function has its error calculated on only the patterns upon which it is active. The calculations required to calculate the error of an individual function are given in figure 4.14. The impact that each dimension has on this calculation is shown with the Parity (6) problem. The number of calculations required is given in figure 4.14. This figure reveals that to calculate the error of the 10 candidate functions requires a total of 1994 calculations made up of multiplication's, divisions, additions and subtraction's. The numbers identified in figure 4.14 are as follows. The number of functions is 10, the

number of patterns in the training set is 64, the number of output units is 2 (identified as 2a), and the number of inputs is 6 (identified as 6b). The other figures given relate to the number of calculations required at each of the stages identified in figure 4.13.

(10*1)+(64*1)+(64*(2a*3))+(64*2*(6b*2)) = Error of Candidate functions

Figure 4.14 - Calculations involved in calculating the error of the 10 candidate functions with the MIF approach.

Once the error of the 10 candidate functions has been calculated, the next step is to calculate the goodness value for each of the functions considered to be active on sufficient patterns. The goodness value is calculated as shown in equation 4.4

$$\text{Error Level} \,/\, \text{Number of Patterns Function Active On} \tag{4.4}$$

This goodness measure is calculated once for each function, and therefore involves ten calculations. The function with the best goodness value is added to the network, and the second function to be added is arrived at by averaging the centres in each dimension of all the functions active on any patterns.

Once the dimensions of the two functions to be added to the network have been determined the approach of the MIF bumptree is similar to that of Omohundro's approach. Both the approaches employ the procedures identified in figure 4.15 for "actual functions" on the two functions to be added to the network. The MIF approach does not require any extra calculations to Omohundro's approach until further functions need to be added to the network and the procedure with the 10 candidate functions recommences. The main steps involved in the MIF approach to training the bumptree are summarised in figure 4.15. These steps are divided into those that are applied to the candidate functions and those that are applied to the functions added to the network. Omohundro's approach utilises only those steps applied to the functions to be added to the network. The n-function bumptree does likewise, except that whilst

Omohundro's approach carries out these steps for two functions the n-function bumptree carries it out for a variant number of functions. The number is determined by the number of input dimensions of the problem. Table 4.11 provides details of the calculations required by the MIF approach, Omohundro's approach, and the n-function bumptree on the problems examined in this study.

Candidate Functions:
1 - Determine initial function centres.
2 - Assign Patterns to functions.
3 - Optimise Alpha and Beta parameters.
4 - Calculate the error of each function.
5 - Calculate a goodness value for each function.
6 - Add functions to the network.

Actual Functions:
7 - Assign Patterns to functions.
8 - Optimise Alpha and Beta Parameters
9 - Calculate the error of each function.

If further functions are required goto step 1, else finish training.

Figure 4.15 - The main steps employed to construct the MIF bumptree.

The figures given in table 4.11 reflect the calculations that the approaches require to add functions to the network. All the approaches need to define the parameters of the initial functions, and Omohundro's approach involves the least calculations. The figure given in table 4.11 reveal that as the input and output dimensionality of the problem increase so the other approaches require more computational effort than Omohundro's approach. The number of calculations required by the N-function bumptree is adversely effected when the input dimensionality of the problem rises. This is to be expected since the number of functions employed increases with the number of input dimensions. The difference in the number of calculations required by the approaches relates purely to their use of different numbers of functions. The MIF bumptree has 10

additional functions to process compared to Omohundro's, and when there are more than two inputs the n-function bumptree has additional functions to deal with. Therefore, in order for the MIF or n-function bumptree to add further functions to the network additional computational effort is required at each addition.

Data Set	MIF Bumptree		Omohundro Bumptree		N-Function Bumptree	
	Candidate functions	**Actual functions**	**Candidate functions**	**Actual functions**	**Candidate Functions**	**Actual Functions**
Iris	**43225**	**14417**	**-**	**14417**	**-**	**21619**
Skin Cancer	**25998**	**8134**	**-**	**8134**	**-**	**10367**
Encoder 8	**10522**	**4370**	**-**	**4370**	**-**	**8984**
Parity 6	**55256**	**18384**	**-**	**18384**	**-**	**36820**
XOR	**1091**	**314**	**-**	**314**	**-**	**314**
Diabetes	**395228**	**164820**	**-**	**164820**	**-**	**395226**
Vowel Data	**112360**	**50912**	**-**	**50912**	**-**	**50912**

Table 4.12 - Calculations required to add new functions to the bumptree. The calculations required to assign the patterns to the functions depending on their activation, to optimise the Alpha and Beta values, and to calculate the error of the functions are considered.

Additional calculations are required when a technique for constraining the radius of the functions is employed. The setting of all the radius to 1 is the least computationally expensive approach, and the use of the Forgy technique is the most computationally expensive approach. However, the impact of constraining the radius of the functions on training computational expense is significant only when the Forgy non-hierarchical clustering technique is employed. Therefore, in the analysis of the computational expense of the various approaches the constraining of radius has not been considered.

Therefore, the amount of computation required by the various techniques differs, with the MIF bumptree employing the non-hierarchical clustering technique being the most computationally expensive.

4.9 Summary

This chapter has addressed the issue of how the problem space should be partitioned. The focus of attention has been on various approaches to positioning the functions on the problem space and constraining the dimensions of these functions. The performance of each of these techniques on various data sets ranging from a vowel recognition task to the diagnosis of skin cancer has been evaluated. Performance has been tested primarily on the ability of the approaches to construct a network that is able to perform well on both training data and generalisation data. In addition, an examination of the computational effort required by the approaches to construct a network to a solution has been examined. Particular attention has been given to the number of functions required to attain a desired level of performance. In addition, the computational effort, in terms of the number of calculations demanded by each technique has been examined. The focus of attention has been on the addition of units to the network, since all the approaches examined in this chapter have used the same learning algorithm and the same output calculation technique.

The approaches have on the whole performed fairly well on the data sets considered. Whilst no individual approach has been able to constantly outperform all the others, some have revealed limitations. For instance, the n-function bumptree has revealed a tendency towards over fitting the training set to the detriment of generalisation performance. In addition, it has been discovered that Omohundro's restriction on the dimensions of the lower level functions causes difficulties when a large number of functions are required to solve a problem. The vowel recognition data set is a good example of this, as is the Parity (6) problem. Limitations with the use of a non-hierarchical clustering technique have also been revealed. On the smaller scale problems the use of a non-hierarchical clustering technique produces a good level of

performance. In particular, on the diabetes problem the best level of performance attained on both the training and generalisation sets is with a network built employing the Forgy technique. However, this approach is susceptible to the problem of local minima. It has proved incapable on some problems of partitioning the data so that good performance on the training and/or generalisation sets can be achieved. In particular, the vowel recognition task has proved a serious problem, and the skin cancer diagnosis task always encountered a local minima.

The best approach to constructing a bumptree neural network considered in this study was the MIF approach, combined with a number of alternate techniques for constraining the dimensions of the functions within the network. The technique that was able to achieve the highest level of performance consistently across all of the problems examined was the one that set all the radii of the functions to 1. This approach overcame the problems of local minima and small radii and consequently allowed sufficient functions to be added to solve the problems without overfitting the training set.

In terms of computational effort, each function addition with the MIF and n-function bumptrees required more effort than when Omohundro's approach was employed. The number of function additions required by each of the approaches was problem dependent, but the trend was for the n-function bumptree to require the most functions, followed by the MIF bumptree with bumptrees developed with Omohundro's approach employing the least functions. This trend is reflected in the results on the data sets examined in this chapter. The n-function bumptree employed too many functions and subsequently on several of the problems overfitted the training data at the expense of generalisation performance. Omohundro's approach was not able to model the more complex problems sufficiently because it employed too few functions. The MIF bumptree was, however, able to produce a reasonable level of

performance on both the training and generalisation data sets on the problems examined in this study.

There are, however, a number of other issues that need to be addressed before a bumptree neural network can be employed. In particular, a learning algorithm that can be applied to each individual function needs to be developed and a technique for calculating the output from the network needs to be devised. Two of these techniques have been introduced in this chapter. In addition, decisions must be made about when to add functions to the network and also when to recognise that further partitioning of the problem space is not desirable and that the training process should be stopped. These issues will be examined in detail in chapter 5, and a comparative study of performance with RBF and MLP networks will be carried out in chapter 6.

Chapter 5

Further Architectural Issues In Developing A Bumptree Neural Network

5.1 Introduction

In order to utilise a bumptree neural network several issues need to be resolved. Chapter 4 examined techniques for positioning the functions and determining their size. The task of partitioning the problem space, whilst being vital to the performance of the network, is only one issue that needs to be addressed in order to implement a bumptree neural network. Other major issues are the nature of the learning algorithm to be applied and the method for calculating the output of the network. In addition, the algorithm developed by the author to construct the bumptree neural network employed some additional mechanisms. These controlled when functions should be added to the network, when the addition should cease and whether a function could be described as "live". Live in this sense refers to whether a function is considered to contribute sufficiently to the network to be involved in calculating its output. All these issues will be examined in this chapter.

This chapter will also introduce the genetic bumptree (Willliams *et al.* 1993, 1994). The genetic bumptree is a variation on the bumptree classifier which, instead of being "trained" using a learning algorithm, develops entirely through a process of evolution, via a genetic algorithm. The genetic algorithm can be used to simultaneously optimises the number of functions employed by the tree, the structure of the tree, the functions centre and radii, and the functions associated weight and bias parameters, in order to minimise a training set error measure. The performance of the genetic bumptree will be described and compared against that obtained with the standard bumptree.

5.2 The Learning Algorithm

The bumptree neural network approaches the task of mapping a problem space in two stages. The first stage involves partitioning the problem space into local areas with gaussian functions. The second stage involves the application to each of these individual areas of a learning algorithm to minimise a training set error measure. These stages are closely interlinked in the process of constructing a bumptree neural network. First, the initial functions are placed on the problem space. The learning algorithm is then applied, and if it is unable to reach an acceptable level of performance additional functions are created to further partition the space before the learning algorithm is applied to the new functions. This procedure continues until an acceptable solution is reached. The partitioning of the problem space was examined in chapter 4, and the second stage of this process, the learning algorithm will now be considered.

The learning algorithm is applied once to each function as it is added to the network, and each time it is applied it only considers those patterns for which the function is active. Hence, each local area of the problem space has the learning algorithm applied to it separately in an attempt to minimise the squared error on the training set. Although a learning algorithm specific to the bumptree was developed by the author, and will be examined below, existing learning algorithms could have been adopted. For instance, it would be possible to employ a multi-layer perceptron to model the space covered by each individual function in the network, so that, each of the functions would have its area modelled by an individual and independent MLP. This is still in compliance with the idea of imposing a local solution, since even though the hyper planes of the MLP will extend far beyond the boundaries of the data that the particular function is concerned with, they will have no effect on the remaining data. However, to employ an MLP at each function

in the network would lead to unacceptably long training times. The time taken would exceed that required to train a single MLP mapping the entire problem space.

A learning algorithm has been developed specifically for the bumptree neural network, which will be shown to be capable of finding a solution quickly and accurately; this will now be examined. The algorithm is a "one-shot" learning algorithm, in contrast to the MLP which employs an iterative procedure. The one-shot algorithm determines the optimum weight and bias parameters for each function by solving a series of equations. This learning algorithm maps the problem space through the use of weight and bias parameters that connect the input and output units. There is one weight, or Alpha, value for each input to output connection and a bias, or Beta, value for each output unit. The Beta value is constant across all the input dimensions. The output of a function on any pattern is given as:

$$ao_{ipz} = \sum_{j}^{jMAX} \alpha_{ijz} * x_j^{(p)} + \beta_{iz} \qquad (5.1)$$

where ao_{ipz} is the output of the zth output unit of the ith function on the pth pattern, j is the input unit, jMAX the total number of input units, α_{ijz} is the Alpha value that connects input unit i to output unit z for the ith function, $x_j^{(p)}$ is the element of the pth training pattern that is concerned with the jth input dimension. Finally, β_{iz} is the Beta value for the zth output unit. The error of each function E_i is given as:

$$E_i = \frac{1}{2} \sum_{p=1}^{pMAX} \sum_{z=1}^{zMAX} (ao_{ipz} - tv_{pz})^2 \qquad (5.2)$$

where E_i is the error of the ith function across all output dimensions (zMAX), for all patterns upon which the function is active (pMAX), ao_{ipz} is the actual zth output of the ith function on the pth pattern, and tv_{pz} is the target output for the zth unit on the pth pattern. Hence, the learning algorithm is required to arrive at Alpha and Beta values for

every function that minimises the error given by 5.2. The Alpha and Beta values for each function i can be derived from the equation given in 5.3, which is simply a combination of 5.1 and 5.2, and this derivation will now be examined.

$$E_i = \frac{1}{2}\sum_{p=1}^{pMAX}\sum_{z=1}^{zMAX}\{\sum_{j=1}^{jMAX}\alpha_{ijz}x_j^{(p)} + \beta_{iz} - tv_{pz}\}^2 \tag{5.3}$$

The equation in 5.3 can be divided into two parts, with the first part, given in 5.4 concerned with the Alpha derivative. The second part concerns the Beta derivative, and is given in 5.5.

$$\frac{\delta E_i}{\delta \alpha_{i'j'z'}} = \sum_{p=1}^{pMAX}\sum_{z=1}^{zMAX}\{\sum_{j=1}^{jMAX}\alpha_{ijz} * x_j^{(p)} + \beta_{iz} - tv_{pz}\}^2 * x_{j'}^{(p)} = 0 \tag{5.4}$$

$$\frac{\delta E_i}{\delta \beta_{i'z'}} = \sum_{p=1}^{pMAX}\sum_{z=1}^{zMAX}\{\sum_{j=1}^{jMAX}\alpha_{ijz} * x_j^{(p)} + \beta_{iz} - tv_{pz}\}^2 * 1 = 0 \tag{5.5}$$

From the derivatives of Alpha and Beta given in 5.4 and 5.5 it is possible to arrive at the Alpha and Beta parameters that provide the best fit to the data. This is achieved through multiplying matrix 1, given in figure 5.1, by the inverse of matrix 3, given in figure 5.3. This provides the matrix of Alpha and Beta values given in figure 5.2. The Matrices derived from 5.4 and 5.5 given in figures 5.1, 5.2 and 5.3 are those required by a problem with 3 inputs and 2 outputs.

$$\begin{array}{cccc}
\sum_{p=1}^{pMax} x_1^{(p)} x_1^{(p)} & \sum_{p=1}^{pMax} x_1^{(p)} x_2^{(p)} & \sum_{p=1}^{pMax} x_1^{(p)} x_3^{(p)} & \sum_{p=1}^{pMaxc} x_1^{(p)} \\
\sum_{p=1}^{pMax} x_2^{(p)} x_1^{(p)} & \sum_{p=1}^{pMax} x_2^{(p)} x_2^{(p)} & \sum_{p=1}^{pMax} x_2^{(p)} x_3^{(p)} & \sum_{p=1}^{pMax} x_2^{(p)} \\
\sum_{p=1}^{pMax} x_3^{(p)} x_1^{(p)} & \sum_{p=1}^{pMax} x_3^{(p)} x_2^{(p)} & \sum_{p=1}^{pMax} x_3^{(p)} x_3^{(p)} & \sum_{p=1}^{pMax} x_3^{(p)} \\
\sum_{p=1}^{pMax} x_1^{(p)} & \sum_{p=1}^{pMax} x_2^{(p)} & \sum_{p=1}^{pMax} x_3^{(p)} & pMax
\end{array}$$

Figure 5.1 - Matrix 1, which is derived from equations 5.7 and 5.8

$$\begin{array}{cc}
\alpha_{11} & \alpha_{12} \\
\alpha_{21} & \alpha_{22} \\
\alpha_{31} & \alpha_{32} \\
\beta_1 & \beta_2
\end{array}$$

Figure 5.2 - Matrix 2, the Alpha and Beta matrix, derived from equations 5.7 and 5.8.

$$\begin{array}{cc}
\sum_{p=1}^{pMax} tv_{p1} x_1^{(p)} & \sum_{p=1}^{pMax} tv_{p2} x_1^{(p)} \\
\sum_{p=1}^{pMax} tv_{p1} x_2^{(p)} & \sum_{p=1}^{pMax} tv_{p2} x_2^{(p)} \\
\sum_{p=1}^{pMax} tv_{p1} x_3^{(p)} & \sum_{p=1}^{pMax} tv_{p2} x_3^{(p)} \\
\sum_{p=1}^{pMax} tv_{p1} & \sum_{p=1}^{pMax} tv_{p2}
\end{array}$$

Figure 5.3 - Matrix 3, the result matrix derived from the equations given in 5.7 and 5.8.

In order to arrive at the Alpha and Beta figures in matrix 2 it is necessary to multiply matrix 3 by the inverse of matrix 1. This procedure will need to be carried out once for each function in the network. The one-shot learning algorithm is able to arrive at the

Alpha and Beta parameters extremely quickly, with a consequent reduction in the training time; this is in contrast to the iterative procedure employed in back propagation. However, there is a drawback with the one-shot learning algorithm, namely the need to invert matrix 1 before being able to calculate the Alpha and Beta parameters.

When the one-shot learning algorithm was first implemented problems with singular matrices were often encountered, which caused training to cease. Such difficulties arose when matrix 1 either contained a number of 0's, or when only a few patterns were being considered. This set of circumstances was found to be problem dependent to a degree, but was also affected by parameters defined for the bumptree training algorithm. For example, in the training algorithm a parameter termed *SMALL* was used to determine whether a function was live and therefore a candidate for training. If a function did not attract more than *SMALL* patterns, it was not a candidate for training. The *SMALL* parameter can, to an extent, prevent problems with singular matrices by ensuring that matrix 1 is not created for functions that are responsible for a small number of patterns. The problem of matrix 1 containing a number of 0 elements, however, still remains.

Experimentation revealed that when matrix inversion was employed as part of the learning algorithm the bumptree had a very limited degree of success. For instance, on the Parity (6), Encoder (8) and XOR problems the bumptree was never able to converge to a solution due to the number of 0's in matrix 1. For the other problems examined in this study, such as the skin cancer diagnosis and vowel recognition data sets, the bumptree was able to reach a solution with an appropriate setting of *SMALL*. However, the accuracy of this solution was not as good as that reported in chapter 6 because the value of *SMALL* was too high to allow the network to model the problem space with sufficient accuracy. The best results attained on the iris problem when pseudo matrix inversion was employed were attained when *SMALL* was set to 7. However, when matrix inversion was

used the problem of singular matrices was encountered, and the bumptree was only able to reach a solution for 70% of the trials when *SMALL* was set to 7. Table 5.1 summarises, the performance on the Iris problem of that version of the bumptree identified in chapter 4 as the one giving the best overall performance. That is, the bumptree which employed the MIF approach with all radii set to 1 and the output calculated using the LAF technique The performance of this bumptree is given for various settings of *SMALL*. Table 5.2 summarises the performance of the bumptree for the skin cancer diagnosis problem. It can be seen that performance is inversely related to the value taken for *SMALL* - a trend that was noted in the results for the other data sets. The standard deviation figures reveal that the bumptree with a higher *SMALL* value is more dependent on its starting configuration. The ideal setting for *SMALL* with the skin cancer diagnosis, vowel recognition, and diabetes diagnosis problems proved to be 1, and this almost always resulted in a singular matrix when matrix inversion was employed.

SMALL setting	Training set				Test set			
	Average	**Max**	**Min**	**Std deviation**	**Average**	**Max**	**Min**	**Std deviatio n**
1	100	100	100	0	96.3	97.3	96	0.33
7	100	100	100	0	97.5	98.7	97.3	0.5

Table 5.1 - The average percentage performance of the MIF bumptree with all radii set to 1 and the output calculated using the LAF approach for different settings of *SMALL* on the iris problem.

SMALL setting	Training set				Test set			
	Average	Max	Min	Std deviation	Average	Max	Min	Std deviation
7	80	87.1	74.2	5.24	80.3	83.9	75.8	1.95
1	79.8	93.6	74.2	5.1	80.8	83.9	77.4	1.13

Table 5.2 - The average percentage performance of the MIF bumptree with all radii set to 1, employing the LAF output calculation technique for different settings of *SMALL* on the skin cancer diagnosis problem.

Table 5.3 summarises the results attained on the Parity (6) problem by the MIF bumptree with all its radii set to 1 that employed the LAF technique for calculating the output of the network. These results follow the trend identified above, namely that performance is inversely related to the value of *SMALL*. Once again, the standard deviation figures reveal that the bumptree with a higher *SMALL* value is more dependent on its starting configuration.

SMALL setting	Training set			
	Average	Max	Min	Std deviation
1	94.1	100	87.1	4.2
0	98.3	100	92.2	4.0

Table 5.3 - The average percentage performance of the MIF bumptree with all radii set to 1 and the output calculated using the LAF approach for different settings of *SMALL* on the Parity (6) problem.

In an attempt to overcome the problem of singular matrices in the matrix inversion procedure alternative techniques were considered. The solution that was finally adopted was to employ a singular value decomposition (SVD) technique to calculate a pseudo inverse instead of the inverse; this overcame the problem of singular matrices. The SVD technique employed in our study used the Gauss Jordan method (VanDer Rest 1992).

Other methods could have been used such as Csanky's method, the bordering method, the twofold partition method and the general r-fold method of partitioning (VanDer Rest 1992). The Gauss Jordan technique was selected because studies had shown that it performs at least as well as any of the other techniques for a wide range of problems (VanDerRest 1992). Experimentation with the Gauss Jordan method showed that it was robust in solving the problem identified above. This enabled the bumptree neural network to attain the levels of performance described in chapter 4. Appendix B provides pseudo code that further details the process of optimising the Alpha and Beta parameters.

The one-shot learning algorithm that employed the SVD technique was consequently adopted to determine the Alpha and Beta parameters, and the results given in chapters 4 and 6 were achieved with bumptrees employing this approach. It was found that this approach gave reduced training times accompanied by improved performance compared to the learning algorithm that employed standard matrix inversion.

5.3 Use of the Bumptree Structure to Calculate the Output of the Network

The partitioning of the problem space through the techniques discussed in chapter 4, combined with the one-shot learning algorithm utilising the Gauss Jordan SVD technique, allowed the construction of a bumptree neural network that gave robust performance. The completed bumptree comprised functions at a number of levels in the tree, with each function active on only a subset of the patterns. With the network fully constructed the next problem concerned the mechanism for calculating the output of the network. In particular, it was necessary to determine which functions should be involved in calculating the output of the network. For a given pattern there will usually be a

number of active functions distributed throughout the tree; figure 5.4 demonstrates a typical activation pattern. Functions that are active on a pattern have their corresponding nodes shaded. The issue then to be resolved is which of the active functions and which, if any, of the other functions in the network are to be allowed to contribute to the network output.

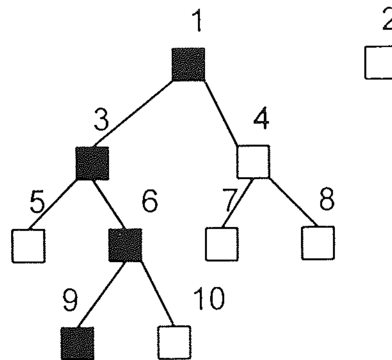

Figure 5.4 - Functions at various levels of the tree are active on individual patterns. This figure demonstrates how 4 of the bumptrees functions are active on the pattern that has just been presented to the network. Here functions 1,3,6 and 9 are active.

Different approaches for calculating the network's output were considered, and these will now be examined. Firstly, the output could be based on the normalised output of every function within the bumptree structure. Using this criterion, the output for any pattern presented to the network in figure 5.4 would be based on the average output produced by all ten functions. However, a better approach is to scale the output of each of the functions. This scaling can be achieved by taking into account the level of activity of each of the gaussian functions on the pattern. The scaling equation is given in 5.9:

$$\sum_{f}^{fMAX}((1/_{GaussTotal})*Gauss_f)*Output_f \qquad (5.9)$$

where f is the function in question, fMAX is the total number of functions in the network, GaussTotal is the summed activation of all the functions in the network, $Gauss_f$ is the activation of the fth function on the pattern, and $Output_f$ is the output of the fth function on the pattern. Although this is a possible approach it was not implemented since it was not considered desirable to have the output of a pattern based partly on the output of functions that had not been trained to deal with the relevant section of the problem space.

A second approach to calculating the output of the network, the all active function (AAF) would be to calculate the output of the bumptree as the normalised output of all the active functions on the pattern in question. Applied to the situation in figure 5.4 the output would be based on that of the functions 1, 3, 6 and 9. The normalised output of these functions could be determined from equation 5.9 by summing over only the active functions.

A third approach, described as the LAF approach in chapter 4, would be to base the output of the network on the output of the active function at the lowest level of the tree. In the case of the situation depicted in figure 5.4 the output of the network would simply be the output of function 9. This approach has the advantage of basing the output on that function determined to provide the best description of that portion of the problem space in which the pattern resides. It might therefore be expected that this approach should give a superior performance as far as training is concerned. Generalisation performance might, however, be degraded by the over reliance on the output of a single activation function. The results of a comparative study of the LAF and AAF approaches is provided below.

A fourth approach would be to sum the output of both the last active function and its "brother", that is, the function created from the same parent function. In the case of the situation displayed in figure 5.4 the output of the network would then be based on the

output of functions 9 and 10, which could be averaged or calculated by a modification of the scaling approach shown in 5.4. This approach was not implemented since it was not considered appropriate to base the output partly on a function untrained for that part of the problem space.

5.3.1 Comparative Results for Output Calculation Techniques

A comparative study of some of the techniques for calculating the output of the network, described in the last section will now be considered for the problems described in chapter 4. For all experiments, the bumptree employed was created with the MIF approach which set all the radius of each function to 1. The performance of each technique was measured initially in terms of its ability to correctly classify the training and generalisation data sets. The same criterion described in chapter 4 to determine the level of performance for the training and generalisation sets was adopted here. Also of interest was the computational time, and expense of required by each approach to reach an acceptable solution.

Figure		Iris	Skin Cancer	Parity 6	Encoder 8	XOR	Diabetes	Vowel Data
Mean	ts	100	79.8	98.3	100	100	76.5	86.5
	gs	97.5	80.8	-	-	-	79.9	73.6
Best	ts	100	93.6	100	100	100	79.5	94.7
	gs	98.7	83.9	-	-	-	81.0	77.2
Worst	ts	100	74.2	92.2	100	100	74.8	73.8
	gs	97.3	77.4	-	-	-	78.0	71.8
Standard	ts	0	5.2	4.0	0	0	1.1	3.0
deviation	gs	0.5	2.0	-	-	-	0.9	1.5

Table 5.4 - The average percentage performance of the MIF bumptree with all radii set to 1 and the output calculated using the LAF output calculation technique.

Table 5.4 summarises the results obtained by using the LAF technique to calculate the output of the network, and table 5.5 shows the results obtained when the AAF technique for output calculation was employed. These results were achieved by bumptrees trained using the one-shot learning algorithm employing the SVD technique. A comparison of the results achieved by these two differing approaches reveals that there was no significant difference in performance on either the training or the generalisation sets for most of the seven problems examined. Employing the LAF approach gave a superior performance by 8% on the Parity (6) problem, but gave an inferior performance of 6% for the skin cancer data. Otherwise the two approaches achieved results within 2% of each other on all the training and generalisation sets. The standard deviation figures demonstrate an equal dependence of both the AAF and LAF output calculation techniques on the initial configuration of the network.

It had been expected that the two differing approaches would produce markedly different results. In particular, it was expected that the approach using the AAF technique would have given better generalisation performance but poorer training performance. The expectation was that because the AAF approach based the output of the network on functions trained across a wider spectrum of the training set the generalisation performance would be better because the output would be influenced by a wider area of the problem space. Conversely, it was felt that since the LAF approach calculated the output based on a single function trained on only a particular area of the problem space the generalisation performance might suffer. However, it was expected that this would improve training performance. That is, it was felt that the LAF approach might result in a degree of overfitting of the training set, whilst the AAF approach might not be able to learn the problems with the same degree of accuracy. The results show, however, that there is little significant difference in performance between the two approaches. The unbalanced nature of the training and generalisation sets, in terms of the proportion of

patterns of each class might have influenced the results. However, with the data sets employed the trend towards parity of results between the two approaches is clear.

Figure	Iris	Skin Cancer	Parity 6	Encoder 8	XOR	Diabetes	Vowel Data
Mean ts	99.6	85.7	90.6	100	100	76.7	86.6
gs	95.6	81.8	-	-	-	79.8	74.1
Best ts	100	93.6	98.4	100	100	79.5	90.0
gs	97.3	83.9	-	-	-	80.7	76.9
Worst ts	97.3	74.2	84.4	100	100	74.5	81.9
gs	94.7	79.0	-	-	-	78.9	70.9
Standard ts	0.7	5.1	3.8	-	-	1.2	3.1
deviation gs	1.0	1.8	-	-	-	0.6	1.6

Table 5.5 - The average percentage performance of the MIF bumptree with all radii set to 1, and the output of the network calculated using the AAF technique.

Problem	Number of functions AAF	Number of functions LAF
Iris	12.6	8.4
Skin Cancer	11.2	3.6
Diabetes	4	3.2
Encoder (8)	2	2
Vowel recognition	164.4	104.2
Parity (6)	35	40.8
XOR	2	2

Table 5.6 - The average number of functions required by the LAF and AAF output calculation techniques.

Another performance metric considered in this study was the number of functions required by each technique in order to optimise performance. As table 5.6 shows, with the exception of the Parity (6) problem, more functions were required when the AAF output calculation technique was employed; this was particularly true for the vowel recognition task. Therefore, the approach that calculates the output using the AAF approach is

computationally more expensive. This is in part because it requires more functions, but also because it is necessary to retain the activation level of the gaussian function for each function that is active on the pattern, as well as retaining the output value that the function produced for the pattern. The process of normalisation can then be carried out on these figures. Calculating the output of the network with the LAF technique simply involves negotiating the tree structure until there exist no further branches and then calculating the output of the final function. Since there is very little difference between the performance of the two techniques the computational requirements would indicate that the output of the network should be calculated on the last active function in the structure.

5.4 Additional Training Mechanisms Included in the Training Algorithm

The approach that the author has developed for building, training and employing the bumptree neural network therefore consists of a technique for partitioning the problem space, a learning algorithm to be applied separately to each function, and a technique for calculating the output of the network. In order for these component parts to be combined satisfactorily to form a complete approach to building and calculating the output of a bumptree some additional mechanisms were introduced. These were concerned with determining when functions should be added to the network, when the process of addition should stop, and when a function added to the network was to be considered live. As defined earlier a function is said to be live if it contributes sufficiently to the network to be considered a part of it for the purpose of calculating its output. In addition, only when a function is live does it need to be considered when determining whether additional functions are required. The mechanisms that were employed for these tasks and the effect that they had on the performance of the network will be examined below.

The first of the additional mechanisms was concerned with the task of deciding when to add functions to the network and when this process should terminate. In order to help make the decision it was decided to use a simple measure of the error for each function for the given training set. If the error was higher than some user defined value, referred to as *ACCEPTABLE*, then the problem space for which the function was responsible was further partitioned by the addition of new functions. The purpose of this approach was to prevent the network adding unnecessary functions in the later stages of the training process, whilst ensuring that function addition took place when required in the earlier stages. The effect on the performance of the network of varying the value of *ACCEPTABLE* was measured for all seven problems. A level of 0.5 was found to give the best overall performance. For instance, for the iris problem when *ACCEPTABLE* was set to 0.5 the bumptree was able to achieve accuracy levels of 100% on the training set and 97.5% on the generalisation set whilst achieving corresponding values of 99% and 94% with *ACCEPTABLE* set to 2.0. The standard deviation values with these results were less than 0.5 for both the training and generalisation sets. When *ACCEPTABLE* was assigned a higher value the network prematurely converged to a solution, whilst a low value tended to increase the number of functions added giving rise to an overfitting of the data. Through extensive experimentation, a value of 0.5 was found to deal satisfactorily with these conflicting requirements.

A mechanism also had to be adopted to determine when a function was live. That is, when it influenced sufficient patterns in the training set to be considered significant. This mechanism employed a parameter termed *SMALL*, and was introduced originally when matrix inversion was an integral part of the learning algorithm. If a function dealt with a greater number of patterns than that assigned to *SMALL* then the function was considered significant; otherwise it was excluded from further splitting and was not involved in calculating the output of the network. The use of an SVD technique largely overcame the

need for the *SMALL* parameter, but the parameter was retained. In only one of the seven problems was a better result attained when *SMALL* was set to a value higher than 1, and this was the iris problem, where a setting of 7 was found to be optimal. In general, a setting of 1 was found to produce the best performance in terms of generalisation as shown in tables 5.1 and 5.2. However, for the XOR, Parity (6) and Encoder (8) problems, as is shown in table 5.3, a setting of 0 was required to attain the best level of performance. The SMALL parameter was used to force a decline in training performance in the hope that generalisation would improve by lowering the number of live functions in the network. Adjusting the parameter upwards only improved performance if the training set was overfitting the data, a problem that was not often encountered with the bumptree.

A mechanism was also introduced to determine whether a network had become trapped in a minimum. This mechanism, which employed a parameter termed *SAME*, was particularly useful when the Forgy non-hierarchical clustering technique was included in the training approach, since this often encountered minima. The ideal setting for this parameter was found through experimentation to be 4, since if the bumptree was still trapped after 4 function additions then escape was not deemed possible. This parameter was used to determine when the bumptree was unable to reach a position where all the active lowest level functions had an error level less than *ACCEPTABLE*.

To conclude, the best parameters for the mechanisms identified above were arrived at through experimentation. These experiments found that on all of the problems studied the best performance was attained when *ACCEPTABLE* was set to 0.5. The iris problem produced the best results when *SMALL* was set to 7, but on the other problems a value of 1 or less for *SMALL* was required. The best setting for the *SAME* parameter was found to be 4.

5.5 The Genetic Bumptree

The bumptree classifier examined in chapter 4 and in the sections above is trained in two stages. Firstly, the structure of the tree, the number of functions, their positions in the tree, and their centres and radii, are determined. Secondly, each function's associated weight and bias (Alpha and Beta) parameters are optimised using a one-shot learning algorithm to minimise an error measure for the training set. A number of variations for both positioning functions on the problem space and for minimising the error of each function have been examined above. A further variation that was examined in this study was the use of a genetic algorithm to arrive at the Alpha and Beta parameters for each function and/or to position the functions on the problem space.

Genetic algorithms (GA's) were discussed briefly in chapter 2, when their use in the optimisation of the MLP was examined. They are optimisation algorithms based on the mechanics of Darwinian evolution that were introduced originally by Holland (1975). They differ from other optimisation algorithms in that they operate on a coding of the parameter space rather than directly on the space itself, and search stochastically from many points at once. In the GA, parameters to be optimised are coded as a string of digits which is usually binary in nature; the string is termed a chromosome. A population of many such chromosomes is initially generated at random. Thereafter, evolution proceeds as follows. Initially, a measure of fitness is calculated for each member of the population, by decoding its chromosome and evaluating the associated point in the parameter-space within the system whose parameters are to be optimised. Then a number of parents are selected from the population, with a chromosome's probability of selection being proportional to its fitness. New offspring chromosomes are generated from fit parents by the application of genetic operators, typically: crossover, which swaps sections of chromosome from two parents, and mutation, which randomly modifies one or more

digits on a chromosome. All offspring so created are introduced into the population, usually replacing the least-fit of the existing individuals, to form the next generation. This process is repeated and with each generation the population's mean fitness increases. The GA has been found to be able to discover near maximal points in high dimensional, discontinuous parameter spaces in a relatively small number of generations.

Recently there has been a growing overlap of research interest between the fields of neural networks and genetic algorithms (GA's). Researchers have sought to combine the GA and neural network paradigms in hybrid learning systems with varying degrees of success (Jones 1993; Yao 1992). The exact nature of the hybridisation has varied, from using a GA to search the weight space of a neural network (Whitley, Dominic and Das 1991), to using the GA to optimise network architecture (Harp, Samad and Guha 1989) to systems in which both architecture and weights are optimised genetically (Bornholdt and Graudenz 1991). One factor which remains constant across the vast bulk of the published work in this area, however, is that the neural network model chosen is the MLP. However, the application of GA's to other types of neural network models is a possibility, and it offers an alternative technique for training the bumptree neural network.

The GA can be employed as a complete learning system to determine both the parameters of the functions in terms of their centres and radius, and the Alpha and Beta values (Williams *et al.* 1993). Alternatively, it can be employed in conjunction with the one-shot learning algorithm. In the later case the GA is used to position the functions on the problem space and the one-shot learning algorithm is used to arrive at the required Alpha and Beta parameters (Williams *et al.* 1994). The GA utilised by the hybrid GA-bumptree whose results are examined below is detailed in Appendix C.

Employing the GA to optimise the bumptree system as a whole had the advantage that the tree structure was not fixed entirely according to the distribution of the training data, but was optimised simultaneously with the function's weight and bias parameters, the whole process being based on the minimisation of a global error. It was hoped that by considering a large number of parameters such a system would be able to provide better performance. Furthermore, the use of a GA would also avoid the problem with singular matrices when matrix inversion was employed. The bumptree that employed the GA to optimise the structure of the tree whilst allowing the one-shot learning algorithm discussed above to determine the Alpha and Beta values was developed to allow an investigation of whether the genetic bumptree was placing the functions in a more advantageous manner. It was also felt that the use of the one-shot learning algorithm would produce better results than basing the Alpha and Beta values on a random, if guided, walk through the problem space by the GA. The genetic bumptree adhered to the constraints imposed by Omohundro's approach (section 4.4) on the size and location of the functions within the network.

The results achieved by the two genetic bumptrees will now be examined. The genetic bumptree that employed a GA to both position the functions on the problem space and to optimise the weight and bias parameters was employed on the Iris problem. This was able to achieve an average performance level of 97% correct on the training set and 90% on the generalisation set over ten trials. In this set of experiments a population size of 400 was used, of which 10% were replaced each generation. In initial tests the genetic bumptree was found to reach near-convergence in around 150-200 generations and so, for these experiments 300 generations were allowed before training was terminated. The same technique described in chapter 4 was employed to determine the point at which the performance of the network should be calculated. The average performance of the various bumptrees described in chapter 4 ranged from 95.5% to 100% on the training set and

from 92% to 97.5% on the generalisation set. Hence, on the whole this type of genetic bumptree produced slightly inferior results compared to bumptrees trained in a more standard manner.

It was hoped that the genetic bumptree that combined the use of a GA to position the functions on the problem space and the one-shot learning algorithm to determine the weight and bias parameters would be able to produce superior results. Table 5.7 summarises the results achieved by this approach for the Iris, Parity (6) and Vowel recognition problems. Once again a population size of 400 was employed with 10% of the generation being replaced. A maximum of 500 generations was allowed and the results averaged over 10 trials for each data set. The results reveal that this genetic bumptree was able to produce better results than the initial genetic bumptree on the Iris problem. Indeed, the results attained on the Iris problem were comparable to the results attained by the standard bumptrees discussed in chapter 4, but its result was found to be inferior for the Parity (6) and vowel recognition problems.

The use of a GA therefore offers an alternative approach for constructing and training a bumptree. The genetic bumptrees developed to date have been constructed to adhere to the constraints imposed by Omohundro on the dimensions of the functions, and it may be possible to achieve improved performance by removing these constraints in a similar way to that discussed in chapter 4. Even if this could be achieved, however, the training time required would increase dramatically even with the GA concerned solely with positioning the functions on the problem space. The scope for the genetic bumptree to improve the performance of the bumptree to a level superior to that attainable by the best standard bumptree developed in chapter 4 may therefore be limited.

Data set	Average performance on the training set	Standard deviation	Average performance on the generalisation set	Standard deviation
Iris	99.7	0.5	96.3	0.8
Parity (6)	77.2	8.4	-	-
Vowel recognition	78.4	1.5	75.0	2.4

Table 5.7 - The average percentage performance of the genetic bumptree that employs the genetic algorithm to determine the dimensions of the functions and the one-shot learning algorithm to optimise the weight and bias values.

5.6 Summary

This chapter has addressed all the issues involved with constructing and using a bumptree neural network other than the task of partitioning the problem space, which was examined in chapter 4. Attention focused initially on the learning algorithm to be employed to optimise the weight and bias (Alpha and Beta) parameters of each individual function to minimise a squared error measure on the training set. The learning algorithm that was developed for this task was a one-shot algorithm that optimised these parameters through a matrix multiplication process applied once to each function. Initial problems were encountered with singular matrices when matrix inversion was incorporated in this technique. However, once the Gauss Jordan SVD technique was used instead of matrix inversion the algorithm was able to produce, alongside the MIF approach to function centering and constraint, networks that were able to produce comparative performance to those produced by MLP and RBF networks. A comparative study with the MLP and RBF networks will be presented in chapter 6.

The second major issue to be examined in this chapter concerned the way in which the output of the network should be calculated. The LAF approach calculated the output as

the output of the live function at the lowest level of the tree active on the pattern. The AAF approach calculated the output as the normalised output of all the live functions that were active on the pattern. Both of these approaches proved to perform well on the seven problems examined in this study, and neither had a real performance advantage in terms of correct classifications over the other. However, the AAF approach was computationally more expensive, since it required more functions to attain a given level of performance. The LAF approach is, therefore, the preferred approach. An important point about the output that the bumptree produces is that it is a winner takes all system, and as such requires one output per class of pattern.

The third issue concerned the mechanisms for determining when to add functions, when to stop their addition and when to consider a function live. The examination of these parameters showed that on the whole the parameters were problem insensitive. That is, the best results were attained on the seven problems with parameters that varied very little.

This chapter was finally concerned with the genetic bumptree. The genetic bumptree was constructed through the use of a genetic algorithm rather than through any of the function centering and constraint techniques discussed in chapter 4. In addition, in the initial version the weight and bias (Alpha and Beta) parameters for each of the functions was arrived at through the use of a genetic algorithm. It proved possible to employ a GA to construct a bumptree neural network, and as the results in section 5.5 demonstrate, its performance is comparable to that attained by the more standard bumptree. There is, however, one issue on which the genetic bumptree fails hopelessly to match the performance of the standard bumptree, and that is training time. The one-shot learning algorithm takes roughly under a second to add a function to the network with the iris problem (on Sparc Classics), and requires at most 9 function additions to solve the

problem. The genetic bumptree requires multiple computers of a similar processing power for periods of time usually exceeding 24 hours to reach a solution. Hence, the genetic bumptree can be seen to take almost a day longer to train to a solution for the simple iris problem. This increase in training time is also seen across the other problems.

The bumptree neural network classifier has been introduced in chapters 4 and 5, as have the various approaches employed to construct, train and employ it. In addition, the performance of the various approaches has been studied. However, in order for the bumptree to be able to make a contribution as a neural network classification system it needs to be compared to existing classifiers; the results of this comparative study are described in chapter 6.

Chapter 6

Comparative Study of the Bumptree, RBF and MLP Networks

6.1 Introduction

In previous chapters we have described the development of the bumptree neural network and shown that it can be successfully applied to a range of problems. As a further investigation of its performance we have conducted a number of comparative studies against the more traditional MLP and RBF networks (Bostock & Harget 1994). This chapter describes the results obtained.

The MLP employed in this study was trained with the standard back propagation learning algorithm (Rumelhart, Hinton and Williams 1988) and employed a momentum term and learning rate that were constant throughout the entire training process. For all problems the learning rate was fixed at 0.25, and the momentum term set to 0.9. For each problem the MLP's architecture consisted of an input layer, an output layer and a single hidden layer. The starting weights and bias values were assigned random values between 2 and -2. There still remained the issue of how many hidden units should be employed for each of the problems, and this figure was arrived at empirically. Each of the problems was solved with different numbers of hidden units and the best performance recorded.

For the RBF network the program included in the Autonet package developed by Recognition Research (1993) was employed. This approaches the task of placing the functions on the problem space by selecting points from either the actual data set or by selecting random points within the area covered by the data set. The positioning of the functions on the problem space was initially carried out using both of these approaches, but the better generalisation performance was attained by using sample points within the

problem space, so this method was chosen for all studies. There was also a choice of four different functions that could be employed by the RBF to partition the problem space, namely, gaussian, multi-quadratic, inverse multi-quadratic and thin plate spline functions; these functions were described in chapter 3. These function types were employed to produce the results discussed below. In addition, the number of functions to be employed was arrived at empirically. Each of the problems was solved with different numbers of functions and the best performance recorded.

A comparative study of network performance was conducted on a wide range of problems, including XOR, Parity (6), vowel recognition, the diagnosis of diabetes and the diagnosis of skin cancer. Particular attention was given to the average number of correct classifications achieved on the training and generalisation sets and the computational time required by the network to train to a solution. Also of concern is the time taken by the trained network to respond to a query. The decision when to determine the performance of the network during the training cycle was addressed in section 4.8.1, and the solution proposed there has been adopted again in this section.

A potential limitation of this study is that the data sets have not split the patterns of the different output classes evenly between the training and generalisation sets. That is, the percentage of patterns of each output class in the training and generalisation sets differs. This uneven distribution may have had an impact on the results reported in this section, a point to be considered when examining the results. The choice of a classification measure based on the percentage of correct classifications did not compensate for any possible bias introduced by the make up of the training and generalisation sets. A measure of the percentage of each output class correctly classified would have provided additional useful information. However, the split of the data was the same for all the neural network types

and as such allowed a consistent and representative picture to be produced across the different network types.

An additional consideration that is relevant to the results reported in this section is that the performance of the MLP network was achieved by varying only a single parameter, namely the number of hidden units. It may have been possible to achieve improved results by adjusting some of the other parameters. For instance, the learning rate was set to a value of 0.25 across all problems, and it may have been possible to improve performance by altering this figure. In addition, the RBF network was employed in a very "black box" manner. That is, this study did not conduct an in-depth investigation of the RBF network. It instead utilised the standard RBF provided by the Autonet package to attain comparative figures. The number of functions, their radius and the dimensions of these, were altered for the various problems, but the results could possibly have been improved by further adjusting these and other parameters in the training algorithm. The MIF bumptree did not adjust the parameters in its learning algorithm in the course of this study, but these parameters could be considered to be nearly optimal following the investigations conducted in chapters 4 and 5. An additional consideration is that the earlier chapters have revealed that the performance of the bumptree is fairly insensitive to these parameters, whilst the performance of the RBF and MLP networks is known to be dependent on various arbitrary parameters. The insensitivity of the bumptree to parameters in the learning algorithm can be cited as an advantage that it possesses. However, it does not remove the concern that the results provided in this chapter may have been influenced by the decision to only adjust certain parameters of the MLP and RBF networks. Therefore, the results provided in this section can be viewed as providing a representative comparison but cannot be claimed to provide a definitive or critical comparison.

6.2 Highly Non-Linear Problems Without a Test of Generalisation Performance

This section will examine how the three neural networks performed on highly non-linear problems with no accompanying test of generalisation ability. The problems in this class were the XOR, Parity (6) and Encoder (8) problems. The aim with these problems was simply to model the training set to optimal accuracy with as small a network as possible. The problems in this class were highly non-linear in nature, and therefore a good test of the ability of the various network types to model difficult problem spaces during training. Furthermore, they have been widely studied by others and provide a good initial benchmark upon which to compare the performance of the bumptree to that of other network types.

With both Parity (6) and XOR the network was learning to ensure even parity. The XOR problem had two inputs and one output, with the output being concerned with ensuring an even number of ones in total for the three units. That is, if the two input units had the same value then the output unit should register a value of 0, otherwise a value of 1. The XOR problem had four patterns in the training set. The Parity (6) problem employed in this study had six inputs and two outputs. It is more common for Parity (6) to have only a single output, like XOR, but as the bumptree takes the most active of the output units as the output of the network, it requires one unit representing a 0 output and another representing an output of 1. The task of the output units with the Parity (6) problem was once again to ensure an even number of ones in the network. For instance, with an odd number of ones in the input the output unit signifying an output of 1 should be active, otherwise the other output unit should be active. The Parity (6) problem had sixty four patterns in the training set. The Encoder (8) problem required the network's output to directly reproduce the input. A single input unit is active for each of the patterns that

comprised the problem. There are eight input units, eight output units, and eight patterns in the training set with each pattern containing a single value of 1 in a different position.

6.2.1 The Radial Basis Function Network

The performance of the RBF network on the XOR, Parity (6) and Encoder (8) problems will now be examined. For each of these problems the functions were placed on the problem space at points within the area covered by the training set. The performance of the RBF network on the XOR problem was very encouraging, with all the functions, excluding the thin plate spline, giving a completely accurate mapping of the training set. For the thin plate spline function the average performance on the XOR problem was found to be 88.3%. Specifically it achieved 100% accuracy for 53% of the runs, and 75% accuracy for the remaining runs. The performance of the RBF network seemed to be invariant to the number of functions, since for each function type the network was trained with two, three and four functions and similar results were obtained.

The performance of the RBF networks employed in this study on the Parity (6) problem was less encouraging. None of the function types was able to achieve an average performance level of complete correctness for this problem. The best average performance was achieved by an RBF that employed inverse multi-quadratic functions; this achieved an average performance level of 92.1% correct using 60 functions to partition the problem space. Only 10% of the runs were able to produce the correct output for all the patterns in the training set. The non-local thin plate spline and multi-quadratic functions were able to achieve an average performance level of 87.4% and 88.0% respectively. The thin plate spline correctly classified all the patterns in 20% of the runs, whilst the multi-quadratic achieved this in just 10% of the runs. The RBF that performed

worst on Parity (6) was the one that employed gaussian functions to partition the space. This achieved an average performance level of 81.4%, and never correctly classified all the patterns.

The performance of the RBF networks on the Encoder (8) problem was very similar to that attained on the Parity (6) problem. Once again none of the function types was able to achieve an average performance of completely correct classification. The best average level of performance was attained by the RBF network that employed multi-quadratic functions, which gave an average performance level of 82.5%, with all the patterns being classified correctly on 30% of the runs, when 8 functions were employed to partition the problem space. Using gaussian and inverse multi-quadratic functions, with 8 functions, performance levels of 71.3% and 78.8% respectively were attained. The RBF that employed thin plate splines to partition the problem space was only able to attain an average performance of 57.5% on the training set.

Problem	Statistics	Thin plate spline	Multi quadratic	Inverse multi quadratic	Gaussian
XOR	**Average**	88.3	**100**	100	**100**
	Functions	2-4	**2-4**	2-4	**2-4**
	Standard deviation	12.5	**0**	0	**0**
Parity (6)	**Average**	87.4	**88**	92.1	**81.4**
	Functions	60-64	**60-64**	60-64	**60**
	Standard deviation	11.8	**8.0**	4.6	**11.5**
Encoder (8)	**Average**	57.5	**82.5**	78.8	**71.3**
	Functions	8	**8**	8	**8**
	Standard deviation	25.7	**18.6**	12.8	**15.4**

Table 6.1 - The average percentage performance of RBF's employing different function types.

These studies showed that the performance of RBF networks employing the various function types was very much problem dependent and that no particular function type consistently gave the best performance. The average results attained by RBF's employing the various function types are displayed in figure 6.1 and summarised in table 6.1. The standard deviation figures in table 6.1 demonstrate that the performance of the RBF's was very dependent on where the functions were positioned on the problem space.

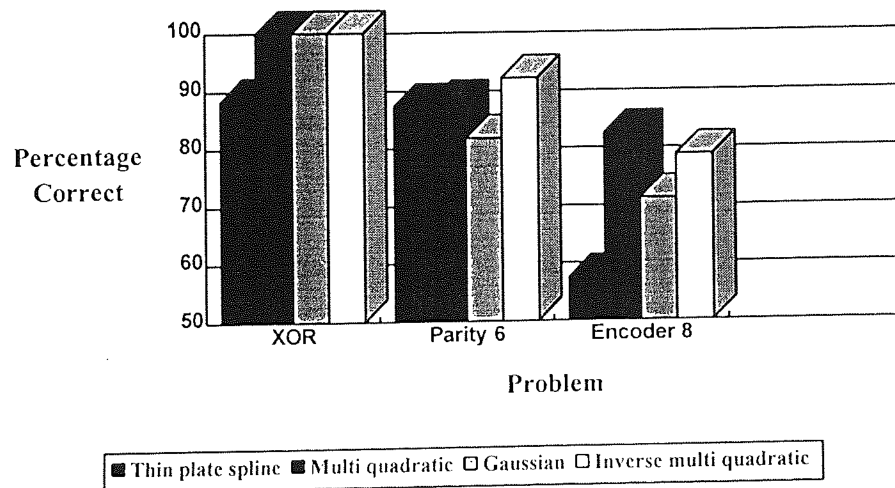

Figure 6.1 - The average percentage performance of RBF networks using the various function types with the optimum number of functions.

Figure 6.2 shows the variation in performance when different numbers of functions were employed on the Encoder (8) problem. The results show an improved performance when the number of functions used is increased. A similar performance trend was observed with the results for the Parity (6) problem. In examining the XOR, Parity (6) and Encoder (8) problems it was decided not to employ more functions than there were patterns in the training set. That is, for Encoder (8) the number of functions was limited to 8. However, it is possible that training performance might have been improved by employing more

than this limited number of functions. Therefore, the results given here may not represent the best possible results that could have been attained with the RBF network.

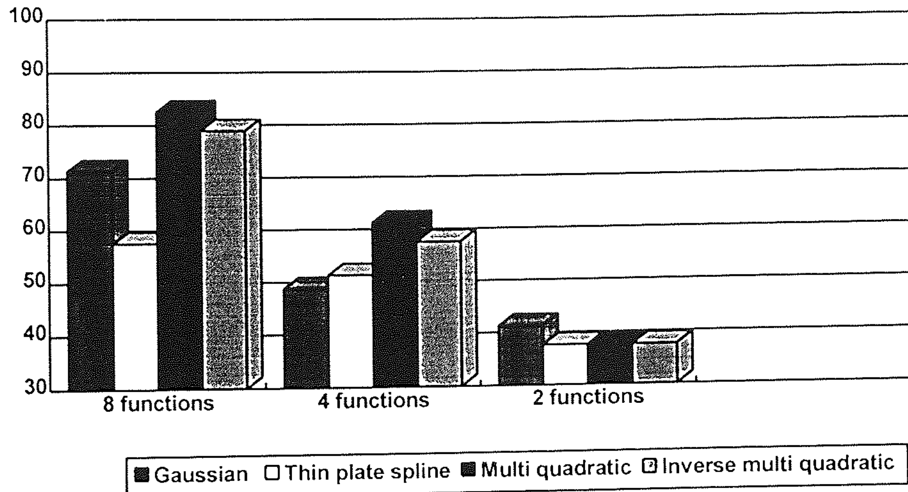

Figure 6.2 - The average percentage performance of RBF networks employing various numbers and types of functions on the Encoder (8) problem.

6.2.2 The Multi-Layer Perceptron

The MLP was tested on the XOR, Parity (6) and Encoder (8) problems with networks that employed differing numbers of hidden units. The performance of the MLP on the XOR problem was excellent when the correct number of hidden units was used. However, when insufficient hidden units were employed the MLP was unable to model the problem space, and performance degraded. Table 6.2 summarises the results obtained. This reveals that a network containing 4 hidden units was required for 100% correctness for all trials. The standard deviation figures given in table 6.2 reveal that the MLP was very dependent on the networks initial configuration when 2 or 3 hidden units were employed. It also

reveals that trials that did reach a level of complete correctness on the problem required fewer iterations when 4 hidden units were employed than when 2 or 3 were used. The average number of iterations given in table 6.2 was calculated only from those runs that correctly classified 100% of the patterns.

Hidden units	100% correct trials	Average % correct	Maximum % correct	Minimum % correct	Standard deviation	Average iterations to convergence
1	0	25	25	25	0	-
2	60	80	100	50	24.5	259
3	70	85	100	50	22.9	343
4	100	100	100	100	0	222

Table 6.2 - The average percentage performance of the MLP on the XOR problem.

The performance of the MLP on the Parity (6) problem was determined for network's containing between one and ten hidden units. The performance was found to be similar to that obtained on the XOR problem; not surprising given the similar nature of the problems. The performance of the different sized networks on the Parity (6) problem is summarised in table 6.3. This reveals that MLP's that employed 1 or 2 hidden units were never able to correctly classify the complete training set. Indeed, when 4 hidden units were employed only 50% of the trials achieved complete correctness. It was not until 10 hidden units were employed that complete correctness was achieved. This trend of improving network performance with increased network size was also observed for the XOR problem. The standard deviation figures in table 6.3 reveal that the MLP became less influenced by its starting configuration when employing more hidden units. The average number of iterations given in table 6.3 was calculated only from those runs that correctly classified 100% of the patterns.

The performance of the MLP on the Encoder (8) problem was once again very good. All the networks that employed two or more hidden units were able to correctly classify all the patterns for all trials. When one hidden unit was employed the MLP was only able to correctly classify 50% of the patterns. As the number of hidden units was decreased the average number of iterations required to achieve 100% correctness increased. For instance, when three hidden units were employed an average of 1937 iterations was required, but only 362 iterations were required when 8 hidden units were used. However, this was compensated for by the fact that with more hidden units each iteration is more computationally complex.

Hidden units	100% correct trials	Average % correct	Maximum % correct	Minimum % correct	Standard deviation	Iterations to convergence
1	0	8.9	65.6	0	26.4	-
2	0	81.1	89.1	32.8	17.1	-
4	50	90.5	100	65.6	11.0	2532
6	10	97.2	100	89.1	2.9	1766
8	90	99.6	100	98.4	0.5	1094
10	100	100	100	100	0	1133

Table 6.3 - The average percentage performance of the MLP on the Parity (6) problem.

These studies have shown that the performance of the MLP is very good when the network contains sufficient hidden units, but degrades sharply when fewer than required are used. They have shown that determining the correct number of hidden units for the MLP network is of considerable importance particularly for problems of a highly non-linear nature. The number of hidden units employed by the MLP also influences the number of iterations required to reach a solution, as illustrated by the Encoder (8)

problem. However, another important point that needs to be considered is that the more hidden units that are employed the more computationally expensive each iteration becomes, because for each iteration there are more weighted connections to update.

6.2.3 Classification Performance of the MLP, RBF and Bumptree Neural Networks

This section will provide a comparative study of the average classification performance of the MLP, RBF and bumptree artificial neural networks on the XOR, Parity (6) and Encoder (8) problems. In considering the results it is necessary to consider that only limited number of parameters for the RBF and MLP have been adjusted, and that improved performance might have been possible if additional parameters had been adjusted. For each network results attained with the optimum architecture will be examined. In the case of the bumptree this means using the MIF technique, with the functions radii in each dimension set to 1, the output of the network calculated with the LAF technique, and the one-shot learning algorithm using the SVD technique employed. This will be referred to as the MIF bumptree. Figure 6.3 provides a graphical summary of how each of these networks performed in terms of the percentage of correct classifications recorded on the various problems.

Overall the most satisfactory performance was achieved for the XOR problem. The MLP was able to correctly classify all the patterns when more than four units were employed in the hidden layer, and the RBF network was able to correctly classify all the patterns when more than two functions were used for all function types except the thin plate spline. The MIF bumptree neural network was able to correctly classify all the patterns; a result that was repeated for the bumptree networks employing any of the centering techniques discussed in chapter 4. Hence, all three approaches were able to correctly classify all

patterns when an optimum network architecture was used. However, in the case of the MLP and RBF networks deviation from the optimum architecture caused a rapid degradation in performance. In contrast, the performance of the bumptree neural network was found to be relatively less sensitive to parameter settings; the same set of parameters being used for the XOR, Parity (6) and Encoder (8) problems.

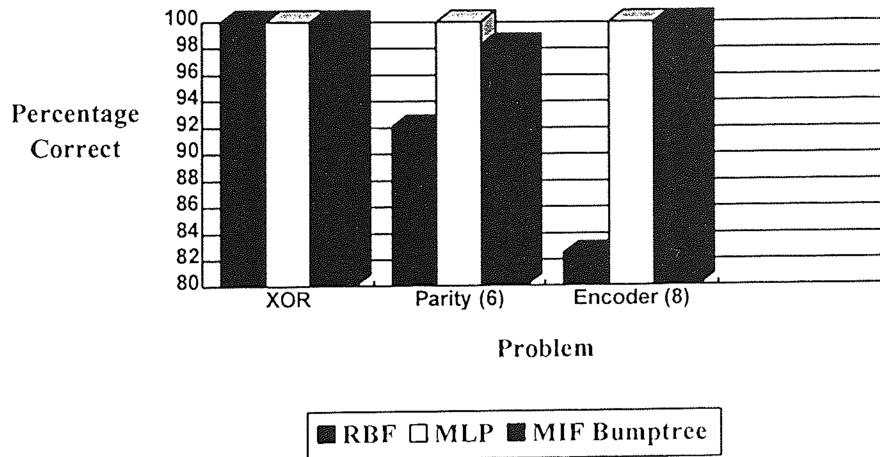

Figure 6.3 - The average percentage performance of the MLP, RBF, and MIF bumptree neural networks on the Parity (6), Encoder (8) and XOR problems.

For the Parity (6) problem the MLP network was able to achieve completely correct classification for the training set when the appropriate number of hidden units were employed. However, as with the XOR problem performance degraded when fewer hidden units were used. The best average performance attained by the RBF network was 92.1% of the patterns correctly classified, achieved when the inverse multi-quadratic function was employed. The MIF bumptree was able on average to correctly classify 98.3% of the patterns. All three networks, therefore, performed to a similar level, but only the MLP achieved 100% correct classification for every trial conducted when using the correct size network. The other two network types were not able to achieve this level of average

performance. The MIF bumptree achieved a better average classification performance to the RBF, although the difference could in part be accounted for by the standard deviation figures, and the limiting of the number of functions in the RBF network to 64.

The Encoder (8) problem once again produced diverse results across the three differing networks. For all trials the MLP was able to correctly classify all the patterns with more than one hidden unit, and a similar level of performance was obtained by the MIF bumptree. The RBF network, however, gave inferior performance; using multi-quadratic functions it could only achieve an average performance of 82.5% correct classifications. The performance of the RBF network was greatly influenced by the positioning of the functions on the problem space, as revealed by the standard deviation figure given in table 6.1.

In conclusion, the classification performance of the MIF bumptree on these highly non-linear problems was found to be very encouraging. The issues of training time and training complexity, examined in section 6.8, reveal that the promising classification results were achieved with a minimum of effort compared to the other approaches. The performance of the MIF bumptree and RBF network were equally dependent on the positioning of the functions on the problem space with the Parity (6) problem. However, for Encoder (8) problem the performance of the RBF was influenced to a greater degree by the initial placing of the functions on the problem space.

6.3 The Diabetes Diagnosis Data Set

Further experiments were conducted in order to test the generalisation performance of the networks, beginning with the diabetes diagnosis data set. This data set concerns the occurrence and diagnosis of diabetes in the Pima North American Indian tribe. In particular, all the patients represented in the data set are females of at least twenty one years of age of Pima Indian heritage living near Phoenix, Arizona, USA. The data set contained 400 patterns in the training set and 368 patterns in the generalisation set with each pattern being described by eight inputs and two outputs. Each of the patients represented in the data set had been diagnosed as either diabetic or non-diabetic, and the problem was to predict whether a patient would test positive according to the World Health Organisation criteria (i.e. if the patient's 2 hour post-load glucose is at least 200 mg/dl) given a number of physiological measurements and medical test results; the attribute details are given in figure 6.4. In order to test the performance of the MLP, RBF and MIF bumptree neural networks on the diabetes problem the data was normalised so that the values for each attribute fell within the range 0-1.

---

1 - number of times pregnant
2 - plasma glucose concentration in an oral glucose tolerance test
3 - diastolic blood pressure (mm/Hg)
4 - triceps skin fold thickness (mm)
5 - 2-hour serum insulin (mu U/ml)
6 - body mass index (kg/m2)
7 - diabetes pedigree function
8 - age (years)

Figure 6.4 - The eight attributes for the diabetes data set.

Of the 768 patterns contained in the data set, 500 (65%) represented diabetic patients and the remaining 268 (35%) non-diabetic patients. The training set consisted of 248 (62%) patterns representing diabetic patients, and 152 (38%) representing non-diabetic patients. For the generalisation set the corresponding figures were 252 (68%) and 116 (32%).

6.3.1 The Radial Basis Function Network

RBF networks employing either localised functions in the form of gaussians, or non-localised functions, in the form of thin plate splines were tested on this problem. The functions were again placed on the problem space at points within the area covered by the data set.

Number of functions	Average % correct training set	Standard deviation	Average % correct generalisation set	Standard deviation
400	100	0	74.7	0
200	88.8	1.3	75.7	1.3
100	82.4	1.0	77.4	1.2
50	79.1	1.1	78.6	1.2
25	76	0.6	78.9	1.1

Table 6.4 - The average percentage performance of RBF networks using thin plate splines on the diabetes data set.

The RBF network that used thin plate splines was constructed with between 25 and 400 functions. The results for these different size networks are summarised in table 6.4. These results show that whilst the best average performance on the training set was obtained with 400 functions, the best generalisation performance of 78.9% was obtained with 25

functions. Figure 6.5 shows a clear trend, in that training performance improves when the number of functions increases, whilst generalisation performance degrades. This suggests that with more functions the RBF overfitted the training set at the expense of generalisation performance.

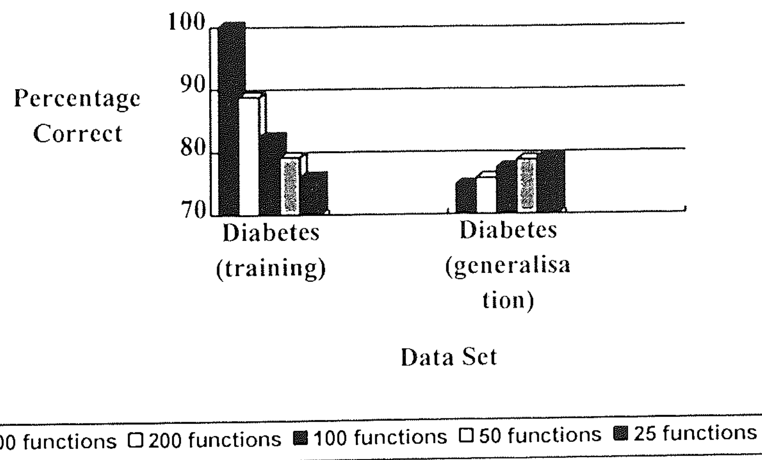

Figure 6.5 - The average percentage performance of the RBF network that employed thin plate splines on the diabetes data set.

The RBF network employed between 25 and 150 gaussian functions, and the results obtained from these differing size networks are given in table 6.5. The best average performance on the generalisation set was 74.7% obtained using 100 gaussian functions, whilst the best training set performance was 83.4% obtained with 150 functions. Our studies showed that the performance of the RBF network was sensitive to the number of functions used whatever their type. When too many functions were employed the RBF network tended to over fit the training set and degrade generalisation performance, whilst if too few were employed the network was unable to achieve satisfactory performance on either data set.

Number of functions	Average % correct training set	Standard deviation	Average % correct generalisation set	Standard deviation
150	83.4	0.5	74.3	1.1
100	78.9	1.0	74.7	2.3
50	73.4	1.5	73.6	1.8
25	70.7	1.2	72.7	1.6

Table 6.5 - The average percentage performance of RBF networks using gaussian functions on the diabetes data set.

6.3.2 The Multi-Layer Perceptron

The MLP was tested on the diabetes diagnosis problem with the hidden layer ranging from 2 to 40 units; the results for these different sized networks are given in table 6.6. The best average generalisation performance of 78.9% was obtained with a network containing 16 hidden units, whilst 40 hidden units were required to give the best performance on the training set of 86.6%. The performance on both data sets improved consistently as the number of hidden units was increased from 2 to 16. When more than 16 hidden units were employed, performance on the training set continued to improve whilst performance on the generalisation set worsened, probably due to an overfitting of the training set. These results bear a strong resemblance to those obtained with the RBF network, even to the extent of giving similar generalisation performance.

Number of hidden units	Average % correct training set	Standard deviation	Average % correct generalisation set	Standard deviation
40	86.6	3.2	78.7	0.8
24	85.6	3.3	78.7	0.7
16	82.5	4.4	78.9	1.5
8	80.9	2.6	78.6	1.7
4	79.7	3.2	78.6	1.4
2	76.5	1.2	78.1	1.5

Table 6.6 - The average percentage performance of the MLP on the diabetes data set.

6.3.3 Classification Performance of the MLP, RBF and Bumptree Neural Networks

This section will provide a comparative study of the average classification performance of the MLP, RBF and MIF bumptree neural networks on the diabetes diagnosis problem. The results summarised in table 6.7 show that the three network types produce similar performance levels. The MIF bumptree gave slightly the best generalisation performance, by 1%, with the MLP and RBF networks giving similar performance. In addition, only the MLP gave better performance on the training set than the generalisation set when the generalisation performance in table 6.7 was achieved. Finally, all three networks were prone to overfitting the training data when too many functions or hidden units were used.

With regards to parameter settings it was found that the same parameter values could be used for the MIF bumptree (*SMALL*=1, *ACCEPTABLE*=0.5) as used in the skin cancer and vowel recognition problems. These figures are similar to those adopted for the XOR, Parity (6) and Encoder (8) problems, and underline the robustness of the bumptree. In contrast, the performance of the RBF and MLP networks was found to be sensitive to parameter settings. This sensitivity was also reflected to a degree in the standard

deviation figures, which demonstrate that the performance of the MLP network was very dependent on the initial configuration of the networks weight and bias parameters. The ability of the network to consistently perform at a given level regardless of initial network configuration is relevant in addressing performance of the networks. The standard deviation figures provide information on this consistency. In addition, they provide information concerning the degree to which the average performance has been influenced by outlying results.

Network type	Average % correct training set	Standard deviation	Average % correct generalisation set	Standard deviation
MLP	82.5	4.4	78.9	1.5
RBF	76	0.6	78.9	1.1
MIF Bumptree	76.5	1.1	79.9	0.9

Table 6.7 - The average percentage performance of the MLP, RBF and MIF bumptree neural networks on the diabetes data set for the networks with the best average generalisation performance.

6.4 The Skin Cancer Diagnosis Data Set

The skin cancer diagnosis data set also allowed generalisation performance to be evaluated. The skin cancer diagnosis data set concerned the occurrence and diagnosis of skin tumours in Britain, and was provided by Ela Claridge[1] and Per Hall[2]. The data set was comprised of 62 patterns in both the training and generalisation sets, with each pattern being described by 3 inputs and 2 outputs. Each of the patients represented in the

[1]Ela Claridge, School of Computer Science, University of Birmingham, Birmingham.
[2] Per Hall, Department of Plastic Surgery, Wordsley Hospital, West Midlands.

data set had had a skin tumour removed and then diagnosed as either benign or malignant. The three inputs represented the bulkiness, the textural fractal dimension and the structural fractal dimension of the tumour. All input data was normalised between the values of 0 and 1.

If the networks could provide high generalisation performance then this would provide a means of detecting melanoma, thus avoiding the unnecessary surgery of benign tumours. In an earlier study with a modified version of the MLP (Bostock *et al.* 1993), only a single output unit was used to indicate whether a tumour was benign or malignant. In this study, however, 2 output units were used, since the output of the bumptree network is given by the output unit with the highest activation. One output unit represented malignant tumours and the other benign. The training set consisted of 37 malignant patterns (60%) and 25 benign patterns (40%), and the generalisation set consisted of 31 (50%) malignant patterns and 31 (50%) benign patterns. This was not balanced in terms of the distribution of patterns from the differing output classes, and the results attained by the different networks may have been influenced by this uneven distribution.

6.4.1 The Radial Basis Function Network

The performance of the RBF network on the skin cancer diagnosis data set will now be examined for networks employing either gaussian, thin plate spline, multi-quadratic or inverse multi-quadratic functions The functions were placed on the problem space at random points within the area covered by the training set.

With thin plate splines a number of networks were constructed with between 10 and 62 functions, and the results are given in table 6.8. These show that the network was able to

correctly classify all members of the training set with 62 functions, and that the best average performance on the generalisation set (79.5%) was achieved when 20 functions were employed. Table 6.8 shows that the average performance on the training set improved as the number of functions increased and that average performance on the generalisation set peaked when 20 functions were employed and deteriorated as the number increased. These results show that the network remained strongly influenced by the initial configuration of the network, regardless of the number of functions employed.

Number of functions	Average % correct training set	Standard deviation	Average % correct generalisation set	Standard deviation
62	100	0	71	0
50	97.6	2.2	73.7	2.8
40	95.2	2.0	74.7	2.5
30	92.9	2.7	77.3	3.1
20	89.7	1.7	79.5	2.9
10	83.2	3.7	78.6	2.9

Table 6.8 - The average percentage performance of RBF networks employing thin plate splines on the skin cancer data set.

A similar set of experiments was conducted with gaussian functions, and the results shown in table 6.9 were obtained. These show that the best average performance on the training set (100%) was achieved when 62 functions were employed, and that between 10 and 20 functions were required for the best average generalisation performance (76.8%). When the number of functions increased beyond 20, the performance on the training set improved, whilst performance on the generalisation set worsened, suggesting that overfitting of the training set had occurred. The results also reveal that the performance of the network was strongly influenced by its initial configuration.

Number of functions	Average % correct training set	Standard deviation	Average % correct generalisation set	Standard deviation
62	100	0	66.1	0
40	94.8	2.6	72.3	3.9
20	89.8	1.8	76.8	2.0
10	85	3.1	76.8	3.0

Table 6.9 - The average percentage performance of RBF networks employing gaussian functions on the skin cancer data set.

The results for the RBF networks that employed multi-quadratic functions are given in table 6.10, and for inverse multi-quadratic functions in table 6.11. Both results show the trend observed in tables 6.8 and 6.9, namely that optimal performance on the generalisation set is achieved with a relatively small number of functions but then degrades as the number of functions increases, probably due to overfitting the training data. Once again the impact of the initial network configuration on performance can be seen to be significant.

Number of functions	Average % correct training set	Standard deviation	Average % correct generalisation set	Standard deviation
30	92.9	1.6	75	3.6
20	90.3	1.7	78.4	6.0
10	84.4	3.2	80.3	6.2
5	79.2	2.1	79.7	3.7

Table 6.10 - The average percentage performance of RBF networks employing multi-quadratic functions on the skin cancer data set.

Number of functions	Average % correct training set	Standard deviation	Average % correct generalisation set	Standard deviation
50	96.6	3.1	68.4	2.7
40	93.6	2.1	68.6	3.0
30	91.4	2.6	68.9	3.5
20	87.4	2.8	69.5	3.9

Table 6.11 - The average percentage performance of RBF networks employing inverse multi-quadratic functions on the skin cancer data set.

A comparison of the performance of the four basis functions reveals that the best average generalisation performance of 80.3% was achieved when 10 multi-quadratic functions were employed, with the poorest being obtained with the inverse multi-quadratic functions. Networks that employed each of the function types were prone to overfitting the training data when too many functions were employed. In addition, the performance of the network was found to be sensitive to the initial configuration of the network.

6.4.2 The Multi-Layer Perceptron

The MLP was trained and tested on the skin cancer diagnosis data set with networks that employed between 1 and 20 hidden units, and table 6.12 summarises these results. The best generalisation performance of 79.3% was obtained with a network employing 4 hidden units, and generalisation performance was to a large extent invariant over the range of 4 to 20 hidden units. The results in section 6.4.1 revealed that when the RBF networks employed too many functions they tended to over fit the training set. A similar trend was found with the MLP, although the deterioration in performance was not so marked. When more than 4 hidden units were employed performance on the training set improved, whilst performance on the generalisation set marginally deteriorated. When

less than 4 hidden units were used the MLP gave poor performance on both the training and generalisation sets, suggesting that the network had insufficient hidden units to adequately map the problem space. The standard deviation figures in table 6.12 reveal that the performance of the MLP was significantly influenced by the initial network configuration.

Number of hidden units	Average % correct training set	Standard deviation	Average % correct generalisation set	Standard deviation
20	93.7	0.9	79	1.5
6	88.7	5.4	79.2	2.3
5	87.3	6.8	77.7	2.5
4	90.7	3.8	79.3	3.0
1	79	0	74.2	0

Table 6.12 - The average percentage performance of the MLP on the skin cancer data set.

6.4.3 Classification Performance of the MLP, RBF and Bumptree Neural Networks

This section will provide a comparative study of the average classification performance of the MLP, RBF and MIF bumptree on the diabetes diagnosis problem. The average level of performance achieved by each network is shown in table 6.13. We have included the performance of the MIF bumptree and Omohundro's bumptree, since it was found to give the best generalisation performance (82.2%) of any of the bumptree variations. The generalisation performance achieved by the different networks was again very similar, although Omohundro's bumptree was found to marginally outperform the other networks. The results indicate a tendency for the bumptree to produce a marginally superior performance on the generalisation set. However, the standard deviation figures,

particularly on the generalisation set, demonstrate the dependency of the networks on the initial configuration. The dependency of the networks on their starting configuration can be considered to play a significant role in determining the average levels of performance. The results reveal that the bumptree is less effected by its initial configuration. They do not, however, demonstrate a significant superiority in terms of performance level.

For the skin cancer diagnosis problem the three networks were able to achieve similar levels of classification performance. The performance on the training set of the MIF bumptree varied significantly and was dependent on its initial configuration, as was that of the RBF and MLP networks. However, on the generalisation set the performance was significantly less dependent on the initial configuration of the network than the other network types. The MIF bumptree proved to be less sensitive to a range of parameter values, since the same parameter values used for the diabetes problem were also used here. In contrast, the MLP and RBF networks had to be re-parameterised in order to optimise network performance. The impact of adjusting the number of hidden units and the number of functions implies that the failure to adjust additional parameters for the RBF and MLP network's might have had a significant impact on the performance level attained.

Network type	Average % correct training set	Standard deviation	Average % correct generalisation set	Standard deviation
MLP	90.7	3.8	79.3	3.0
RBF	84.4	3.2	80.3	6.2
MIF bumptree	79.8	5.24	80.8	1.95
Omohundro's bumptree	79.0	3.52	82.2	1.28

Table 6.13 - The average percentage performance of the MLP, RBF and bumptree neural networks on the skin cancer data set.

6.5 The Iris Data Set

The iris data set was concerned with distinguishing between three different sorts of flowers, and was provided for use in this study by D. Bounds[3]. The data set contained 75 patterns in both the training and generalisation sets, with each pattern being described by 4 inputs and 3 outputs, with each type of flower represented by one of the outputs. The data was normalised in the range 0 to 1. In its entirety the data set consisted of 150 patterns, with 50 patterns representing each class of flower. Both the training set and the generalisation set consisted of 75 patterns, with 25 instances of each class of flower.

6.5.1 The Radial Basis Function Network

RBF networks were constructed using gaussian, thin plate spline, multi-quadratic, or inverse multi-quadratic functions with the functions once again being placed on the problem space at random points within the area covered by the training set.

An RBF network employing between 25 and 75 thin plate spline functions gave the performance shown in table 6.14. These results show that the network was able to correctly classify all the patterns in the training set when between 35 and 75 functions were employed. The best average generalisation performance of 93.7% was achieved with 35 functions. There was a slight weakening of generalisation performance as the number of functions was increased, but even with 75 functions performance was still satisfactory.

[3]Professor D. Bounds, Head of Department, Department of Computer Science, Aston University, Birmingham.

Number of functions	Average % correct training set	Standard deviation	Average % correct generalisation set	Standard deviation
75	100	0	91.6	1.5
65	100	0	92.1	0.9
45	100	0	93.3	1.3
35	100	0	93.7	1.0
25	99.7	0.7	93.3	0.4

Table 6.14 - The average percentage performance of RBF networks employing thin plate splines on the iris data set.

The results obtained for an RBF network using between 35 and 75 multi-quadratic functions are shown in table 6.15. The results on the training set show that when 75 functions were employed the network was able to correctly classify all the patterns for all the trials, but that performance degraded when fewer functions were used. In contrast generalisation performance was found to improve slightly as fewer functions were used, although the actual changes were very small, and possibly explained by the standard deviation figures given in table 6.15.

Number of functions	Average % correct training set	Standard deviation	Average % correct generalisation set	Standard deviation
75	100	0	92.7	0.8
45	99.9	0.3	94.1	0.7
40	99.5	0.4	94.3	0.9
35	99.5	0.3	94.7	1.0

Table 6.15 - The average percentage performance for RBF networks employing multi-quadratic functions on the iris problem.

Number of functions	Average % correct training set	Standard deviation	Average % correct generalisation set	Standard deviation
75	100	0	89.5	1.9
60	99.1	1.2	91.6	1.6
50	97.8	1.0	92.5	2.6
35	95.3	2.0	92.1	1.8

Table 6.16 - The average percentage performance of RBF networks employing inverse multi-quadratic functions on the iris problem.

RBF networks constructed with between 35 and 75 inverse multi-quadratic functions gave the results shown in table 6.16. The results are similar to those obtained using multi-quadratic functions. The network was able to correctly classify all the members of the training set when 75 functions were used and generalisation performance improved when the number of functions decreased. In this instance the best average generalisation performance of 92.5% was achieved with a network using 50 functions.

When gaussian functions were used, networks employing 50 and 75 functions were examined, and both of these networks correctly classified all the members of the training set. With 50 functions the generalisation performance was found to be 94% and for 75 functions 96%. Overall this was the best performance obtained by an RBF network. Once again, it was decided not to employ more functions than one per pattern in the training set.

In summary, all the RBF networks were able to achieve completely correct classification of the training set - although the best performance on the generalisation set was not achieved by the thin plate spline, multi-quadratic and inverse multi-quadratic functions with this level of performance on the training set. To correctly classify all the training patterns it was necessary for these RBF networks to over fit the training data at the

expense of generalisation performance. The best level of performance on the generalisation set was obtained with an RBF with gaussian functions, when a correct classification of all the patterns in the training set was obtained.

6.5.2 The Multi-Layer Perceptron

The MLP was tested on the iris pattern classification task with networks that employed between 2 and 50 hidden units; the results for the generalisation set are given in figure 6.6. The best average generalisation performance of 95.7% was achieved when 4 hidden units were employed. For the iris problem the size of the hidden layer made hardly any difference to the performance of the network, unlike the other problems examined so far. When between 2 and 50 hidden units were employed the MLP was able to correctly classify all members of the training set and achieve a classification performance exceeding 95% on the generalisation set.

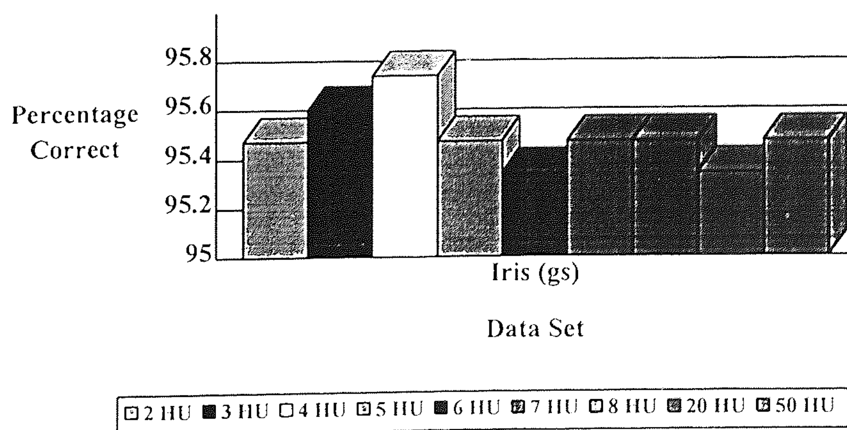

Figure 6.6 - The average percentage performance of the MLP on the iris pattern classification problem.

6.5.3 Classification Performance of the MLP, RBF and Bumptree Neural Networks

This section will provide a comparative study of the average classification performance of the networks on the iris pattern classification task. The best level of average performance attained by each of the networks is given in figure 6.7. This shows that the MIF bumptree achieved the best generalisation performance (97.5%). The corresponding figure for the MLP was 95.7% with four hidden units, and 96% for the RBF network employing 75 gaussian functions. Figure 6.7 illustrates that all networks achieved 100% correct classification of the training set when this level of generalisation performance was achieved. As previously observed, the RBF network once again suffered from the problem of overfitting the training data when too many functions were used, except when gaussian functions were employed. The MLP in this instance was not troubled by any problems of overfitting the training data.

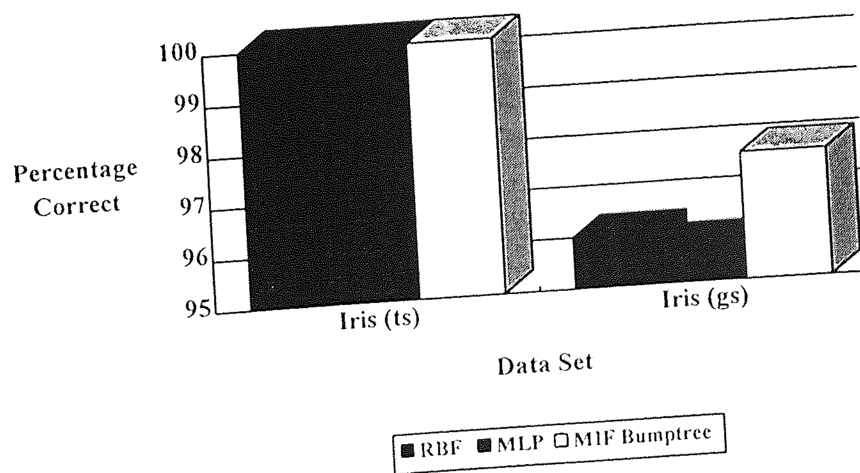

Figure 6.7 - The average percentage performance of the RBF, MLP and MIF bumptree neural networks on the iris pattern classification problem.

The overall results and trends for the iris pattern classification task were consistent with those achieved for the other problems. The MIF bumptree produced a classification performance comparable to that obtained with the MLP and RBF networks, and an analysis of the time required to train the network to a solution carried out in section 6.7 reveals that the training process was less computationally complex. The response times of the trained networks to queries was similar in the case of the MLP and MIF bumptree networks, but much slower for the RBF network.

In contrast to the previous studies the performance of the MIF bumptree was found to be sensitive to the parameter values taken, since the best performance was obtained with a value of 7 for the parameter *SMALL*. The performance of the RBF network was once again found to be dependent on the number and type of function employed, whilst the MLP was able to produce a similar performance on both the training and generalisation data sets for an architecture consisting of between 2 and 50 hidden units.

6.6 The Petersen and Barney Vowel Data

The Petersen and Barney vowel data set (Petersen and Barney 1952) concerned the classification of the letters A-J, and was provided for this study by David Bounds[4]. The data set contained a number of occurrences of the letters A-J, and consisted of 653 patterns with each pattern being represented by two inputs and three outputs. The patterns were divided into 320 patterns in the training set and 333 patterns in the generalisation set, with each of the letter classes being evenly represented in both data sets. All the input and output values were normalised to values between 0 and 1. This was a complex

[4]Professor D.Bounds, Head of Department, Department of Computer Science, Aston University, Birmingham.

problem to solve, since the different classes of patterns were very closely grouped together, with the various classes overlapping on the problem space.

6.6.1 The Radial Basis Function Network

The performance of the RBF network on the vowel recognition task will now be examined. RBF networks were constructed using either thin plate spline, gaussian, multi-quadratic or inverse multi-quadratic functions, with varying numbers of functions. The functions were once again placed on the problem space at random points within the area covered by the training set.

Figure 6.8 provides the results for the RBF network using between 60 and 320 gaussian functions. These show that the best training performance (99.4%) was achieved with 320 functions and the best generalisation performance (85.4%) was achieved with 80 functions. As the number of functions increased generalisation performance degraded as the network overfitted the training set, with a severe degradation being observed with 320 functions. Performance on the training set was also found to be somewhat sensitive to the number of functions employed, the results showing a clear trend of improving performance with increasing network size.

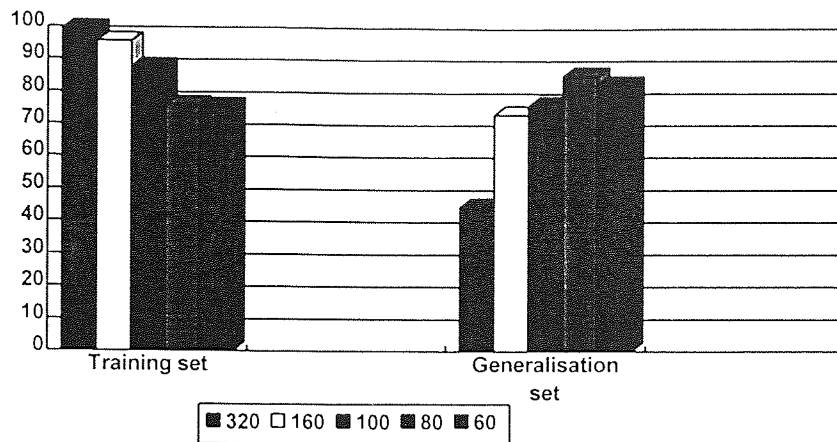

Figure 6.8 - The average percentage performance of RBF networks employing gaussian functions on the Petersen and Barney vowel data.

Similar results were obtained for RBF networks using thin plate spline, multi-quadratic and inverse multi-quadratic functions. As the number of functions increased towards 320 performance on the training set improved, whilst performance on the generalisation set peaked with between 100 and 200 functions; this is shown in figures 6.9-6.11. The RBF network using thin plate splines achieved the best training performance (96.2%) when approximately 300 functions were used, and the best generalisation performance (77.9%) when between 50 and 75 functions were used. The best training performance for the RBF network employing multi-quadratic functions (95.1%) was achieved with approximately 300 functions as was the best generalisation performance (76%). When less than 290 functions were employed the performance on both the training and generalisation sets deteriorated. The RBF using inverse multi-quadratic functions obtained its best average performance level of 93.2% on the training set when between 290 and 300 functions were employed. The best average generalisation performance (76.4%) was achieved when between 200 and 210 functions were used. When less than 200 functions were employed performance on both the training and generalisation sets deteriorated.

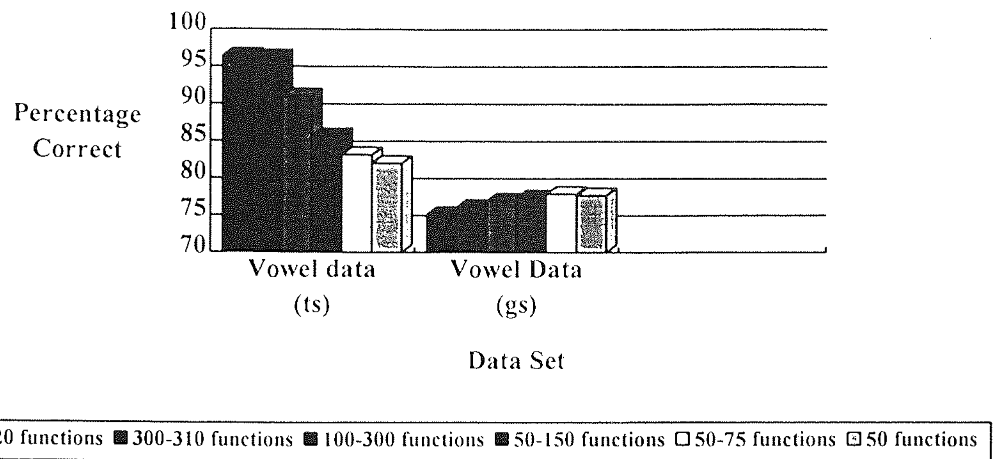

Figure 6.9 - The average percentage performance of RBF networks employing thin plate splines on the Petersen and Barney vowel data.

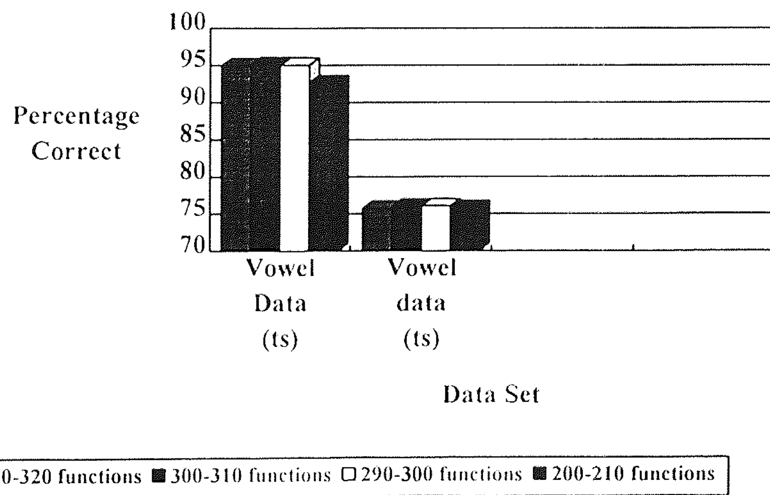

Figure 6.10 - The average percentage performance of RBF networks employing multi-quadratic functions on the Petersen and Barney vowel data.

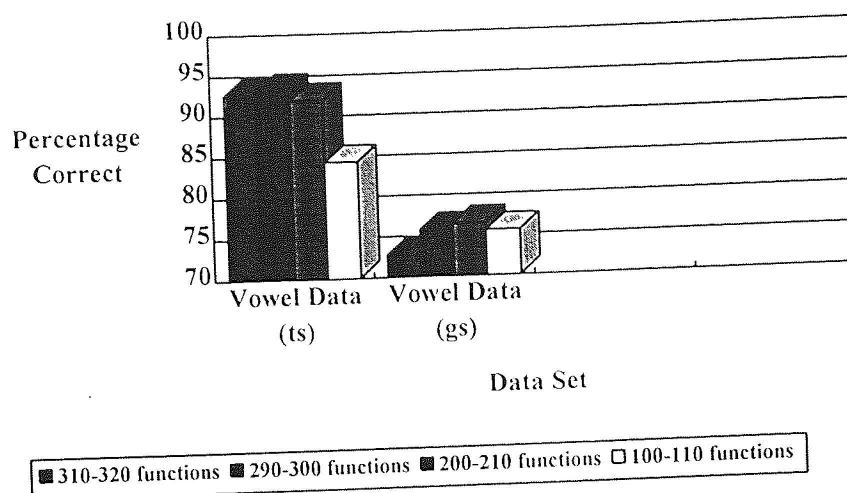

Figure 6.11 - The average percentage performance of RBF networks employing inverse multi-quadratic functions on the Petersen and Barney vowel data.

The performance of the RBF network on the Petersen and Barney vowel data showed the same trends between performance and the number of functions present in the network as were observed for the other problems studied. That is, the performance on the training set improved as the number of functions increased, and the network was able to memorise the training patterns. The performance on the generalisation set improved upto the point when the network possessed sufficient degrees of freedom to map the problem, and deteriorated thereafter as the network began to simply memorise the training patterns.

6.6.2 The Multi-Layer Perceptron

The MLP was tested on the Petersen and Barney vowel recognition data with networks consisting of between 8 and 20 hidden units; the results are given in table 6.17. Given the complexity of the problem the MLP required a large number of hidden units in order to effectively partition the problem space. The largest number of hidden units employed in

the network was 20, and this gave a training set performance of 82.4% and a generalisation set performance of 77.1%. In order for the network to correctly classify more than 75% of the patterns in the training set at least 16 hidden units were required. Further trials were carried out that employed more than 20 hidden units, but performance on the generalisation set was not significantly improved, whilst training time was extended greatly.

Number of hidden units	Average % correct training set	Standard deviation	Average % correct generalisation set	Standard deviation
20	82.4	4.0	77.0	3.7
16	81.2	4.2	76.0	4.3
12	78.8	4.2	74.5	4.0
8	74.4	4.2	70.7	3.6

Table 6.17 - The average percentage performance of the MLP on the Petersen and Barney vowel data.

6.6.3 Classification Performance of the MLP, RBF and Bumptree Neural Networks

This section will provide a comparative study of the average classification performance of the various neural networks on the Petersen and Barney vowel recognition problem. The level of performance achieved is shown in figure 6.12. This shows that the RBF network using 80 gaussian functions was able to attain the best performance level on the generalisation set of 85.5%, followed by the MLP with 77.1% and the bumptree trailing with 73.6%. The RBF network correctly classified over 90% of the patterns in the training set for all four function types when a sufficient number was used, although generalisation performance deteriorated. The MLP attained the best performance on both

the generalisation and training sets when 20 hidden units were employed. The performance of the MIF bumptree on both the training set and the generalisation set improved as more functions were used, but was disappointing.

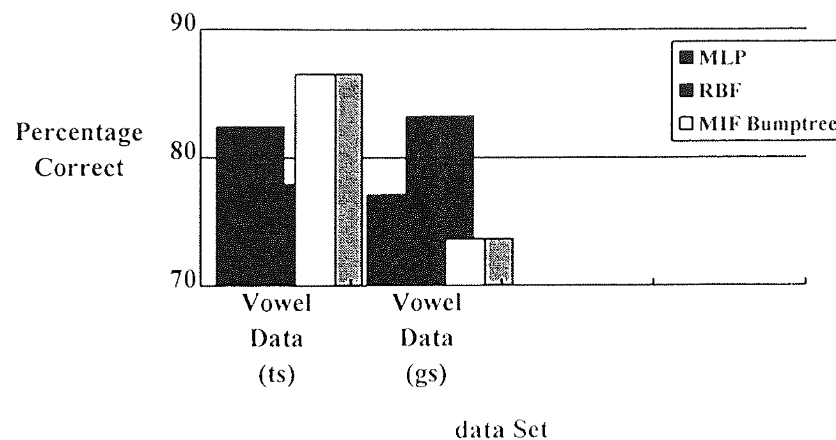

Figure 6.12 - The average percentage performance of the MLP, RBF and MIF bumptree neural networks on the Petersen and Barney vowel recognition task.

It was expected that with a problem where the different classes overlapped to a large degree, as was the case with the Petersen and Barney data, the MLP, with its long range solution, would produce the worst generalisation performance. However, this was not the case. The RBF was able to produce superior performance to the MLP, but the MIF bumptree gave the poorest performance. It was not able to match the generalisation performance of the other networks on this problem, and this raises the issue of its ability to deal with difficult problems. This problem is particularly difficult because of the manner in which the different classes overlap. The MIF bumptree is able to learn complex problems, as revealed by the Parity (6) problem and by the performance on the training set with this problem. However, it was not able to produce good generalisation performance on this, the most complex data set examined. It was able to reach the

recorded level of performance relatively quickly, but from this point was not able to improve generalisation performance.

6.7 Computational Complexity Of the MLP, RBF and MIF Bumptree Networks.

The average classification performance of the RBF, MLP and MIF bumptree neural networks is one performance measure that can be utilised. An additional performance measure is the computational effort required to reach a solution. The issue of computational complexity involves both the size of the network required to reach a solution and the mathematical operations required to train the network. A similar approach to that employed in chapter 4 has been adopted to compare the computational complexity of the RBF and MLP networks to the MIF bumptree. The data sets examined are those that allow a valid test of the generalisation performance of the network, since the main concern is to identify the degree of computational complexity required to train the network to the level of generalisation performance given above.

The computational complexity of the RBF, MLP and MIF bumptrees will be analysed both in terms of the size of network required to reach an adequate solution and in terms of the mathematical operations required to train the network. The latter measure provides more useful information, since a network that employs 1 function but takes thousands of calculations is inferior in terms of its computational complexity to one with 20 functions that requires a hundred calculations. Table 6.18 summarises the number of hidden units or functions each network type required to attain the level of average generalisation performance given above.

	Iris	Skin Cancer	Diabetes	Vowel
MLP	**4 hidden units (262 iterations)**	**4 hidden units (4425 iterations)**	**16 hidden units (740 iterations)**	**20 hidden units (9800 iterations)**
RBF	**75 functions**	**10 functions**	**25 functions**	**80 functions**
MIF Bumptree	**8 functions**	**4 functions**	**3 functions**	**104 functions**

Table 6.18 - The size of networks to produce the best generalisation performance on the problems with a valid test of generalisation.

Table 6.18 suggests that the MIF bumptree generally requires fewer functions than the RBF network with the exception of the Vowel Recognition problem. This is also the problem where the MIF bumptree performs worse than the other networks. It appears that if the MIF bumptree is unable to reach a solution with a relatively small network the level of performance deteriorates in comparison to the other network types. That is, although the performance of the MIF bumptree improves on the Vowel Recognition problem until on average 104 functions are employed it is not able to match the performance of the other networks. The inability of the MIF bumptree to perform adequately on the vowel recognition problem was raised earlier, and its performance brings into question its ability to perform adequately when numerous function additions are required. The MLP employed a relatively small number of hidden units for all the problems, and the significance of these and the number of iterations required on the complexity of the training process will be examined below.

In terms of network size the MIF bumptree generally employed fewer functions than the RBF network. In addition, it generally employs a similar number of functions as the MLP employs hidden units. The exception to this is the Vowel recognition problem, and this have been discussed above. Although network size is an important issue in considering

the computational complexity of the approaches, of more significance is the number of mathematical operations required to arrive at a solution. Chapter 4 examined the number of mathematical operations (multiplication's, divisions, additions, subtraction's) required by the MIF training operation. The number of calculations required by the MLP and RBF can also be categorised in this manner, which enables the scale of any differences in computational complexity to be identified.

In the case of the MIF bumptree, the focus of attention is on adding functions to the network, and the subsequent training of these functions. The pseudo code given in Appendix A provides a basis for this analysis, and chapter 4 examines this process in detail. To arrive at the total number of calculations required to reach a solution for the MIF bumptree it is necessary to consider not only the calculations required to train each function, but it is also necessary to consider the number of function additions required to reach a solution.

---

1 - Assign the weight and bias's their initial random values.
2 - Calculate the error of the network.
3 -While the error level is not satisfactory.
    4 - Adjust the weight and bias values.
    5 - Calculate the error of the network.
6 - End training.

Figure 6.13 - The main steps employed in training the MLP.

The training of the MLP concerns the iterative updating of the weight and bias values employed by the network. That is, it commences with a network of fixed size and updates the weight and bias values until an acceptable level of performance is reached. Therefore,

the important issues are the number of hidden units employed, the number of iterations required, and the number of weight and bias values to be updated. The steps involved in training the MLP network employed in this study are summarised in figure 6.13.

---

1 - Determine initial function centres.
2 - Calculate activation of each function on all patterns.
3 - Optimise Alpha and Beta parameters for patterns where the function has an activation level that is non zero.

Figure 6.14 - The main steps employed to construct the RBF network.

The RBF training process employed in this study bears similarities to both the MLP and MIF bumptree approaches. Similar to the MLP, the size of the RBF network is fixed at the commencement of the training process and every update of the parameters has to consider the impact of every pattern in the training set. However, the RBF training process is not an iterative procedure. Instead the RBF, like the MIF bumptree, can employ a one-shot learning algorithm. The training process employed by the RBF used in this study is summarised in figure 6.14. The RBF network employing gaussian functions will be considered because the bumptree considered in this section employed gaussian functions. There are similarities between the RBF approach and the MIF bumptree training process which is summarised in figure 6.15. The training process for each function with the MIF bumptree is not iterative, but the entire process can be seen to be repeated each time an additional function is required. The RBF network can employ the one-shot learning process once only for all the functions in the network. The absence of the need for a routine to calculate the error of the RBF network is an obvious difference between the approaches. This is not required since the size of the network and the

position of the functions is fixed for the duration of the training process, and the error of the network does not impact on the training process. It can simply be used at the conclusion of the training process to measure the success or failure of the approach. In addition, it is usual for the training algorithm for the RBF to contain an element concerned with positioning functions on the problem space. However, in this study experimentation was employed to determine acceptable positions for the functions.

Candidate Functions:
1 - Determine initial function centres.
2 - Assign Patterns to functions.
3 - Optimise Alpha and Beta parameters.
4 - Calculate the error of each function.
5 - Calculate a goodness value for each function.
6 - Add functions to the network.

Actual Functions:
7 - Assign Patterns to functions.
8 - Optimise Alpha and Beta Parameters
9 - Calculate the error of each function.

If further functions are required goto step 1, else finish training.

Figure 6.15 - The main steps employed to construct the MIF bumptree.

The calculations required by the MIF bumptree to add two functions to the network for each of the Iris, vowel, diabetes and skin cancer diagnosis data set were examined in chapter 4. Chapter 4 provided an analysis of the calculations required by the main steps involved in this training process, namely those concerned with assigning patterns to the functions, optimising the Alpha and Beta parameters, and calculating the error of the functions. In order to arrive at the total calculations required by the MIF approach to reach a solution for each of these problems it is also necessary to consider the number of

function additions required by the MIF bumptree to reach a solution. Table 6.18 listed the number of functions required for each of the problems and table 6.19 provides a summary of the total calculations required to solve each of the problems in question. This table reveals that the vowel recognition problem is the most expensive problem in terms of computational complexity for the MIF bumptree. This is primarily because the network is required to make 52 function additions as opposed to the 2 or 4 additions required for the other problems examined in this section.

	Iris	Skin Cancer	Diabetes	Vowel Data
Number of function additions	4	2	2	52
Calculations per function addition	57642	34132	560048	163272
Total Calculations	230568	68264	1120096	8490144

Table 6.19 - Total calculations required to train the MIF bumptree to a solution.

In order to arrive at the total number of calculations required to train the MLP and RBF networks to a solution for the problems examined in table 6.19 it is necessary initially to identify the calculations required by each stage of the training process for these networks. Appendix D provides the pseudo code for the main steps of the learning algorithm required by the MLP training process summarised in figure 6.13. The calculations required by each of the main steps in the MLP learning algorithm for the problems examined in this section are outlined in table 6.20.

	Iris	Skin Cancer	Diabetes	Vowel Recognition
Network Parameters	**4 hidden units 262 iterations 75 patterns**	**4 hidden units 4425 iterations 62 patterns**	**16 hidden units 240 iterations 400 patterns**	**20 hidden units 9800 iterations 320 patterns**
Calculate Error Adjust Weights	**75*72*262 75*197*262**	**62*64*4425 62*177*4425**	**400*392*240 400*1081*240**	**320*248*9800 320*749*9800**
Total Calculations	**5,285,850**	**66,118,350**	**141,408,000**	**31,265,920,000**

Table 6.20 - Total calculations (additions, subtractions, multiplications, divisions) required by the MLP to achieve the desired level of performance.

The figures given in table 6.20 break down the main steps of the MLP training procedure into numbers of simple mathematical operations. These figures reveal that as with the MIF bumptree the most expensive problem in terms of computational complexity was the vowel data problem, followed by the diabetes diagnosis problem. A comparison of the mathematical operations required by the MLP with those required by the MIF bumptree reveals that the iterative MLP training procedure is far more computationally complex than the MIF bumptree training procedure. For the iris problem, the MIF bumptree required 230,568 mathematical calculations to train to a solution, whilst the MLP required 5,285,850 mathematical calculations. The MLP, therefore, required 23 times the mathematical calculations the MIF bumptree required. To solve the diabetes diagnosis problem the MLP required 126 times the calculations required by the MIF bumptree. To solve the skin cancer diagnosis problem the MLP required 968 times the calculations required by the MIF bumptree. Finally, to solve the vowel recognition problem the MLP required 3682 times the calculations required by the MIF bumptree. These figures

demonstrate the advantage that the MIF bumptree possesses over the MLP in terms of the computational complexity of the training process.

```
For Patterns = 1 to Num Patterns in Training Set
   For Functions = 1 to Functions in the Network
         Temp 5 =1
         For Units = 1 to Num Input Units
                  Temp1 = 1/(1*(sqrt(3.141592654*2)))
                  Temp2 = 0.5 *(Input [Patterns][Units]-Function centre[Units])/1)^2
                  Temp3 = exp(-Temp2)
                  Temp4 = Temp1*Temp3
                  Temp5 = Temp5*Temp4
         End
         Function[Functions] Activation Level = Temp5
   End
End
```

Figure 6.16 - Pseudo Code detailing the steps involved in calculating the activation of the functions in the RBF network.

The procedure for training the RBF network is summarised in figure 6.14. It is able to employ similar procedures for these steps to those employed by the MIF bumptree examined in chapter 4. However, the RBF network has no candidate functions to consider and therefore, these steps are carried out once for each function. The steps required to calculate the activation of the RBF functions on each pattern are summarised in figure 6.16. The activation of each of the functions on the patterns is particularly important to the RBF network, since the functions only require their weights updating for those patterns upon which they are active. In addition, when the network is fully trained the functions only contribute to the networks output for those patterns upon which they are active.

Once the activation of the functions has been calculated it is necessary to optimise the weight and bias parameters of each function. This can be done with a procedure very similar to that employed by the MIF bumptree, examined in chapter 4. Each function has its weight and bias parameters optimised across all the patterns in the training set. However, only those patterns for which a function is active will impact on the weights of the function. In addition, the degree to which a pattern alters the weights of a function depends on the activation level of the function on it. The process is similar to that of the MIF bumptree summarised in Appendix B. The main difference is that whilst the MIF bumptree assigns each pattern to only one function, the RBF network allows each pattern to have multiple functions active on it. The weight and bias values for the functions can again be arrived at through the matrix multiplication and pseudo inversion employed to train the MIF bumptree. The difference is that the RBF network needs to consider every pattern for every function, whilst this is not the case with the MIF bumptree. As with the MIF bumptree, Matrix1 can be populated in two stages. The first stage populates an area equal to the square of the number of input dimensions. For each function every slot is filled by a process using 1 multiplication and 1 addition for every pattern in the training set. The number of times these calculations need to be carried out for each function is given as the number of input dimensions squared, multiplied by the number of patterns in the training set, even though the activation of some of the functions may be zero on some patterns they still have to be considered. The second stage of populating Matrix1 involves every function for every input dimension having an addition carried out for every pattern. This process is carried out twice. Each time the calculations required can be given as the number of input dimensions multiplied by the number of patterns in the training set, even though many will add a zero value because the function is not active on them.

Matrix3 is populated through a very similar process to Matrix1. It is again populated in two stages. The first stage populates an area of the matrix for each function equal to the

number of input dimensions multiplied by the number of output dimensions. For each of the functions, every slot is filled by a process using 1 multiplication and 1 addition for every pattern in the training set. The number of times these calculations need to be carried out for each function is given as the number of input dimensions multiplied by the number of output dimensions, multiplied by the number of patterns in the training set The second stage of populating Matrix3 considers every function in every output dimension having an addition carried out over every pattern. This process is carried out twice. The calculations required for each function can be given as the number of output dimensions multiplied by the number of patterns in the training set, even though some of the patterns will not contribute to the final total because the function is not active on them.

Once the two matrices have been populated it is necessary to produce the pseudo inversion of matrix1, and this is done using the Gauss Jordan method. This involves identifying the pivot row and carrying out the elimination process, but this process is not computationally intensive. Once the pseudo inverse of Matrix1 has been calculated, it is necessary to multiply matrices 1 and 3 for each function. For every row in the result matrix, a process involving a single multiplication is carried out. The number of calculations is given as Input dimensions +1 multiplied by the number of output dimensions.

The training process for each function considers all the patterns in the training set, although in practice the function is unlikely to be active on each of the patterns. The impact that each of the patterns has on the weight and bias parameters for each function is scaled depending on the level of activation of the function on the pattern. However, since the purpose of this study is not to carry out an in-depth analysis of the RBF approach to training a network but simply to attain comparative figures in terms of training complexity this issue does not need to be examined. The steps required to fill stage 1 of

matrix 1 can be given as the number of input dimensions squared, multiplied by the number of patterns in the training set, multiplied by the number of functions in the network, multiplied by the number of calculations required, in this case 2. In order to fill stage 2 of matrix 1 the number of calculations is equal to the input dimensions multiplied by the patterns in the training set, multiplied by the number of functions in the network. The steps required to fill stage 1 of matrix 3 can be given as the number of input dimensions multiplied by the number of output dimensions, multiplied by the number of patterns in the training set, multiplied by the number of functions in the network, multiplied by the number of calculations required, in this case 2. The steps required to fill stage 2 of matrix 3 can be given as the number of output dimensions multiplied by the number of patterns in the training set, multiplied by the number of functions in the network. Table 6.21 summarises the calculations required by the two major procedures in the RBF training algorithm.

	Iris	Skin Cancer	Diabetes	Vowel Recognition
Activation of functions	247,500	20,460	880,000	563,200
Optimisation of weight & bias values	354,375	21,700	1,700,000	1,587,200
Total Calculations	601,875	42,160	2,580,000	2,150,400

Table 6.21 - The total calculations required to train the RBF network to an acceptable solution.

The results provided in table 6.21 reveal that to train the RBF network to a position where it is able to produce the level of performance discussed in this chapter, requires less computational complexity than is required to train the MLP. The RBF network requires 2.6 times as many calculations to train to a solution on the iris problem as the MIF

bumptree. On the diabetes problem it requires 2.3 times as many calculations to train to a solution as the MIF bumptree. However, on the skin cancer problem the RBF requires only 62% of the calculations required by the MIF bumptree, and on the vowel recognition problem the RBF network requires only 26% of the calculations required by the MIF bumptree. The number of calculations required to train to a solution provides an indication of the computational complexity of the various approaches, and reveals interesting information on the three approaches. As expected the MLP requires the largest number of calculations and on two of the four problems the RBF network requires significantly more calculations. However, on the skin cancer problem and the vowel recognition problem, the RBF requires less calculations than the MIF bumptree. This is due in part because the RBF network does not require an error calculation routine as part of its training algorithm. Another factor is the use of multiple initial candidate functions by the MIF bumptree. This adds greatly to the number of calculations required by the bumptree neural network. The difference in calculations between the RBF and MIF bumptree on the vowel recognition problem is also impacted upon by the fact that the RBF network requires significantly fewer functions than the MIF bumptree.

In addition to the issue of training time, there exists the issue of the time for the trained network to respond to a query. It was anticipated that the MLP would provide the best performance on this since the other two networks require the additional calculation of the activation levels of the functions in the network. The approach to producing an output for a query adopted by the MLP is outlined in figure 6.17. The calculations that this approach requires for each of the problems examined in this section of the study are summarised in table 6.22.

```
For Units in layer 2
        temp=0
        For Units in layer 1
                temp=temp+(output[unitlayer1]*weight[unit, layer1 to layer2])
        End
        netoutput[unit in layer 2]=temp+bias[unit in layer2]
        output [unit in layer2]=1.0/(1.0+exp(-netoutput[unit in layer2]))
End
For Units in layer 3
        temp=0
        For Units in layer 2
                temp=temp+(output[unitlayer3]*weight[unit, layer2 to layer3])
        End
        netoutput[unit in layer 3]=temp+bias[unit in layer3]
        output [unit in layer3]=1.0/(1.0+exp(-netoutput[unit in layer3]))
End
```

Figure 6.17 - The procedure required to produce the output to a query for the MLP.

	Iris	Skin Cancer	Diabetes	Vowel Recognition
Network Size	4-n-3	3-n-2	8-n-2	2-n-10
Hidden Units	4	4	16	20
Weights	28	20	128	240
Bias's	7	6	18	30
Total calculations	80	60	392	600

Table 6.22 - The calculations required to respond to a query of a trained MLP network.

Table 6.22 reveals that the MLP requires most calculations to respond to a query for the vowel recognition problem. It demonstrates how the computational complexity of

producing a response to a query increases dramatically as the size of the network increases. The ratio of calculations per weight and bias value remains relatively constant across the problems, it is simply the number of weight and bias connections in the network that leads to the increased calculations.

```
temp5=0.0
For Input Units
        temp1=1.0/(functions centre * (sqrt(3.14159*2.0)))
        temp2=0.5* ((patterns input -functions centre)/ functions radius)^2)
        temp3=exp(-temp2)
        temp4=temp1*temp3
        temp5=temp5*temp4
End
functions activation = temp5
```

Figure 6.18 - Routine to calculate the activation of gaussian functions.

The approach of the MIF bumptree to respond to a query is to calculate the activation level of the functions in the network in the branches of the tree where the active functions reside. That is, the parent functions both have their activation calculated, and following this the two functions on the selected branch have their activation calculated. This procedure continues until there are no further branches in the tree that remain to be examined. When there are no further levels in the tree structure the weight and bias parameters are utilised to produce the required output value. Figure 6.18 summarises the manner in which the activation of the gaussian functions is calculated, and figure 6.19 summarises the approach to calculating the actual output value of the network.

```
For output units
        temp2=0.0
        For input units
                temp=weighted connection * input pattern
                temp2=temp2+temp
        End
        temp3=temp2+bias value
        Output for output unit =temp3
End
```

Figure 6.19 - Routine for calculating the response to a query of the MIF bumptree.

The approach adopted by the MIF bumptree can also be adopted by the RBF network, although the routine to calculate the activation of the functions differs when other than gaussian functions are employed. In addition, whilst the MIF bumptree only calculates the activation of those functions that are potentially active on the pattern and the output of the final of these, the RBF network needs to calculate the activation of all the functions in the network and the output of each of these since they all contribute to the final output of the network. The degree to which the output value of each function impacts on the final output of the network is determined by the activation level of the function on the pattern. Functions with the highest activation levels have the greatest impact. Table 6.23 summarises the calculations required by the MIF bumptree and RBF network to provide a response to a query for the problems examined in this section.

	Iris	Skin Cancer	Diabetes	Vowel Recognition
MIF Bumptree	204	146	386	1238
RBF (gaussian)	5400	470	3000	5760

Table 6.23 - The calculations required by the MIF bumptree and RBF networks to respond to a query.

The figures provided in tables 6.22 and 6.23 reveal that the RBF network is outperformed by all the other networks on all the problems. The biggest differential is on the Iris problem where the RBF network requires 67 times the calculations required by the MLP required, and 26 times the MIF bumptree calculations. The reason for this additional complexity is that the output of the network instead of needing to be calculated once as with the other approaches needs to be calculated once for each function in the network. The performance of the MIF bumptree is similar to that of the MLP network, since it only requires the output of the network to be calculated once and in doing so less calculations are required than when calculating the output of the RBF. It is the calculation of the activation of the gaussian functions that requires the extra computational expense compared to the MLP. The calculations required to respond to a query on the diabetes data set reveals that when the network size is appropriate the MIF bumptree is able to produce a response to a query quicker than the MLP.

6.8 Summary

This chapter has focused on a comparative study of the performance of the MLP, RBF and MIF bumptree neural networks over a wide range of problems. The differing approaches have been compared in terms of their performance on a generalisation set, the time taken to train to a solution, and the time taken to respond to a query. The average classification performance of the MIF bumptree is comparable to that of the other networks on the problems examined in this study. The vowel recognition problem is an exception, where it is unable to match the performance of the other approaches. In terms of training time the MIF bumptree substantially outperformed the other two networks on the Iris and Diabetes diagnosis problems, but is outperformed by the RBF on the skin

cancer and vowel recognition problems. Training time is one of the major advantages of the MIF bumptree approach. In terms of the response time to queries the MIF bumptree was able to outperform the RBF network on all the problems examined in this study, and at least matched the performance of the MLP on some.

To conclude, the MIF bumptree showed in this study that it was able to perform adequately across a wide range of problems, and its performance was comparable to that achieved with the MLP and RBF networks. The training speed of the MIF bumptree was found to be a major advantage, so much so, that for problems where a quick solution is required the MIF bumptree is clearly an attractive candidate.

Chapter 7

Conclusion

The work presented here has been concerned with the impact of architecture on the performance of neural networks, and its impact has been examined across a range of problems for a number of network types. The work has initialy examined the architectural issues associated with the MLP network. It has then demonstrated how it is possible to employ alternate network architecture's to the MLP that implement a more local solution, with attention focusing in particular on the development of the bumptree neural network,

The main focus of attention of this study was to examine the impact of employing an alternate network structure to the MLP. The alternate techniques examined attempted to improve network performance by partitioning the problem space in a different manner to the MLP. In particular, attention turned to structures that employed a more local solution than the MLP, namely the RBF and bumptree neural networks. Particular attention focused on the MIF bumptree. This embodied an idea with its foundations in the decision tree algorithms of machine learning, namely the utilisation of a tree structure to partition the problem space. It was hoped that the use of a more local solution would allow the production of networks that would generalise better and train to a solution quicker.

The main issues to be resolved with the bumptree neural network concerned the manner in which the functions that made up the tree structure were to have their dimensions determined, the learning algorithm to be employed, and the manner in which the output of the network was to be calculated. A number of alternatives were presented in chapters 5 and 6, with the preferred solutions being embodied in the MIF bumptree which eradicated the issue of determining the radius of the functions in each dimension by employing a constant unit radius. The learning algorithm employed after examination of

alternatives was the one-shot learning algorithm described in chapter 5, and the output of the network was calculated using only the output of the function furthest from the root level that was active on the pattern. This approach is referred to as the LAF output calculation technique in chapter 5.

The MIF bumptree was able to produce performance comparable to that achieved by MLP and RBF networks in terms of average correct classifications on the problems examined in this study. The bumptree did, however, prove to have an advantage over the other connectionist models examined. This was its training time, and in some instances the time to access a trained network. With the bumptree it is only necessary to compute the activation of two functions at each level of the tree, in addition to calculating the output of the last active function, in order to classify a point. In contrast an RBF network requires the calculation of the activation of all functions in the network. An MLP with n hidden units requires n hyperplanes to be evaluated each time a point is classified. In contrast, a complete bumptree with n functions requires $2\log_2 n$ function evaluations to classify a point. Thus, the bumptree can offer an access-speed advantage over the MLP when networks employ similar numbers of hyperplanes and functions, with the advantage increasing exponentially with increasing network size. During training, the bumptrees output weights are minimised in a single pass through the training set, whereas the MLP typically employs an iterative gradient descent method requiring thousands of passes. In addition, each function in the bumptree is trained only on a subset of the patterns in the training set and does not need to consider other functions when the learning algorithm is being applied to it. This is unlike the RBF where the functions are all trained on all the patterns (although not all the patterns exert equal influence). Hence, the bumptree neural network exhibited a considerable improvement in training time, and in some case recognition time, compared to the other networks examined.

In addition to this work a genetic bumptree was developed with the initial intent of producing an alternative bumptree which had a learning algorithm not effected by possible problems with singular matrices. However, the one-shot learning algorithm employing an SVD technique that was developed meant that this problem was resolvable with more standard techniques, and the performance of the initial genetic bumptree that employed a genetic algorithm both to determine the dimensions of the functions and to determine their weight and bias values was not particularly encouraging. It was, however, perceived that the genetic bumptree would achieve better performance when it only employed a genetic algorithm to determine the dimensions of the functions and utilised the one-shot learning algorithm employing an SVD technique to determine the required weight and bias values. When a genetic bumptree was developed that approached the task in this manner it was able to achieve performance comparable to the standard bumptree.

This study has shown that a bumptree neural network has certain advantages over the type of MLP and RBF networks studied. The bumptree neural network produced performance comparable to the other networks in terms of correct classifications whilst displaying the advantage of reduced training times, and comparable query response times. However, the study compared the performance of an optimised bumptree against the performance of an MLP network where the only parameter altered was the number of hidden units. In addition, attempts to improve the performance of the RBF network focused on altering the number of functions employed. Hence, this study provides a representative comparison of the approaches, although it may not be a definitive comparison. The same point can be made with regard to the nature of the training and generalisation data sets. The patterns of the various output classes are unevenly distributed between the data sets, and whilst this does not impact on the representative nature of the study, it does mean that the comparison cannot be viewed as definitive.

The bumptree neural network is still in its infancy compared to the other more established networks examined in this study, and there are a number of areas in which it may be possible to further improve the performance of the network. It may be possible to employ different learning algorithms for the functions within the tree structure without adversely affecting the networks performance. One option is to employ an individual MLP for each function, although this would drastically affect training time. Another issue that can be addressed in the future concerns the possibility of doing away with the tree structure once the network has been constructed, and simply retaining the lowest level functions to calculate the response of the network to patterns.

Therefore, in conclusion the bumptree neural network revealed an alternative network to currently existing ones that was able to produce a level of performance at least comparable to that attained by the RBF and MLP networks on the problems examined in this study.

References

Ash, T. (1989), "Dynamic Node Creation in Back-Propagation Networks", *ICS Report 8901*, Institute for Cognitive Science, University of California, San Diego, California.

Astion M.L. & Wilding P. (1992), "A Comparison of Neural Network and Other Pattern Recognition Approaches to the Diagnosis of Low Back Disorders", *International Journal of Neural Networks*, 3, pp.583-591.

Bornholdt, S. & Graudenz, D. (1991), "General Asymmetric Neural Networks and Structure Design by Genetic Algorithms", *Neural networks*, 5, pp327.

Baum, E.B., & Haussler, D. (1989), "What Size Net Gives Valid Generalization ", *Neural Computation*, 1 (1), pp.151-160.

Bostock, R.T.J., & Harget, A.J. (1994), "A Comparative Study of a Modified Bumptree Neural Network with Radial Basis Function Networks and the Standard Multi-Layer Perceptron", *Advances in Neural Information Processing*, 6, pp.240-247, Morgan Kauffman.

Bostock, R.T.J, Claridge, E., Harget, A.J. & Hall, P.N. (1993), " Towards a Neural Network Based System for Skin Cancer Diagnosis", *IEE Third International Conference on Artificial Neural Networks*, pp.215-220, Watkiss Studios Limited, Biggleswade, Bedfordshire, England.

Broomhead D.S. & Lowe D. (1988), "Radial Basis Functions. Multi-Variable Functional Interpolation and Adaptive Networks", *Complex Systems*, 2, 3, pp.321-355.

Buntine, W. (1991), "About the IND Tree Package", *Report*, NASA Ames Research Centre, Mail Stop 269-2, Moffet Field, CA 94035.

Chauvin, Y. (1989), "A Back-Propagation Algorithm with Optimal use of Hidden Units", *Advances in Neural Information Processing*, 1, pp.519-526.

Chen, S., Cowan, C.F.N, & Grant, P.M. (1991), "Orthogonal Least Squares Learning Algorithm for Radial Basis Function Networks", *IEEE Transactionsa Neural Networks*, 2, pp.302-309.

Chiu, W.C., & Hines, E.L. (1991), "A Rule-Based Dynamic Back-Propagation (DBP) Network", Neural Engineering Laboratory, Department of Engineering, Warwick University, Warwick.

Chester, D.L., (1990), "Why Two Hidden Layers Are Better Than One", In *Proceedings Of The International Joint Conference on Neural Networks*, 1, pp.265-268.

Cybenko, G., (1989), "Approximation by Superpositions of a Sigmoidal Function", *Mathematics of Control, Signals, and Systems*, 2(4), pp.303-314.

Fahlman, S.E. & Lebiere, C. (1990), "The Cascade-Correlation Learning Architecture", School of Computer Science, Carnegie Mellon University, Pittsburgh.

Fahlman, S.E. (1988), "Faster-Learning Variations on Back-Propagation. An Empirical Study", *Proceedings of the 1988 Connectionist Models Summer School*, pp.38-51.

Frean, M., (1990), "The Upstart Algorithm : A Method for Constructing and Training Feed Forward Neural Neural Networks", Department of Physics and Centre for Cognitive Science, Edinburgh University, Edinburgh, Scotland.

Forgy E.W., (1965), "Cluster Analysis of Multivariate Data Efficiency Versus Interpretability of Classifications", *Biometric Society Meetings*, Riverside, California, Abstract in Biometrics 21, No.3, pp.768.

Funahashi, K.I. (1989), "On the Approximate Realization of Continous Mappings By Neural Networks", *International Journal of Neural Networks*, pp183-192.

Gentric, P. & Withagen, H.C.A.M. (1993), "Constructive Methods for a New Classifier Based on a Radial Basis Function Network Accelerated by a Tree", *Report*, Eindhoven Technical University, Eindhoven, Holland.

Hanson, S.J., & Pratt, L.Y. (1989), "Comparing Biases for Minimal Network Construction with Back-Propagation", *Advances in Neural Information Processing Systems*, 1, pp.177-185.

Harp, S., & Samad, T. (1991), "Genetic Synthesis of Neural Network Architecture". In Handbook of Genetic Algorithms, Van Norstrand Reinhold : Chapter 15, pp.202-221.

Harp, S., Samad, T., & Guha, A. (1989), "Towards the Genetic Synthesis of Neural Networks", *Proccedings of the Third International Conference on Genetic Algorithms*.

Hirose, Y., Yamashita, K., & Hijiya, S. (1991), "Back-Propagation Algorithm Which Varies the Number of Hidden Units", *International Journal of Neural Networks*, 4, pp61-66.

Holland, J.H. (1975), Adaption of Natural and Artificial Systems, The University of Michigan Press, Ann Arbor.

Jones, A.J. (1993), "Genetic Algorithms and Their Application to the Design of Neural Networks", *Neural Computing and Applications*, 1, pp.32-45, Springer-Verlag, London.

Kohonen T. (1988), "Self-Organising Feature Maps", in Self-Organisation and Associative Memory, 2nd edition, Springer-Verlag, pp.119-157.

Le Cun Y., Boser B., Denker J.S., Henderson D., Howard R.E., Hubard W. & Jackel L.D. (1990), "Handwritten Digit Recognition with a Back-Propagation Network", *Advances in Neural Information Processing Systems*, 3, Morgan Kaufmann, pp.395-404.

Le Cun, Y., Denker, J.S., & Solla, S.A. (1990), "Optimal Brain Damage", *Advances in Neural Information Processing Systems*, 2, pp.598-605.

Le Cun, Y. (1987), "Modeles Connexionnistes de l'Apprentissage", *PhD thesis*, Universite Pierre et Marie Curie, Paris, France.

Leung, H.C., Glass, J.R., Phillips M.S., & Zue, V.W. (1991), "Phonetic Classification and Recognition Using the Multi-Layer Perceptron", *Advances in Neural Information Processing Systems*, 3, pp.255-261.

Leung, H.C., & Zue, V.W. (1989), "Applications of Error Back-Propagation to Phonetic Classification", *Advances in Neural Information Processing Systems*, 1, pp.206-215.

Lippmann, R.P., (1987), "An Introduction to Computing With Neural Nets", *IEEE Acoustics, Speech and Signal Processing Magazine*, 4(2), pp.285-288.

Lowe D., & Webb, A.R. (1991), "Time Series Prediction by Adaptive Networks: A Dynamical Perspective", *IEE Proceedings-F*, 128(1), Feb, pp17-24.

MacQueen J.B., (1967), "Some Methods for Classification and Analysis of Multivariate Observations", *Proceedings of Symp.Math.Statist.Probability*, 5th, Berkeley, 1, pp.281-297, AD 669871, University of California Press, Berkeley.

Marshall, S.J., Harrison, R.F. (1991), "Optimisation and Training of Feed Forward Neural networks by Genetic Alkgorithms", University Of Sheffield, England.

Miller, G.F., Todd, P.M., & Hedge, S.U. (1989), "Designing Neural Networks Using Genetic Algorithms", *Proceedings of theThird International Conference on Simulation of Adaptive Behaviour*, pp.366-375, The MIT Press.

Minsky, M.L. & Papert, S.A. (1969), Perceptrons, Cambridge, MIT Press.

Moody, J. & Darken, C. (1989), "Fast Learning in Networks of Locally-Tuned Processing Units", *Neural Computation*, 1, pp.281-294.

Moody, J. & Darken, C. (1988), "Learning with Localized Receptive Fields", *Research Report YALE/DCS/RR-649*.

Mozer, M.C., & Smolensky, P. (1989), "Skeletonization: A Technique for Trimming the Fat from a Network Via Relevance Assessment", *Advances in Neural Information Processing Systems*, 1 , pp.107-116.

Musavi, M.T. (1992), "On the Training of Radial Basis Function Classifiers ", *Neural Networks*, 5, pp.595-603.

Nadal, J.P. (1989), "Study of a Growth Algorithm for a Feed Forward Network", *International Journal of Neural Systems*, 1, pp.55-59.

Omohundro S.M. (1991), "Bumptrees for Efficient Function, Constraint and Classification Learning", *Advances in Neural Information Processing Systems*, 3, Morgan Kaufmann, pp.693-699.

Omohundro, S.M. (1990), "Geometric Learning Algorithms", *Physica D*, 42, pp.307-321.

Omohundro, S.M. (1989), "Five Balltree Construction Algorithms", *International Computer Science Institute Technical Report TR-89-063*.

Omohundro, S.M. "Efficient Algorithms with Neural Network Behaviour", *Complex Systems*, 1, pp.273-347.

Petersen, G.E., & Barney, H.L. (1952), "Control Methods Used in the Study of Vowels", *Journal of Acoustic Soc. Am.*, 24, pp.175-184.

Poggio, T., & Girosi, F. (1989), "A Theory of Networks for Approximation and Learning", *MIT AI MEMO 1140*.

Powell M.J.D. (1985), "Radial Basis Functions for Multivariate Interpolation", A Review. *Technical Report* DAMPT 1985/NA12, Dept. of App. Math. and Theor. Physics, Cambridge University, Cambridge, UK.

Radcliffe, N. (1991), "Genetic Set Recombination and its Application to Neural Network Topology Optimisation", *PCC Report No EPCC-TR-91-21*, University of Edinburgh, Edinburgh.

Recognition Research (1993), "Manual for the Use of the Autonet Neural Network Package", Recognition Research, Aston Science Park, Bitmingham, England.

Renals, S. & Rohwer, R.J. (1989), "Phoneme Classification Experiments Using Radial Basis Functions, *Proceedings of the IJCNN*, pp.461-467.

Rosenblatt, F. (1959), " Two Theorems of Statistical Seperability in the Perceptron", *in Mechanisms of Thought Processes: Proceedings of a Symposium held at the National Physical Laboratory*, November, 1, pp.421-456, London, HM Stationary Ofice.

Rumelhart D.E., Hinton G.E. & Williams R.J. (1988), "Learning Internal Representations by Error Propagation", in Parallel Distributed Processing. Exploration in the Microstructure of Cognition. (eds Rumelhart D.E. & McClelland J.L.), vol.1, The MIT Press, pp.319-362.

Sietsma, J., & Dow, R.J.F. (1991), "Creating Artificial Neural Networks that Generalize", *International Journal of Neural Networks*, 4, pp.67-79.

Sietsma, J. (1990), "The effect of Pruning a Back-Propagation network", DSTO Materials Research Laboratory, Melbourne, Australia.

Van Der Rest, J.C. (1992), "A Comparative Study of the Use of Non-Hierarchical Clustering Techniques to Partition Data", Bsc Final Year Project, Department of Computer Science, Aston University, Bitmingham, England.

Whitley, D., Dominis, S., & Das, R. (1991), "Genetic Reinforcement learning with multilayer neural networks", *Proceedings of the Fourth International Conferenceon Genetic Algorithms*, pp.562-569, Morgan Kauffman.

Whitley, D., & Hanson, T. (1989), "Optimising Neural networks Using Faster, More Accurate Genetic Search", *Proceedings of the Third International Conference on Genetic Algorithms*.

Widrow, B. (1987), "ADALINE and MADALINE - 1963", *Proceedings of the IEEE 1st International Conference on Neural Networks*, Plenary Speech, 1, pp.143-158.

Williams, B.V., Bostock, R.T.J, Bounds, D.B. & Harget, A.J. (1993), ""A Normalised Chromosome Representation for a Genetic Bumptree Classifier", *Proceedings of the BNNS Symposium on Artificial Neural Networks*, To be published.

Williams, B.V., Bostock, R.T.J, Bounds, D.B. & Harget, A.J. (1994), "Improving Classification Performance in the Bumptree Network by Optimising Topology with a Genetic Algorithm", *Proceedings of the IEEE Conference On Evolutionary Computation*, To be published.

Williams, B.V. (1992), "The use of genetic algorithms in the optimisation of neural networks", *First year Phd report,* Department of Computer Science, Aston University, Birmingham, England.

Wynne-Jones, M., (1992), "Node Splitting: A Constructive Algorithm for Feed-forward Neural Networks", Research Initiative in Pattern Recognition, Malvern.

Wynne-Jones, M., (1991): *Constructive Algorithms and Pruning : Improving the Multi Layer Perceptron* : Proceedings of the 13th IMACS World Congress on Computation and Applied Mathematics : P747-750.

Yao, X., (1992), "A Review of Evolutionary Neural Networks", *Commonwealth Scientific and Industrial Research Organisation Technical Report,* Victoria, Australia.

Yu, Y. & Simmons, R.F. (1990), "Extra Output Biased Learning", *Technical Report AI90-128*, Department of Computer Science & Artificial Intelligence Laboratory, Yaylor Hall 2.124, University of Texas at Austin, Austin, Texas.

Appendix A

Pseudo Code for the MIF bumptree. The learning algorithm employs the Gauss Jordan singular value decomposition technique rather than matrix inversion.

STEPWISE LEVEL 1

```
1- Create the initial functions to be considered for addition to the network.
   For functions = 1 to 2 (the original functions to be added to the network).
   {
       If function(functions) is active
       {
2-             Optimise the Alpha and Beta values.
       }
   }
   While training is not satisfactorily concluded
   {
       For all functions that have not previously had their level of error calculated.
       {
               Calculate the level of error for function(functions).
               Test Performance of function(functions).
               If  performance is unsatisfactory
               {
3-                     Add in further functions with function(functions) as the parent .
               }
       }
   }
   Display the performance of the network.
```

STEPWISE LEVEL 2:

Further refining of the steps identified in level 1.

The first process to be further broken down is that identified above as 1, namely the process to Create the initial functions to be considered for addition to the network.

```
Create centres for the 10 initial functions to be considered for addition to the network.
For patterns =1 to number of patterns in the training set
{
    For the 10 functions to be considered for addition to the network
    {
            Calculate the activation of function(functions)
    }
    Assign pattern(patterns) to the function with the highest activation level.
}
For the 10 functions to be considered for addition to the network
{
    If function(functions) is active
    {
            Optimise the Alpha and Beta Values.
    }
}
For the 10 functions to be considered for addition to the network
{
    Calculate the level of error for function(functions)
    Calculate the goodness measure for function(functions).
}
Select the best function.
Attain the other function to be added to the network.
For the 2 functions to be added to the network
{
    For patterns =1 to number of patterns in the training set
    {
            For the 2 functions to be considered for addition to the network
            {
                    Calculate the activation of function(functions)
            }
            Assign pattern(patterns) to the function with the highest activation level.
    }
}
```

The second process to be further broken down is that identified above as 2, namely the process to Optimise the Alpha and Beta values for the functions presented to the routine. The routine can be employed for the 2 functions added to the network when an addition takes place. It can also be employed for the 10 functions that are considered for addition to the network. The approach given here employs the Gauss Jordan singular value decomposition pseudo matrix inversion technique.

Create matrix 1.
Create matrix 3.
Carry out the singular value decomposition on matrix 1 to arrive at the pseudo inverse of this.
Multiply matrix 3 by the pseudo inverse of matrix 1 to create matrix 2 which contains the optimised Alpha and Beta values.

The third process to be further broken down is that identified above as 3, namely the process to add in further functions when the original functions have proved unable to satisfactorily partition the problem space. This process is almost the same as that involved in placing the initial functions on the problem space. The fundamental difference concerns the positioning of the functions. When the initial functions are added to the network they can be positioned anywhere in the problem space. The functions added at this stage of the learning algorithm have to have their centre point situated within the area covered by the patterns upon which their parent function was active.

Create centres for the ten functions to be considered for addition to the network (using the procedure identified in step2).

```
For patterns =1 to number of patterns in the training set
{
    For the 10 functions to be considered for addition to the network
    {
            Calculate the activation of function(functions)
    }
    Assign pattern(patterns) to the function with the highest activation level.
}
For the 10 functions to be considered for addition to the network
{
    If function(functions) is active
    {
            Optimise the Alpha and Beta Values.
    }
}
```

```
 For the 10 functions to be considered for addition to the network
 {
    Calculate the level of error for function(functions)
    Calculate the goodness measure for function(functions).
 }
Select the best function.
Attain the other function to be added to the network.
 For the 2 functions to be added to the network
 {
    For patterns =1 to number of patterns in the training set
    {
            For the 2 functions to be considered for addition to the network
            {
                    Calculate the activation of function(functions)
            }
            Assign pattern(patterns) to the function with the highest activation level.
    }
 }
}
```

Appendix B

Pseudo Code for the MIF bumptree. The pseudo code given in this appendix provides more detailed information on the process of Optimising the weight and bias, or Alpha and Beta parameters. The optimising process identified in this appendix is that employed when optimising the 10 candidate functions considered for addition to the network with the MIF bumptree. However, the same process is utilised by the other bumptree approaches, and by the MIF approach when optimising the functions actually being added to the network. The difference is that instead of the process being carried out 10 times it is carried out for as many functions as require it. When it is carried out for the addition of the final two functions to the network it is carried out for two functions. However, with the n-function bumptree it can be carried out for any number of functions.

1 - Assign Values to the Elements of Matrix 1 for each function

```
For F = 1 to 10
 if Function[F] active
  x1 = 1
  Temp 2 = 0
  For Row = 1 to Num Input Units
   x2 = 1
   For Column = 1 to Num Input Units
    For Pattern = 1 to Num Patterns Function [F] active on
     Temp=Function[F] InputPattern[pattern][x1]*Function[F]InputPattern[pattern][x2]
     Temp2=Temp2 + Temp
    End
    Function.Matrix1[row][column] = Temp2
    x2 = x2 + 1
    Temp2 = 0
   End
  x1 = x1+1
 End
 For Row =1 To Num Input Units
  Temp = 0
  For Pattern = 1 To  Num Patterns Function [F] active on
   Temp = Temp + Function[F].InputPattern[Pattern][Row]
  End
  Function[F] Matrix1[Row][Num InputUnits + 1]
 End
 For Column = 1 To Num Input Units
  Temp = 0
  For Pattern = 1 To  Num Patterns Function [F] active on
   Temp = Temp + Function[F].InputPattern[Pattern][Row]
```

```
  End
  Function[F] Matrix1[Num Input Units + 1][Column]
 End
End
```

2 - Assign Values to the Elements of Matrix 3 (the result matrix) for each function

```
For F = 1 to 10
 if Function[F] active
  For Row = 1 to Num Input Units
   For Column = 1 to Num Input Units
    Temp 2 =0
    For Pattern = 1 to Num Patterns Function [F] active on
     Temp=
      Function[F]OutputPattern[pattern][x1]*Function[F]InputPattern[pattern][x2]
     Temp2=Temp2 + Temp
    End
    Function.Matrix3[row][column] = Temp2
   End
  End
  For Column = 1 To NumOutput Units
   Temp = 0
   For Pattern = 1 To  Num Patterns Function [F] active on
    Temp = Temp + Function[F].OutputPattern[Pattern][Row]
   End
   Function[F] Matrix3[Num Input Units + 1][Column] = Temp
  End
 End
```

3 - Assign Values to the Elements of Matrix2 (the Alpha & Beta matrix) for each function

This procedure is concerned with inverting matrix1 to allow it to be multiplied by Matrix3, which is the result matrix, to arrive at the optimal Alpha and Beta values. This procedure employs the Gauss Jordan singular value decomposition in place of a standard matrix inversion rechnique. This is to avoid problems with singular matrices. The steps are not broken down into great detail since it is simply a case of employing the Gauss Jordan singular value decomposition technique.

```
For Pass = 1 To Num Input Units + 1
        Choose pivots within Matrix1
        Utilise the pivot value to produce pseudo inverse of Matrix1
End
```

Appendix C

This appendix contains details of the genetic bumptree introduced in section 5.5.

The Genetic Bumptree

In order to represent a bumptree structure as a chromosome suitable for manipulation by genetic operators, various issues needed to be addressed, and these are examined below. An ideal representational scheme for a GA should have properties of completeness, closure and low-redundancy - and these will now be considered in relation to the bumptree. First, the issue of completeness. An ideal chromosomal representation scheme should be complete in the sense that the genotype should be capable of representing any point in the space of the possible phenotypes. In the case of the abstract notion of the bumptree, there was obviously an infinite number of possible tree structures which could be defined in a space of given dimensionality, and hence the chromosome would have to be of infinite length to truly be complete. However, this could be made of finite length if the space of possible bumptrees is of finite volume describing discrete points in which the bumptrees must exist.

Second, the issue of closure. An ideal genetic representation should be closed with respect to the space of possible phenotypes to be represented. For the bumptree this means that it should be impossible for any genetic string to define an illegal tree structure. That is, if Omohundro's approach to centring and constraining the functions was employed the bumptree structure would have to be binary in nature and each function below the top level would have to be fully enclosed by its parent function. Hence, for a representation to be closed it must only produce representations that describe valid states in the specified problem. If a representation is not closed, it becomes necessary to check the validity of

each new chromosome which arises through mutation or crossover, and to decide on a suitable scheme to deal with invalid chromosomes. In the case where invalid chromosomes represent a significant proportion of those arising, evolutionary paths through genetic space become discontinuous, and the search ability of the GA may be greatly reduced, whilst computational demands are greatly increased.

Third, the issue of low-redundancy. In an ideal genetic representation there should be a one to one mapping between the space of possible chromosomes and the space of possible phenotypes. For the bumptree neural network, no two different chromosomal representations should ever yield identical bumptrees. If the situation exists where two identical bumptrees can be constructed from quite different chromosomes, then a child produced by combining genetic material from two such identical parents could be very different to both. This effect reduces the GA's ability to discover and combine good genetic 'building blocks'. Redundancy in the representation scheme has proved to be a big obstacle in the application of GA's to neural network problems in general and has led to the so-called 'permutation problem' in optimising the MLP (Radcliffe 1991).

The GA approach employed to optimise the bumptree neural network structure will now be examined. It was decided that chromosomes of fixed length would be used to represent the bumptree. It is possible to allow the chromosomes to vary in length, but this demands highly specialised genetic operators, with an attendant increase in complexity. The chromosome employed consisted of floating-point genes, arranged as a concatenation of 30 sets of parameters, each set defining a single function. It was not necessary for all 30 functions to be present in the final bumptree. The first two blocks of parameters on the chromosome define the root-level functions and thereafter the presence or absence of further functions is dictated by a single parameter associated with each function which determines whether that function is a terminal function or contains two child functions. All

potential functions have fixed positions on the chromosome, and if function n is non-terminal, its children will be functions $2n+1$ and $2n+2$, as is shown in figure C1.

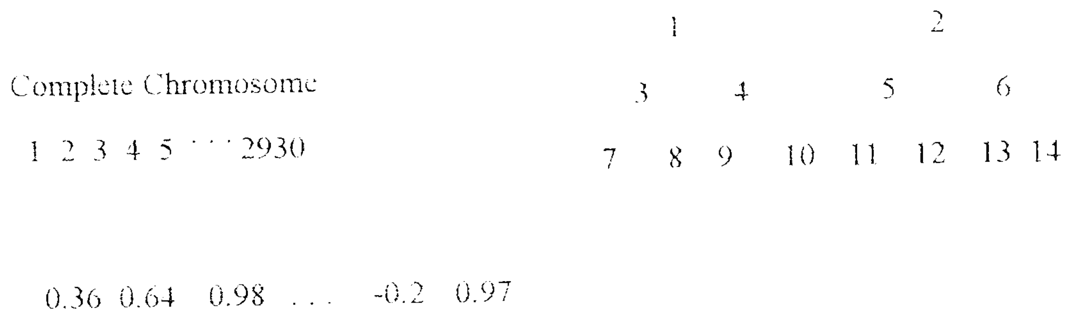

Figure C1 - Chromosome representation of bumptree parameters.

The problem with encoding structural parameters for the bumptree is that there exists a strong dependency between the parameters which are subject to certain criteria being fulfilled. The genetic bumptree as originally developed adhered to the constraints imposed by Omohundro's approach (section 4.4) on the size and location of the functions within the network. Hence, a functions maximum radii and the space in which its centre must exist were not known *a priori*, but depended on the centre and radii of its parent function. A naive coding scheme, where each function's centre and radii are coded as real values in a global co-ordinate system therefore becomes untenable. It becomes necessary to check and possibly adjust the value of every gene on any new bumptree chromosome, to ensure that all the functions are correctly enclosed by their parents. A reduction in a particular function's radii, arising as a result of a mutation, may necessitate adjustment of the radii and possibly the centres of all the functions below it in the tree.

In order to employ a GA to optimise a bumptree neural network a simple, but novel, chromosomal representation has been introduced. In this representation, every gene is a single floating point value normalised to a value in the range + or -1. The process of translating a chromosome into a bumptree neural network commences by defining the volume of the problem space within which the bumptree is to be placed. The genetic bumptree defines this area as a hyper-ellipse which encloses all the data points in the training set. A co-ordinate system based on this volume was then calculated, in which the origin is at the volume's centre. The genes which code the centre co-ordinates of each of the two root-level functions in the tree are interpreted in terms of this co-ordinate system. In a given dimension, the value of a gene which codes a root-level function's radius is mapped onto the region between the function's centre and the perimeter of the volume within which the bumptree is constrained. Having constructed the two root-level functions, each non-root level function in the tree is constructed in a similar manner, the function's co-ordinate system being defined by the volume enclosed by its parent.

It can be shown that this representation scheme has the desirable properties outlined earlier of completeness, closure and low-redundancy. Given the constraint that some maximum number of functions is to be allowed, the chromosome is able to represent all possible bumptrees within a given discrete space. The representation is, therefore, complete. The normalised relative coding of function centres and radii ensures that the integrity of the tree structure is inherent in the coding - it is impossible to represent an illegal bumptree structure, so the representation is closed. Moreover, as well as ensuring that chromosomes always produce legal bumptrees, this coding helps the important structural qualities of parent bumptrees to be preserved when chromosomes are recombined by crossover or modified by mutation, as there is a smooth and continuous mapping between the space of the chromosomes and the space of bumptrees. Finally, as each potential function has its parameters coded as a fixed position on the chromosome there is very little redundancy

inherent in the representation scheme. For a given bumptree structure, there exist only two different possible chromosomes which could have yielded that structure, which contain identical genetic information for the two main branches of the tree, but with the root-level functions transposed.

Having examined the normalised representation scheme that has been employed in the genetic bumptree, it is now necessary to describe the other details of the genetic bumptree classifier. In the genetic bumptree, each chromosome of the population has associated with it a unique co-ordinate on a two dimensional grid. This spatial relationship between chromosomes is used to implement local, rather than population wide selection, reproduction and replacement strategies. This approach has been found to be beneficial in preventing the GA from converging prematurely due to loss of genetic diversity, by allowing the emergence of sub-populations of chromosomes, isolated by distance from each other and with a consequently limited rate of exchange of genetic material.

In each generation, a fixed number of chromosomes is replaced, and the rest of the population is unaffected. This replacement occurs as follows. Initially a random point on the population grid is selected. A parent chromosome is chosen, being the fittest chromosome encountered during a fixed length walk across the grid from the initial starting point. A second parent chromosome is chosen in the same manner, during a second random walk from the same initial starting point. These two parent strings are combined by a genetic crossover operator to form a single child chromosome, which is then subjected to slight random mutation. A third random walk is made from the initial starting point and the least fit chromosome encountered during this walk is replaced with the new child. Chromosomes that have already been replaced in the current generation may not be selected as parents or as candidates for replacement. This entire process is repeated from different initial random starting points on the population grid. After a certain proportion of

the population has been replaced in this manner, all newly created child strings are evaluated for fitness, and this completes one generation of the GA.

The crossover operator employed in the genetic bumptree was a variation of uniform crossover, in which many small sections of genetic material, sampled randomly and with uniform probability along the length of one parent chromosome were exchanged with corresponding sections from the second parent chromosome. In addition, every new child chromosome created is subjected to 'creeping' mutation, which is effected by randomly selecting a number of genes along the length of the chromosome (around 5% of the total number of genes in the chromosome) and perturbing their floating point values by adding or subtracting small random quantities, usually in the range of + or - 0.2. Checks were then performed to ensure that the values remained in the range of + or - 1.

Appendix D

This appendix contains the pseudo code for the backpropagation learning algorithm that was employed by the MLP used in this study. Step 1 sets out the basic steps involved in the learning algorithm.

STEP 1

```
Initialise the weight and bias values
A1 - Calculate the error of the network.
While the error level is not satisfactory Do
B1 -    Adjust Weights
        Calculate Error
End While
```

STEP2

This step provides the pseudo code required to calculate the error of the network.

```
errorsum=0.0;
for pattern = 1 to num patterns   (tp)

        clamp input units to their required value
A2 -    calculateoutput
        for units in layer 3   (j)
                Calculate the error over the entire training set AS
                [errorsum=errorsum+ ((targetvectors[tp][j]-output[(numunits-
unitsinlayer3)+j]) * (targetvectors[tp][j]-output[(numunits-unitsinlayer3)+j]))]
        End
        error=errorsum*0.5;
End
```

This step further breaks down A2 - the process to calculate the output of the network.

```
for units in layer 2   (j)
        temp=0.0;
        for units in layer 1   (i)
                temp=temp+(output[i]*weight[j][i]);


/* To get the value referred to in the literature as NETpj, it is necessary */
/* to add the bias value to the temporary value calculated above            */

                netoutput[j]=temp+bias[j];

/* From this net output it is possible to calculate the actual output of the */
/* unit. A threshold has been employed in order to stop figures going to     */
/* unacceptable values                                                       */

                output[j]=1.0/(1.0+exp(-netoutput[j]));

        End
End
for units in layer 3   (j)
        temp=0.0;
        for units in layer 2   (i)
                temp=temp+(output[i]*weight[j][i]);

/* To get the value NETpj, it is necessary */
/* to add the bias value to the temporary value calculated above            */

                netoutput[j]=temp+bias[j];
                output[j]=1.0/(1.0+exp(-netoutput[j]));
        End
End
```

STEP 4

This step further breaks down the process required to adjust the weights.

```
/* Calculate the weight change on the connections to the output units */

total=numunits;
for units in layer 3    (j)
        delta[j]=(targetvectors[tp][j-(total-unitsinlayer3)] -output[j]) * output[j] * (1.0-
output[j]);
End
B2 -    weight_change

/* Calculate the weight changes on the connections to the hidden units */

for units in layer 2   (j)
        sum=0.0;
        for  units in layer 3    (k)

/* For all units directly above the hidden layer pass back the error */

                sum=sum+(delta[k]*weight[k][j]);
        End
        delta[j]=sum*output[j]*(1.0-output[j]);
End
total=numunits-unitsinlayer3;
weight_change

/* Update the bias for the relevant units (all except the input units) */

for  units in layers 2 and 3    (units)
        deltabias[units]=(learningfactor*delta[units]) +
(momentumfactor*olddeltabias[units]);
        bias[units]=bias[units]+deltabias[units];
End
```

The next step further breaks down the process identified as weight change above.

```
for units in top layer  (j)
        for units in bottom layer  (i)
                deltaweight[j][i] = (learningfactor*delta[j]*output[i])
+(momentumfactor*olddeltaweight[j][i]);
                weight[j][i]=weight[j][i]+deltaweight[j][i];
        End
End
```