Extending and Benchmarking Cascade-Correlation

Extensions to the Cascade-Correlation architecture and benchmarking of feed-forward supervised artificial neural networks

by

Samuel George Waugh, BSc (Hons)

Submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

University of Tasmania, August, 1995

Cent
Thesis
WAUGH
Ph.D
Comp. Sci
1997

Abstract

This thesis is divided into two parts: the first examines various extensions to Cascade-Correlation, and the second examines the benchmarking of feed-forward supervised artificial neural networks, including back-propagation and Cascade-Correlation.

The first extensions to the training mechanism of Cascade-Correlation involve the inclusion of patience to stop the addition of hidden nodes and the introduction of alternative methods for training the candidate pool. These methods greatly improve the training speed of the algorithm. Secondly, reducing the number of connections within Cascade-Correlation networks is examined: by the introduction of hidden nodes with limited connection strategies, and by the pruning of the fully-connected hidden nodes and the output layer. Three methods of stopping the pruning process are briefly investigated. It is shown that adding limited connected hidden nodes is effective in altering the style of network topology, if not reducing the number of connections. Pruning within Cascade-Correlation drastically reduces the number of connections required without affecting the classification performance of the networks developed. Furthermore, all the different methods of halting the pruning process are shown to be effective.

The second part of the thesis concentrates on benchmarking feed-forward supervised artificial neural networks, in particular Cascade-Correlation. The earlier part of the thesis highlights the need for effective benchmarks, as a large number of real-world problems do not require anything more than a single layer of weights to achieve near optimal performance given the available data. The second part initially investigates two new real-world problems. Although both turn out to be useful problems to examine — testing many of the features of Cascade-Correlation described earlier — they too do not require much more than a single layer of weights, and hence do not test the power of Cascade-Correlation or other systems which allow the use of hidden nodes. Two methods of generating artificial data are then examined as ways of producing increasingly complex data sets. The application of these benchmarks to the comparison of various artificial neural network methods is examined. The generated data sets are effective in highlighting the differences between the algorithms, for example it is shown that Quickprop and the activation function offset methods of accelerating training are not always useful, and provide more detailed results on the various Cascade-Correlation modifications.

Statements of originality and access

This thesis contains no material which has been accepted for a degree or diploma by the University or any other institution, except by way of background information and duly acknowledged in the thesis, and to the best of my knowledge and belief no material previously published or written by another person except where due acknowledgment is made in the text of the thesis.

[illegible]

This thesis may be made available for loan and limited copying in accordance with the *Copyright Act 1968*.

[illegible]

Acknowledgments

Thanks to my supervisors Tony Adams and Phil Collier: their help and insightful comments were invaluable. Thanks particularly to Tony, who cheerfully put up with me annoying him every week. Thanks also to Scott Fahlman who was of great assistance during the initial development of this work, and who is always ready to answer questions. Further thanks to the anonymous examiners for their helpful comments and suggestions.

Thanks also to the members of the Artificial Neural Network Research Group and the department's postgraduate students — particularly Julian Dermoudy, Carl Lewis, Peter Vamplew, Tim Freeman and Lee Arnould — who helped my postgraduate studies go a little faster.

Acknowledgment and thanks must also go to other groups for their assistance in producing this thesis. Thanks to the Tasmanian State Government Department of Education and the Arts for providing after-hours access to their machines, resulting in over 13000 hours of simulations being completed. In particular, Jim Palfreymann and Dr John Gilbert deserve special mention. To the Tasmanian Government Department of Primary Industry and Fisheries Marine Research Laboratories, in particular Warwick Nash, for generously providing the abalone data. To the University of Newcastle Centre for Linguistic and Literary Studies, in particular John Burrows and Hugh Craig, for the data on Romantic and Renaissance tragedies. Finally thanks to Michael Fraser, Simon Talbot and Tony Adams, respectively, for suggesting these resources in the first place.

The patience and assistance of the many proof readers who made it through the earlier drafts also deserves recognition, in particular Tony Adams, Trudy Steedman, Cristina Cifuentes, Phil Collier and Julian Dermoudy.

On a more personal level, a great deal of thanks to my parents, for getting me to the stage where I could attempt this at all. Finally, thanks to Trudy Steedman for always being my best friend.

Contents

1 Introduction

In recent years there has been an enormous increase in the amount of research conducted in artificial neural networks. This may be loosely divided into two complementary areas: firstly, the application of computational methods to the development of realistic models of neural functions, and secondly the application of the distributed computation methodology to solving problems, not necessarily in a biologically plausible manner.

One of the most developed and researched areas in the applications part of artificial neural networks is inductive learning — the learning of a theory from individual examples presented to the system. In particular, supervised learning — where an answer is known and used to improve performance — is particularly popular. The back-propagation algorithm [Rumelhart, Hinton & Williams 1986] is easily the most frequently used artificial neural network model, not only because of its simplicity, but also because of its effectiveness at producing good solutions to a wide range of problems.

One of the difficulties with the back-propagation algorithm, and others like it, is that details of the network structure need to be decided prior to training. This requires a priori knowledge of the problem to obtain good performance, gathered either from knowledge of the problem domain or from experimentation using the learning algorithm.

In response, attempts have been made to develop algorithms which change their internal structure as well as training the network weights, with the aim of removing the onus on the user of selecting the network topology. An artificial neural network which dynamically alters its topology, not only alleviates the need for human intervention, but also potentially gives extra flexibility which allows the training algorithm to more effectively find a solution [Baum 1989].

One of the more promising algorithms for dynamically altering artificial neural network topologies is Cascade-Correlation (Cascor) [Fahlman & Lebiere 1989]. This algorithm starts with a minimal network architecture, to which hidden nodes are added as required, forming feature detectors within the network. The first part of this thesis examines this algorithm, extending the methods of training and examining further ways of altering the final network topology.

A further difficulty with the development of inductive learning via artificial neural networks is the frequent reliance on minimal testing to measure the performance of various techniques. The currently available benchmarks are not entirely suitable, as will be seen from the results of Part I, and comparatively little literature is devoted to the development of benchmarks for inductive learning systems. A large number of generated benchmarks are too simple to be realistic, and are thus not able to test the algorithms such as Cascor.

Hence the second part of this thesis examines the area of benchmarking supervised inductive learning — in particular artificial neural networks. Two new real-world problems and two methods for generating complicated artificial problems are examined and assessed.

1.1 Organisation of thesis

To limit the size of this thesis it is assumed that the reader has background knowledge of inductive learning, particularly classification which involves the separation of examples into distinct classes; and supervised feed-forward artificial neural network methods.

The main body of this thesis is in two major sections. The first part involves alterations made to the Cascor neural network architecture in an effort to improve its performance. This consists of three chapters. Chapter 2 reviews methods of dynamically altering the structure of feed-forward fully-supervised artificial neural networks, and then details an outline into which all such algorithms fit. The chapter is concluded by giving a description of the Cascor algorithm, the parameters and data sets used, and the results of Cascor as applied to nine problems used for benchmarking the first part of the thesis. Chapter 3 examines methods for assisting and speeding the training process: a method used to halt training when little performance increase is being achieved; and alternative methods for training the candidate nodes. Chapter 4 examines methods of reducing the number of connections within a Cascor network — the aim being to produce a smaller classifier which will generalise at least as well and possibly better by using fewer free parameters. This is addressed in two ways: by the addition of hidden nodes which have a limited initial connection strategy, and by the pruning of hidden nodes and the output layer to reduce the number of connections after a suitable amount of training is completed.

The second part of this thesis examines methods of benchmarking artificial neural network inductive learning systems. In Chapter 5, a review of the literature is presented which highlights the important features of data sets which need to be considered. This is followed by a summary of different benchmarks that have been presented: both those containing real-world problems, and those containing artificially generated data. Finally, an examination of the performance of Cascor on a number of these benchmarks is given. Chapter 6 examines two new real-world problems, using Cascor as the major development tool, in an attempt to find tasks which require the processing power of a reasonable number of hidden nodes to be solved. The problems relate to the ageing of abalone shellfish from Tasmanian waters, and the separation of Romantic and Renaissance tragedy authors. Chapter 7 examines two methods of generating complicated data sets, and their application to comparisons between various artificial network training methods.

Finally, Chapter 8 concludes the work in the thesis, and suggests further work which may be conducted in both the areas of examining Cascor and benchmarking strategies. Full details of the experiments undertaken in Part I are detailed in Appendices A through D. Appendix E is an abridged version of the manual for the simulator used to perform the Cascor experiments [Waugh 1995c], and Appendix F gives the complete bibliography.

1.2 Inclusion of papers

For clarity the papers which have been included within this thesis as part of the author's own work are outlined with references to the relevant sections. Firstly, those which have been accepted in refereed conferences are given:

[Waugh & Adams 1993]	§5.3
[Collier & Waugh 1994]	§5.3
[Waugh & Adams 1994]	§4.1
[Adams & Waugh 1995]	§8.1
[Waugh 1995a]	§3.1 and §3.2
[Waugh 1995b]	§7.1
[Waugh & Adams 1995]	§4.2

Secondly, unrefereed works are outlined:

[Waugh 1994a]	§2 and §4.2
[Waugh 1995c]	§E

Part I Extensions to Cascade-Correlation

2 Background to dynamic learning

One of the major criticisms of fully-supervised feed-forward artificial neural networks is their failure to cope with requirements for different topologies. Usually only a simple, fixed network structure is used: namely one hidden layer with no shortcut connections, forming two processing layers. The problem is not due to the limitations of particular weight training algorithms, such as back-propagation [Rumelhart, et al. 1986], but rather is due to the limits of the network's structure and how this is developed [Baum 1989]:

> ... it is unlikely that any algorithm which simply varies weights on a net of fixed size and topology can learn in polynomial time. ... obstructions to rapid learning can be avoided if one considers algorithms with the power to add neurons and synapses, as well as simply varying synaptic weights.

An artificial neural network has a set number of inputs and a set number of outputs, as defined by the problem being addressed. However, the internal hidden connections, weights and nodes may be altered in any way by the training algorithm.[1] This includes deciding what connections are present between nodes, whether there should be distinct layers of nodes, and so on. Overall, the number of free parameters, or the ability of a network to model further data set features, corresponds roughly with the number of connections [Cortes, Jackel & Chiang 1995]. Thus, the modification of network features allows for more parameters to be added to model the underlying data set function, or the removal of parameters to avoid over-specialisation on the given training data.

This chapter investigates the dynamic alteration of network topologies during the training of fully-supervised feed-forward artificial neural networks. It is suggested by many researchers (for example, [Baum 1989; Fahlman 1990]) that dynamically altering networks presents good opportunities for developing optimal network architectures that generalise well. This chapter gives an overview of past methodologies, both constructive and destructive; gives a general reasoning as to why certain types of topology-changing algorithms are successful based on a framework developed from the literature; and concludes by giving a more detailed description of Cascor and introducing the remaining chapters in this part of the thesis.

[1] In this thesis, the term connection is used to indicate the presence of a link between two nodes, whereas the term weight is used to indicate the numeric strength of the connection.

2.1 Current literature on dynamic neural networks

The main aim of dynamic neural network algorithms is to produce a network which effectively solves the problem at hand. This is done in two basic ways: by either removing unnecessary features to make the network smaller, or adding features to a minimal network as required. The fewer free parameters that exist in the network, the more likely that they will be correctly estimated from the available training data. The greater the number of free parameters, the more likely the network will have the ability to model all of the data. Thus the task of the dynamic neural network algorithm is to produce the most appropriate number of parameters in a form which models the function underlying the data, without allowing for over-specialisation.

Few papers summarise the major construction and pruning strategies. Wynne-Jones concentrates on weight decay methods, and node construction and pruning — the paper does not examine the ideas of connection pruning in any detail [Wynne-Jones 1991a]. Hertz, Krogh and Palmer briefly examine connection pruning, weight decay and node construction algorithms [Hertz, Krogh & Palmer 1991]. Reed gives a very good overview of the different pruning strategies, identifying the two main groups of pruning algorithms: sensitivity calculation methods, and penalty-term methods [Reed 1993]. Fiesler provides a very brief tabulated overview of many methods of changing topology within perceptron-style networks and others [Fiesler 1994]. The general perceptron-style topology altering methods are described more fully below.

2.1.1 Removing connections — saliency methods

There are two main ways of removing connections between nodes: by pruning using saliency measures, or by pruning using penalty-term methods to reduce weights to zero. Removing weights by the use of penalty terms will be considered in the next section.

The saliency of a connection is the change in error after the removal of that connection, or the sensitivity of the network to the removal of that connection [Mozer & Smolensky 1988]:

$$\text{Saliency} = \text{Error (connection removed)} - \text{Error (connection present)} \qquad (2.1)$$

The higher the saliency value, the more important the connection is. Low saliencies indicate that a connection has little importance, and negative saliencies indicate a weight that is doing more harm than good. Often the saliency is estimated in some way to speed the removal of connections. The actual saliency or its estimate may then be used to decide which connections to prune or remove.

Thodberg examines the removal of connections by a process of direct elimination: each connection is pruned in turn, and the resulting network is retrained for a short period [Thodberg 1991]. If the network still performs reasonably the change is kept, otherwise the connection is returned along with the original network weights. This method may be time consuming — to the point of being computationally intractable — but it has reasonable success in removing extra connections and.retraining existing ones. A saliency estimate is not calculated as the weights are individually removed, and the effect on the network is evident after training.

Skeletonization [Mozer & Smolensky 1988; Mozer & Smolensky 1989] is a technique which removes nodes by assessing the relevance of their connections. This process may be simply extended to the removal of connections, as it actually estimates the error after removing a single connection, which is combined to give the error after the removal of a node. Karnin notes that Mozer and Smolensky's sensitivity measure is defined to be used with a particular linear error measure, and he goes on to describe a sensitivity measure specifically for network connections which is independent of the error function used [Karnin 1990].

Another sensitivity measure is Optimal Brain Damage (OBD) [Le Cun, Denker & Solla 1989] and the subsequent method Optimal Brain Surgeon (OBS) [Hassibi & Stork 1992]. OBD calculates saliencies by comparing the results of the main diagonal of a Hessian matrix, or the second derivatives, of the change in error with respect to the weights. A Taylor expansion of the error results in four groups of terms: the first term is assumed to be constant, the second term is assumed to be zero as the network has been trained to a local minimum and the slope is constant, the third term results in the Hessian matrix which calculates the quadratic approximation or curvature of the error surface, and the higher order terms are ignored and assumed to be negligible. The main diagonal of the Hessian matrix gives an estimate of which connections are required. OBS follows this up with improvements, mainly by using the full Hessian matrix, in contrast to using only the diagonal. This has the advantage of requiring no retraining after the changes have been made whereas OBD does require retraining. Nevertheless, these computations can be quite expensive. A number of papers have examined these algorithms further, with comparisons between OBD and OBS, and some improvements and modifications to the algorithms — particularly to the OBS algorithm (for example, [Gorodkin, Hansen, Krogh, Svarer & Winther 1993; Hassibi, Stork & Wolff 1993; Tolstrup 1995]).

A further method called Principle Components Pruning (PCP) has also been developed [Levin, Leen & Moody 1994]. As the name suggests, this method prunes connections by the use of principle components analysis to calculate their relative worth. The paper shows that PCP has a computational complexity much less than OBS, but greater than OBD. It also claims that PCP is likely to produce better results than OBD, although this is not backed up with results.

Another method [Tsaptsinos, Mirzai & Leigh 1992] uses correlation analysis for the removal of unnecessary connections; and there are a number of papers which optimise the network architecture by using genetic algorithms (for example, [Nolfi & Parisi 1991; Hancock 1992; Kendall & Hall 1992; Kendall & Hall 1993]), though it is not obvious from the papers that this is an efficient process, especially for larger networks [Hertz, et al. 1991].

Sensitivity measures also have their critics [Reed 1993]:

> ... most of the sensitivity methods ... don't detect correlated elements. ... An extreme example is two nodes which cancel each other out at the output. As a pair, they have no effect on the output, but individually each has a large effect so neither will be removed.

Retraining may break such a deadlock, but this will not necessarily result in an optimum solution.

2.1.2 Modifying weights — penalty terms

Another way of removing a connection, as mentioned previously, is by changing the weight of that connection so that the connection has no effect. Pruning of weights, or regularisation, is performed by adding a penalty term affecting the network complexity to the network error term which is being minimised, with the purpose of changing the magnitude of the network weights. With this method generally weights are reduced to remove their effect. This relies on the weight training algorithm to reduce the weights by minimising the overall network cost:

$$\text{Network cost} = \text{Network error} + \text{Network complexity} \tag{2.2}$$

The minimisation of the overall cost results in the training of the network weights and the alteration of the weights to minimise the term specifying the network complexity.

Weight decay and weight elimination are mentioned quite extensively in the literature. Less frequently mentioned is the use of weight enhancement to generate weights from zeroed connections [Chauvin 1988].

Krogh and Hertz give a good overview of the area of weight decay [Krogh & Hertz 1991] — the idea being initially attributed to Rumelhart [Hanson & Pratt 1988; Wynne-Jones 1991a] — and then go on to show that weight decay is an improvement over standard gradient descent back-propagation weight training. Weight decay in its basic form is simply the gradual reduction of the smaller weights to minimise their effect in relation to the larger weights in the network. Hanson and Pratt, and Burkitt and Ueberholz also examine this method, with the latter attempting to separate the learning from the weight reduction phases [Hanson & Pratt 1988; Burkitt & Ueberholz 1993].

Weigend, et al. propose a system for weight elimination which subsumes much of the work done in weight decay [Weigend, Rumelhart & Huberman 1990; Weigend, Rumelhart & Huberman 1991]. This system, similarly operates by training to a set minimum error for a particular problem and trades off complexity and the network error. The method allows for the alteration of the weight cost function so that smaller weights or larger weights become relatively expensive.

Nowlan and Hinton describe a further penalty term method called soft weight sharing [Nowlan & Hinton 1992]. Under this scheme an alternative penalty term is used which favours the reduction of smaller weights. The penalty term involves the combination of two Gaussian functions. One function is used to reduce the smaller weights, while the other targets larger weight values — the latter, in the limiting case, may be replaced with a uniform distribution. The penalty term is reduced by allowing the means and variances of the Gaussians used to adapt such that the variances shrink, drawing the weights into having similar values, which in turn implements a 'soft' version of weight sharing, whereby a single weight is used by several connections. By starting the penalty-term Gaussians with high variances, all the weights influenced by the respective Gaussians will be forced to have similar values. The wide variance at the beginning means that the Gaussians will not adversely affect the training process. The sharing of weights results in a reduction of the degrees of freedom that the network may use for over-fitting the data. There is, of course, a greater cost with the increased complexity of the weight optimisation process.

Not everyone is in favour of these penalty-term methods. Mozer and Smolensky state [Mozer & Smolensky 1989]:

> ... our impression is that it is a tricky matter to balance a primary and secondary error term against one another. ... In our experience, it is often impossible to avoid local minima — compromise solutions that partially satisfy each of the error terms.

Karnin also notes that penalty-term methods can 'interfere with the learning process' [Karnin 1990]. Hanson and Pratt indicate that weight decay is not effective at removing hidden units [Hanson & Pratt 1988]. To remove connections other methods would have to be employed to zero small weights at some stage during training, as the weight decay methods are not effective in reducing the weight values to absolute zero [Sietsma & Dow 1988].

2.1.3 Changes to the number of hidden nodes

The large majority of papers with respect to topology altering algorithms consider changes in the number of hidden nodes, rather than changes to the connections or weights between nodes. Most of these papers concentrate on the introduction of new nodes when the network is not capable of solving the problem at hand. Only a few examine the removal of nodes. These different styles will be considered in turn.

2.1.3.1 Construction — adding hidden nodes

Many techniques are based on the standard configuration for a back-propagation style of network, with two layers of processing nodes. The idea of splitting nodes in the hidden layer, or simply adding extra nodes to the hidden layer is very common (for example, [Ash 1989; Hanson 1989; de le Maza 1991; Platt 1991; Refenes & Vithlani 1991; Wynne-Jones 1991b]).

Several methods have been developed which grow layers as well as the number of nodes in a single layer. The majority have been designed for problems with binary inputs, but could be extended to cover more general cases. Gallant presents three concepts of network growth which involve the addition of individual nodes [Gallant 1986]: growing nodes with connections to the previous node and the inputs (Tower Construction), growing nodes connected to all previous nodes and inputs (Inverted Pyramid Construction), and adding static nodes in layers. No results are given as to the effectiveness of these ideas. The Tiling algorithm [Mézard & Nadal 1989] builds another layer on the network outputs if the previous layer does not separate the classes in the problem. Along a similar vein is the Extentron algorithm [Baffes & Zelle 1992] which forces the separation of examples by extending a standard perceptron. The Upstart algorithm [Frean 1990] produces a binary tree of nodes which correct the values of the outputs for all training examples, the purpose being to correct any mistakes by adding extra positive and negative signals to the output node. This adds as many nodes as required to correct the error. The Upstart algorithm performs better than the Tiling method [Wynne-Jones 1991a], however both suffer from the limitation that only binary tasks are addressed.

Cascor is not limited to binary problems nor to a certain number of layers [Fahlman & Lebiere 1989]. It allows the addition of hidden nodes as required which have connections from all previous hidden nodes and the inputs, and are connected in turn to all outputs — hence giving the Inverted Pyramid Construction identified by Gallant. The network starts as a single output layer with full connections between the network inputs and outputs. Training occurs until there is no further improvement, as measured by patience parameters, much like the method employed by Ash [Ash 1989]. Cascor is then able to individually install hidden nodes into the network. The hidden node is selected from a pool of trained candidate nodes. The node with the highest correlation to the network error after the candidates have been trained is installed. The weights of this hidden node are then frozen and the output layer is retrained with the extra node connected to it. This process is cyclical and continues until either the training set is classified correctly or the maximum number of hidden nodes has been added. This produces all possible feed-forward connections, and the ability of hidden nodes to connect to other hidden nodes allows for the possible formation of advanced feature detectors. The Cascor algorithm has been extended a number of times [Littmann & Ritter 1992; Simon, Corporaal & Kerckhoffs 1992; Simon 1993] .

One limitation of Cascor is that it is not effective when examining regression style problems. The correlation machinery tends to over-compensate which means that the results, though effective for classification, tend to over-shoot on regression problems [Fahlman 1993; Hwang, You, Lay & Jou 1993; Freeman 1994; Adams & Waugh 1995].

Projection Pursuit Learning (PPL) [Hwang, et al. 1993; Hwang, Lay, Maechler, Martin & Schimert 1994] involves a single hidden layer of nodes with adaptable activation functions. Nodes are added to the single hidden layer when required, and the activation functions are altered to solve the required problem, rather than adding more nodes with fixed activation functions to a deepening network as with Cascor. Simulations indicate that PPL is extremely effective at solving regression problems, as the activation functions adapt to fit the shape of the problem structure.

2.1.3.2 *Pruning — removing hidden nodes*

Three main methods are employed for node pruning: heuristic solutions, saliency measures which are extensions of the methods used to prune connections, and node decay based on weight decay methods. Sietsma and Dow take the heuristic approach by comparing nodes based on the network outputs of all training patterns [Sietsma & Dow 1988]. The idea is to remove those nodes which have little effect — non contributing units; or whose effect is duplicated by other nodes — the unnecessary-information units. It also removes layers by determining if they are redundant. Shamir et al. consider the reduction of hidden nodes by merging neurons with similar functional behaviour, hence preserving functionality [Shamir, Saad & Marom 1993]. Statistical results are presented to support the algorithm. Chung and Lee also develop a node pruning algorithm which removes four styles of unnecessary nodes: non-contributing, duplicated, inversely-duplicated and inadequate nodes [Chung & Lee 1992].

As was mentioned previously, Skeletonization is one method of pruning nodes from a network [Mozer & Smolensky 1988; Mozer & Smolensky 1989]. Segee and Carter show that this method is quite effective, even though a relevant node may have weights which are irrelevant [Segee & Carter 1991]. Ramachandran and Pratt extend the Skeletonization idea by basing a node pruning method on an information measure from the inductive learning literature [Quinlan 1986b; Ramachandran & Pratt 1991]. Similar ideas are presented by Dunne et al. with regard to nodes which attempt to separate only one class from the rest of the classifications in the problem [Dunne, Campbell & Kiiveri 1992]. Adams and Jones also examine a node's relevance in relation to function interpolation with success in creating minimal single layered networks [Adams & Jones 1992].

Chauvin examines the removal of nodes by a weight decay method which is altered to encompass all weights connected to the one node, rather than operating on individual weights [Chauvin 1988]. Ji et al. also consider the removal of nodes using a penalty term, along with the reduction of weight magnitudes at the same time [Ji, Snapp & Psaltis 1990].

2.1.4 Combinations of different strategies

As the methods mentioned above may be applied in different phases of network training, combinations of different methods have occurred. Sietsma and Dow examine one method which combines the use of growing and pruning algorithms [Sietsma & Dow 1991]. One deficiency with their method of heuristic pruning [Sietsma & Dow 1988] is that although it may produce a minimal number of nodes in a layer, the output of the layer may not be linearly separable. This may require the introduction, and hence growth, of extra hidden layers to overcome the newly created problems with the hidden layer or layers. The result is a transformation of a wide and shallow network into a thin, deep network which may generalise better than the originally trained network. The problem is that this method is used over and above the training of the initial network, thus increasing the time required for training.

Wynne-Jones favours the combination of constructive algorithms and pruning methods to overcome the problems of obtaining the minimal network, by allowing the training process to increase the size of the network, and then reduce it when the task has been learnt [Wynne-Jones 1991a]. No specific system is presented in the paper. There have been algorithms developed which both add and delete hidden nodes from a single layer (for example, [Murase, Matsunaga & Nakade 1991; Wang & Hsu 1991]).

There are a number of additions to the Cascor algorithm to include further topology changing methods. Klagges and Soegtrop which examines the use of limited and randomly connected hidden nodes in the Cascor style of network [Klagges & Soegtrop 1992]. The idea of growing a single hidden layer using Cascor has also been considered [Sjogaard 1991; Yeung 1991].

2.1.5 Further comments

In the case of using pruning algorithms, where initially excess connections or nodes are in the network, often the extra free parameters aid in the learning process as well as the speed of learning [Mozer & Smolensky 1988; Izui & Pentland 1990; Thodberg 1991; Wynne-Jones 1991a].

Not all people agree with this approach [Ash 1989]:

> There are some shortcomings to the pruning approach. Since the majority of the training time is spent with a network which is larger than necessary, this method is computationally wasteful.

This training speed problem does not occur with methods such as Cascor [Fahlman 1993], as it seems to be related to the 'herding' problems that have been identified within standard back-propagation style networks [Fahlman & Lebiere 1989]. Since in back-propagation all hidden nodes are active at any point in time during the training of the hidden layer, all the nodes are trying to correct the same error. To ensure that a solution is reached, a greater than optimal number of hidden nodes is required for training to ensure that the nodes are well spaced by the initial random allocation of weights. Cascor, for example, does not suffer from this problem as only one hidden node is trained at a time so that the maximum error is reduced by one node and then its weights are frozen. Nevertheless, it is not guaranteed that this 'greedy algorithm' will produce a minimal network [Fahlman 1990], as it is trying to remove as much of the error as possible using a single hidden node.

A further criticism of pruning is that the reduction in the number of connections may lead to a corresponding reduction in the fault tolerance of networks, inherited from the way in which knowledge within the network is distributed. Work done on the effect of pruning on the fault tolerance of networks indicates that after pruning a network's ability to cope with being damaged is not decreased [Segee & Carter 1991]. This should be taken in the context that generally, back-propagation gradient-descent trained networks do not have significant fault tolerance capability, as usually individual weights may have a great bearing on the end result [Bolt 1992].

Further work is also being performed on network architectures which differ from the standard feed-forward network design (for example, [Fiesler 1994; Deffuant 1995]). However, the above review does show definite trends which form the basis for the next section outlining an abstraction of methods for changing network topology. The majority of connection alteration methods involve the pruning of unnecessary connections or weights. The majority of node alterations involve the addition of new nodes to account for further data features. Most algorithms also require the retraining of the network after pruning.

2.2 Abstraction of topology changing methods

This section outlines an abstraction of possible ways of changing network topologies developed from the literature presented above. Feed-forward networks can be considered to be directed acyclic graphs. There are two features of a general acyclic graph: the set of vertices, and the set of directed edges between those vertices. Artificial neural networks can be mapped to acyclic graphs such that nodes are regarded as the vertices, and connections being the directed edges. The weight of a connection can be considered to be a strength of the directed edge. From this it can be seen that there are two general topological features of artificial neural networks, namely the nodes and the connections with their associated weights. The distinction between connections and weights may be considered to be arbitrary, but it reflects a difference in methods presented in the literature.

Disregarding whether nodes, connections, or weights are being examined, there are three common ways in which topologies are altered: by the use of constructive algorithms to add features to the network, using destructive algorithms which remove or prune features from the network, and a combination of constructive and destructive algorithms. Any algorithm which considers the alteration of a network topology will then be adding or removing nodes, connections, or weights. These will be considered in turn below. Note that if a node is added or removed, so are all the connections to that node, so changes which include whole nodes amount to the addition or removal of blocks of network connections.

2.2.1 Changing connections and weights

Firstly, adding or removing connections or weights is examined. Consider that there are n potential inputs to a particular node, then there are 2^n possible connection strategies to that node (see figure 2.1). In a construction algorithm, a base case of minimal connections to a node is needed to which new connections can be added. As the number of inputs to a node increases, the number of possible starting connection strategies to that node increases dramatically. The large number of possibilities would mean that the best initial connection strategy would often be missed if only a few were selected randomly. Though there are advantages to having nodes with limited fan-in, the large number of combinations make their use prohibitive.

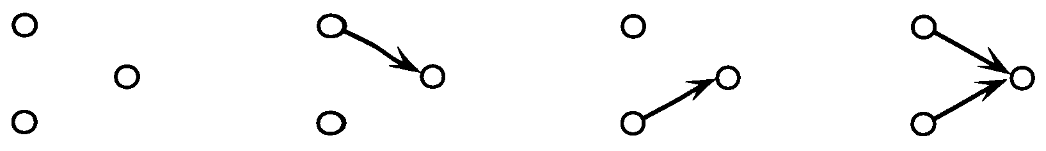
Figure 2.1 — Example of the possible connection strategies with two inputs (possibly one being a bias connection) to the one node

For example, consider the case where each node has a maximum of two connections. If there are *n* network inputs there will be:

$$\frac{n!}{2 \cdot (n \pm 2)!} \tag{2.3}$$

possible node connection strategies to the input layer, without even considering the need to train multiple nodes with different random weights to avoid local minima. Therefore for 10 inputs, there would be 45 different connection strategies for each node to be used.

It may also be possible to employ some form of weight enhancement to add in connections when they are required from zeroed weights. The problem of what is a sensible initial connection strategy comes up again, as most weight training methods require the setting of random initial weights. If a network required the majority of weights — or all weights as is the obvious choice — to be set to zero, allowing weight enhancement to be employed at some later stage, there will be little variation to avoid local minima. Further, it is difficult then to start the training process and decide which weights should be allowed to vary.

A pruning algorithm for connections would start at the more obvious position of all possible connections to a node being present and could then decrease the number of connections according to some pruning strategy. Although there are the same number of possible connection strategies, a unique base case exists to start training on, and it is easier to remove connections which have no effect than add connections which may have an effect. Likewise, weight decay methods seem more reasonable than weight enhancement as they can gradually reduce weights that are already in place. One concern with choosing a pruning approach is that initially training the extra connections would require more computational time for little gain than a more limited connection strategy, especially when a large number of connections are redundant.

2.2.2 Changing the application of hidden nodes

Finally, additions and deletions to the number of hidden nodes are considered. Whereas the number of connections has an upper limit set by the maximum number of inputs to the particular node, the number of hidden nodes has no upper limit. The base case is naturally no hidden nodes at all — perhaps forming just a perceptron-style output layer. Thus a node construction algorithm seems quite attractive as long as the initial conditions are sensibly set. This will require at least one node in each hidden layer or some strategy for forming new layers.

To use a destructive scheme for removing nodes more nodes are required than are necessary in the final network. The problem with such a destructive algorithm is that at some point the initial topology needs to be decided: what is the maximum number of hidden nodes required, how many layers are needed and how are the nodes to be connected to proceeding and succeeding layers. A sensible possibility would be to start with a fully connected layered network and remove unnecessary nodes. This relies on the weight training, possibly over a large number of layers, to find a reasonable solution in the first place and also to be reasonably quick; and puts the onus on the user to allocate the initial topology to be just larger than the necessary final network, to reduce the overall training time.

Thus the most promising approaches seem to be to add hidden nodes and remove connections. The ideal of using the smallest number of layers may also be incorporated into this scheme if this is judged as being important for the network application. This abstraction is mirrored by the methods presented in the literature.

2.3 Standard Cascade-Correlation

Cascor is examined as one of the most promising node construction algorithms, as it is able to develop networks with multiple layers creating advanced feature detectors, and it is able to examine problems with real-valued inputs. Its real strengths lie in the area of classification where the outputs are binary and it is this excellent performance which warrants further consideration. The algorithm's performance with real valued outputs is less than optimal [Adams & Waugh 1995]. Furthermore, it is a prime candidate for the use of methods to reduce the number of connections, as the algorithm adds hidden nodes with all possible feed-forward connections, many of which may be redundant. This leads to a natural combination of growing and pruning methods, in line with the trends evident in the literature.

2.3.1 Overview of Cascade-Correlation

The Cascor algorithm cycles between two phases to train a network: the first phase involves training and further retraining of the weights to the output nodes; and the second involves the gradual addition of hidden nodes to the network (see figure 2.2). The second phase is the more complicated whereby candidate nodes are trained to maximise their correlation with the network error, and the best of these nodes is installed into the network.

Figure 2.2 outlines the ordering of these processes, and the following subsections describe them in more detail. The details described in this section are taken from Fahlman's paper, the released Cascor software and the author's TasCas simulator (see [Fahlman & Lebiere 1989; Crowder & Fahlman 1991; Waugh 1995c] and §E).

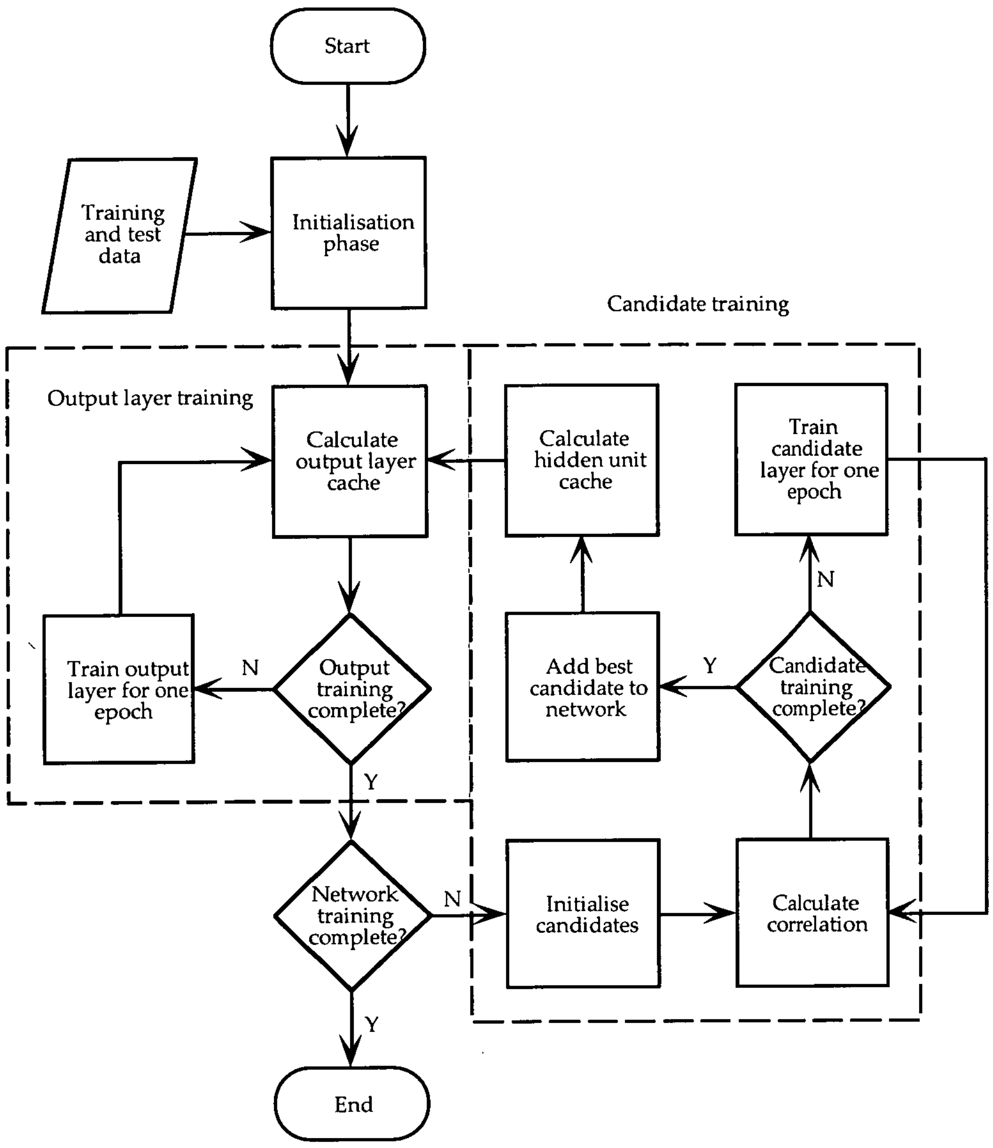
Start
Training and test data
Initialisation phase
Candidate training
Output layer training
Calculate output layer cache
Calculate hidden unit cache
Train candidate layer for one epoch
N
Train output layer for one epoch
N
Output training complete?
Add best candidate to network
Y
Candidate training complete?
Y
Network training complete?
N
Initialise candidates
Calculate correlation
Y
End

Figure 2.2 — Flow-chart outlining the Cascor algorithm

The initialisation phase of the algorithm is simple: the network consists of the required number of inputs and outputs as determined by the problem. For classification problems it is usual to have one output node per class and train the network to fire one output node at a time. For unseen data the output node with the largest response is taken as the example classification. The network is created by the allocation of the required memory for the connections between the input and output nodes, the initial weights of the network are randomly set, and the data required for training and testing is read in to appropriate data structures for use throughout the training process. Note that the test set data is not used in any way to select features of the network. It is there purely to monitor the generalisation of the network.

2.3.2 Output layer training

Cascor starts developing a network by initially training a layer of weights between the input nodes and the output nodes. This single layer is fully-connected and prescribed by the problem and data representation chosen. The output layer configuration, with random weights set during the initialisation process, is trained to produce a minimal error (see §2.3.5).

For efficiency and speed considerations the training process initially involves caching the required values from the evaluation of the network: namely the network outputs for each example, the error values for each output and example and the overall network error. The error for each output and example value is as follows:

$$e_{kp} = y_{kp} \pm t_{kp} \qquad (2.4)$$

where e_{kp} is the error over all outputs k and all training patterns p, y_{kp} is the output node value and t_{kp} is the expected output value.

The output and error values are cached for later use in the training process, especially during the training of the candidate nodes where these values are not altered over several iterations. The caching is not necessary, but it greatly speeds up training if the machine memory is available. Otherwise the values and errors of the output nodes must be recalculated for each example when required.

Once the values have been cached, it is possible to say whether the goals of training have been met, as the most recent network has been evaluated. At this point it is decided whether training the output layer should continue. This process is described more fully below (see §2.3.4). If these goals have been met — which is unlikely if no changes have been made to the weights — then the output layer training phase is complete. If this does not occur then the output layer weights are trained using Quickprop (see §2.3.5) and the algorithm cycles to evaluating and caching the output layer information. This process continues until training of the output layer is considered to be complete.

The error to be back-propagated (δ) is calculated as follows:

$$\delta_{kp} = f'_{kp} \cdot e_{kp} \qquad (2.5)$$

where f'_p is the derivative of the activation function f for pattern p, in this case for the output unit k, with respect to the sum of its inputs. The released Cascor code [Crowder & Fahlman 1991] uses the above slope value for some error calculations instead of (2.4). This usually includes an activation function offset (see §2.3.5). The error function in (2.4) is used for all error calculations within the experiments presented within this thesis. The learning error rate used is:

$$\frac{\eta}{t} \tag{2.6}$$

where η is the nominal learning rate and t is the total number of training examples. This scaling is for the benefit of Quickprop to keep the updates within a sensible range.

Once the output layer training is complete as mentioned previously, the algorithm checks to see if the conditions of training the entire network have been met (see §2.3.4). If they have, the algorithm stops, otherwise candidate units are trained and one of those units is installed in the network. The output layer is then retrained with connections to this new hidden unit, and the process cycles. The process of training the candidates is described next.

2.3.3 Candidate training

The training and installing of a hidden node is performed in a similar manner to the output layer training: a number of candidate nodes are given initial random weights, and they are then trained independently to maximise their correlation to the network error; the total number of candidates is specified by the user. The candidates are connected to all the input nodes and all the previously installed hidden nodes.

The training cycle for candidate nodes begins by calculating the correlation[2] of each candidate node with the residual error at the output nodes. The original Cascor paper [Fahlman & Lebiere 1989] gives the following formula for the correlation calculation (S) for which is to be maximised each candidate:

$$S = \sum_{k=1}^{m} \left| \sum_{p=1}^{t} \left(v_p \pm \bar{v}\right)\left(e_{kp} \pm \bar{e_k}\right) \right| \tag{2.7}$$

where v_p is the value of the candidate for example p, and $\bar{v}$ and $\bar{e_k}$ are the averages of the candidate values and the errors for each output respectively. This results in the following derivative with respect to the candidate's input weights which are being trained:

$$\frac{\partial S}{\partial w_i} = \sum_{k=1}^{m} \sigma_k \cdot \sum_{p=1}^{t} \left(e_{kp} \pm \bar{e_k}\right) f'_{jp} \cdot x_{ip} \tag{2.8}$$

---

[2]As noted in Fahlman's paper the correlation calculation is strictly a covariance, as no normalisation of the calculation is performed. Fahlman has indicated that the normalisation process does not improve training performance [Fahlman & Lebiere 1989].

where σ_k is the sign of the correlation between the candidate's value and output k, x_{ip} is the input the candidate receives from the unit i for the pattern p, j is the index for the candidate nodes, and w_i is the weight to the candidate from the input layer. The unit i may be a network input or a previously installed hidden node. In the actual implementation released by Fahlman, and the subsequent TasCas simulator [Crowder & Fahlman 1991; Waugh 1995c] error normalisation is implemented for correlation values. This amounts to having the following formulas instead of (2.7) and (2.8):

$$S = \frac{\sum_{k=1}^{m} \left| \sum_{p=1}^{t} v_p \cdot e_{kp} \pm \bar{v} \cdot \bar{e_k} \right|}{\sum_{k=1}^{m} \sum_{p=1}^{t} e_{kp}^2} \tag{2.9}$$

$$\frac{\partial S}{\partial w_i} = \frac{\sum_{k=1}^{m} \pm\sigma_k \cdot \sum_{p=1}^{t} \left(e_{kp} \pm \bar{e_k}\right) \cdot f'_{jp} \cdot x_{ip}}{\sum_{k=1}^{m} \sum_{p=1}^{t} e_{kp}^2} \tag{2.10}$$

These formulae are used for the calculation of the correlation and the subsequent modification of candidate weights within the candidate training process.

If, as calculated from the correlation calculations, training is not complete (see §2.3.4) then the candidate node weights are modified by gradient ascent to maximise their correlation with the output nodes. The learning rate is normalised in a similar manner to the output layer training rate to keep the Quickprop updates within a sensible range [Crowder & Fahlman 1991]:

$$\frac{\eta}{t \cdot (n + h + 1)} \tag{2.11}$$

where n is the total number of inputs, and h is the number of hidden nodes installed so far, and η again is an arbitrary constant representing the learning rate.

Once the candidates are trained the candidate with the largest correlation is installed in the network. Its input weights are added to the network and frozen so they will not be altered. The freezing is effected by only training the output layer weights during the output training phase, and not back-propagating the errors past the output layer. The output layer weights for the newly installed hidden nodes are set using minus the last correlation calculated as an initial guess ($-S$). Fahlman has found this to be more effective than just setting random weights [Fahlman 1993]. The hidden unit cache may then be updated with the values produced by the new hidden node. As the hidden node weights are frozen, these will not alter during the rest of training.

The output layer is then retrained with the newly installed hidden node as an extra input to all the output nodes. This process cycles, adding in further hidden nodes, until the training is completed. The extra connections to the previous hidden nodes generate a very deep network with one node per hidden layer and all possible shortcut connections installed. The maximum number of hidden nodes which can be installed is again specified by the user.

2.3.3.1 *Correlation derivation*

The following is the derivation of the correlation *S* with respect to the candidate's input weights [John 1995]:

$$\begin{aligned}
\frac{\partial S}{\partial w_i} &= \frac{\partial \sum_{k=1}^{m} \left| \sum_{p=1}^{t} \left(v_p \pm \bar{v}\right)\left(e_{kp} \pm \bar{e}_k\right) \right|}{\partial w_i} \\
&= \sum_{k=1}^{m} \frac{\partial \left| \sum_{p=1}^{t} \left(v_p \pm \bar{v}\right)\left(e_{kp} \pm \bar{e}_k\right) \right|}{\partial w_i} \\
&= \sum_{k=1}^{m} \sigma_k \frac{\partial \sum_{p=1}^{t} \left(v_p \pm \bar{v}\right)\left(e_{kp} \pm \bar{e}_k\right)}{\partial w_i} \quad \text{where } \sigma_k \text{ is the sign of the correlation} \\
&= \sum_{k=1}^{m} \sigma_k \sum_{p=1}^{t} \frac{\partial \left(v_p \pm \bar{v}\right)\left(e_{kp} \pm \bar{e}_k\right)}{\partial w_i} \qquad (2.12) \\
&= \sum_{k=1}^{m} \sigma_k \sum_{p=1}^{t} \frac{\partial \left(v_p \pm \bar{v}\right)}{\partial w_i}\left(e_{kp} \pm \bar{e}_k\right) \quad \text{as } e_{kp} \text{ does not depend on } w_i \\
&= \sum_{k=1}^{m} \sigma_k \sum_{p=1}^{t} \left(\frac{\partial v_p}{\partial w_i} \pm \overline{\frac{\partial v}{\partial w_i}}\right)\left(e_{kp} \pm \bar{e}_k\right)
\end{aligned}$$

The error is independent from the candidate weights as this is dependent on the output weights only. From the definition of the network:

$$v_p = f\left(\sum_{i=1}^{n+h} w_i \cdot x_{ip}\right) \qquad (2.13)$$

the following may be calculated:

$$\begin{aligned}
\frac{\partial v_p}{\partial w_i} &= \frac{\partial f\left(\sum_{i=1}^{n+h} w_i \cdot x_{ip}\right)}{\partial w_i} \\
&= f' \frac{\partial \left(\sum_{i=1}^{n+h} w_i \cdot x_{ip}\right)}{\partial w_i} \qquad (2.14) \\
&= f' \cdot x_{ip}
\end{aligned}$$

which may be used within the last equation of (2.12) to give:

$$\frac{\partial S}{\partial w_i} = \sum_{k=1}^{m} \sigma_k \sum_{p=1}^{t} \left(f' \cdot x_{ip} \pm \overline{f' \cdot x_i}\right)\left(e_{kp} \pm \bar{e}_k\right) \qquad (2.15)$$

This formula can be shown to be equivalent to the original calculation of the derivative (2.8) as the $\overline{f' \cdot x_i}$ term sums to zero. Removing this additional term may lead to some problems of precision, although empirical evidence does not indicate that this has had any major effect [John 1995].

2.3.4 Stopping Training

There are three points within the Cascor algorithm where decisions need to be made as to whether training should continue. These are at the end of each output layer training epoch, at the end of each candidate pool training epoch, and for the entire network at the end of each output layer training phase.

At these different points different methods are used to decide when training is complete:

- an arbitrary limit — which is used for output layer and candidate training setting a maximum number of epochs which training may take, or is used by the overall network training by setting the maximum number of hidden units which may be installed;
- a correctness limit — which is used for stopping output layer training and the entire network training; and
- a patience limit — which is used to halt output layer and candidate training when training is not resulting in an effective improvement in network performance.

If any one of these limits is met on a phase of training in which it is being employed, then training halts. Thus if, for example, the epoch limit is met on output layer training, then training halts regardless of whether the patience limit or correctness limit has been met. The arbitrary limit, irrespective of whether it measures the number of hidden nodes or the number of epochs, is a rough measure of training time; hence this may be regarded as a time limit.

The correctness limit is determined in two ways. The first is more appropriate for classification problems, where a minimum number of error bits is set for a goal of training and often this minimum is set to zero. An error bit is where a value for an output for a particular example is outside a specified range, and is hence considered to be in error. This is counted as one error bit. If the number of error bits is zero then 100 percent correctness is said to be achieved. The maximum number of error bits is therefore the number of training examples multiplied by the number of outputs. The allowable range from the expected value is specified by an error threshold. Thus, for example, if a symmetric sigmoid activation function is used with values between –0.5 and 0.5 and an error threshold of 0.4 set, then a correct maximum value will be between 0.1 and 0.5, and a correct minimum value will be between –0.5 and –0.1.

The second method for judging correctness, which is more appropriate for regression problems, is to simply set an arbitrary error value which the network error must fall below:

$$\mathrm{MSE} = \frac{\sum_{k=1}^{k} \sum_{p=1}^{t} \left(y_{kp} \pm t_{kp}\right)^2}{m \cdot t} \qquad (2.16)$$

Fahlman provides further normalisation of this value within the released simulator (see [Crowder & Fahlman 1991] and §E.7.3).

Patience is a measurement of minimal activity specified by the two patience parameters:

- patience error — the change in error required over a set period to continue training; and
- patience period — the period over which the change in error is measured.

Thus if there has not been a change in the network error greater than the patience percentage of the error given — or the maximum correlation for training the candidate pool — over the patience period, then the network runs out of patience and that phase of training is completed. The code for implementing the patience calculation is given in figure 2.3.

```
Initialise by:
quitEpoch = currentEpoch = 0;
stillPatient = true;

At the end of each training period:
/* note that currentEpoch++ has occurred and currentError is set */
if (quitEpoch == 0) {
  lastError = currentError;
  quitEpoch = patienceLength;
} else
if (fabs(currentError - lastError) > patiencePercentage * lastError) {
  lastError = currentError;
  quitEpoch = currentEpoch + patienceLength;
} else
if (currentEpoch >= quitEpoch)
  stillPatient = false;

At the completion of training:
totalEpochs += currentEpoch;
```

Figure 2.3 — C code for calculating the patience stopping criterion

2.3.5 The Quickprop algorithm

Quickprop is the name given by Fahlman to a quasi-Newton method of minimising a function using an heuristic estimate of the curvature of the error function to improve performance over gradient descent back-propagation [Fahlman 1988a].[3] Any function can be expanded about a known point in a Taylor series. For simplicity consider the one dimensional case:

$$f(x_0 + h) = f(x_0) + h.\left.\frac{\partial f}{\partial x}\right|_{x_0} + \frac{1}{2}h^2.\left.\frac{\partial^2 f}{\partial x^2}\right|_{x_0} + \dots \qquad (2.17)$$

The first term is proportional to the function evaluated at the known point, the second to the first derivative evaluated at that point and multiplied by the distance from it, the third to the second derivative evaluated at that point and multiplied by the square of the distance from it, and so on.

If the expansion is about a minimum, the curve in the vicinity of the minimum can be reasonably approximated by a constant and a term quadratic in the distance from the minimum, h. This is because the first term gives the value at the minimum, the second term is proportional to the slope of the curve which at the minimum is zero and the third term is proportional to the curvature at the minimum. At the minimum the curve is symmetric and hence all the terms proportional to the odd powers of h are zero. If h is small then even terms of order h^4 and higher will be small compared with the quadratic term.

In minimising the error function in back-propagation this analysis cannot be directly applied since the position of the minimum is what is required rather than what is known. However, if near a minimum it is reasonable to take the shape of the surface to be quadratic. Fahlman makes the assumption that the surface is quadratic but applies it at all times rather than only near a minimum [Fahlman 1988a]. Fahlman makes the further 'risky' assumption that the weights are independent, thus changes to a weight do not affect the other weights in the network. The use of the partial derivative is the implementation of this assumption.

The standard gradient descent weight change, including the momentum term which need not be used, is as follows:

$$\Delta w(t) = \pm\eta \cdot s(t) + \alpha \cdot \Delta w(t \pm 1) \qquad (2.18)$$

[3]The technical report referenced here is available by ftp. The published version of the paper [Fahlman 1988b] is also referenced but it is not as readily available.

where η is the learning rate, α the proportion of momentum used and $s(t)$ is the slope at time t. The Quickprop algorithm essentially changes this to:

$$\Delta w(t) = \frac{s(t)}{s(t \pm 1) \pm s(t)} \Delta w(t \pm 1) \tag{2.19}$$

leading to a crude approximation of the optimum value which gets increasingly better as the minimum is approached. The derivation of this is outlined below.

2.3.5.1 *Derivation of Quickprop update*

In back-propagation style networks the error function is a function of the weights and each weight is dealt with individually. At each step in the iterative process of minimising the error function the value of the error and the gradient of the error function at that point are known. Quickprop uses the current and previous gradients and the values of the weights to estimate the position of the minimum based on the quadratic approximation. Figure 2.4 illustrates the situation.

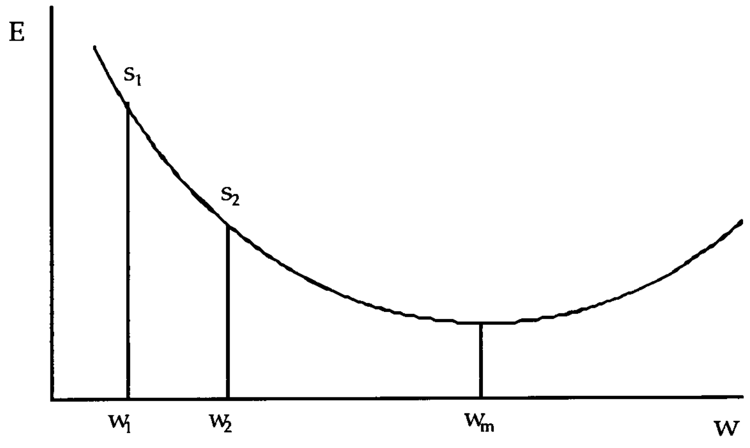

Figure 2.4 — The error function, E is a function of a weight, w. At w_1 and w_2 the gradients of the curve are s_1 and s_2 respectively. w_m is the position of the minimum

The error function is assumed to be a quadratic function of the weight, w, namely

$$E = a + bw + cw^2 \tag{2.20}$$

The slope of this curve is given by:

$$\frac{\partial E}{\partial w} = b + 2cw \tag{2.21}$$

Two points are known, namely

$$\frac{\partial E}{\partial w} = s_1 \text{ at } w = w_1 \text{ and } \frac{\partial E}{\partial w} = s_2 \text{ at } w = w_2 \tag{2.22}$$

At the minimum we have:

$$\frac{\partial E}{\partial w} = 0 \text{ at } w = w_m \tag{2.23}$$

Substituting (2.22) into (2.21) we have:

$$s_1 = b + 2cw_1 \text{ and } s_2 = b + 2cw_2 \tag{2.24}$$

from which we derive:

$$b = \frac{s_1 w_2 \pm s_2 w_1}{w_2 \pm w_1} \text{ and } c = \frac{1}{2}\frac{s_2 \pm s_1}{w_2 \pm w_1} \tag{2.25}$$

Substituting these values into (2.21) and (2.23) we have:

$$\begin{aligned} w_m &= \pm \frac{1}{2}\frac{b}{c} \\ &= \pm \frac{s_1 w_2 \pm s_2 w_1}{w_2 \pm w_1} \cdot \frac{1}{2} 2 \frac{w_2 \pm w_1}{s_2 \pm s_1} \\ &= \frac{s_2 w_1 \pm s_1 w_2}{s_2 \pm s_1} \end{aligned} \tag{2.26}$$

Finally, if we interpret the subscripts to mean that w_1 and s_1 are measured at time (t - 1) and w_2 and s_2 at time (t) and introduce two further parameters:

$$\Delta w(t) = w_m \pm w_2 \text{ and } \Delta w(t \pm 1) = w_2 \pm w_1 \tag{2.27}$$

we can rewrite (2.26) as:

$$\Delta w(t) = \frac{s(t)}{s(t \pm 1) \pm s(t)} \Delta w(t \pm 1) \tag{2.28}$$

which is the derived Quickprop formula [Fahlman 1988a]. Since it involves the (reciprocal of the) difference of two gradients divided by the distance between them, Fahlman is able to claim that this is an approximation to a second order algorithm.

2.3.5.2 *Practical considerations*

Since at a point a great distance from a minimum the quadratic assumption may be poor Fahlman has introduced a series of heuristic rules to deal with the cases where this assumption does not work well.

At the beginning of the training there is only one value of w and s known and hence simple gradient descent with learning rate η is employed as the full Quickprop update formula will be ineffective. Likewise when the previous weight change is zero, gradient descent is used to continue the training process if required.

Furthermore, the standard gradient descent term is added to the Quickprop quadratic update if the current slope will cause the weight to move down the error slope in the same direction as the previous change, helping to push the value toward the minimum. This additional term is not used if the slope changes: hence the weight is near the minimum where the quadratic weight update should be most effective.

If the current slope is close to or larger than the previous slope and moving in the same direction — unlike the parabola that Quickprop formula models — a jump to the minimum may result in an overly large step. To avoid this Fahlman introduces a term μ, the maximum growth factor, to limit the step size. In such a situation μ times the previous weight change is used instead of the Quickprop update formula, where Fahlman suggests a value of 1.75 for μ. This or the Quickprop formula are thus only used if the previous weight is non-zero.

A shrink factor is calculated from μ to test if the current slope is as large or larger than the previous slope. This is used to avoid taking steps which are too large. The shrink factor is defined as follows:

$$\text{shrink factor} = \frac{\mu}{1 + \mu} \tag{2.29}$$

Finally a weight decay term is added to the slope prior to these calculations to limit the growth of the weights, if required, giving the following cost function for the output layer as an example:

$$E = E_0 + \frac{1}{2}\gamma \sum_{k=1}^{m} \sum_{i=1}^{n+h} w_{ki}^2 \tag{2.30}$$

where γ represents the strength of the weight decay. A small decay value will ensure that the weights do not grow too large, for both the output layer and candidate weights.

All these modifications lead to the implementation shown in figure 2.5. For stability considerations this algorithm only works as a batch training method, which requires the presentation of a group of examples to update the weights. In this case each batch is considered to be a complete run through the training set, after which the errors are used to update the weights.

An offset to the activation function derivative to stop this getting close to zero, is also often employed in conjunction with the Quickprop algorithm [Fahlman 1988a]. Briefly the activation function offset adds 0.1 to the derivative of the activation function it is applied to. The purpose of this is to ensure that the derivative does not become close to zero for values at the extremes of the activation function. The weight update, by definition, is multiplied by the derivative, and thus there is no update if the derivative is zero or is near zero. This increases the effect of any error in the region of the activation function. In Cascor an activation function offset is used in training the output layer, but no activation function offset is used with the candidate nodes as this confuses the correlation machinery [Fahlman

& Lebiere 1989]. Adams and Lewis have shown that for function evaluation, which is related to finding the maximum of the correlation, the use of the offset is not useful.

```
nstep = 0.0;
s += decay * w;
if (pd < 0.0) {
  if (s > 0.0) nstep -= eta * s;
  if (s >= shrink * ps) nstep += mu * pd;
                   else nstep += pd * s / (ps - s);
} else if (pd > 0.0) {
  if (s < 0.0) nstep -= eta * s;
  if (s <= shrink * ps) nstep += mu * pd;
                   else nstep += pd * s / (ps - s);
} else
  nstep -= eta * s;

pd = nstep;
w += pd;
ps = s;
```

Figure 2.5 — the C code for a single weight Quickprop update: *s* is the current slope, *ps* the previous slope, *pd* the previous weight change, *w* is the actual weight, *eta* is the learning rate, *mu* the maximum growth factor, *decay* is the weight decay term, *shrink* is the shrinkage factor, and *nstep* is a variable for calculating the next step by the Quickprop algorithm

2.3.6 Diagrams

Since Cascor is able to install a large number of shortcut connections the usual layered network diagram becomes unmanageable and unable to convey the full network information. To overcome this problem Fahlman and Lebiere developed an alternative diagram for displaying cascaded neural networks [Fahlman & Lebiere 1989]. Examples of networks — both a standard layered network and a Cascor network — are given in figure 2.6. The shaded nodes are the non-processing inputs, and the nodes at the top of each diagram are the outputs. The boxes indicate frozen hidden node connections within the Cascor network and the crosses indicate trainable connections: for the Cascor network this involves only output layer connections. The vertical lines to a node indicates the nodes' inputs and horizontal lines from a node indicate outputs from that node. If no box or cross occurs on the intersection of a horizontal and a vertical line, then the relevant connection is not present. This method of displaying artificial neural networks is useful for displaying any variety of feed forward network.

2.3.7 Summary

To recap the entire Cascor training process: initially a layer of weights between the input and output nodes is trained to minimise the overall network error. This is performed using the Quickprop algorithm, which requires several parameter values over a phase of training:

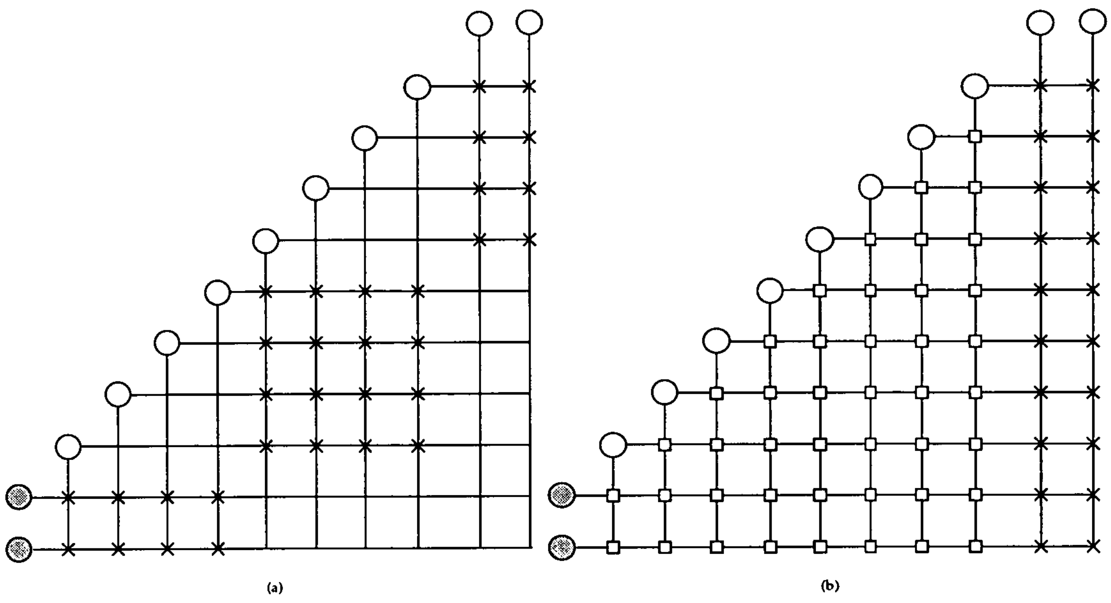

Figure 2.6 — Examples of (a) a standard two hidden layer network, and (b) a standard Cascor network

- η — the learning rate;
- μ — the maximum growth rate which limits the size of any change in weights performed after each presentation of the training set; and
- γ — a standard weight decay parameter which may be used to ensure that the weights do not grow excessively large.

Once progress is no longer made, the maximum number of epochs of training has been performed or the network achieves a correct result, output training is complete. If this results in a correct network, or the maximum number of hidden nodes has been installed, then the network training is complete. Otherwise a pool of candidate nodes is trained to maximise their correlation with the residual error of the network. When the maximum number of epochs has been reached or no progress is been made in the training of the candidate nodes, the best candidate node is installed into the network and the output layer is retrained. The algorithm thus cycles through installing hidden nodes and retraining the output layer, and these phases themselves cycle through training regimes. The user of Cascor has to decide what parameters to use for the specific application of the algorithm.

2.4 Experimental design

This section provides more specific details regarding the design of experiments throughout this thesis. The assumptions and general algorithm parameters used for experiments are given, followed by how network and general classifier performance is measured. The data sets employed to test the ideas presented in the first part of the thesis are then outlined, and finally details of the results of basic simulations using these data sets and parameters with standard Cascor are presented. Modifications to the Cascor algorithm are presented in the remaining chapters in the first part of the thesis.

2.4.1 Standard Cascade-Correlation option settings

To test the performance of the modifications presented in this thesis, a standard experimental construction is used. Each result is the median of 100 randomly seeded trials performed on the selected data sets (see §2.4.3), using the parameters outlined in tables 2.1 and 2.2, unless otherwise specified. The median is reported to avoid problems of skewed results, since there is no guarantee that the results produced will follow a normal distribution. A maximum of 25 hidden nodes may be added to each network. Symmetric sigmoid activation functions, with values within the range of –0.5 and 0.5, are used for all processing nodes, unless otherwise specified. The initial weights are set evenly over the interval –1 and 1. Each class is represented by a single output node.

Table 2.1 — Default values for candidate and output layer training parameters

Parameter	Candidate value	Output value
η	1.0	0.35
μ	1.75	1.75
γ	0.0	0.0
Patience percentage	3%	1%
Patience period (epochs)	50	50
Epoch limit	500	500
Activation function offset	0.0	0.1

Table 2.2 — Default values for network training parameters

Parameter	Value	Parameter	Value
Number of candidates	10	Network trials	100
Candidate node limit	25	Percentage allowable error bits	0.0
Default connection strategy	Full	Error threshold	0.4
Default candidate node activation function	Symmetric sigmoid	Expected value buffer	0.0

Thus the only difference between tests is the actual initial weights for the individual trials and the parameters under investigation. All the experiments are performed using the author's Cascor simulator (see Appendix E).

2.4.2 Measures of performance

There are six often quoted measures of performance of a classifier [Bratko 1990; Weiss & Kulikowski 1991; Zheng 1993]:

- the performance or prediction accuracy of the classifier;
- the speed of classification;
- the speed of learning the classification;
- the complexity or size of the final theory;

- the explanation ability of the final theory; and
- the ability of the theory to stand up to partial corruption.

These measures do not include other important factors, such as the cost of measuring the attributes, which is problem dependent. The speed of the final classification is often negligible — certainly in comparison to learning times — and will not be considered here. The ability of the network to stand up to partial corruption and the explanation ability of the final classifier will also not be considered, although they are areas warranting further detailed investigation.

The most important measure is the prediction accuracy or the generalisation ability of the final network, as this is the goal of the training. This is the ability to learn the underlying function of a population from the examples that have been presented to the system. Weiss and Kulikowski give an excellent introduction to measuring generalisation [Weiss & Kulikowski 1991]. Within this thesis — unless otherwise indicated — generalisation will be measured by a separate unseen test set. Though this may give a biased — usually pessimistically biased — measure of performance, this measure was chosen as it is computationally cheaper to employ than full cross-validation.

The complexity or size of the final theory also needs to be considered. It is preferable to obtain a system which solves the problem correctly, but which is also the smallest possible, giving the best chance for sensible generalisation. The size of the resulting theory depends on the method used to solve to the problem. For example, the number of hidden nodes, or the total number of connections are most often used to specify the complexity of a network.

Thus the results presented in this thesis take three forms:

- correctness or performance of the final theory — measured by the percentage correct on an unseen test set;
- complexity or size of the resulting theory — measured by the number of hidden nodes, or the total number of connections in the network; and
- training time — for artificial neural network methods, measured by the number of connection crossings or epochs required to train the network.[4]

---

[4]An epoch is considered here to be the presentation of the same number of examples as there are in the training set. Each connection crossing is the multiplication of a network weight by an input. The number of connection crossings is a more accurate measure than the number of training epochs in an architecture changing environment, as the extra work required with the introduction of more hidden nodes which have more inputs is taken into account as the network grows [Fahlman & Lebiere 1989].

2.4.3 Benchmark data sets

The following classification tasks are chosen to test the performance of Cascor on the alterations presented in the first part of this thesis:

- the Monks problems [Thrun, et al. 1991] — an artificial benchmark of three problems used to compare various methods of machine learning: these problems are based on a simple set up of enumerated attributes leading in each case to a binary classification problem;
- Two Spirals problem (TS) [Fahlman & Lebiere 1989] — a well known problem for showing the learning ability of Cascor — involves two interlocking spirals of different classes, with 192 training and test examples;
- Double Helix problem [Waugh & Adams 1994] — an extension of the Two Spirals concept: the Double Helix data set is generated using two full spirals of radius one with each spiral being one unit in length, one hundred samples were taken at evenly spaced intervals along each spiral, and the test set includes the points shifted by 0.1;
- LED — recognising LED displays from examples with ten percent noise added [Breiman, Friedman, Olshen & Stone 1984] using 2000 training examples and 500 test examples; and
- problems from the Proben1 benchmark [Prechelt 1994a] — three examples of real-world problems without substantial missing values are selected from this benchmark: Cancer1 from the University of Wisconsin Hospitals, Diabetes1, and Glass1.

Copies of all but the Double Helix data set are available from the UCI machine learning database repository [Murphy & Aha 1994]. Thus, nine problems are used to benchmark the methods outlined in the first part of this thesis. Six of them are artificial, and three are taken from real-world environments, all of which have prescribed test sets for generalisation. These tasks are selected as they are commonly known problems available to all researchers (Monks, LED and Proben1 data sets), or they are difficult tasks for standard artificial neural networks with sigmoid-like activation functions (Two Spirals and Double Helix). For example the Two Spirals problem is very difficult to solve using standard non-constructive methods. Fahlman and Lebiere note that at least two hidden layers are required to solve the problem, often with specialised architectural features, such as short-cut connections between hidden layers [Fahlman & Lebiere 1989]. The training also requires a large number of training epochs: of the order of 60000 to 200000.

It also accounts for the extra work involved when the number of candidates in the hidden node pool for Cascor is altered.

The three Proben1 problem are chosen as they are classification tasks from the benchmark which do not contain major arbitrary encodings of missing data. In a related work Zheng details a number of benchmarks taken from the UCI repository for general machine learning tasks [Zheng 1993]. In total he identifies 13 data sets, including Monks2, Cancer1, Diabetes1 and LED. Although a complete coverage of all the data sets mentioned in these benchmarks is not made, a reasonable number of each is examined while still examining two data sets requiring the addition of hidden nodes. The issue of benchmarking is further addressed in Part II of this thesis.

2.4.4 Performance of standard Cascade-Correlation

To give a baseline measurement, table 2.3 outlines the performance of Standard Cascor on the benchmark data sets outlined above, given the parameters outlined in tables 2.1 and 2.2. It is evident that a large proportion of these data sets have training sets which are completely solved by the addition of hidden nodes, although this may be the result of over-training. Two of the problems also reach the maximum number of hidden nodes, indicating that the problems are not completely solvable, and thus may contain classifications which cannot be explained given the information available. However, the results achieved using Cascor may be improved with modifications.

Table 2.3 — Results from application of standard Cascor to the benchmarking problems: shown are the name of the data set, the training and test set performance, the number of hidden nodes and connections required, and the number of connection crossings the training took (measured in millions)

Data set	Train %	Test %	Hidden nodes	Connections	CCs (M)
Monks 1	100	97.69	1	50	4.5
Monks 2	100	99.7	1	50	5.8
Monks 3	100	88.89	2	69	16.1
Two Spirals	100	95.83	12	132	123.3
Double Helix	100	100	6	59	63.5
LED	76	71.8	25	830	4770.7
Cancer1	100	95.98	5	90	178.7
Diabetes1	98.48	68.49	25	593	1962.9
Glass1	100	66.04	17	468	407.7

The data sets used for the experiments should present an interesting array of results. For example, the Two Spirals problem requires the installation of hidden nodes to be solved, and the LED problem is linearly separable but not completely solvable in that 100 percent cannot be achieved on the training set.

For comparison previous performances on these data sets are summarised in table 2.4. These results are taken from the previously referenced papers, apart from the LED problem [Quinlan 1987]. The results on the Proben1 data sets presented here are the best results of several trials. The performances given in table 2.3 may be comparatively worse as these involve different techniques or more fine-grained optimisation of the algorithm than has been attempted here. Optimisations of the parameter values are not examined as the experiments are only comparing modifications to standard Cascor and not a detailed comparison to other techniques. Nevertheless, the performance levels achieved are near the maximums obtained on these data sets.

Table 2.4 — Results from recorded literature on the benchmarking problems (except Double Helix): shown are the name of the data set, the training and test set performance, the name of the learning method used to achieve the performance reported, and any noteworthy differences in the exact data used; with 'N/A' indicating unavailable results

Data set	Train %	Test %	Method	Comments
Monks 1	100	100	Cascor	1 Gaussian hidden node
Monks 2	100	100	Cascor	1 Gaussian hidden node
Monks 3	100	95.4	Cascor	3 Gaussian hidden nodes
Two Spirals	100	N/A	Cascor	Median of 15 sigmoid hidden units
LED	N/A	72.6	C4	Optimal rate 74%
Cancer1	N/A	98.62	RProp	Two hidden layers
Diabetes1	N/A	75.9	RProp	Two hidden layers
Glass1	N/A	67.3	RProp	One hidden layer

3 Extensions to Cascade-Correlation training

This chapter presents modifications to standard Cascor which are shown to improve the training mechanism. Some of the results in this chapter have been reported previously [Waugh 1995a]. The first section examines the application of patience parameters to the addition of hidden nodes with the aim of halting network training. The second section examines methods for altering where patience is applied to the candidate pool in standard candidate training: training candidates in subgroups of the same node style and training candidates individually, instead of training the whole candidate pool; and by changing the maximum criteria of candidate selection.

3.1 Stopping the addition of hidden nodes

One problem with artificial neural networks is deciding when to stop training. Three commonly used methods are:

- correctness — checking the classification accuracy: training is halted when a certain number of the training examples, often 100 percent, are classified correctly by the network;
- time — checking an epoch or connection crossing limit: training stops when an arbitrary amount of training has been completed[1]; and
- validation set (separate selection test set) — a separate pool of examples is used for checking when overtraining is occurring: this set is independent of the training set used to set the network weights and test set used to evaluate the final network's generalisation ability [Prechelt 1994a].

The most effective of these methods is the validation set. However, there are often not enough examples available for training, let alone for creating two test sets. A preferable stopping criterion is one which does not require a validation set, and which will work when it is not obvious what is the optimum training time or what is the highest possible correctness.

Within Cascor, as mentioned previously, Fahlman and Lebiere rely on another method for halting output layer or candidate pool training,-which is a hybrid of the time and correctness stopping criteria [Fahlman & Lebiere 1989]:

[1]Methods for checking the optimal time limits as opposed to arbitrary time limits [Hamey 1991] will not be considered here.

- patience — if training continues over a period of time with little change in the network performance, the network 'runs out of patience' and training is halted.

Patience is set by two parameters: the percentage change in network error required to continue training, and the length of patience time, which is usually measured in epochs.

3.1.1 Description of node patience

This section introduces a new use of patience applied to the installation of hidden nodes to halt the entire Cascor training process — *node patience.* The idea has been identified independently of previous work [Squires & Shavlik 1991]. Once there has been no improvement in accuracy with the installation of the most recent *n* hidden nodes, training is halted. Node patience will only have an effect when the introduction of new hidden nodes does not assist the network. The node patience is set by the following parameters:

- node percentage change — of the network error, as in standard patience calculations within Cascor; and
- node patience period — over the number of hidden nodes added.

An epoch limit for the node patience period is not appropriate as a varying number of epochs of training may occur in the addition of different hidden nodes, resulting in the use of the hidden nodes installed as the time period. Thus there are two distinct uses of patience within Cascor which are applied in three places within the Cascor algorithm: standard patience which is used to halt the training of the output layer or candidate pool, and node patience which halts the addition of nodes to the network.

The implementation of node patience is simple. The code from figure 2.3 is applied at the end of each output layer training phase, and node patience may then be used with the other network stopping criteria. The network error is used to calculate whether the node percentage change has occurred, and the number of hidden nodes installed is used for the period. The additional code required is thus limited to the variables and code for the patience call, and the additional check to see whether network training is complete.

Node patience may be extended by later removing the nodes which added little to the performance of the network, a procedure termed *rollback.* If training is halted by node patience, the last *n* hidden nodes added are removed, where *n* is the node patience period. This is done in two ways: by either storing the output layer weights to be re-used if required, or by simply retraining the output layer again for one training phase with the unnecessary hidden nodes removed. If the output layer weights are to be stored, the final weights of the last *n* phases of output layer training need to be stored, where *n* is the node patience period. In such cases retraining is unnecessary, resulting in the same training times for the networks with and without rollback. Retraining the output layer is a simpler method to implement, but forces extra unnecessary training. Here rollback is implemented by the second method: once the entire network training is complete, if node patience is used and rollback is required, the relevant candidates and their output layer weights are removed and the output layer is retrained for another phase of training. This completes the network training, and further check of the entire network training conditions is made.

The application of node patience should aid in stopping network training. It involves the addition of a new stopping criterion which will not interfere with the previous criteria. It will only decrease the amount of training performed, but it is an open question of whether node patience will hinder network training. Node patience should reduce the overall execution time by avoiding unnecessary training of candidate nodes, and should also reduce the final network size without jeopardising the network integrity. In fact it may even result in better generalisation as the size of the network will be reduced.

3.1.2 Results and discussion

For assessing the performance of node patience the standard experimental set up is used with the values of the required node patience percentage change (1 percent to 5 percent or 10 percent for the Two Spirals problem), and the node patience period (1 to 5 hidden nodes added) being varied. Full results — showing the median percentage correct on the unseen test set, number of hidden nodes, and number of connection crossings required for training over 100 trials — are presented in Appendix A.

The results of the use of node patience vary markedly depending on the problem addressed. Three different effects of node patience are seen on the benchmark problems: no effect, a hindering effect, or a beneficial effect. The use of node patience on Cancer1, Double Helix and the Monks data sets has no effect (see tables A.1.1 to A.3.3, A.5.1 to A.5.3, and A.7.1 to A.7.3). Cascor simply solves each problem — to an accuracy of 100 percent on the training set — by adding in the required hidden nodes. The percentage change made by the addition of each hidden node is sufficient to ensure that training continues.

Some problems are hindered by the use of node patience. For example, the Two Spirals problem is not effectively solved when a high patience percentage (greater than 2 percent) and a low patience period (less than 3 hidden nodes) are used in some combinations (see figure 3.1). This corresponds to a dramatic reduction in the nodes installed, but the extra nodes are necessary to solve the problem. In this case node patience may be over-used to the point where it halts useful training. Node patience is not necessary, but it does not hinder training when used sensibly, as in the cases where it has no effect on the majority of Two Spirals trials.

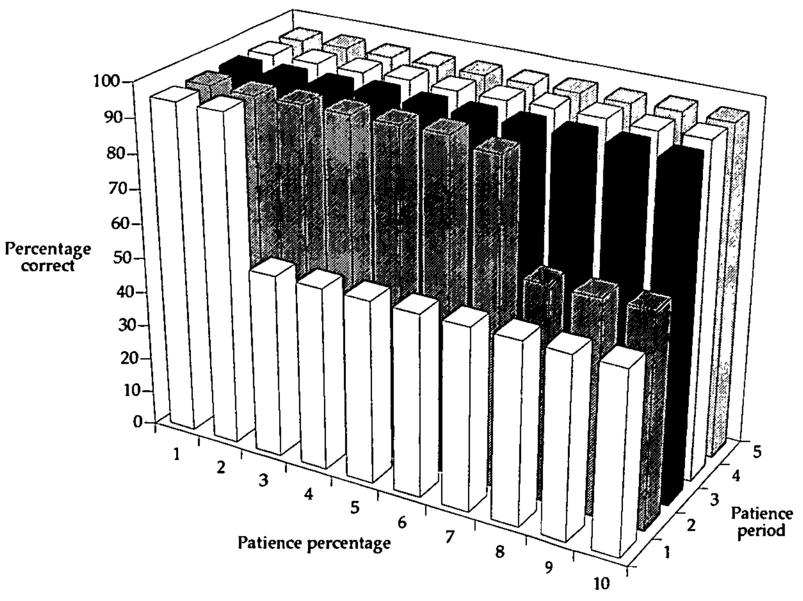

Figure 3.1 — Percentage correct on the test set for Two Spirals problem

On other problems, such as the LED and Diabetes1 data sets (see tables A.6.1 to A.6.3 and A.8.1 to A.8.3), the use of node patience is extremely beneficial. For the LED problem the test set classification performance is not affected by the application of node patience (staying around 72 percent), however the training required drops dramatically (see figure 3.2) as less hidden nodes are installed. The maximum number of allowed nodes is never installed on the LED problem while node patience is used under the given experimental conditions. Applying standard Cascor to the LED problem will always install the maximum number of hidden nodes, thus taking longer to train for the same result. A similar trend is also evident in solving the Glass1 data set (see tables A.9.1 to A.9.3), although this is not as significant as the problem is also solved to 100 percent accuracy on the training set.

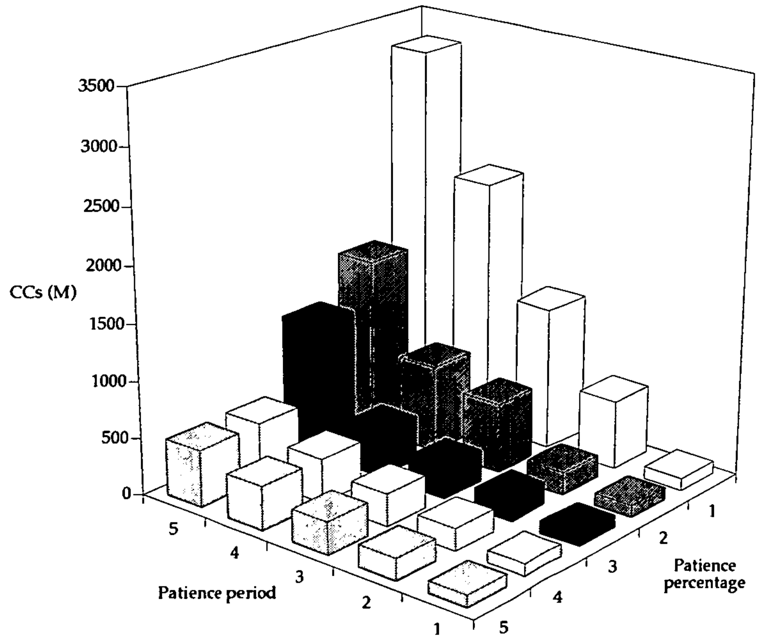

Figure 3.2 — Connection crossings (millions) for LED problem (note the difference in the labelling of the axes from figure 3.1 and 3.3)

Likewise for the Diabetes1 problem a similar effect is seen, except that where less hidden nodes are added the test set performance actually increases, avoiding over-training (see figure 3.3).

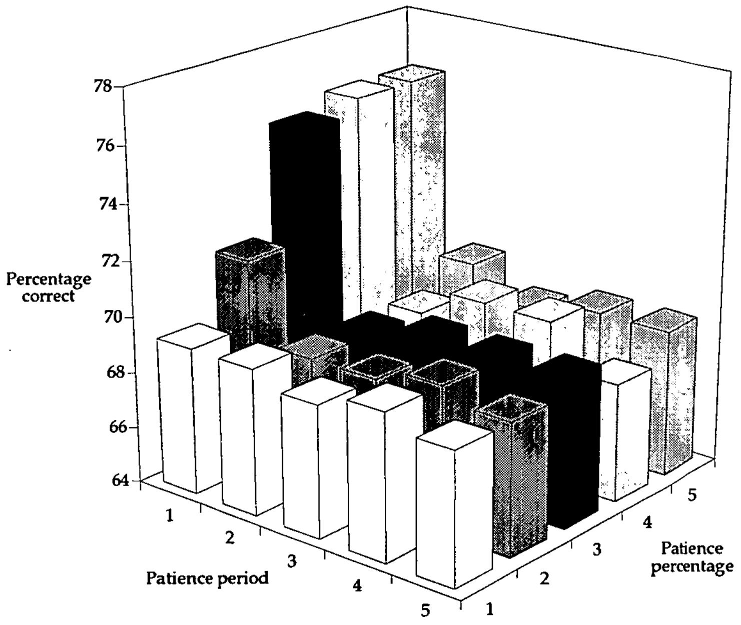

Figure 3.3 — Percentage correct on the unseen test set for the Diabetes1 problem

Predictable results occurred when rollback is added to the best trials on two of the data sets where node patience was effectively employed (see table 3.1). In both cases the number of hidden nodes is reduced, and a better classification performance is obtained on the Diabetes1 data set. Note that the node patience parameters used with the Diabetes1 data set were determined after the experiments presented above — simply another trial was performed using the parameters given which resulted in one less hidden node being added to the final network.

Table 3.1 — Results of rollback experiments on the LED (node patience 1% and 1 node) and Diabetes1 (node patience 6% and 1 node) data sets

	LED		Diabetes1	
	Node patience	+ Rollback	Node patience	+ Rollback
Test set performance	72%	72%	76.04%	77.6%
Number of hidden nodes	1	0	1	0
Connection crossings (M)	108.8	127	26.7	28.0

3.1.3 Need for hidden nodes

Following on from the node patience experiments performed above, it is worth asking whether hidden nodes are actually required at all with the data sets selected for benchmarking. Two problems definitely do not require hidden nodes, as shown by the rollback trials. To answer this question, trials are made of each data set with the restriction that no hidden nodes may be added to the network (see table 3.2). Thus only the single output layer is used for the classification tasks, and thus no node patience is required.

Table 3.2 — Results with no hidden nodes: shown are the problem name, the training and test set performances, the number of connections, and the number of connection crossings required for training (measured in millions)

Data set	Train %	Test %	Conn's	CCs (M)
Monks 1	84.68	75.23	32	0.64
Monks 2	63.31	62.27	32	0.67
Monks 3	94.26	96.76	32	0.73
Two Spirals	50	50	6	0.13
Double Helix	50	50	8	0.37
LED	75.15	72	80	31.68
Cancer1	96	98.28	20	2.99
Diabetes1	77.6	77.08	18	3.4
Glass1	70.81	66.04	60	3.13

In comparing the results of table 3.2 to table 2.3 it is evident that for a number of problems there is no performance improvement to be gained by adding hidden nodes, without further optimisation of the other network parameters. Monks3, LED, Cancer1, Diabetes1 and Glass1 all achieve as good, if not better, results on the test set without hidden nodes being added. The training speed is remarkably improved, as would be expected, and the size of the resulting classifiers has also been reduced. In all cases the performance on the training set is increased by the addition of hidden nodes, but mostly this is over-training. None of the real-world data sets require the addition of hidden nodes to achieve better performance given these training parameters. Only the Monks1, Monks2, Two Spirals and Double Helix data sets require the addition of hidden nodes — which is not an encouraging result as these are all constructed data sets.

3.1.4 Summary

Node patience is able to limit unnecessary training, ensuring that overtraining of the network is minimised. It will only stop training earlier when there are no relatively large data set features to be learnt; training will not continue longer than standard Cascor. It would also seem, from further experiments, that many problems may not require hidden nodes to be solved. Better performance using hidden nodes may be achieved, but only with much greater cost in optimising the network training parameters.

Some criticisms of the node patience method may be made. It does contribute to the number of parameters Cascor requires. However, node patience may be used independently of the other Cascor features as it only partially controls stopping the entire network training. For example, these studies indicate that it is unnecessary to consider node patience periods greater than 2 nodes, and it is rare that a percentage change greater than 5 percent would be required to achieve reasonable results.

Node patience is not a substitute for the use of a validation set. This still remains the best method of ensuring that the correct network size has been achieved. If, however, insufficient training examples are available to produce a validation set then node patience may assist greatly in producing a superior classifier by cutting down excessive training and classifier size.

3.2 Alternative candidate node training schemes

After examining alternative methods of stopping the installation of candidate nodes in Cascor, the next stage is to examine possible improvements to the training mechanism, in particular the candidate training where a large proportion of computational time is used. Here, again, the target is to reduce overall training time and network size, and improve the network classification performance.

Unlike the output layer training where there is the network error, there is no natural combined error measure which can be used to halt the training of the candidates. To decide when to stop training candidates in the normal Cascor system, the patience criterion is applied to the maximum of the candidate node correlations. This particular method of selecting the correlation score is not explicitly stated [Fahlman & Lebiere 1989], but it is inferred from publicly available software [Crowder & Fahlman 1991]. Selecting the maximum of the correlations at each stage is not necessarily the optimum method of choosing a value to apply patience, though it is an obvious choice.

The problem with this standard form of candidate training is that nodes which have different features — such as activation functions, connection strategies, or even different random weights — may train more quickly than the other nodes in the pool. This may hinder the network by forcing the use of a quickly trained node when there may be a better alternative. It also means that the training of one candidate node is influenced by the rest of the nodes in the pool. This final point is especially problematic as the candidates are supposed to be independent. Thus methods for avoiding such a situation are examined.

3.2.1 Description of alternative candidate training methods

Two methods for overcoming these problems are proposed and examined in detail. Both involve changes where the patience stopping criterion is applied to stop candidate training. The first is *independent candidate training*: each candidate unit has its own patience parameters so the candidate trains until it, not the entire pool, runs out of patience. The second is *subgroup candidate training*: subgroups of the candidate pool which have the same properties are trained in a block. For example, half the candidate pool may have Gaussian activation functions and the other half may have sigmoid activation functions, so the pool would be trained in two portions, each having its own patience parameters. The only differences between candidates in a subgroup will be their random weights. The advantage of allowing different activation functions and other network features is that more suitable nodes which match part of the feature space more concisely may be added as required.

The implementation is relatively straight forward. In standard candidate training the entire pool of nodes is trained for a period controlled mainly by the patience criterion applied to the maximum correlation of all the nodes. This code is simply generalised to perform the same function on contiguous candidate nodes: meaning that a lower and upper bound within the candidate pool may be given and all the candidates within that bound are trained as a pool. Standard candidate training is then implemented by calling this subgroup training function with the bounds of the candidate pool: namely one to the number of candidate nodes. Independent candidate training is affected by calling the subgroup training function *n* times, where *n* is the number of candidates: thus the nodes are trained as a number of one node pools. It is assumed for subgroup training that candidate nodes which are of the same style are placed with adjacent array positions, which is performed during the initialisation of the candidates. Determining the subgroups then simply involves checking the activation function — and connection strategy if required — for each candidate node, and training in a subgroup those which are the same (see figure 3.4).

```
noderange l, u;    /* lower and upper bounds for candidates */
noderange n;       /* total number of candidates */
candidateinfo *c; /* variable representing remaining candidate info */

if (independentTraining) {
  for (u = 0; u < n; u++)
    trainCandidateSubgroup(c, u, u);
} else
if (subgroupTraining) {
  l = u = 0;
  while (u < n) {
    while (u < n && c->activationFunc[u] == c->activationFunc[l]) u++;
    trainCandidateSubgroup(c, l, u);
    l = u;
  }
} else
  trainCandidateSubgroup(c, 0, n-1);
```

Figure 3.4 — C code for implementing the alternative candidate training methods, with subgroup training based only on activation function similarities

The training of candidate nodes either in subgroups or independently has several advantages. Training will not be forced to stop prematurely based on the results of a different node or node subgroup, so the candidates should train to produce better results. Likewise, nodes or subgroups may finish training when they would normally be forced to continue, saving time on a serial computer. Conversely, independent or subgroup training may allow the candidates to train excessively, especially when other nodes have already achieved much better results. Likewise, training may cut out too early when local minima are found — the extra training forced by other nodes during standard candidate training may produce a better hidden node. It is not obvious which factors will prevail.

One further method for altering the training mechanism is to alter the function which selects the value used for patience. Fahlman suggests using the sum of the candidate correlations for the patience calculations rather than the maximum [Fahlman 1994]. This ensures that all nodes are allowed to train while it is still possible for them to make reasonable progress. It is expected that the method will allow better training to be performed, but will be slower than the standard maximum criterion. The summing of the correlation scores has no effect when candidates are trained independently.

The implementation of the summation criterion is also straightforward. The sum of the candidate correlations is given to the patience calculation rather than the maximum. The maximum of the candidates is still required to select the final candidate to be installed, as is the case with subgroup and independent candidate training.

3.2.2 Experimental design

The following experiments were performed on data sets which require the addition of hidden nodes based on the experiments presented in tables 2.3 and 3.2: namely the Monks1, Monks2, Two Spirals, and Double Helix problems; and also upon the Monks3 and Cancer1 data sets as examples of what may occur when hidden nodes are not strictly required.

The first group of experiments involves training hidden nodes with only symmetric sigmoid activation functions using normal, independent, and summing candidate training methods. This examines the effect of the different methods given a homogeneous candidate pool, with candidates likely to give similar results, as the only differences between them are the initial random weights. If all the nodes are the same style, subgroup training reverts to standard candidate training; hence this is not examined here.

In the second group of experiments, a variety of different activation functions are used (symmetric sigmoid, asymmetric sigmoid, tanh and Gaussian) on candidate nodes, so it is possible to test subgroup as well as standard and independent candidate training methods. For each of the experiments the candidate nodes are allocated different activation functions so that the number of each style is as even as possible, given that there are four activation functions used and differing numbers of candidates. In the case of there being only four nodes in the candidate pool, subgroup candidate training reverts to individual candidate training.

To clarify the choice of activation functions, the following are the formulas and derivatives for the symmetric sigmoid, asymmetric sigmoid, tanh and Gaussian functions:

$$\mathrm{sig(x)} = \frac{1}{1+e^{\pm x}} \pm \frac{1}{2} \text{ and } \mathrm{sig'(x)} = \frac{1}{4} \pm \mathrm{sig(x)}^2 \tag{3.1}$$

$$\mathrm{asig(x)} = \frac{1}{1+e^{\pm x}} \text{ and } \mathrm{asig'(x)} = \mathrm{asig(x)}\left(1 \pm \mathrm{asig(x)}\right) \tag{3.2}$$

$$\mathrm{tanh(x)} = \frac{e^{x} \pm e^{\pm x}}{e^{x} + e^{\pm x}} \text{ and } \mathrm{tanh'(x)} = 1 \pm \mathrm{tanh(x)}^2 \tag{3.3}$$

$$\mathrm{Gauss(x)} = e^{\pm\frac{1}{2}x^2} \text{ and } \mathrm{Gauss'(x)} = \pm\mathrm{Gauss(x)} \cdot \mathrm{x} \tag{3.4}$$

The actual implementation of the activation function includes bounds to avoid overflow and underflow errors. Although the first three activation functions — symmetric sigmoid, asymmetric sigmoid and tanh — are similar in form, there are enough differences to lead to different training patterns. The asymmetric sigmoid obviously has a range from zero to one, thus being centred around 0.5. The symmetric sigmoid and tanh function are related mathematically with both being centred around zero (tanh(x) = 2sig(2x)), but the tanh function has double the range of the symmetric sigmoid function. A further difference between these two functions is that the slope at zero for the tanh function is four times the symmetric sigmoid slope at the same point. Although the slope may be modified by changing the weights of the node, the initial variation one way or the other may be beneficial depending on the task the node needs to complete. These differences are small, but they lead to greater variation than just the differences in the random weights.

In both experimental groups the length of the standard patience period for candidate training is altered (using 10, 20 and 50 epochs) since the new candidate training methods will affect the training time. Furthermore, the size of the candidate pool is altered to give an indication of which method of candidate training performs better for small, medium and large candidate pools (4, 10 and 20 candidate nodes respectively). It is expected that the larger candidate pool will give better performance per node installed, but will require much more training. When there are 10 candidates in the pool, three Gaussian and three symmetric sigmoid activation functions are used, with two asymmetric sigmoid functions and two hyperbolic tangent functions. Node patience is not used in these experiments.

Full results of these experiments incorporating the percentage correct on the unseen test set, the number of hidden nodes added, and the number of connection crossings are presented in Appendix B. The results of using only a single activation function are detailed in Appendix B.1, and Appendix B.2 details the results of varying the activation functions.

3.2.3 Results and discussion — single activation function

The results of the experiments are reasonably consistent across the different data sets, including those which did not require hidden nodes. As would be expected, training times and classification performance drop with a decrease in the candidate pool size and a decrease in the patience period of the candidate training. The difference on all measures between a candidate patience period of 20 epochs and a period of 50 epochs is nowhere near as marked as the difference between the 10 and 20 epoch limits. A further increase in the candidate patience period to above 50 epochs would be unlikely to accrue any great benefit. The classification performance, the number of hidden nodes installed and the training speed are now considered in turn.

3.2.3.1 Classification performance

For the majority of the data sets there is very little difference between the trials when examining the performance on the unseen data set. Those differences which do occur may be accounted for by differences in the seeds for the trials, or the fact that excess training may produce slight over-training. For the Monks problems and the Cancer1 data set similar results are achieved regardless of the variation in the training parameters (see tables B.1.1.1, B.1.2.1, B.1.3.1 and B.1.6.1).

However, on the Two Spirals and Double Helix problems — both of which require the addition of multiple hidden nodes — there is a marked difference between the candidate training methods. Although at the 20 and 50 epoch patience period for candidate training there is no difference between the trials on percentage correct (see tables B.1.4.1 and B.1.5.1), the 10 epoch range shows distinct differences in the training pattern (see figure 3.5). On the Two Spirals problem, summation training outperforms standard candidate training by a few percent, but independent candidate training is able to greatly increase the performance of the classifier. More effective training is being performed and solutions near the top of the range of possible test results are achieved. The maximum number of hidden nodes are installed on the other trials, stopping training from continuing. If node patience were employed these trials would probably not install the same number of hidden nodes [Waugh 1995a]. Similar results are obtained on the Double Helix problem except that summation candidate training produces similar performance to standard candidate training of around 50 percent, which is no better than chance.

3.2.3.2 *Hidden nodes installed*

This difference in ability between the candidate training methods is also evident upon examination of the numbers of hidden nodes installed into the network. Most of the data sets show that, for patience periods of 20 or 50 epochs on the candidate training, the number of hidden nodes is similar (see tables B.1.1.2, B.1.2.2, B.1.3.2, B.1.4.2 and B.1.5.2). The Two Spirals problem also shows the benefit of using a larger candidate pool with less nodes being required when more candidates are trained (see table B.1.4.2). Over the 20 and 50 epoch cases the only data set which shows any difference between the methods of candidate training is the Cancer1 problem, where independent and summation candidate training add slightly fewer candidate nodes (see table B.1.6.2).

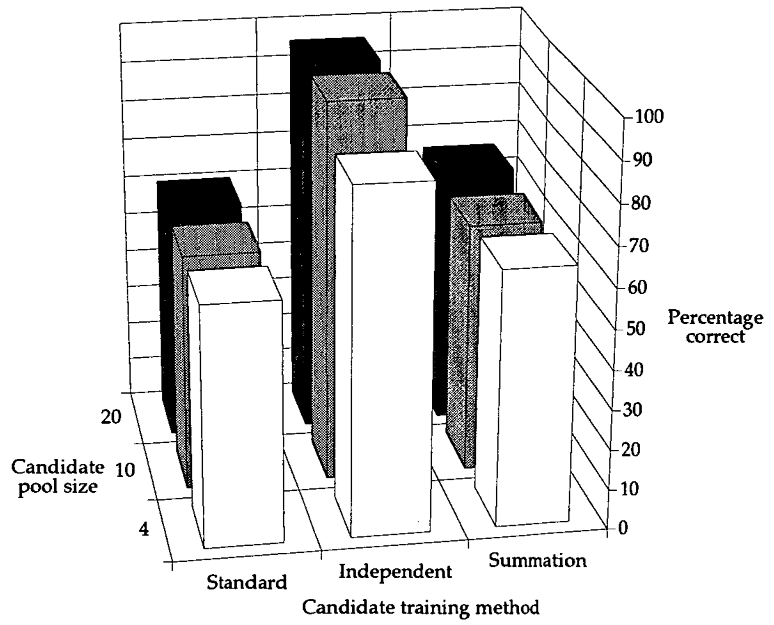

Figure 3.5 — Percentage correct on the unseen test set for the different candidate training methods on the Two Spirals problem, examining the results of the 10 epoch patience period on candidate training

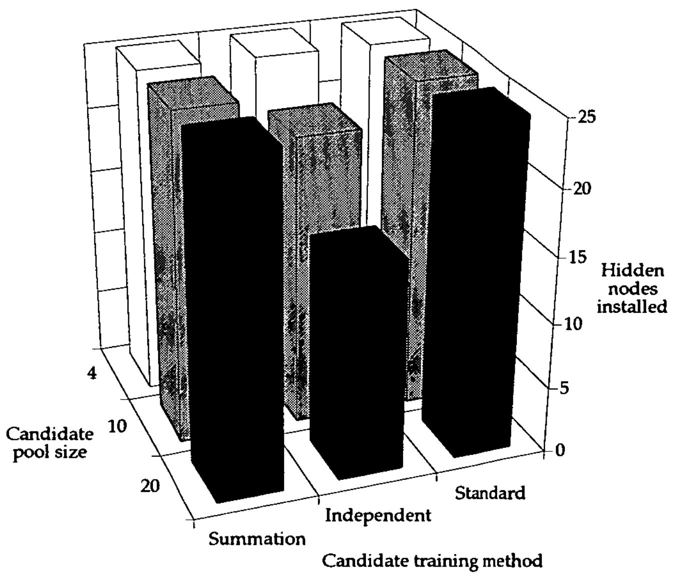

Figure 3.6 — Hidden nodes installed for the different candidate training methods on the Two Spirals problem, examining the results of the ten epoch patience period on candidate training

The real differences between the methods is again shown by examining the 10 epoch patience period trials (see tables B.1.1.2, B.1.2.2, B.1.3.2, B.1.4.2, B.1.5.2 and B.1.6.2). The Monks problems indicate that summation and independent candidate training install less hidden nodes than standard candidate training. The results from the Two Spirals (see figure 3.6), Double Helix and Cancer1 data sets indicate that independent candidate training is superior to the other two approaches, as fewer hidden nodes are installed indicating — along with the percentage correct results — that more effective nodes are being installed.

As mentioned previously, the trials on the Two Spirals data using a 10 epoch patience period would not have continued to the point of adding in 25 hidden nodes had node patience been used [Waugh 1994b]. The lack of performance gain by each hidden node would result in a lack of patience prior to the installation of the maximum number of hidden nodes.

3.2.3.3 *Training speed*

Finally the training speed, as measured by the number of connection crossings, needs to be considered. There is no consistent trend across all data sets as to which method is faster. The Monks1 and Monks2 data sets show that standard candidate training is faster for 20 and 50 epoch patience periods, where candidates are not trained for as long as with independent candidate training (see tables B.1.1.3 and B.1.2.3) as these problems do not require hidden nodes. The latter is faster for 10 epoch periods, as more effective training is performed in the shorter period, leading to less hidden nodes being installed. For the Monks3 problem independent candidate training produces results faster across all the trials (see table B.1.3.3).

For the Cancer1 and Two Spirals data sets (see table B.1.6.3 and figure 3.7 respectively) faster training results are achieved for the 20 and 50 epoch patience period cases using independent candidate training. For the 10 epoch patience period trials more effective training is being performed using the independent training, resulting in higher generalisation ability and less hidden nodes, and also more training time is required to achieve this level (see figure 3.7). Both standard and summation training in this case result in the maximum number of hidden nodes being installed.

The Double Helix problem is mostly solved most quickly by the standard candidate training, followed by independent and then summation candidate training (see table B.1.5.3). The Double Helix data set is unique in this context as, although it requires the addition of hidden nodes, it is simple to solve to 100 percent accuracy using Cascor. This may indicate why standard training is faster than independent candidate training: the standard method stops training earlier avoiding unnecessary candidate development.

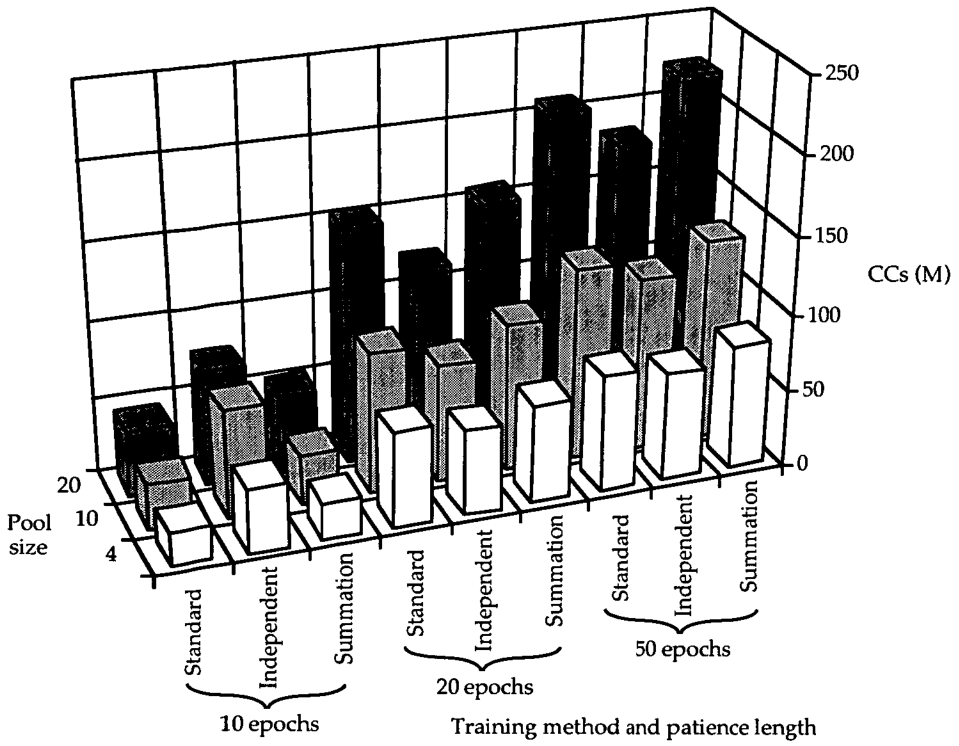

Figure 3.7 — Connection crossings (millions) required for the Two Spirals problem where only a single activation function used within the candidate pool

3.2.4 Results and discussion — multiple activation functions

The results from using multiple activation functions closely follow those from a single type of candidate activation function. The classification performance, the number of hidden nodes and the amount of training required will again be considered in turn.

There is little difference between the methods in terms of the generalisation ability, indicating that there is a reasonable amount of room for variation of parameters while still obtaining good performance. In fact, unlike the previous experiments, a mixture of hidden node activation functions aids in finding a solution, and all methods found reasonable networks (see tables B.2.1.1 through to B.2.6.1).

With regard to the number of hidden nodes added on the Monks1 and Monks2 problems there is little to distinguish the methods (see tables B.2.1.2 and B.2.2.2). However, differences occur on the Monks3, Two Spirals, Double Helix and Cancer1 data sets (see tables B.2.3.2, B.2.4.2, B.2.5.2 and B.2.6.2). The Monks3 data set shows that independent candidate training may result in one less hidden node being installed than for both standard and subgroup candidate training. This trend is repeated for the Two Spirals (see figure 3.8), Double Helix and Cancer1 data sets for both independent and subgroup candidate training.

On the Monks1 and Monks2 the speed of standard candidate training is usually slightly better than both independent and subgroup candidate training (see tables B.2.1.3 and B.2.2.3) again accounted for by the complexity of the problems. However, on the Monks3, Two Spirals (see figure 3.9) and Cancer1 data sets the reverse is true: better training times are achieved by the independent candidate training with subgroup candidate training usually between the other two methods on speed of training (see tables B.2.3.3, B.2.4.3 and B.2.6.3). This is a more significant result as these data sets require more hidden nodes to be added to a network, thus requiring more substantial candidate training. The first two Monks problems, on the other hand only require a single hidden node to be added.

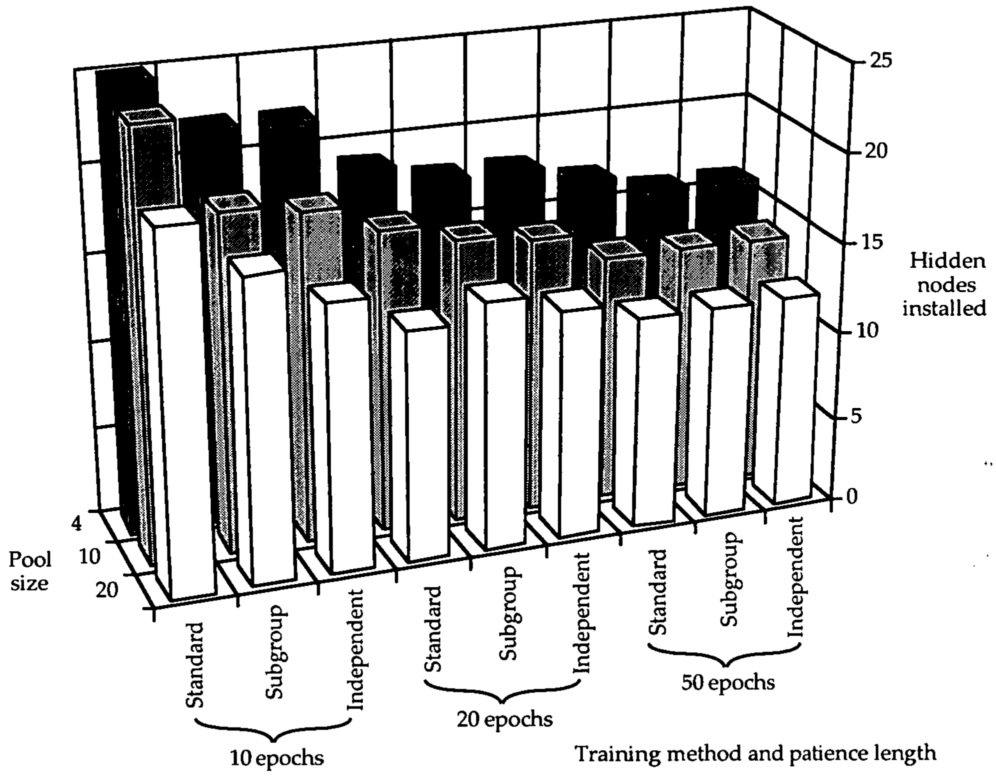

Figure 3.8 — Hidden nodes installed in solving the Two Spirals problem with multiple activation functions in the candidate pool

An exception on the Two Spirals problem is when 20 candidates are used: here the extra candidate nodes with multiple activation functions lead to enough variability that standard candidate training performs well enough without the assistance given to the other methods. This results in better training times (see figure 3.9).

Finally the Double Helix problem shows a mixture of results: when only a 10 epoch patience period is allowed, independent and subgroup training easily outstrips the performance of standard candidate training (see table C.2.5.3). However, when a greater patience period is used, standard training performs slightly better, indicating that the extra training allowed is not required.

3.2.5 Summary

The trials using a single activation function for the candidate nodes indicate that independent candidate training is generally much faster than standard or summation training methods, as it performs more effective training resulting in a smaller and better classifier. Summation training is not a major improvement over standard maximum selection candidate training. The experiments on multiple activation functions show that subgroup candidate training, whilst not performing to the same standard as independent training, also performs better than standard candidate training.

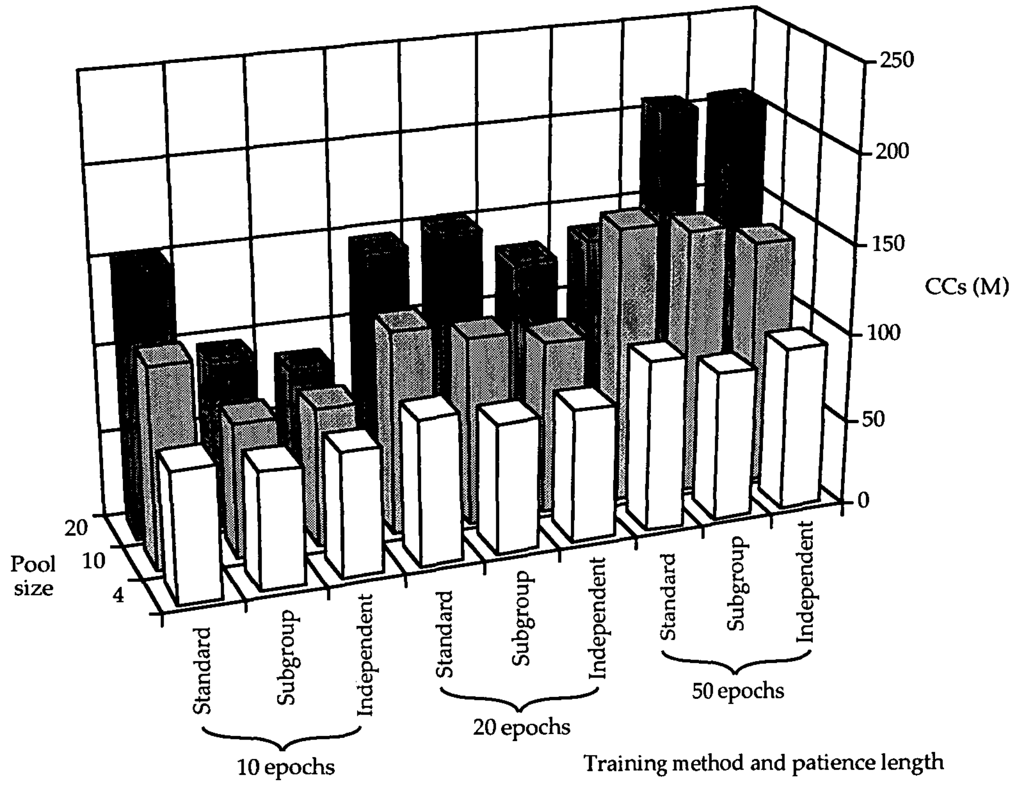

Figure 3.9 — Connection crossings (millions) required for Two Spirals problem with multiple activation functions in the candidate pool

With regard to the results presented here, using only the single set of conservative parameters to train the output layer may bias the results slightly, as there is a greater penalty than is strictly necessary for introducing another hidden node. Nevertheless, the experiments still give a solid indication of the comparative training performances.

Training candidate units independently — in particular — or in subgroups also allows different styles of nodes (different activation functions and connection strategies), as well as different training methods (such as employing pruning or different weight training methods), to be included in the one candidate pool. 'Faster' training nodes will not force training to halt because independent nodes or subgroups set their own training pace. More effective training is performed allowing decreases in candidate pool size and candidate patience periods, while achieving the same level of results.

4 Altering connection strategies within Cascade-Correlation

One of the criticisms of Cascor is that it enables too many connections to be added to a network. This chapter examines methods for reducing the number of connections in Cascor networks: firstly by adding nodes with a limited number of connections, and secondly by the pruning of candidate nodes and the output layer. Results for limiting the number of connections to candidates have been reported earlier [Waugh & Adams 1994], as has the work on pruning within Cascor [Waugh 1994a; Waugh & Adams 1995]. The motivation behind this work is to produce a smaller network which solves the task at hand and is then more likely to provide better generalisation. This may result in reductions to the network depth which may be necessary for certain evaluation speed increases in applications, although this will not be directly considered here.

4.1 Limiting connections by growth

This section investigates a number of different connection strategies for the hidden nodes that Cascor inserts into a network, by limiting the connections a candidate node may make. By definition Cascor starts new candidate nodes with full connections from all inputs and previously added hidden nodes, and to later all hidden nodes and the outputs (see figure 4.1, a replication of figure 2.6 for convenience). One sensible opportunity to change this is to change the connection strategy of the candidate nodes. A more limited topology may improve learning speed as more hidden nodes are added, and the fewer connections could lead to greater generalisation ability as less parameters need to be estimated.

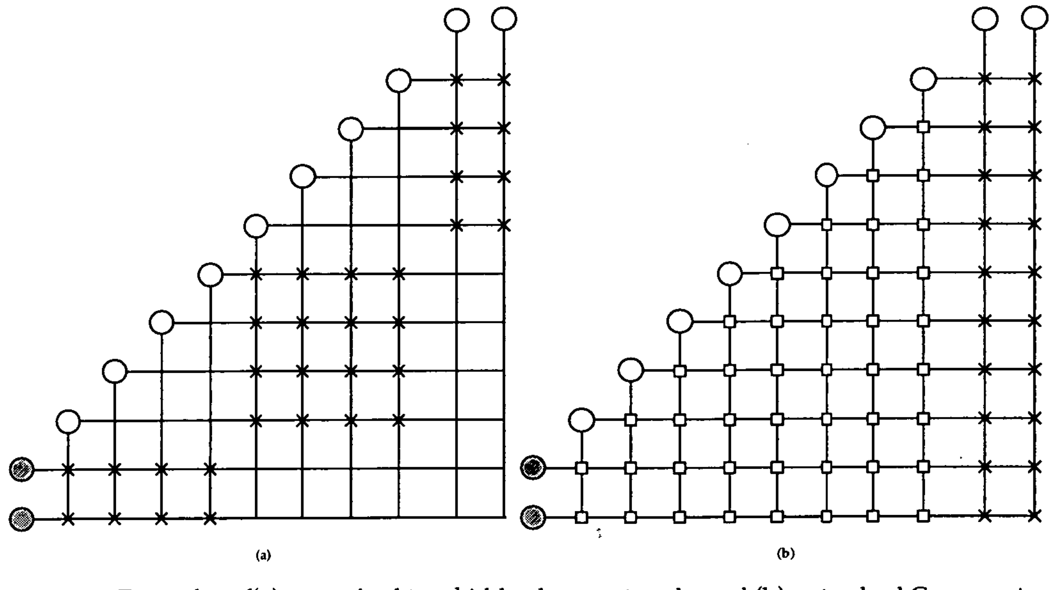

Figure 4.1 — Examples of(a) a standard two hidden layer network, and (b) a standard Cascor network shown in the traditional Cascor format

4.1.1 Alternative node connection strategies

Three different techniques for limiting the number of connections in hidden nodes are examined to obtain an indication of the power of limited connection strategies. These are growing layered networks, growing networks with limited shortcuts, and growing networks with random candidate fan-in.

One method for altering the connection strategy — and one which is often mentioned in the literature (for example, [Mézard & Nadal 1989; Marchand, Golea & Rujian 1990; Sjogaard 1991; Yeung 1991; Baluja & Fahlman 1994]) — is the idea of adding nodes to layers, rather than simply increasing the network by one one-node layer at a time, as is performed by standard Cascor. Thus some or all of the candidate nodes are trained with no connections to a number of the previous hidden nodes, leading to a layer of nodes with inputs only from previous layers (see figure 4.2). This has the benefit of creating networks which are not as deep as a fully cascaded style of topology, with several hidden nodes forming a single layer.

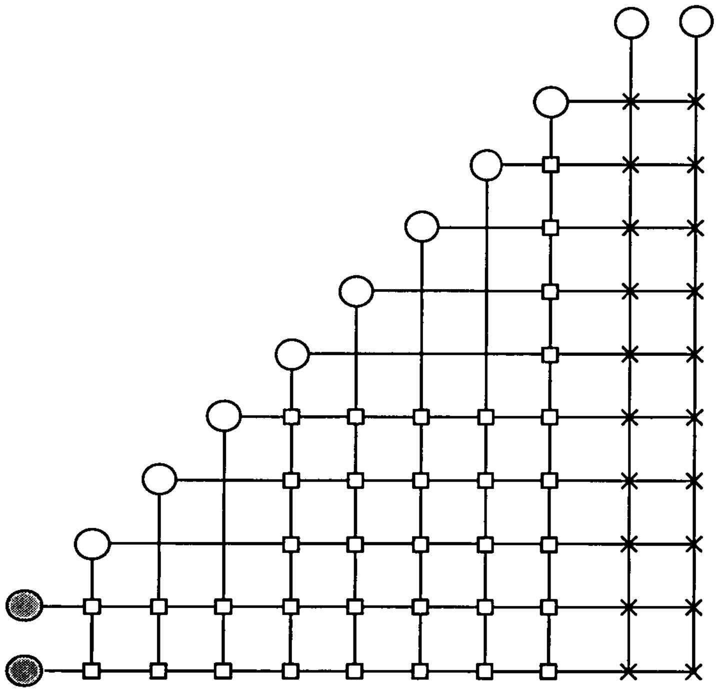
Figure 4.2 — Layered network with shortcuts: a network with layered nodes forming three hidden layers with 3, 4 and 1 nodes respectively

The next method examined uses a minimal number of shortcuts between layers, where shortcuts from hidden nodes to non-adjacent hidden nodes are not used [Waugh & Adams 1994] similar to the Tower construction suggested by Gallant [Gallant 1986]. This means that a hidden node will only receive connections from the immediately previous hidden node and the inputs, and is in turn only connected to the next hidden node and the output nodes (see figure 4.3). This greatly decreases the number of connections required, although it does not decrease the number of layers.

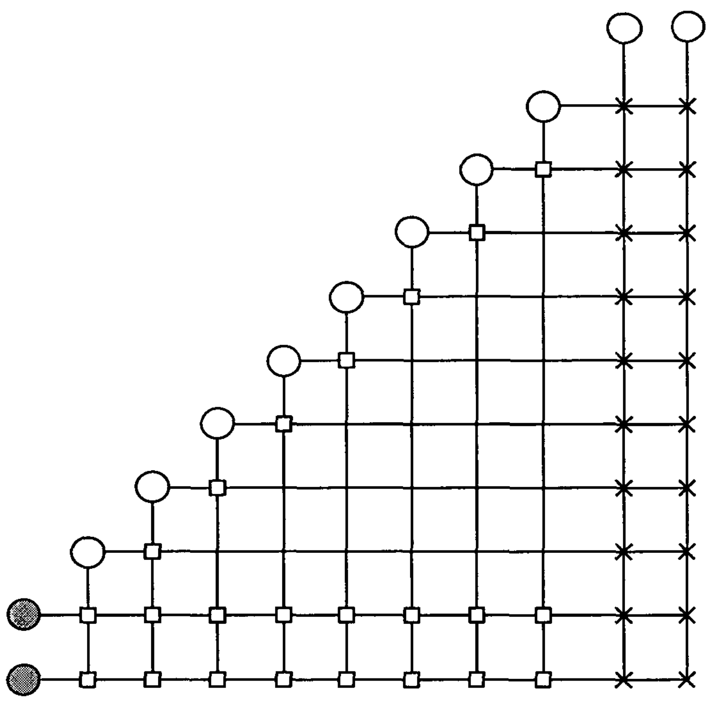

Figure 4.3 — Minimum shortcut network: a network where all nodes have a minimal number of shortcut connections

Using a pool of candidates with a limited number of randomly chosen connections is the final node topology to be examined — also termed Limited Fan-in Cascade-Correlation (LFCC) [Klagges & Soegtrop 1992]. A number of connections from both the inputs and previously installed hidden units are selected randomly for use. This is done in two distinct ways: a random number of connections (Rand-LFCC) [Waugh & Adams 1994]; and a fixed number of connections, in this case two connections (2-LFCC).

The implementation of the different connection strategies is relatively straight forward. The network data structure is extended to not only include the network weights, but also to include a boolean variable for each weight flagging whether there is a connection present or not. The number of required nodes of each style is set and the appropriate connection strategies are allocated by initialising these connection flags. The code for performing this initialisation is outlined in figure 4.4.

4.1.2 Node forcing and experimental design

Experiments are conducted to test the effects of limiting the number of connections to candidate nodes. The experiments are conducted by altering the candidate pool in two ways. Firstly, the candidate pool is split in half: one half with standard fully connected hidden nodes, and the other half with the limited connection nodes. Secondly, the candidate pool solely uses the particular limited connection nodes.

```
fullyConnected = true;
for (i = 0; i <= n + h; i++) {
  if ((connectionStrat == Layered  && i > n + h - nodesInLayer) ||
      (connectionStrat == MinShort && i > n && i < n + h) ||
      (connectionStrat == Random   && i > 0 && !in(randConns, i)) )
  {
    candidateWeight[i] = 0.0;
    candidateConnection[i] = fullyConnected = false;
  } else {
    candidateWeight[i] = randomWeight;
    candidateConnection[i] = true;
  }
}
if (connectionStrat != Full && fullyConnected) connectionStrat = Full;
```

Figure 4.4 — C code implementing the initialisation of candidate node connections for each node; where "in" is a function returning true if that connection number is one of the connections to be present

The first pooling method is further modified to allow forcing the use of limited connection nodes, when the correlation of the best limited connection node is near that of the best standard cascaded node. This gives a higher priority to the limited connection nodes, and is achieved by adding to these candidates a percentage of their own correlation score. The purpose of this is to bias the candidate training in favour of the candidates with limited connection strategies and thus guide the network structure towards the required form, although the method may be used to increase the chance of any node variety being used. This method has been independently investigated by Baluja and Fahlman [Baluja & Fahlman 1994]. Three different forcing factors — the percentage by which the specified nodes are biased over others — are used in these experiments: 0, 10 and 50 percent. The implementation involves, in this case, setting a default connection strategy of full connection, and multiplying the correlation score of any non-default node by the forcing factor, thus increasing its chances of selection. The node with the highest modified correlation score is then chosen as the candidate to be installed as well as that value being used for the correlation calculations.

These series of experiments are conducted on the Monks problems, Two Spirals, Double Helix and the Cancer1 data sets. Independent candidate training is used for these trials.

4.1.3 Results and discussion

The full experimental results are presented in Appendix C, giving the percentage correct on the unseen test set, the number of limited hidden nodes and total hidden nodes added, the number of network connections, and the total connection crossings as an indication of training speed.

4.1.3.1 *Classification performance*

The performance of the different styles with respect to their generalisation ability is even (see tables C.1.1, C.2.1, C.3.1, C.4.1, C.5.1, and C.6.1). This is to be expected, as the aim of training is to produce a network which solves the problem at hand.

There are only two exceptions: on the Two Spirals problem, for example (see figure 4.5), both layered nodes and random connection nodes with only two links have difficulties in solving the problem when all nodes have those connection strategies. It is well known that a network without advanced feature detectors has a great deal of difficulty in solving the Two Spirals problem. Hence a fully layered network, forming one hidden layer with shortcut connections from the inputs to the outputs, will not be able to solve the problem completely. The two random connection nodes have a similar problem in that it is unlikely that, out of a group of ten candidate nodes, the right connections will be obtained, let alone that the connections will have the right starting weights. Hence the performance is lower on the Two Spirals and the Monks2 problems (see tables C.3.1 and C.4.1), although it may be improved by greatly increasing the size of the candidate pool.

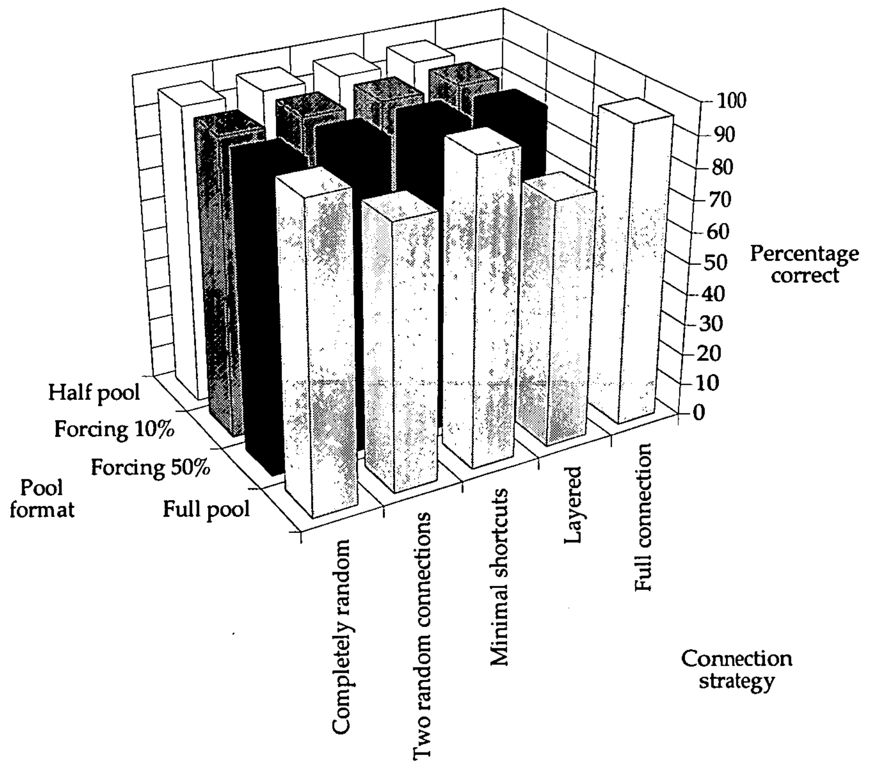

Figure 4.5 — Percentage correct on the unseen test set for the Two Spirals problem: note that the full pool candidate training is only valid method for fully connected nodes

Nevertheless, these results imply that a large number of the network connections may be redundant, as the limited connection strategies can do equally well as fully connected nodes.

4.1.3.2 *Network structure*

Upon examining the numbers of hidden nodes installed, further interesting trends become evident. Firstly, the Monks problems are not completely suitable for testing the limited connection strategies as they require two or less nodes to be added to the network to be solved (see tables C.1.2, C.2.2 and C.3.2). This creates difficulties in that the layered nodes do not come into effect until after a first hidden node has been added, and it takes two hidden nodes to be added before there is any effect from the introduction of minimal shortcut nodes. The Monks problems indicate that having pools of nodes with only two random connections may require more nodes to be installed, or alternatively not solve the problem by the time the maximum number of hidden nodes has been reached given the current pool size. Further, nodes with a random number of random connections are able to introduce nodes with less than maximum connections quite easily, although this does not guarantee a large reduction in connections (see tables C.1.3, C.2.3 and C.3.3).

With regard to the other benchmarks, the fact that the nodes with less than the full connections are chosen without forcing their use is an indication that a fully connected node is not necessarily the best option (see tables C.4.2, C.5.2 and C.6.2). Indeed, a choice of the limited connections may be beneficial to the network. Having said that, no large numbers of hidden nodes have been replaced, which indicates that the weight training algorithm may be removing the effect of unnecessary connections without the need to cut them absolutely. For example, on the Two Spirals problem over 100 trials a maximum of two limited nodes are used out of between 12 and 14 needed to solve the problem when the node usage is not being forced (see figure 4.6).

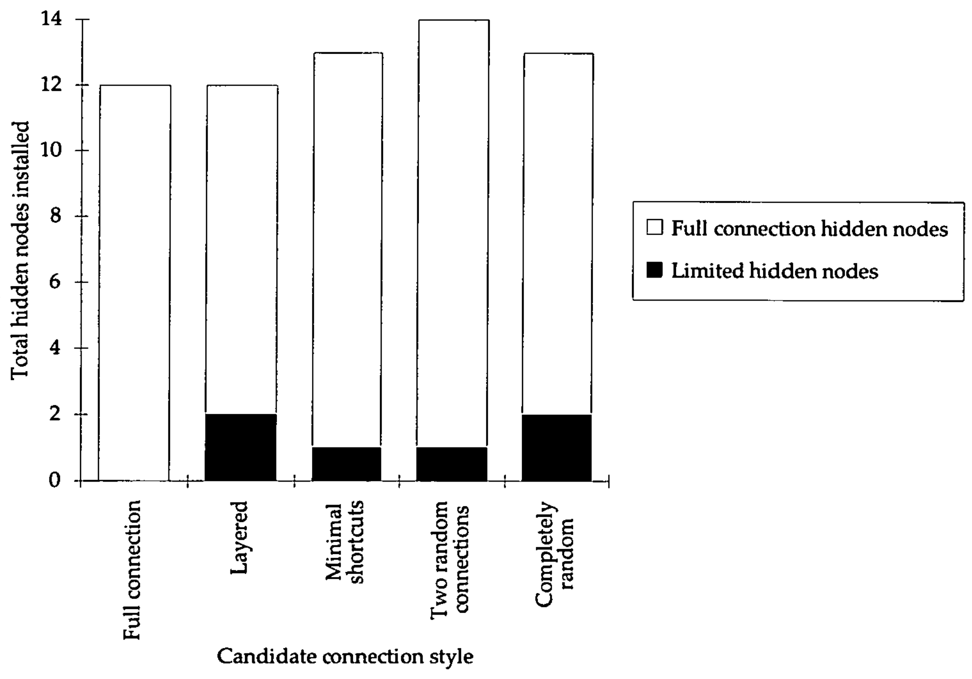

Figure 4.6 — Hidden nodes and limited hidden nodes installed on the Two Spirals problem without forcing

Adding a forcing factor to help choose particular nodes is a workable method for biasing in favour of a desired network topology. Limited forcing strongly increases the number of specialised nodes, with only a small increase in the total number of nodes employed. For example, again the Two Spirals problem shows a great increase in the number of limited nodes used when 50 percent forcing is applied (see table C.4.2). The layered nodes installed increases from 2 to 9 while the total number of nodes needed increases by only 2 to 14 (see figure 4.7). Nevertheless, using only the one limited node style in the candidate pool may hinder the network — as the results on the Two Spirals problem show with great increases in the number of nodes installed — as some node connection strategies are not able to solve the problem. The results from the Double Helix and Cancer1 data sets mirror these findings, except that fewer nodes are required to solve these benchmark problems (see tables C.5.2 and C.6.2).

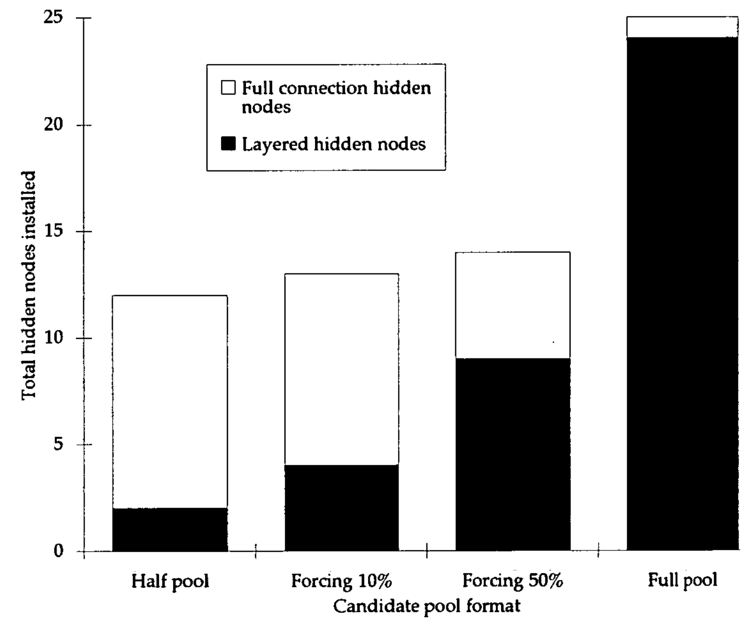

Figure 4.7 — Layered nodes installed with forcing for the Two Spirals problem

Nevertheless, these limited connection strategies are useful if a particular network structure is required. For example, when layered nodes are used on the Two Spirals problem, the following number of layers were required: 11 for the half pool without forcing layering, 10 with 10 percent extra forcing, 7 with 50 percent extra forcing, and of course 2 when using a full pool of layering candidates. A total of 13 layers were employed by standard Cascor.

The number of nodes required gives only part of the picture. The number of connections in the final networks must also be examined. The Monks problems show very little variation in the number of connections required, except to show that using only two random connections may often backfire and require a much greater number of connections to be used (see tables C.1.3, C.2.3 and C.3.3). On the Double Helix and Cancer1 problems, all limited connection methods produce a slightly smaller network than the standard Cascor network, with the best method being the layering of hidden nodes (see tables C.5.3 and C.6.3).

However this trend does not continue with the more difficult Two Spirals problem (see table C.4.3). For example, a random number of random connections does not decrease the number of connections needed to solve the problem at any stage. The only method to have some reasonable success at producing a smaller network is the minimal shortcuts (see figure 4.8 and C.4.3).

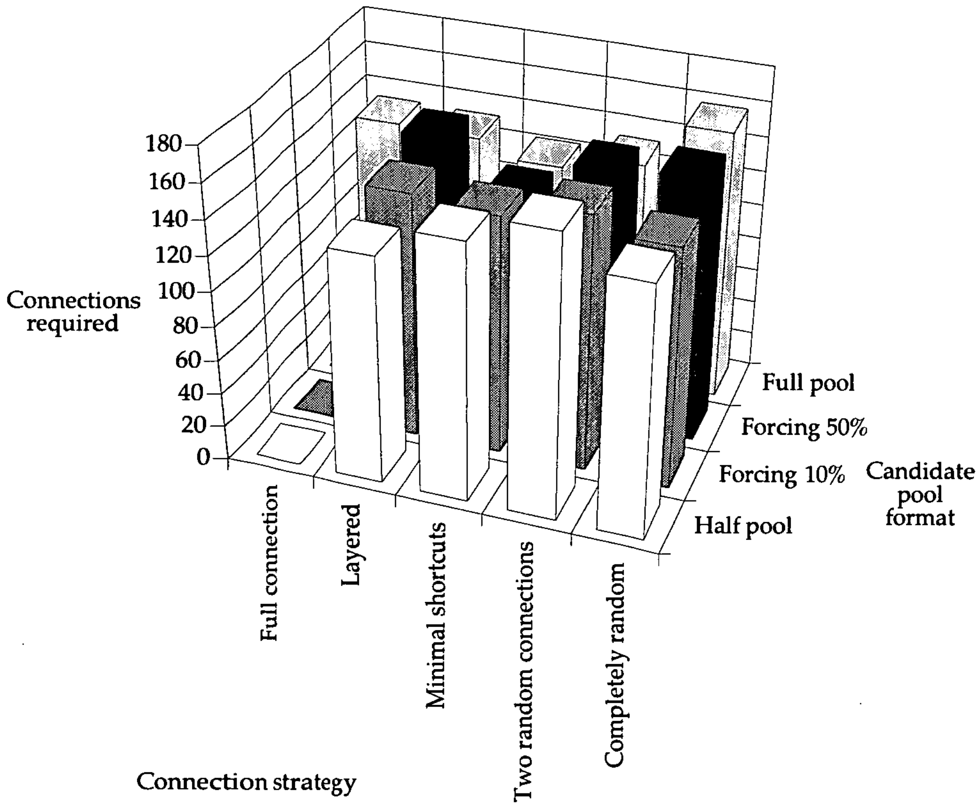

Figure 4.8 — Connections required in solving the Two Spirals problem

4.1.3.3 *Training speed*

With regard to training speed, once again the Monks and Cancer1 problems are similar in nature (see tables C.1.4, C.2.4, C.3.4 and C.6.4). Layered and minimal shortcut nodes show little difference in training when compared to standard fully-connected nodes. Both the random connection techniques show a decrease in training times due to fewer connections being present: two-connection nodes taking the smallest amount of training except when used in a full pool, as would be expected. On the Double Helix problem, most trials decreased the training time, with layered and completely random nodes showing the largest and most consistent decreases (see table C.5.4). On the Two Spirals problem, again most trials decrease the learning time, with minimal shortcuts and two random connection nodes being the most effective (see table C.4.4 and figure 4.9).

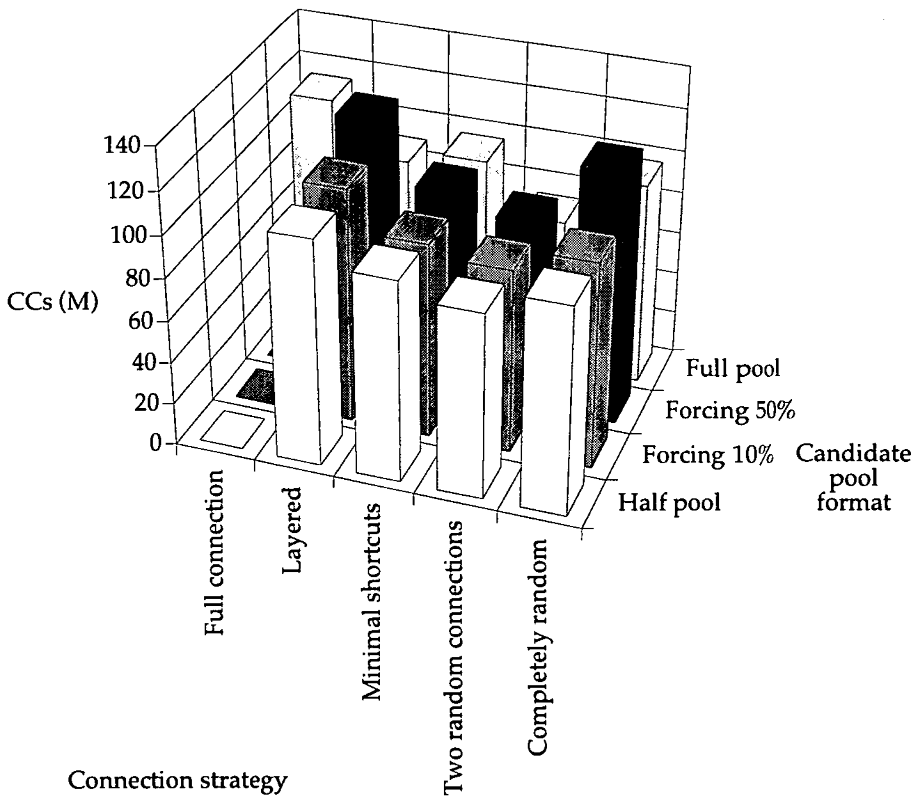

Figure 4.9 — Connection crossings required in training on the Two Spirals problem

An interesting point is that generally forcing limited connection nodes by 50 percent increases the training time (see figure 4.9). This is due to forcing the use of the limited-connection nodes more often, basically halving the candidate pool size. This of course means that less useful nodes are being installed and that further training has to be completed at a later stage. The training times generally drop off when the entire pool is limited connection nodes.

4.1.4 Summary

It is possible to add limited connection hidden nodes to a Cascor network without any decrease in classification performance. The network structure is influenced but there is generally little reduction in network size. Further limited connection strategies other than the ones selected are possible, but perhaps not as obvious. These different strategies appear to be particularly good for certain problems but not for others, so a more general method of reducing connections should be developed to be usable for all problems.

Forcing is an effective method of ensuring that the preferred node styles are used above another style. Overall it would seem that forcing only certain types of connection strategies, such as layering, may hinder the network's performance, although for the most part this does not occur. Allowing the use of other node styles solves this deficiency.

A further drawback is that the methods examined here do not alter the training requirements for the Cascor output layer in any way. This means that for a large number of problems, which require few hidden nodes to be effectively solved, the methods mentioned here will have little effect. These techniques do allow the Cascor user to guide the way in which the final network is to be shaped, and are effective in this respect. However other methods of reducing connections need to be examined.

4.2 Limiting connections by pruning

As opposed to the methods presented in the previous section, where the cascaded networks have been altered by allowing limited connected nodes to be added to the networks, this section considers reducing the size of the network by training fully connected nodes and pruning connections from them. As the number of hidden nodes increases, covering all cases of limited connection hidden nodes as in §4.1 becomes impossible. Therefore, fully connected hidden nodes may need to be used to ensure that the smallest possible network may be generated. These nodes, including the output layer nodes, may be pruned later. The pruning of connections is a more sound approach to the reduction of the size of Cascor networks, as is identified in chapter 2, and is more likely to produce a smaller classifier in line with the aims expressed at the beginning of this chapter.

4.2.1 Pruning algorithm

The choice of pruning method is not obvious without a study of which is most effective. Although some such studies have been performed [Thimm & Fiesler 1995] the results are less than conclusive given the large number of pruning and regularisation algorithms available. Thus an arbitrary decision has been made to use Karnin's connection pruning algorithm [Karnin 1990]. This involves calculating a saliency measure of the importance of each weight, and pruning the weights with the lowest sensitivities. The derivation and justification of the sensitivity calculation from Karnin's paper is given below.

The sensitivity *S* of a network to the removal of a weight *w* is:

$$\begin{aligned} S &= E(0) \pm E(w^f) \\ &= \pm \frac{E(w^f) \pm E(0)}{w^f \pm 0} \cdot w^f \end{aligned} \qquad (4.1)$$

where w^f is the final value of the connection on completion of the training phase, and *E* is the network error expressed as a function of *w* assuming all other weights are fixed in their final state. Assuming further that the initial random weight value will substitute for the error with the weight zeroed, the sensitivity *S* may be approximated by:

$$S \approx \pm \frac{E(w^f) \pm E(w^i)}{w^f \pm w^i} \cdot w^f \tag{4.2}$$

where the final value, w^f, and the initial weight, w^i, are values which are available during training. The numerator in (4.2) may be estimated in turn by the following integral:

$$E(w^f) \pm E(w^i) \approx \int_I^F \frac{\partial E(w)}{\partial w} \cdot dw \tag{4.3}$$

where I and F are the initial and final positions of w respectively. This may in turn be approximated by a summation giving the entire sensitivity calculation:

$$S \approx \pm \sum_{e=0}^{N \pm 1} \frac{\partial E}{\partial w}(n) \cdot \Delta w(n) \frac{w^f}{w^f \pm w^i} \tag{4.4}$$

where N is the number of epochs of training, and the gradient and weight change are available during training. This means that there is little cost in implementing the saliency measure calculation. The previously calculated values are simply stored in shadow arrays until they are required for pruning. Only the initial weights and the current sum of the epoch information has to be stored. Note that if the weight is the same as the initial weight, the sensitivity is assumed to be zero.

The sensitivity measure works equally well for correlation calculations: at the beginning of candidate training the initial weights are stored and the derivative and weight change use the correlation instead an error measure. Since Cascor has two quite different training phases, it is sensible to use two sets of pruning parameters, one on the candidates, and one on the output layer. Karnin's saliency measure does not adjust the other weights so the pruned nodes need to be retrained.

The aim of the following experiments is to test Cascor networks pruned with Karnin's sensitivity measure against the standard Cascor network training. There are two questions to be answered about how to incorporate pruning into the Cascor architecture: where to prune and how to stop pruning. These issues and their effects on network performance will be examined in turn.

4.2.2 Where to prune?

Since Cascor training is cyclic it is not immediately obvious where pruning should be applied to the trained connections. Pruning the hidden nodes at the completion of network training is only feasible with the subsequent retraining of those nodes, which is not practical when the network is very deep. Thus there are two options of where to prune:

- prune each candidate pool when trained and prune the output layer after training is complete; or
- prune each candidate pool and prune the output layer after each output training phase.

This choice only has an effect when hidden nodes are being added to the network.

Experiments on where to prune are performed on the Two Spirals data set, as an example of a problem requiring a deep network. Here only one parameter is used for both phases of training. Two arbitrary pruning levels (0.01 and 0.06) are used on both the candidates and the outputs to give an indication of the results expected for low and high pruning levels. The usual range of pruning levels for the Karnin measure is approximately between 0 and 0.1. Any connection with a sensitivity less than the specified level is removed from the network (see table 4.1).

Table 4.1 — Results of when to prune (no pruning, output layer at end of training, or every output layer training phase) using two pruning levels (0.01 and 0.06) on the Two Spirals problem, giving the percentage correct on the test set, the number of hidden nodes, the number of network connections and the number of connection crossings (millions) training required

Where to prune?	Prune level	Test set %	Hidden	Connections	CCs (M)
No pruning	N/A	95.83	12	132	112.8
Output once	0.01	95.83	12	104	127.7
	0.06	95.83	14	89	148.0
Every output	0.01	90.62	19	162	171.0
	0.06	83.33	25	100	214.1

Table 4.1 indicates that pruning can be effective in reducing network size without damaging classification ability. Although pruning requires more training, the trade off to obtain a smaller network and thus a more concise classifier may be worthwhile as this process reduces the number of free parameters within the network. There is, however, no indication here of increased generalisation ability. This is not surprising as the Two Spirals problem is not a good test for generalisation, but rather memorisation [Baluja & Fahlman 1994]. The extra requirement to prune the output layer after each training phase damages the final network by prematurely removing connections which may be used later. The conclusion from these experiments is that the best pruning method is to prune connections after training of the particular nodes, whether that be the candidate nodes or the output layer, has been fully completed.

4.2.3 Stopping pruning

Three methods have been developed to stop the pruning process: two absolute measures and a third relative measure.

4.2.3.1 *Arbitrary choice and percentage change*

The obvious way of removing connections, as mentioned in §4.2.2, is to pick an arbitrary cut-off sensitivity level, and remove all connections with saliencies below that level. This will then relate to either the change in error with the removal of an output layer connection, or the change in correlation if pruning candidate nodes. Given the saliency measure algorithm (see §4.2.1), the implementation of this is trivial.

A further method is to remove connections whose expected change in the error or the maximum correlation is less than a fixed percentage. The maximum correlation is used to prune all of the candidates so that the saliency of each connection is being standardised across the entire candidate pool. A node with a lower correlation will not have connections regarded more highly because of that node's correlation score. Both these methods measure a change relative to the initial error or correlation values, before processing occurs. The implementation of this method is also simple: the minimum sensitivity level is multiplied by the current error or maximum correlation.

4.2.3.2 *Relative saliency measure*

Deciding when to stop pruning is a similar problem to the decision of when to stop training in Cascor: at some stage a decision to stop must be made. Some mechanism similar to patience may be used, as opposed to simply picking an arbitrary level. However, unlike training where it is possible to use the patience criterion and to train for a few extra epochs to check when to stop, pruning a few extra connections without any regard for their effect may destroy a node's functionality. There is no buffer of extra connections to sacrifice to a patience criterion to start the process of deciding when to stop pruning.

The following algorithm has been developed to allow for a patience-like method of stopping connection pruning. It is assumed that the saliency measure is a relatively accurate measure of the importance of each connection:

1. calculate saliencies of all connections after training has been completed;
2. remove connections with zero or negative saliency, thus decreasing the network error;
3. sort the remaining saliencies;
4. remove connections from lowest to highest saliency until the predicted error or correlation change is too large, using the training error or correlation as a starting point; and
5. retrain if required.

Deciding when to stop pruning falls to a measure of change in the sorted saliencies: if the change is too large then pruning is stopped. Unlike the patience criterion, no patience period is used, but the increase in saliency compared to the removal of the previous connection is checked. Such a period, measured across the range of connections, is not necessary due to the sorted nature of the saliencies. It would force the removal of at least n-1 connections where n is the length measured in connections removed. The use of a term such as 'reverse patience' [Waugh & Adams 1995] is thus inappropriate as there is no longer any 'time' period over which the error change is measured.

As the saliencies are assumed to be a measure of the change in error or correlation caused by the removal of a connection, it is necessary to calculate the saliency changes in relation to the current error or maximum correlation level, not just using the actual saliency value. Thus this level is used as a starting point for these calculations. This will ensure that the saliency's relevance is taken into account with respect to the network error or the maximum correlation.

Connections from layers are pruned together rather than pruning each node independently, as it is possible, for example, to have as little as two connections to a candidate or output node. Combining the connections from otherwise independent nodes will give a more significant sample from which to draw saliencies to decide which connections to prune.

4.2.2.3 *Results and discussion*

Results of experiments conducted are presented in Appendix D detailing the percentage correct on the test set, the number of connections required, and the number of connection crossings training required for training.

The three methods for stopping pruning are tested on the nine benchmarking problems with Monks3, Cancer1, Diabetes1, Glass1 and LED resulting in the installation no hidden nodes, meaning that only the output layer needs to be pruned. The other problems — Monks1, Monks2, Two Spirals and Double Helix — include trials on pruning the candidate nodes as well. All trials consist of giving pruning levels of 0.0 to 0.1 in steps of 0.01 to the absolute, percentage and relative pruning methods. These results are summarised below.

There is little to distinguish between the methods of stopping pruning. All are effective (see tables D.1.2, D.2.2, D.3, D.4.2, D.5.2, D.6, D.7, D.8 and D.9), although the relative method may over-prune candidate nodes when high levels are used (see figure 4.10 and tables D.1.1, D.2.1, D.4.1 and D.5.1). This results in the maximum number of hidden nodes being installed, as the data set features cannot be learnt under such conditions. The over-pruning using saliency changes is simply due to the pruning depending on the previous connection removed; the relative nature of the algorithm means that more connections are cut as long as the relative difference is not too great.

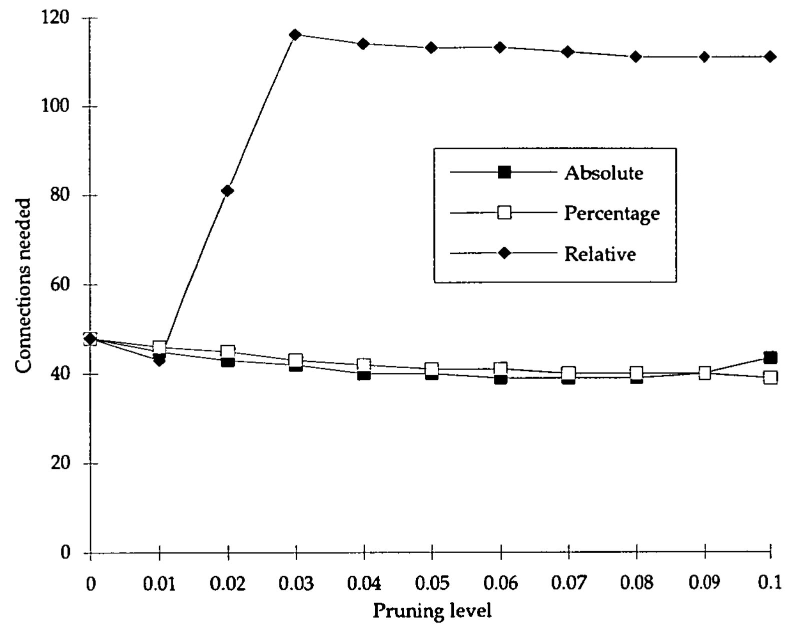

Figure 4.10 — Candidate node pruning on the Monks1 problem: comparison of absolute, percentage and relative pruning

One feature of the results that is immediately evident is that little or no connections are pruned from the output layer of networks (see figure 4.11 and see tables D.1.2, D.2.2, D.3, D.4.2, D.5.2, D.6, D.7, D.8 and D.9), even with the relative saliency method. The worth of these connections is greater than those of a hidden unit as they directly influence the output error as opposed to indirectly through the correlation to the output error, but it is unexpected that so few would be removed. One reason may be that all of the inputs are required to solve the network. Cancer1 requires all 20 possible connections — taking into account biases, Diabetes1 uses all 18 possible connections, Glass1 uses 57 to 58 of a maximum possible 60, LED requires 71 to 72 of a maximum of 80 (see figure 4.11), Monks1 uses 43 to 44 of a total of 50 and so on. Unnecessary connections are removed early and the rest are required to solve the problem.

An alternative explanation for the low pruning of the output layer is that the actual pruning algorithm used is not effective in estimating the saliency of connections. This is difficult to quantify without further comparison between differing methods of pruning.

When used to reduce the complexity of candidates being introduced into the network the performance of the pruning methods is much better than that of limited connection hidden nodes (see tables D.1.1, D.2.1, D.4.1 and D.5.1). Classification performance is not affected when, for example, the number of connections in the Two Spirals networks are reduced from 132 to 92 using absolute pruning, without employing any connection pruning on the output layer (see figure 4.12). This compares with a minimum of 131 connections obtained using limited connection candidates under similar conditions. A further point is that as more pruning is performed more training is required to compensate to solve the problem by greater alterations to existing connections or by the addition of further hidden nodes. In this situation the candidate nodes are being over-pruned, although Cascor is still able to find solutions.

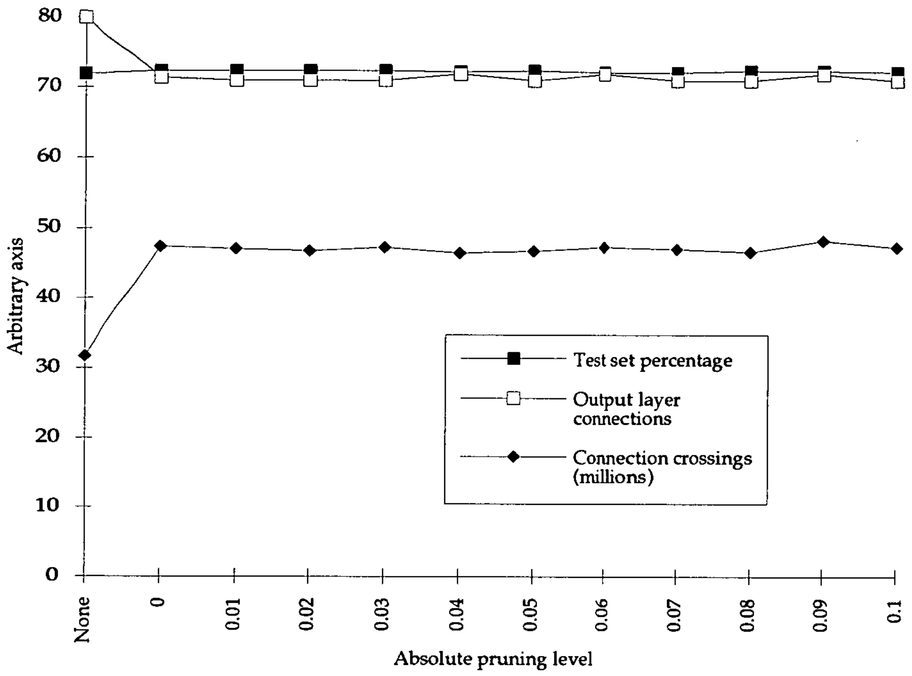

Figure 4.11 — Output layer pruning using an absolute level on the LED problem

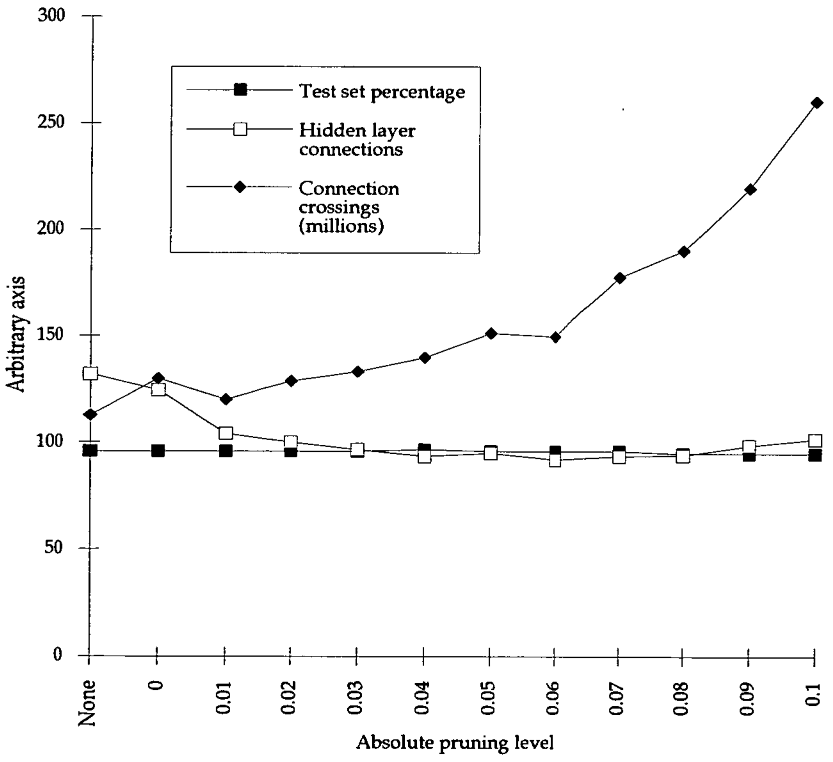

Figure 4.12 — Candidate node pruning using absolute level on the Two Spirals problem

A final point is that although a large number of connections have been removed the actual generalisation ability of the networks is not increased or decreased by the removal of connections — in fact no change beyond random variation is evident (see figures 4.11 and

4.12). The lack of increase may be due to the networks being so small initially that further reduction of free parameters is not necessary to achieve good generalisation capabilities, and so it may be peculiar to the Cascor training method.

4.2.3 Summary

Pruning is a more principled approach to removing connections which are not required than picking an arbitrary hidden node connection strategy. Simple pruning can remove a large number of connections from a standard Cascor network, especially from hidden nodes, with no change in the classification ability and only a small amount of extra training. The removal of the unnecessary connections allows for the possible extraction of knowledge from networks to occur more easily [Tolstrup 1995], as well as reducing the number of free parameters which in turn reduces over-training. The methods used to stop pruning may be applied to other artificial neural networks, and all are effective in stopping the pruning process. The level of pruning may need to be determined empirically depending on the problem at hand in the same way that the learning rate is determined. Nevertheless, it is possible to employ a small amount of pruning without jeopardising the quality of the final results.

Part II Benchmarking Cascade-Correlation

5 Background to benchmarking databases

In the previous two chapters several extensions to Cascor have been examined. In doing so, it became evident early on that the benchmarks selected to test the extensions are less than sufficient. The majority of problems appear to tend towards two extremes: linearly separable or unsolvable given the available information; or they are contrived and do not effectively test generalisation. The possible problems which may be solved by algorithms such as Cascor need to be understood in abstract terms. However, more immediately there is a demand for problems which will test the different capabilities of learning algorithms.

In general, one of the faults of current research into inductive learning, particularly in the artificial neural network field [Prechelt 1994b], is that new learning methods are not benchmarked in a consistent or sufficient manner. The trials of a learning method are often performed on a single data set, which may not be readily available to other researchers, or which is overly simple, such as the ubiquitous xor problem. Hence, it is important to develop benchmarks for testing new and variations on existing methods.

The aim of this part of the thesis is to develop new benchmarks for artificial neural network classifiers. Although this study was motivated by the examination of Cascor and other artificial neural networks, the benchmarks shown in this part of the thesis may be applied to any inductive learning system.

In this chapter, a background to the area of benchmarking data sets is given. This covers what features of data sets need to be considered, examples from the literature of real-world and constructed benchmarks, and examples of the performance of Cascor on some of these sets. This is followed by chapters giving new examples of the different benchmarking styles.

5.1 Features of data sets

To create a benchmark some consideration of what features are important within data sets is necessary — a description of the data character. What sort of structure the data sets can entail is outlined below, thus looking at the content of the data set and the complexity of the underlying functions. How these data set features are presented is then examined, both through the dimensions of the problem and the effect of sampling.

5.1.1 Underlying problem structure

Obviously one of the important features of data sets is the structure of the underlying problem, which gives the difficulty in learning — sometimes termed the concept character or the class distribution for classification problems [Rendell & Cho 1990]. This is the distribution of examples in the instance space, as defined by the measured attributes — the shape of the class regions or partitioning in relation to each other, or the shape of the surface formed by a regression problem. The attributes — otherwise known as the features, variables tests or inputs — are the measurements or observations recorded about each example. Each example or instance is one case drawn from the population under consideration. Whether examples are present only in certain areas of the attribute space, or whether examples occur in an uniform distribution across the attribute space, for example, is part of the underlying problem structure. The attribute space is the geometric space formed by using the attribute values as axes of measurement, meaning that each example forms a point in the space.

It is worth noting that the distribution of classes throughout the attribute space may lead to some interesting formations with interlocking classes. However, it is more likely that class regions may simply not meet, and thus may be solved with a simple classifier; or they may overlap resulting in an unsolvable situation. The area of each data set which may be solved using a more powerful classifier over a simple linear classifier may be very small.

Rendell and Cho examine a number of features of classification problems which relate to these ideas: the size of the concept — the amount of the feature space it covers; the concentration of peaks in the one class — whether a number of peaks formed by examples are distributed around the feature space, or whether there is only one peak of class membership; conformation — whether the peaks of a class are normal in shape or involve 'all-or-none' class membership; and whether there is some higher order regularity in the distribution of class peaks [Rendell & Cho 1990]. Although a process of generating artificial data sets is described in this paper, the actual details of the examples used are unclear and only single data sets are used to test differences in the concepts.

A number of papers consider concepts which are logical combinations of attribute values (for example, [Lounis & Bisson 1991; Hickey 1992]). These papers are more directed at testing the capabilities of symbolic machine learning systems, such as C4.5 — a decision tree inductive learning methodology [Quinlan 1993a].

Quinlan [Quinlan 1993b] identifies two styles of problems: S-type which are suitable to be learnt by sequential classification methods such as C4.5, and P-type which are suitable to be learnt by parallel classification methods such as gradient descent artificial neural networks. These are best characterised by the following [Quinlan 1993b]:

At one extreme are *P-type tasks* where all the input variables are always relevant to the classification. ... At the other extreme are the *S-type tasks* in which the relevance of a particular input variable depends on the values of the other input variables.

Quinlan considers one specific example of each type, which are then used to compare symbolic and connectionist learning methods.

There have been attempts to characterise what sort of problem structures are difficult for different supervised artificial neural networks to learn, such as Lippmann's classification of network capability [Lippmann 1987]. This states, for example, that a two-layer network can only solve problems with convex open and closed decision regions. This has been shown to be false in particular contrived cases [Sjogaard 1991].

The majority of papers consider only single problems without any variation in the underlying structure. This is to be expected with problems taken from the real-world where a specific problem is considered, but it also often occurs with generated data [Lounis & Bisson 1991; Thrun, et al. 1991; Hickey 1992; Quinlan 1993b]. Many of these papers mention methods for generating further data sets, but again only specific examples are considered.

Note that the actual shape of the underlying problem structure is a vague concept which is based on the sample or data set selected from the population, and the attributes that have been measured. For example, the xor problem is simple to solve if different attributes are given — such as the number of true values modulo two. The measurements that have been made will often be only a small portion of the overall picture. An analogy may be drawn between the visible light and the rest of electromagnetic spectrum, which is there but invisible to humans. Thus, the underlying structure cannot be completely separated from how the problem is presented, which is considered next.

5.1.2 Factors affecting the data presentation

It is also important to consider factors which may affect the view of the population structure: the representation and data reliability. The inputs, outputs, examples and each example's values need to be considered (see figure 5.1). The inputs, and outputs outline the problem dimensions; whereas the actual examples selected from the population give the sampling dimensions [Rendell & Cho 1990].

Variations in any of these features will occur when information is missing or if extra information is available. Figure 5.2 draws a distinction, shown on two axes, between different types of information which may be gained or lost by the addition or removal of a data set feature: relevant information; irrelevant information; or a combination of the two leading to a continuum of attributes with differing predictive powers. Extra information may be redundant if full information required to solve the problem is available, meaning that it is duplicated information, or irrelevant, meaning that the information is not relevant to the learning task. A reduction in relevant information may lead to missing information necessary to solve the problem.

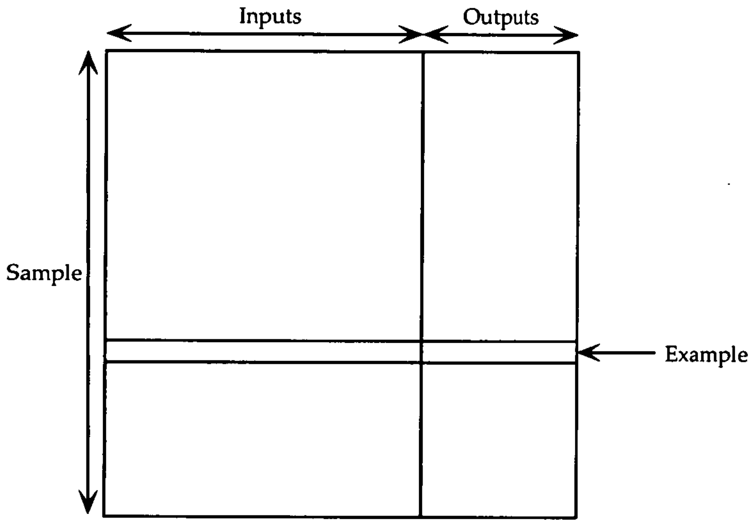

Figure 5.1 — Graphical view of a data set for supervised inductive learning, showing the features considered: the inputs, outputs, examples and an actual example

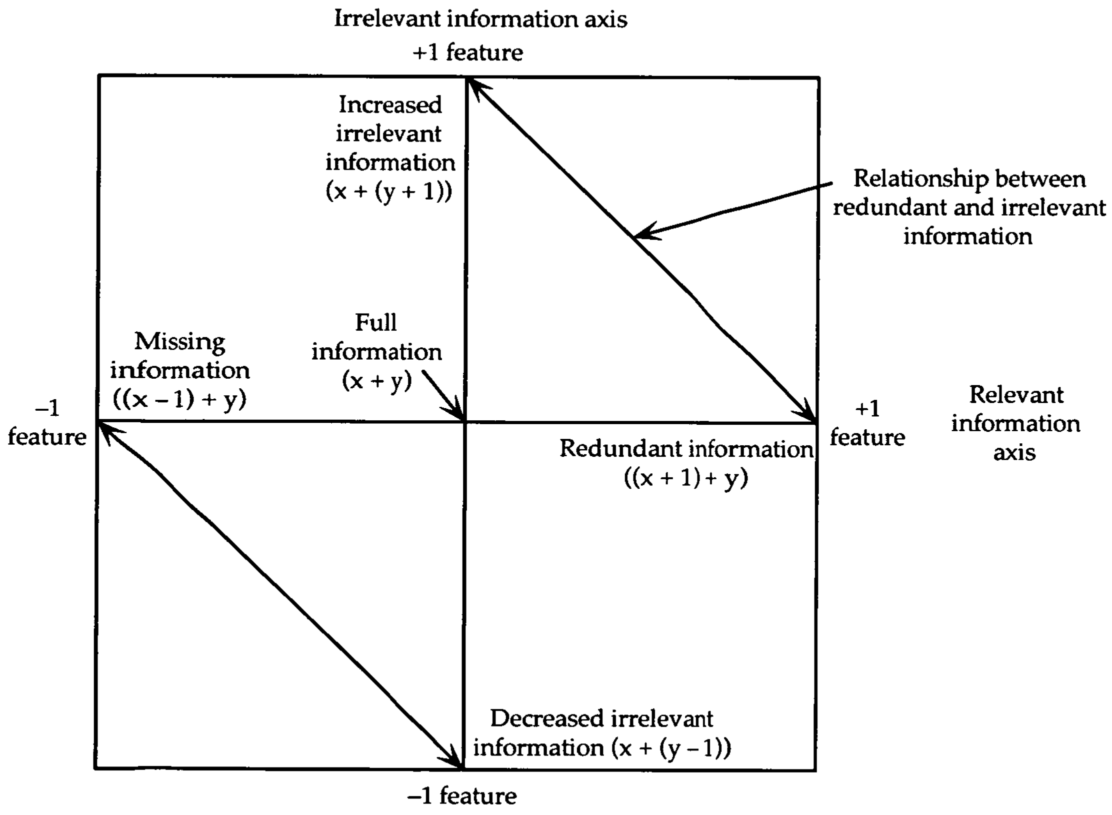
Irrelevant information axis
+1 feature
Increased irrelevant information (x + (y + 1))
Relationship between redundant and irrelevant information
Missing information ((x – 1) + y)
Full information (x + y)
–1 feature
+1 feature
Relevant information axis
Redundant information ((x + 1) + y)
Decreased irrelevant information (x + (y – 1))
–1 feature

Figure 5.2 — The relationship between needed information (shown on the x axis) which is redundant if enough information to solve the problem is available (as shown), and unnecessary information (shown on the y axis) given full information: the sum of x and y will give a whole number relating to the number of features under consideration

For example, if a new input is added to a data set which contains all of the inputs necessary to solve the problem, the input may be solely redundant information, solely irrelevant information or a combination of the two. Often this may show up as a poor or unrepresentative input — only partially measuring the features which are required to solve the problem — as not all the required information is available. Similarly, any reduction in the required inputs will mean that information necessary for the solution will be missing or that irrelevant information is removed. A large number of data sets may then be considered a combination of these factors: missing information necessary to fully describe the problem, but containing poorer quality information that partially contains the required information.

5.1.2.1 *Problem Dimensions*

Consider the variations that may occur with respect to the inputs as one part of the problem dimensions:

- number of inputs — dictates the size of the problem by specifying how features are being measured for the learning system;
- nominal / tree-structured / ordinal / interval inputs — the type of inputs is also important, as more information may be available from different styles;
- missing inputs — it is possible that not enough or not the right inputs have been selected for the data set to solve the problem;
- redundant inputs — a number of inputs may not be required because their information is duplicated by others; and
- irrelevant inputs — an input may not be required, but unlike a redundant input it contains no relevant or useful information.

Nominal data has no order: such as binary or enumerated types; tree-structures have a partial ordering; ordinal is ordered discrete values; and interval includes integer and real values giving extra complexity and extra information [Rendell & Cho 1990]. A large proportion of the generation methods shown in the literature rely only on integer [Rendell & Cho 1990] or nominal [Lounis & Bisson 1991; Thrun, et al. 1991; Hickey 1992] inputs.

The outputs are similar in structure and contain the following elements:

- number of outputs — also dictates the size of the problem, showing what is expected at the output; and
- nominal / tree-structured / ordinal / interval outputs — similar to the inputs, the outputs of a learning system may take on a number of different styles.

A problem with real outputs is a regression problem, whereas a classification problem entails nominal outputs. Ordinal or integer outputs may be considered to form constrained regression problems. Here the focus is on classification problems, thus regression problems are not considered, though many of the difficulties are the same. The majority of benchmarks concentrate on classification problems, with only a few regression data sets (for example, [Prechelt 1994a]). It is possible to consider missing, redundant or irrelevant outputs as well, but it is usual to consider only a single relevant problem at a time.

The problem dimensions form the structure of the problem: what information is available. The process of feature extraction gives the attribute space structure. Good feature selection will lead to a problem that is simple to solve using a linear discriminant function for example. Unfortunately finding such attributes is a difficult and error prone process. The final attributes which are used may require a more complicated classifier as the underlying problem structure is more convoluted, or even unsolvable given the known information.

5.1.2.2 *Sampling dimensions*

Now the actual examples that will be presented to the system are considered. The sampling dimensions do not effect the underlying problem structure, but they effect how well that structure may be learnt. Many of these considerations are the duals of those given above. However, they are distinct as they refer to individual examples with values across all inputs and outputs, rather than considering an input or output which has values across all examples. Variations in the entire data set are considered:

- number of training examples in sample — to train a system to recognise the underlying function, the particular function needs to be sampled enough to obtain the required information;
- redundant or irrelevant examples in the sample [Quinlan 1986a] — there is a problem that extra examples may be presented, which may either contain no information or misleading information, or they may be redundant resulting in biases toward one class or another; and
- missing examples from the sample — likewise there may be examples of important cases which may be missing from the sample.

An under-sampled problem will lead to poor generalisation as there are not enough examples to train the system properly. Over-sampling may, with some systems, lead to excessive training times, which is a lesser problem. A few papers examine changes in these sampling dimensions [Rendell & Cho 1990; Collier & Waugh 1994]. Redundant, irrelevant or missing examples point to problems in the sampling or measuring processes.

It is also important to outline differences between the example values which may occur:

- noise or errors within an example — the extra fluctuations in the measurements of all values when considering numeric values, or wrong nominal values which are not appropriate; and
- missing values within an example — due to a number of reasons a particular value may be missing: further separation of data may be possible given the

actual value, or there is a reduction in numerical results possibly leading to a smaller range of attribute values, and thus a weaker indicator relative to other measures.

These changes may occur to individual examples only, or may be alterations to the values of all examples. Noise [Quinlan 1986a] can have a number of sources, such as problems with the measuring equipment affecting all examples, or being an aberration affecting only a single example. In some cases, especially with binary or enumerated inputs, or even the actual classes, these fluctuations may lead to an erroneous value which does not reflect the example taken from the population.

It is preferable if the examples are selected independently, giving a proportional view of the entire population. Furthermore, the minimum classification rate expected is the proportion of examples in the largest class. If the largest class accounts for 95 percent of the data set, there is no point accepting any level of performance below this minimum.

All of these factors can affect the performance of a learning system in the development of a classifier or predictor. Extra information, in the form of redundant or irrelevant information, may bias the learning a system performs, as will missing information. Combinations of these cases may lead to considerable difficulty in learning a task. This is on top of the ability or biases of learning systems toward learning certain tasks, and will be considered next.

5.1.3 Inductive bias

Inductive bias is how a particular learning system, in learning a set problem, affects the final classifier produced. Not only do these biases stem from how learning methods cope with the underlying data set structure, but also from how different methods are affected by noise, missing values, irrelevant and redundant data and so on. If you are given a particular data set, a learning system will develop one solution over another on the basis of how that method works. There are two forms of inductive bias [Collier & Waugh 1994]:

- restricted hypothesis space bias — the possible theories which can be generated; and
- preference bias — each learning system generates theories in preference to others consistent with the training set.

The first form of inductive bias is important as it indicates what sort of data sets may be learnt. For example, C4.5 is biased in that it may only represent theories which involve Boolean combination of attributes in conjunctive normal form. C4.5 cannot combine several inputs to generate a classifier when this is required to solve a problem efficiently. Another example is standard back-propagation, which with only a single hidden layer and a restricted number of hidden nodes has difficulty in distinguishing between regions which curve around each other, such as the Two Spirals problem presented earlier [Fahlman & Lebiere 1989].

The second form of inductive bias is equally as important. Consider back-propagation: this learning algorithm may generate a large number of possible classifiers, each with the potential to solve a particular problem. However, frequently the final solution is biased, for example, by the initial random weights, often quite easily [Adams 1994]. Likewise the training algorithm is biased towards smaller weights as a by-product of minimising the total network error, and this may affect the generation of solutions.

There is a trade-off between the two different sources of bias which have been identified as the 'bias/variance dilemma' [Geman, Bienenstock & Doursat 1992]. Briefly, the more freedom a learning method has, the more variations are possible, requiring more training with a greater number of examples. If a learning method is restricted — such as a parametric statistical technique — then less examples and training are required, but the learning method is biased in what solutions it may develop, possibly resulting in a less suitable classifier.

5.2 Real-world and constructed data sets

From the above considerations of how problems are presented and how different learning systems may produce different results, it is worthwhile considering particular benchmarks and problems that have been presented throughout the literature.

There have been a number of studies into defining benchmarks for inductive learning. These often centre on either data sets from real-world problems, or upon constructed data sets where the domain has been created artificially. There is a difference in opinion as to which style of benchmarking provides more relevant results to those developing new learning methods. Some examples of each benchmarking style are considered in turn.

5.2.1 Constructed data set benchmarks

A number of artificial benchmarks for inductive learning tasks have been developed. The majority of data sets have been presented through the literature as single problems created mostly on an ad hoc basis, often for testing particular features of a learning method (for example, [Solla 1988; Fahlman & Lebiere 1989; Sjogaard 1991; Baluja & Fahlman 1994]). The Two Spirals data set [Fahlman & Lebiere 1989] is an example of this, as it is a difficult problem for artificial neural networks with sigmoid activation functions to solve. In this respect the benchmark is very good, hence the application of it in the first part of this thesis. However it is hindered in that the test set — and for that matter the training set — is unrealistic and does not test generalisation [Baluja & Fahlman 1994].

Famous collections of benchmark problems include Breiman et al. which includes problems such as recognising waveforms and LED displays with added noise [Breiman, et al. 1984]. One of the first artificial benchmarks for neural networks includes the parity, symmetry, encoder, T-C and addition problems [Rumelhart, et al. 1986].

An example of an artificial benchmark of several specific data sets developed for testing inductive learning systems is the Monks problems [Thrun, et al. 1991]. This suite consists of three data sets which are all binary classification tasks. The tasks are variations on the same input space which consists of six inputs with two to four possible values for each input. Hence the problems are completely enumerable with a total of 432 possible cases. In each data set only a limited number of these cases are available for training. The first problem is in disjunctive normal form, the second is similar to parity problems and the third is another disjunctive normal form problem with added noise. The aim of the benchmark is to describe the performance of a variety of learning algorithms on these standard problems, and thus provide a good comparison between the various methods.

Rendell and Cho develop a number of benchmark problems, generated to test various characteristics that they wished to examine [Rendell & Cho 1990]. These include variations in the attribute and class errors, the size of the concepts, the number of class peaks and their shape, the scales of the attributes and the number of training examples. The actual method of data set generation is outlined, though no examples are given, and only integer attribute values are used.

Lounis and Bisson also consider the generation of artificial benchmarks [Lounis & Bisson 1991]. They justify their usage by stating that with artificial benchmarks the availability of data is no longer a problem, translation is simple, interpretation of the results may be performed without an expert in a particular area, and that it is easier to answer questions such as 'what happens if the application domain is different?' Specifically they consider attribute value logic and predicate logic methods for the generation of concepts, concentrating on a single problem. Hickey also considers the benefits of generating artificial data [Hickey 1992]. The approach taken is to consider a specific problem in conjunctive form and model the introduction of noise to such a system. Both of these methods use only nominal attributes.

Further papers consider two problems involving overlapping Gaussian distributions [Kohonen, Chrisley & Barna 1988; Rögnvaldsson 1993]. The distributions have different standard deviations, and one problem has the same mean value for the distributions, whereas the other is offset in one dimension. The problem is to distinguish between the distributions which are described by two to eight continuous-valued attributes — leading to a total of 14 problems to compare various methods.

As mentioned previously Quinlan identifies two styles of problems: S-type and P-type, and uses real-valued attributes in the generation of problems [Quinlan 1993b]. Quinlan shows that decision-tree methods are unsuitable for P-type problems, and that artificial neural networks — specifically back-propagation — requires an inordinate amount of learning time for S-type problems. This shows the inductive bias of artificial neural networks favours solving P-type problems. Further work has been conducted in this area [Collier & Waugh 1994] which confirms Quinlan's findings, and extends these by showing that irrelevant — and, to a lesser extent, redundant — attributes adversely affect connectionist learning systems and noise affects symbolic learning systems. The work also shows that fewer training examples is more of a problem for symbolic methods than connectionist methods.

Quinlan's work could be extended to consider problems on a continuum whereby they are not S or P-type, but S and P-type to some degree [Collier 1995]. Adding irrelevant or redundant attributes influences problems to be more S-type in structure as those attributes are not necessary. On the other hand, adding noise or providing fewer training examples makes a problem more P-type-like in structure as individual attribute values are less reliable.

Not all are in favour of constructed data sets. Of the benchmarks developed in the 1980s Prechelt [Prechelt 1994a] states:

> all of these problems are purely synthetic and have strong a-priori regularities in their structure; for some of them it is unclear how to measure in a meaningful way the generalization capabilities of a network with respect to the problem; most problems can be solved 100% correct, which is untypical for realistic settings.

With respect to problems which have a stochastic element to their generation, two faults are identified:

> First, there is still the danger to prefer algorithms that happen to be biased towards the particular kind of data generation process used. ... Second, it is often unclear what parameters for the data generation process are representative of real problems in any particular domain.

Prechelt states that although generated data of a realistic nature has its place in the development of new algorithms, real data sets are preferred as the results produced will be applicable to at least a 'few' real domains [Prechelt 1994a].

5.2.2 Real-world data set benchmarks

A number of attempts have also been made at producing benchmark sets which include real-world problems. The first collection, which is the basis for most others, is the UCI Repository for Machine Learning databases [Murphy & Aha 1994]. This collection has no set structure and is made up of donated data sets from a large number of people with varying backgrounds. Though not an actual benchmark in itself, the databases contained there formed the basis of most other real-world benchmarks. The main reason for this is that it is simply too expensive and time consuming to develop new data sets.

Proben1 is a well constructed benchmark to use with artificial neural networks for benchmarking both classification and regression style methods [Prechelt 1994a]. It relies on a number of real-world problems available from the UCI Repository [Murphy & Aha 1994], using set encodings. The problems are presented in a consistent format which allows for easy and direct comparison between methods.

A further benchmarking suite has been developed using databases from the UCI Repository. Zheng's database collection may be used to benchmark classification methods [Zheng 1993]. The collection is an effort to cover the widest possible set of problem types by examining which benchmarks should be used, based on a number of measures. These are the type of attributes, the number of attributes, the number of different nominal attribute values, the number of irrelevant attributes, the data set size, the data set density, the level of noise in attribute values, the level of noise in class memberships, the frequency of missing values, the number of classes, the default accuracy, the entropy, the predictive accuracy, the relative accuracy, the average information score, and the relative information score. From all these factors 13 data sets were selected in the final benchmark. Note, though, that about five were generated or artificial in nature.

Lee and Lippmann also consider a combination of two artificial problems and two speech recognition tasks [Lee & Lippmann 1989]. The aim of their paper is to measure the performance of various pattern recognition algorithms with the following view in mind:

> A shortcoming of much recent neural network pattern classification research has been an overemphasis on back-propagation classifiers and a focus on classification error rate as the main measure of performance. This research often ignores the many alternative classifiers that have been developed ...

Waugh and Adams have also examined data sets for benchmarking neural networks [Waugh & Adams 1993]. A large number of the problems were again from the UCI Repository, and a number of the problems were constructed rather than naturally occurring. One of the main results of this work was that when using Cascor only one of the 14 data sets used required more than two hidden nodes to be installed to solve the problem: that being the Two Spirals problem.

The above result, along with previous experience of a large number of UCI databases not reported here, implies that the majority of real-world data sets rarely require any of the power of adding hidden nodes that Cascor possesses. A similar point of view is expressed by Holte using tree induction methods [Holte 1993]. Holte examines 16 UCI problems and concludes that frequently very simple classification rules perform almost as well as more complicated learning methods. He considered the development of single-level decision trees whereby a single attribute is used to split the data — all the data sets being considered involve binary classifications. He states:

> Of particular concern are the datasets. One does not intuitively expect "real" classification problems to be solved by very simple rules. Consequently, one may doubt if the datasets used in this study are "representative" of the datasets that actually arise in practice.

Not all concur with the above opinion. Elomaa argues that, although the prediction accuracy differences between the one-level decision trees and C4.5 are small, the differences are significant: 'High baseline success is achieved by simple means, but further advances require much more effort,' [Elomaa 1994]. Furthermore, it is stated that methods such as C4.5 are more robust in the solutions they generate.

Elomaa also makes some interesting points about the quality of real-world data that is available:

> It is essential to test machine learning approaches on data drawn from real-life in order not to lose sight of the real goal of our field. Nevertheless, manufactured data suits the purpose too: it is easier to control self-made data in the sense that monitoring the effects caused by changes in data, e.g., to prediction accuracy is easier. ... Many of the differences [with Holte's work] basically stem from the fact that we did not accept the UCI repository data sets to be representative of most typical application domains of decision tree learning. ... Holte has, rather, succeeded in proving that the current collection of standard test data for inductive learning is not up to its function.

Holte's paper is not inconsistent with these views.

5.3 Application of previous benchmarks

This section briefly outlines the application of some of the previously mentioned benchmarks to Cascor. The first point is to recount the result of the experiments performed in the first part of this thesis (see table 5.1).

Even without the application of the methods developed within Part I, it is easy to see the deficiencies with these benchmarks. As would be expected, the addition of hidden nodes increases training set performance. However, this is at the expense of test set performance in most cases, and in all cases a large amount of extra training has to be performed. Only two of the Monks problems, the Two Spirals and Double Helix data sets require the addition of hidden nodes given the training parameters used: the problems from the Proben1 benchmark — Cancer1, Diabetes1 and Glass1 — not requiring any such nodes. The first two Monks problems, which have an increased performance from the addition of hidden nodes, only require a single such feature detector. Finally the Two Spirals and Double Helix data sets, though they require the addition of large numbers of hidden nodes, have also been criticised as being extremely unrealistic.

Table 5.1 — Results from application of standard Cascor with and without hidden nodes to the benchmarking problems from Part I: the name of the data set, the training and test set performance, the number of hidden nodes required, and the number of connection crossings the training took (measured in millions) are given

Data set	Train %		Test %		Hidden		CCs (M)	
	Stand.	No hid.	Stand.	No hid.	Stand.	No hid.	Stand.	No hid.
Monks 1	100	84.68	97.69	75.23	1	0	4.5	0.64
Monks 2	100	63.31	99.7	62.27	1	0	5.8	0.67
Monks 3	100	94.26	88.89	96.76	2	0	16.1	0.73
Two Spirals	100	50	95.83	50	12	0	123.3	0.13
Double Helix	100	50	100	50	6	0	63.5	0.37
LED	76	75.15	71.8	72	25	0	4770.7	31.68
Cancer1	100	96	95.98	98.28	5	0	178.7	2.99
Diabetes1	98.48	77.6	68.49	77.08	25	0	1962.9	3.4
Glass1	100	70.81	66.04	66.04	17	0	407.7	3.13

Further results have been generated using Cascor on another benchmark for the comparison of the algorithm with back-propagation, Quickprop and C4.5 [Waugh & Adams 1993]. This examined some of the above problems, as well as others from the UCI Repository and one obtained from within the Department of Computer Science at the University of Tasmania. One common benchmark, the encoder problem [Rumelhart, et al. 1986], was considered within the group, but it is simply not suitable for testing Cascor as it requires the encoding of the inputs through a specified hidden layer for further decoding at the output layer. The architecture of Cascor does not allow the use of this problem. Of the nine further problems — discounting those considered above — none required the addition of more than two hidden nodes by Cascor. Comparisons were possible on the problems, but there seemed to be no difficulty in solving the presented tasks.

Further studies of the performance of Cascor have also concentrated on specific problems. One considers character recognition on an eight by eight grid [Hamamoto, Kamruzzaman & Kumagai 1992], another further problems from UCI [Yang 1991], and a further paper which examines three artificial data sets and one well known real-world data set: majority7, parity6 and Mackey-Glass, and heart disease [Squires & Shavlik 1991]. The following comment is made [Hamamoto, et al. 1992]:

> This task is made difficult by the non-availability of non-proprietary data sets from real-world domains that are complex enough to adequately challenge generative learning algorithms (note that all the data sets used in this study required the generation of relatively small numbers of hidden units).

5.4 Summary

The ability to generalise to unseen cases is very much problem dependent, as well as learning system dependent. If there is no useful information in the data set which may guide a classification system, then there is no way that any method can produce a good result. Likewise the final result is biased by the learning system — symbolic, statistical and artificial neural network methods all learn in different ways.

It is possible to identify causes of differences between data sets, splitting these into the underlying structure of the data set and factors which affect the measurement of those underlying structures. Having done so, it is easy to see that it is difficult to produce some sort of benchmarking suite which covers this entire area, though reasonable attempts have been made.

People working in the development of new learning methods require specific ways of sensibly comparing their methods against other established techniques, involving both artificial and real-world problems. The rest of this thesis examines new real-world problems and methods of creating data sets for benchmarking, specifically in respect to Cascor.

6 Real-world data sets — two new examples

In this chapter two new real-world data sets are examined. The purpose of this process is to examine new data using Cascor as one of the tools, and to see if either of the new data sets contains features which require the use of hidden nodes. Given the complexity of previously examined real-world data sets, the chances of finding a problem which is solvable by the introduction of hidden nodes is unlikely, but this still needs to be considered.

6.1 Example one — ageing abalone

Abalone shellfish are a major industry in Tasmania. Sales of abalone are worth millions, as are commercial licences to catch the shellfish. The Marine Research Laboratories of the Tasmanian State Government Department of Primary Industry and Fisheries have an ongoing research interest in managing the fishery stock. Part of this research involves the catching and measuring of large numbers of the shellfish for analysis [Nash, Sellers, Talbot, Cawthorn & Ford 1994]. However, determining the age of the abalone is relatively time consuming, and hence expensive. The aim here is to develop a classification system which will give a reasonable estimate of the abalone age from the other measured attributes of each shellfish.

To this end, data from abalone captured in two regions of the state are examined. This data is generously provided by the Marine Research Laboratories. The differences in the regions are due mainly to the type of abalone captured: the first region, Bass Strait, contains a large number of samples which have stunted growth patterns; the second region, St Helens, contains samples which are predominantly fast growing.

6.1.1 Initial data preparation

For each example supplied by the Marine Research Laboratories the following information is assessed:

- area — the area of collection within the region (string containing name);
- site — number of the actual site (integer value of site);
- sex — the sex of the abalone: male, female, infant or trematode (nominal value);
- length — the length of the abalone (in millimetres);
- diameter — basically the width of the abalone (in millimetres);
- height — height of abalone (in millimetres);
- whole weight — the weight of the abalone after capture (in grams);
- shucked weight — the weight of the abalone meat (grams);

- viscera weight — the weight of the gut, this is after the abalone has been bleeding, and hence the weights do not total (in grams);
- shell weight — weight of the dried shell, the shells being porous can otherwise carry a lot of water (in grams);
- rings — number of rings through the abalone shell; and
- age — the number of rings plus 1.5, as determined by previous experiments.

The number of rings give the age of the abalone. The shell needs to be dried, cut, stained and the rings counted under a microscope — the process takes around five minutes per shell. This is the most expensive part of the information gathering, and hence the target for the classification. Only discrete ages are then available, hence the choice of using a classification system as opposed to a regression network.

The area and site information are ignored, as a classifier which will work for any abalone caught in Tasmanian waters is preferred. The age is also ignored as this is a simple calculation from the number of rings in the abalone shell — no example has an age without the ring information being present. Thus the problem involves eight attributes, seven of which are continuous numeric values and one of which, the sex, is an enumerated variable with four values; and the result of the classifier is the number of rings in the abalone shell.

Table 6.1 details the structure of the data set. This indicates that very few of the samples which contain missing values can be used as training vectors for the classifier. 8233 examples are available, and 4203 have no missing values. Of those examples with missing values, 495 could be used for training given some form of input encoding for a neural network [Vamplew & Adams 1991], as with the rest the number of rings is the missing value. This is due to a large proportion of the shells from the Bass Strait area being damaged by natural causes to the extent that the number of rings cannot be determined. No examples with missing values are used in the experiments.

Table 6.1 — Numbers of examples and their breakdown

	Bass Strait	St Helens	Total data
Total samples	4754	3479	8233
No missing values	1621	2582	4203
Missing values	3133	897	4030
Missing rings/age	2921	614	3535
Missing other values	212	283	495

The next point to consider is the distribution of the classes. Table 6.2 outlines this information for the examples without missing values, detailing the number of examples in each age group for the different regions. From this information it was decided to examine the data in three ways: the first being trying to classify all of the examples with their given class; the second involving grouping the data into three new classes; and the third trying to classify the four classes with the most examples, namely 8, 9, 10 and 11 rings. The three new classes of the grouped data set are created by collecting examples with 1 to 8 rings in class one; examples with 9 or 10 rings in class two; and examples with greater than 10 rings in class three. Though it may be possible to separate all of the examples, this is unlikely due to how few examples in some classes are available for training. Hence the development of the two extra data sets.

Table 6.2 — Number of examples in each ring group, with the shaded region shows the examples used in the restricted data, and the borders indicating the divisions of the grouped data ('N/A' is used to indicate not applicable cases throughout this chapter)

Number of Rings	Bass Strait	St Helens	Total
1	1	N/A	1
2	1	N/A	1
3	8	7	15
4	24	33	57
5	49	67	116
6	60	199	259
7	96	297	393
8	101	471	572
9	165	530	695
10	185	451	636
11	155	336	491
12	153	116	269
13	155	50	205
14	113	15	128
15	97	6	103
16	66	1	67
17	55	3	58
18	42	N/A	42
19	33	N/A	33
20	26	N/A	26
21	14	N/A	14
22	6	N/A	6
23	9	N/A	9
24	2	N/A	2
25	1	N/A	1
26	1	N/A	1
27	2	N/A	2
29	1	N/A	1
Total	1621	2582	4203

Note that these data sets could naturally be translated as regression problems given that the classifications used here are only discrete versions of the continuous age of the abalone. However this does not preclude the examination of these particular data sets, which will indicate whether the underlying structure may be solved as a regression problem.

The majority of the attributes are numeric, so it is important to get an idea of the range of results (see table 6.3). Since the ranges of the data are so large — even discounting some obvious errors where an abalone would have to be twice as high as it is long — the examples are scaled, simply by dividing by 200. This gives the attributes small ranges which may be handled by artificial neural networks more easily, as there are no extremely large values which force the activation functions to be hard on or hard off.

Table 6.3 — Minimum and maximum ranges of attributes over examples with no missing values

	Bass Strait		St Helens		Total	
Attribute	Min	Max	Min	Max	Min	Max
Length	15	160	27	163	15	163
Diameter	11	126	21	130	11	130
Height	2	50	0	226	0	226
Whole weight	0.4	510	2.8	565.1	0.4	565.1
Shucked weight	0.2	214.1	1.1	297.6	0.2	297.6
Viscera weight	0.1	118	0.1	152	0.1	152
Shell weight	0.3	201	0.8	179.4	0.3	201

The only non-numeric attribute is sex (see table 6.4). Even here there is a bias with very few of the examples being trematodes — animals which have been de-sexed. These examples have been removed.

Table 6.4 — Totals of each sex over samples with no missing values

Sex	Bass Strait	St Helens	Total
Male	653	875	1528
Female	624	683	1307
Infant	340	1002	1342
Trematode	4	22	26
Total	1621	2582	4203

To summarise, the missing value examples have been removed, as have the trematode examples. Six data sets are considered, three distinct problems both scaled and unscaled: the complete class data; the grouped class data; and the restricted class data. In each case the data set has been created by randomly selecting three quarters of the examples to create a training set and one quarter to be an unseen test set. The number of examples in each of the training and test sets is outlined in table 6.5, along with the minimum expected percentage based on the largest class (scaled and unscaled data sets have the same characteristics).

It is also worth checking the correlations between the attributes (see table 6.6). This information shows that a number of the attributes may be redundant, for example the length and diameter are very closely related. It also shows that a single attribute may not be used to solve the problem, and that combinations of attributes may be required. The correlation between the number of rings and the other attributes does not suggest that the task may be simply solved. There is most likely redundant information being presented here, but there is no point removing information for this initial study even though this may lead to less non-linear features being evident in the data.

Table 6.5 — Details of the training and test set sizes of the data sets extracted from the Abalone data, along with the minimum required percentage correct calculated by the percentage of the largest class

Data Set	Training Set		Test Set		Totals
	Size	Min	Size	Min	
Full data	3133	16.16%	1044	16%	4177
Grouped data	3133	34.34%	1044	37.07%	4177
Restricted data	1783	28.77%	595	29.58%	2378

Table 6.6 — Correlations between attributes calculated from all data examples

	Diameter	Height	Whole	Shucked	Viscera	Shell	Rings
Length	0.9868	0.8276	0.9253	0.8979	0.903	0.8977	0.5567
Diameter	N/A	0.8337	0.9255	0.8932	0.8997	0.9053	0.5747
Height	N/A	N/A	0.8192	0.775	0.7983	0.8173	0.5575
Whole	N/A	N/A	N/A	0.9694	0.9664	0.9554	0.5404
Shucked	N/A	N/A	N/A	N/A	0.932	0.8826	0.4209
Viscera	N/A	N/A	N/A	N/A	N/A	0.9077	0.5038
Shell	N/A	N/A	N/A	N/A	N/A	N/A	0.6276

6.1.2 No hidden nodes

A number of experiments on the six data sets using Cascor have been conducted. Cascor was trained, using the standard parameters outlined in Part I, to classify the problems. Two further restrictions are enforced: no hidden nodes are installed and the training of the output layer is restricted to 100 epochs (see table 6.7).

Table 6.7 — Results of using Cascor to build classifiers on the six Abalone data sets: 100 clock-seeded trials, giving the median (upper result) and the interquartile range (lower result)

Abalone Data Set	Training %	Test %	Connections	CC(M)
Full, Unscaled	13.50	13.41	308	194
	5.94	7.76	0	0
Full, Scaled	27.18	24.86	308	194
	0.45	0.86	0	0
Grouped, Unscaled	44.69	43.53	33	20.8
	20.71	23.18	0	0
Grouped, Scaled	64.28	61.40	33	20.8
	0.22	0.29	0	0
Restricted, Unscaled	28.77	29.58	44	15.8
	5.16	5.21	0	0
Restricted, Scaled	39.99	37.98	44	15.6
	0.79	0.67	0	1.1

These results indicate that the performance on the scaled data is much greater than the unscaled data. The unscaled data is not doing much better than chance — the high range of the attribute values prevents the network from learning. If the network weights are generated in the bounds of 1 and –1, as in this case, then large input values will in turn give a large value to the squashing function input, resulting in near extreme values for the squashing function output. It has been noted previously [Fahlman 1988a] that this causes very slow learning, due to the slope near extreme values of the activation function being close to zero. This leads to the changes to the weights being very small, as they are proportional to the slope. In fact, in this case, on the full data the performance has dropped below that obtained by selecting the largest class. In comparison, the results on the scaled data are a third to twice the minimum performance level.

The results also indicate that the unscaled data is much more unstable, with the interquartile ranges showing a far larger spread of results. The error on the training set for one trial on each of the restricted data sets is traced for both the scaled and unscaled data (see figure 6.1), and it is obvious from this that the scaled data is more stable to the point that training ceases early due to the lack of patience. From now on only the scaled data will be considered.

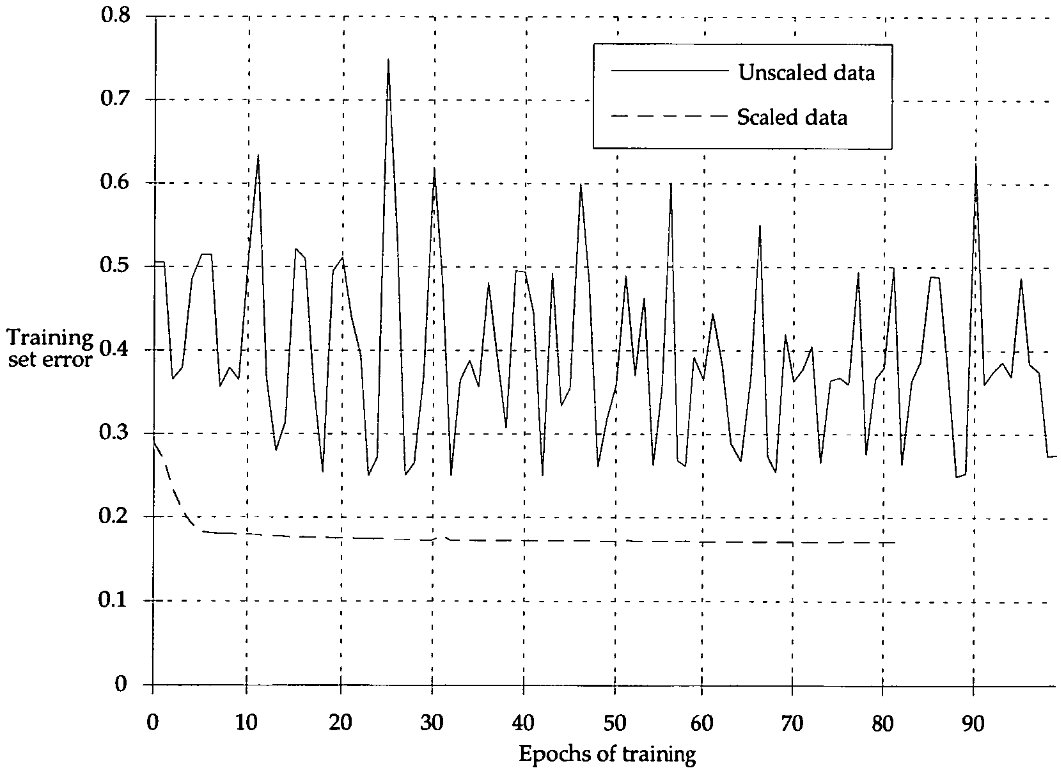

Figure 6.1 — Training error measured against time (epochs) for a single trial of the restricted data sets

6.1.3 Hidden nodes

The performance without hidden nodes, even on the scaled data, is not as good as is required, with the highest performance around 60 percent correct on the more general grouped data. The next trials to examine are the introduction of hidden nodes within Cascor to pick up any non-linear features in the data sets. Trials have been conducted on the scaled data sets, allowing up to 10 hidden nodes to be added whilst using independent candidate training and node patience: 3 percent change over a single node period. Candidate training is restricted to 200 epochs and output layer training is restricted to 100 epochs (see table 6.8).

Table 6.8 — Results on the three scaled Abalone data sets applying node patience (3%, 1 node), giving the median (upper result) and the interquartile range (lower result) over 100 clock-seeded trials

Data Set	Training %	Test %	Hidden	Connections	CC(M)
Full	27.66	24.90	1	347	491.0
	0.48	0.77	0	0	32.9
Grouped	66.36	64.85	2	62	296.9
	1.21	1.34	1	15	155.5
Restricted	41.00	39.24	1	59	100.2
	0.73	2.35	0	0	3.8

Comparing these results against the networks without hidden nodes (see table 6.7) is not encouraging. The performance of these single trials is not that much better, indicating that the problem involves overlapping classes and only minor improvement may be made by adding hidden nodes. This result is verified by further trials which forced the usage of five hidden nodes, although the generalisation performance is slightly better (see table 6.9), a difference which is accounted for by the simplistic application of node patience. No further improvement is made by introducing up to 20 hidden nodes.

Table 6.9 — Results on the three scaled Abalone data sets installing 5 hidden nodes, giving the median (upper result) and the interquartile range (lower result) over 100 clock-seeded trials

Data Set	Training %	Test %	Hidden	Connections	CC(M)
Full	29.36	26.25	5	513	1631.3
	0.57	1.25	0	0	105.4
Grouped	67.22	65.61	5	113	851.3
	0.41	0.77	0	0	30.4
Restricted	43.07	39.33	5	129	502.5
	0.84	1.34	0	0	17.8

6.1.4 Optimal Performance

It is preferable at this point to show that no possible further improvement may be achieved by the addition of hidden nodes. Techniques are available for estimating the optimal performance of any learning method on any problem — thus the performance which may be achieved if an unlimited amount of data is available [Cortes, et al. 1995]. The methods were developed to examine problems where 'the data collection was not designed for the task at hand and prove inadequate for constructing high performance classifiers.' This is applicable to this problem.

The independent variables in this process are the capacity of the classifier and the number of learning examples. The capacity may be roughly defined as the power of the learning system to model the data, and in an artificial neural network the capacity is related to the number of free parameters — namely the number of weights and layers. In a standard back-propagation network the capacity is fixed, whereas in a Cascor network the capacity is varied by adding more hidden nodes with the resulting connections.

Briefly, increasing the capacity of a learning system, given a certain amount of data, produces a distinctive pattern: the training error continues to drop as more capacity is allowed, whilst the test set error initially drops and subsequently increases due to the overtraining allowed by the increased capacity resulting in the memorisation of the training set. Thus the increase in capacity causes the following stages to be met: undertraining, a good capacity for modelling the data, and then overtraining.

Furthermore, if a learning system of a fixed capacity is trained with increasing numbers of examples in the training set, then the training set error increases as more examples need to be replicated, and the test set error decreases as the classifier becomes more robust. This means, given an infinite number of training examples, that the training and test set errors converge toward the predicted error for that capacity: the asymptotic error E_∞. This may be shown by averaging the training and test set errors, and extrapolating the error limits.

The combination of these two features of learning means that the intrinsic noise level of the data set may be determined, giving an estimate of the optimal performance of any learning algorithm which may be obtained from particular data given an unlimited number of examples. This may be achieved by plotting the asymptotic error rate against the change in capacity. The curve that plotting the error rate follows is limited from below by the intrinsic noise level.

This method is of particular interest here as Cascor allows the generation of results regarding different capacities from a single trial. By saving the performance results and error after the installation of increasing numbers of hidden nodes, it is possible to generate the required results for a single data set size, without requiring the retraining of another network or other learning system.

Figure 6.2 shows the results of a single trial on the grouped data with up to 100 hidden nodes installed, measuring the mean squared error on both the training and test sets. This indicates that the optimal performance is achieved after the introduction of only a few hidden nodes. Although this single trial does not show all the asymptotic error rates, the errors may be considered to approximate the asymptotic error rates, and they indicate that no further improvement may be expected after this point.

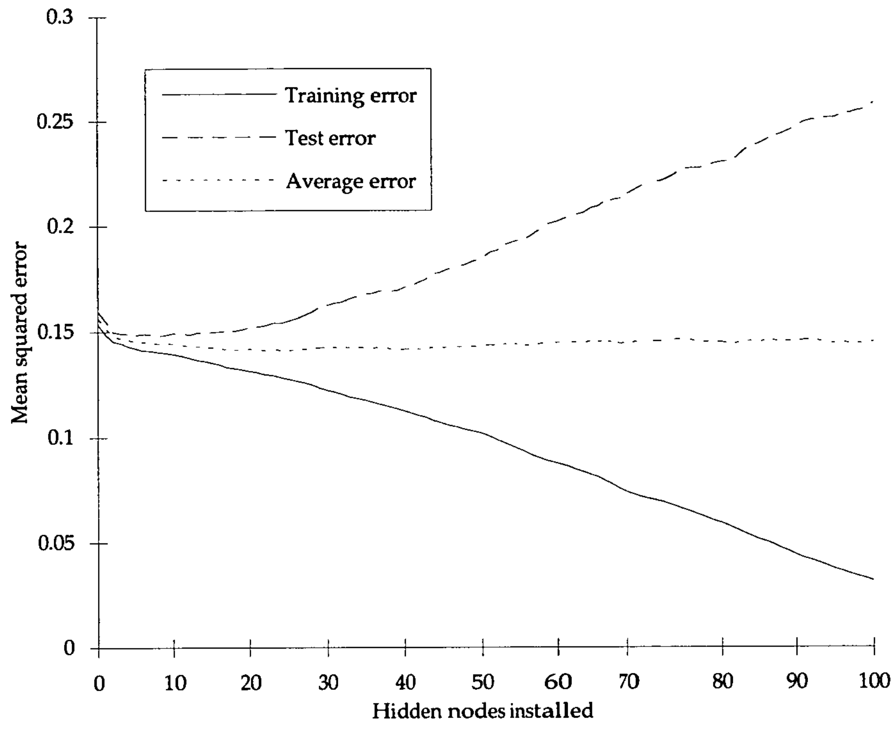

Figure 6.2 — Errors on a single trial of the grouped data after the introduction of each hidden node, up to a total of 100 hidden nodes

The results of measuring the classification performance rather than the mean squared error give similar results, though slightly rougher in nature given that the network is trained on the error rather than the percentage correct.

6.1.5 Confusion matrices

How the examples in the data sets are being separated may be checked by the closer examination of single trials. Confusion matrices [Weiss & Kulikowski 1991] are produced for the grouped, restricted and full data from single trials (see tables 6.10, 6.11 and 6.12 respectively). Although the results are reasonably spread, there is a considerable amount of overlap. The grouped data shows reasonably good selection of classes one and three, but the performance on class two is poorer. This may be due to the small range of samples in the second class. The results on the restricted data shows that it is difficult for adjacent classes to be distinguished. The full data also indicates the problem of overlapping classes, though a definite trend in training is evident.

Table 6.10 — Final training and test set confusion matrices for a single trial on the grouped data: columns show predicted values (shown by labels across table), rows the actual class of the examples

	Training			Test		
	Class 1	Class 2	Class 3	Class 1	Class 2	Class 3
Class 1	858	171	47	259	49	23
Class 2	253	376	368	85	105	136
Class 3	96	187	777	40	76	271

Table 6.11 — Final training and test data set confusion matrices for a single trial on the restricted data: columns show the predicted values, rows the actual class of the examples

	Training				Test			
	8 rings	9 rings	10 rings	11 rings	8 rings	9 rings	10 rings	11 rings
8 rings	226	145	50	9	74	46	16	2
9 rings	152	187	133	41	55	62	49	10
10 rings	80	135	178	88	29	36	56	32
11 rings	41	73	130	115	27	21	47	33

Table 6.12 — Final test set confusion matrix for a single trial on the full data: columns show the predicted values, rows the actual class of the examples, blank cells contain no examples, bold numbers show the correct examples

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
3			2	1											
4		1	**6**	4	2	1									
5			3	**3**	6	12									
6	1			4	**8**	46	4	6							
7				1	2	**37**	15	24	2						
8					2	25	**52**	48	13						
9						14	34	**68**	50	1					
10						5	23	45	**82**	4					
11						4	16	38	72	**9**	1				
12						2	4	30	27	3	**1**				
13							6	12	37	4					
14							3	8	20	1		1			
15							2	10	12	2					
16							1	7	10		2	2			
17							2	2	5	1	1				
18								1	9			1			1
19							1		6	1					
20								2	2					1	
21									1						1
22															
23															
24											1	1			

6.1.6 Pruning

Pruning will not necessarily produce a better classifier, but it may result in a much smaller network, and would indicate that a number of the attributes are not required. Simple trials are conducted on solving the problems using the output layer only, using absolute pruning to firstly remove connections with saliencies below 0.0, and secondly removing connections with saliencies below 0.05 (see table 6.13). As expected, the addition of pruning does not greatly improve the performance of the classifier, although there is a performance increase with all problems. However it does show that a reasonable proportion of the connections are not required at all, as they are simply removed by low level pruning. This in turn identifies many of the inputs that are not required, which is supported by the correlation information (see table 6.6). The increased level of pruning does not reduce the number of connections further as these, according to the saliency measure, are required to solve the problem.

Table 6.13 — Results of pruning experiments, showing the data set, the pruning level, and median and interquartile results for the training and test set percentage correct, the number of connections, the maximum number of possible connections and the number of connection crossings (in millions)

Problem	Pruning	Train	Test	Conns	Maximum	CCs (M)
Full	0.0	27.51	24.90	273	308	304.2
		0.41	0.77	6	N/A	46.4
	0.05	27.61	24.90	273	308	313.6
		0.51	0.86	8	N/A	53.6
Grouped	0.0	64.60	62.07	26	33	34.5
		0.35	0.86	2	N/A	5.5
	0.05	64.67	62.07	26	33	34.4
		0.29	0.96	2	N/A	5.6
Restricted	0.0	40.38	38.32	33	44	22.4
		0.62	1.68	3	N/A	1.3
	0.05	40.38	38.40	32	44	22.1
		0.5	2.35	3	N/A	1.5

By counting which connections are pruned it is possible to get an idea of which connections are actually important. Figure 6.3 demonstrates this by showing which attributes were used the most over 100 trials of the full data — giving percentage usage on all possible (2800) connections per attribute. This pattern is repeated for the other data sets. It seems that the abalone length, and shucked and shell weight are the most important indicators in forming the classifier, although no attribute stands out as being completely redundant.

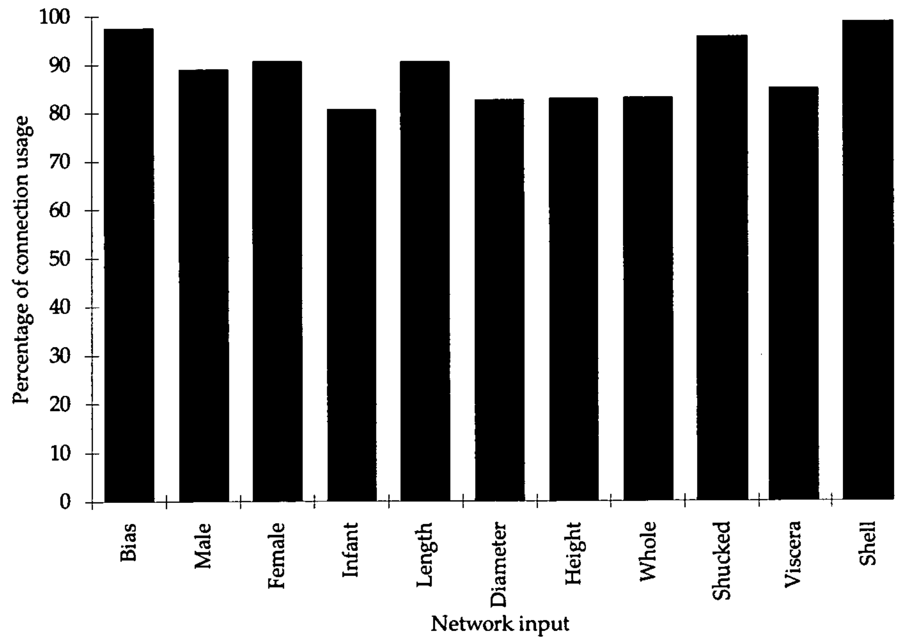

Figure 6.3 — Percentage usage of connections over 100 solutions to the full abalone data

6.1.7 Other classification methods

One decision tree method and two statistical methods are used to classify the normalised data sets: C4.5 (see table 6.14), and linear discriminant analysis (LDA) and (k = 5) nearest neighbour (5-NN) (see table 6.15). For reference the results of applying Cascor without hidden nodes to the data sets is also given (see table 6.14).

Table 6.14 — Results of trials using C4.5 and Cascor (previously generated): the training and test set percentages correct and the number of nodes in the tree for C4.5 are shown

	C4.5			Cascor — no hidden nodes	
Data set	Training set	Test set	Nodes	Training set	Test set
Full	76.6	21.5	1817	27.18	24.86
Grouped	89.3	59.2	874	64.28	61.40
Restricted	83.1	30.8	862	39.99	37.98

The performance of C4.5 on the data sets is not as good as Cascor, although the results are comparable. The training set performance is much higher indicating a great deal of over-specialisation which is irrelevant to further unseen cases. This is also evident from the large sizes of the final trees produced. C4.5 must not be sold short in that it does have a restricted hypothesis space and the speed of the actual learning is faster than that of Cascor, with no requirements for setting any training parameters. The process is deterministic, thus requiring only a single trial. The performance of C4.5 is not affected by whether the data has been scaled or not.

Table 6.15 — LDA and 5-NN trials results: training and test set performances on the normalised data

Data set	LDA		5-NN	
	Training set	Test set	Training set	Test set
Full	0.03	0.0	7.14	3.57
Group	33.61	32.57	90.33	62.46
Restricted	26.36	26.22	82.29	35.93

The performance of LDA on the abalone data is not good. For whatever reason the results are well below those of Cascor and C4.5, though the result on the restricted data set is comparable. The large spread of examples in the full and grouped data appear to cause problems for LDA in learning.

The performance of 5-NN is also poor, especially on the full data set where there are a large number of classes — a relatively large proportion of the classes containing less than five examples for training. The bias of the algorithm is such that the performance on all the data sets is less than the results from Cascor: the overlapping data seems to degrade the performance of nearest neighbour. This is supported by the performance of 5-NN on the grouped data, which performs better than Cascor without hidden nodes. Although Cascor with hidden nodes out performs the level achieved by 5-NN, the nature of the grouped data

— where possibly overlapping ring groups have been placed in the one class — results in a better level of performance from 5-NN.

From this brief examination of non-neural methods, it seems that a better level of performance may not be obtainable. This supports the results gained by checking the optimal performance using Cascor. This must be taken in the context that the statistical methods examined are very simple — better results may be obtained from more sophisticated methods.

6.1.8 Summary

A new problem has been examined, and it has been shown that the maximum performance on this data set is achievable with Cascor. However, the problem is not solvable from the information available, and the requirement for the addition of hidden nodes is limited. This may be due to the duplication in the data sets of related attributes, as well as the unreliability of prediction masking non-linear features. Nevertheless, the performance of Cascor is higher than the other methods examined. The problem is useful for testing basic learning performance, as well as being a problem of interest in its own right.

Of course this data is not ideally suited for analysis as classification tasks. The measure of the number of rings giving the classification may also be translated as a function approximation problem. However, a classification problem is as equally valid as the number of rings is divided into discrete values. Nevertheless, there is the indication that the problem being examined has a great deal of overlap between the classes. The results on the grouped data and from the confusion matrix examples indicate that it is possible to get estimates of the number of rings from the other attributes, however exact matching is not possible.

Further information is required to obtain a useful classifier, as the information available is not sufficient to perform the necessary classification. For example information on the site of where the abalone was captured may provide the required information. This may be generalised to information such as, for example, whether the abalone grew in an area exposed to colder ocean water — a factor quite important to the abalone growth rates. In a further trials, site information was included and used to train Cascor networks, resulting in improvements of up to five percent in the classification performance as expected. Thus such information in a more general form would be invaluable for further work.

6.2 Example two — identifying authors

In further attempt to examine data which is difficult in nature, examples of text word frequencies were generously provided by the University of Newcastle Centre for Literacy and Linguistic Computing (CLLC) for the purpose of distinguishing between Renaissance and Romantic tragedy authors [Burrows & Craig 1994]. The reason why this may be a difficult problem is that the attribute information is based on word counts from passages of text, but the classification is based on the authors of the text passages.

Previous work has been conducted on the stylometry identification of authors using artificial neural network methods [Matthews & Merriam 1993; Merriam & Matthews 1994; Singh & Tweedie 1995], however this is the first to use a topology changing algorithm such as Cascor. The previous work has not considered whether it is possible to solve such problems using only a single layer of weights, which is addressed by the application of Cascor.

6.2.1 Details of author data

Each example is a section from one of a number of plays (see table 6.16): text blocks of close to 2000 words are used and the most frequent words — from throughout all the passages — within each block are counted (see table 6.17). From the selected plays there are 188 examples, classed as being Romantic (80 examples) or Renaissance (108 examples), with 100 attributes each representing one of the most frequently used words. Thus each attribute value is the number of occurrences of a particular word within the corresponding example.

Table 6.16 — Plays used for analysis, giving the author(s), the name of the play, the number of blocks of text extracted, and the number of words in each block: the first ten are from the Renaissance era, and the second ten are Romantic plays

Author	Play	Samples	Words per sample
Kyd	The Spanish Tragedy	10	9 by 2000, 1 by 2773
Shakespeare	Hamlet	14	13 by 2000, 1 by 3218
	Macbeth	8	7 by 2000, 1 by 2674
	King Lear	12	11 by 2000, 1 by 2752
	Othello	12	11 by 2000, 1 by 2895
Middleton	Women Beware Women	13	12 by 2000, 1 by 1925
	Hengist	10	9 by 2000, 1 by 3455
Middleton/Rowley	The Changeling	9	8 by 2000, 1 by 2498
Rowley	All's Lost by Lust	8	7 by 2000, 1 by 2491
Webster	Duchess of Malfi	12	11 by 2000, 1 by 1744
Scott	The House of Aspen	6	5 by 2000, 1 by 2365
	Auchindrane	7	6 by 2000, 1 by 3244
	Halidon Hill	5	4 by 2000, 1 by 1980
Byron	Marino Faliero	14	13 by 2000, 1 by 2281
	Manfred	5	4 by 2000, 1 by 2296
	Werner	13	12 by 2000, 1 by 2777
Shelley	The Cenci	9	8 by 2000, 1 by 2913
Coleridge	Osorio	7	6 by 2000, 1 by 2700
Keats	Otho	7	6 by 2000, 1 by 2349
Sheridan	Pizarro	7	6 by 2000, 1 by 3549

Table 6.17 — The list of the 100 most common words in descending order of frequency of the 10 Renaissance and 10 Romantic tragedies (from left to right and down the table)

the	and	I	of	a
you	is	my	it	in (preposition)
not	to (infinitive)	to (preposition)	me	but
be	have	with	he	this
will (verb/modal)	his	your	for (preposition)	as
thou	what	him	all	are
that (demonstrative pronoun)	thy	now	if	that (relative pronoun)
do	that (conjunction)	thee	we (not royal plural)	shall
then	from	by (preposition)	which (relative pronoun)	was
or	no (adjective)	would	they	on (preposition)
at	our (not royal plural)	there	can	o
more	must	their	am	lord
she	here	her (adjective)	them	so (adverb of degree)
when	one	yet	how	let
know	upon (preposition)	were	may	sir
well	had	such	should	come
so (adverb of manner)	good	see	who (relative pronoun)	man
an	her (pronoun)	some	us (not royal plural)	for (conjunction)
too	these	why	like (preposition)	has
make	where	say	love	life

Two different forms of the data were provided by the CLLC: the raw data which simply contains the word counts of the most frequent words for each play section, and a normalised data set. The normalisation process involved taking the raw data, dividing each example's attribute values by the total number of words in that example and turning each attribute value into a percentage of the total number of words. This standardises the examples by removing the number of words in the block as a factor removing irrelevant information.

The immediate concern with this data is that there are very few examples available to train a classification system given that there are 100 different attributes. This is particularly a concern with artificial neural networks given that the number of parameters to be estimated within the classifier is proportional to the number of attributes and the number of hidden nodes. On this basis the performance of any classifier trained on this data when confronted with new examples may be doubtful. Further, the number of examples may lead to a biased estimate of generalisation ability given the small number available for training and later testing. Thus the first experiments performed will use the full data for training: examining the need for hidden nodes and pruning within Cascor, and giving the apparent error rate calculated from the training set [Weiss & Kulikowski 1991].

6.2.2 Full data Cascade-Correlation experiments

Two data sets are examined, the original raw data and the normalised data, and are used to train classifiers using Cascor with pruning at two levels (0.0 and 0.05 absolute level pruning) and without pruning with all the examples being used for training (see table 6.18). For all of the 100 randomly seeded trials on each data set and training method, hidden nodes were not required in developing the final classifiers.

Table 6.18 — Results of training Cascor using the raw and normalised data, showing the median (upper) and interquartile range (lower) for the percentage correct on the training set, the number of connections (maximum of 202), and the number of connection crossings of training

Technique and Data Set	Training %	Connections	CCs (M)
Cascor	100	202	14.5
Raw	0	0	10.6
Cascor	100	202	3.0
Normalised	0	0	0.3
Cascor, pruning 0.0	100	136	15.4
Raw	0	22	9.9
Cascor, pruning 0.0	100	119	3.4
Normalised	0	8	0.5
Cascor, pruning 0.05	100	138	15.7
Raw	0	19	10.2
Cascor, pruning 0.05	100	115.5)	3.4
Normalised	0	9	0.4

These results demonstrate a number of interesting points. The training set, when all the data is used to train a network, appears linearly separable. This does not seem to be a difficult problem, the only difference between trials being to the random starting points which leads to a great variation in the training times.

The performance on the normalised data is much more stable. As is expected reducing the size of the attribute values and removing the reliance on the number of words in each example greatly speeds the learning process. The presence of the word count within each example of the raw data results in training difficulties as this information is irrelevant to and disguises the relative word frequencies. Further experiments only considered the normalised data.

Simple pruning reduces the size of the network dramatically, up to 40 percent of the connections are removed with no change in network performance. Figure 6.4 shows the results of pruning connections on the normalised data, removing connections with a saliency below 0.0. Over 100 trials, almost 50 percent of the connections are not required for half the solutions. These results indicate a large number of redundant attributes. There is little point in trying to examine which particular attributes are being pruned out from both outputs, as there is so much flexibility in choosing connections due to the large number of attributes (see figure 6.4).

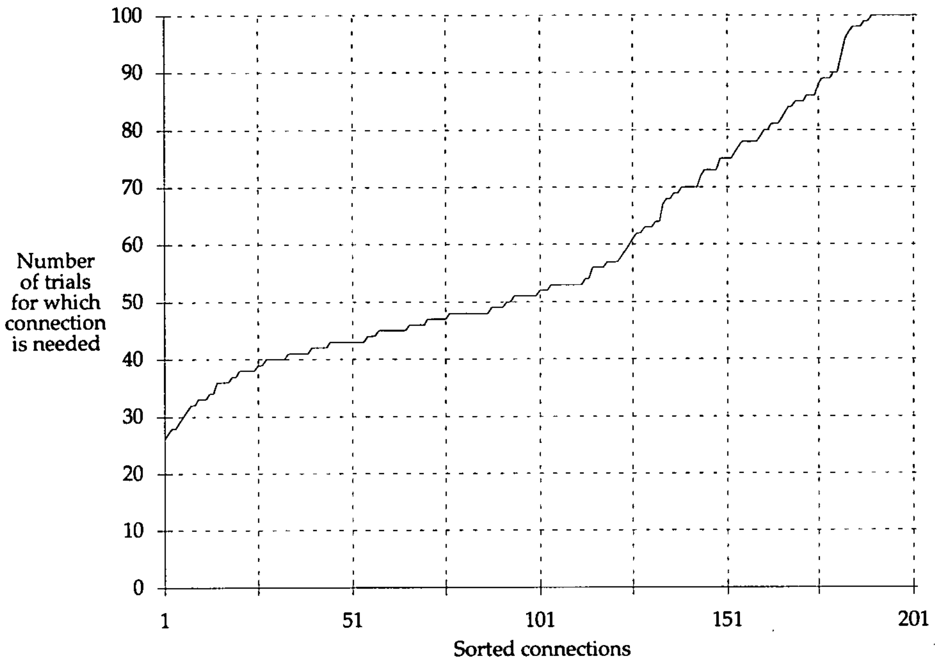

Figure 6.4 — The number of times each of the 202 possible connections (including the two bias connections) are required over 100 trials of the full normalised data

The redundant attributes do not necessarily lead to a reduction in network performance, indeed in the presence of noise the extra attributes will result in better predictions. However poor or non-critical redundant attributes may adversely affect the training performance by overweighting unimportant features thus biasing training [Weiss & Kulikowski 1991]. It is not obvious whether such redundant attributes are valuable or not, although the results from table 6.18 give a weak indication that reducing the number of attributes has no effect. This may only be examined further by testing or estimating the true error rate.

6.2.3 Cross-validation error estimation

It is difficult to see how well a learning method is generalising without a test set to check the performance on unseen cases. Unfortunately there are not enough examples available to produce a separate test set. Rather cross-validation is used to estimate the true error rate of the population [Weiss & Kulikowski 1991]. In this case the leaving-one-out method of cross-validation is used, whereby 188 different data sets are created: each contains a one example test set and a 187 example training set, and the average of these test set results gives the estimate of the true error rate. This is further complicated by the random nature of the neural network starting points. Hence the median of 100 trials is used as the error value for each test set. There is little difference between the median and mean results over the 100 trials. The results are displayed in table 6.19.

Table 6.19 — Results of cross-validation training of Cascor, showing the average of the median results over 100 trials: including the percentage correct on the training and test sets, the number of connections (maximum of 202) and the number of connection crossings of training (in millions)

Technique	Training %	Test %	Connections	CCs (M)
No pruning	100	98.4	202	2.98
Pruning 0.0	100	98.4	118.43	3.4
Pruning 0.05	100	98.4	115.3	3.39

The performance of various methods, without the installation of hidden nodes which are not required, is high (see table 6.19). Cascor is able to distinguish between the play segments to a high level of accuracy, although a level of 100 percent is not achieved. The introduction of pruning, removing the influence of a large number of attributes, does not result in performance degradation, although training time is increased. This indicates that the redundant attributes present are not degrading or improving the classification performance, and there may be no effect in reducing a large number of attributes. This will be tested next.

6.2.4 Restricted attributes

This section examines restricting the number of attributes as a crude method of determining the attribute redundancy in the data set. If a large number of the attributes are redundant, a smaller theory, from artificial neural networks especially, may be produced by reducing the attributes. This will also test whether the data is noisy in nature — resulting in a decrease in classification performance — or whether the redundant attributes adversely affect training — resulting in an increase in classification performance in these experiments. Simple reductions in the number of attributes will be used to test the extent of attribute redundancy.

To start with, the previous pruning experiments may be examined. If the frequency an attribute is used by either output node after pruning (0.0 level) is totalled (see figure 6.5), it is evident that the relative frequency of the word occurrence is not a factor in deciding which words separate the examples. If more examples were to be classified a larger number of attributes may be required. However, this evidence points towards a large number of the attributes being redundant. Further examination of the saliency of each connection is also possible giving a more detailed measure of relative worth, but without any justification of the validity of the Karnin saliency measure, the value of such an analysis is minimal.

A total of ten data sets were created from the normalised data such that four data sets had 25 attributes missing, two data sets had 50 attributes missing and the final four data sets had 75 missing attributes. The attributes are simply partitioned, and the results presented are from the leaving-one-out cross-validation of the median of 100 trials (see table 6.20). Note that some of the trials with only a quarter of the attributes remaining required the addition of hidden nodes, resulting in the differences in the number of connections. More sophisticated techniques for reducing the number of attributes [Catlett 1992; Kira & Rendell 1992; Caruana & Freitag 1994] will not be considered here.

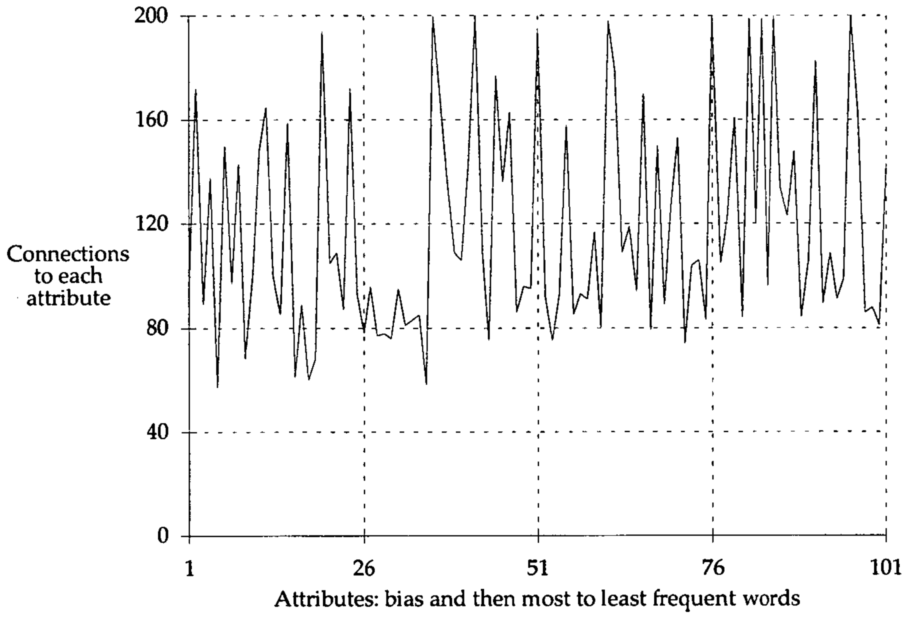

Figure 6.5 — The usage of different attributes from most to least frequent words over 100 trials.

Table 6.20 — Cross-validation results of the median of 100 Cascor nets using the normalised reduced attribute data, showing the training and test set percentage correct, and the number of connections and connection-crossings of training (in millions)

Data Set	Training %	Test %	Connections	CCs (M)
25% missing — 1st 25	100	97.87	152	2.07
25% missing — 2nd 25	100	94.41	152	5.13
25% missing — 3rd 25	100	98.4	152	2.58
25% missing — 4th 25	100	95.74	152	3.16
50% missing — 1st 50	100	92.02	102	5.33
50% missing — 2nd 50	100	97.34	102	2.54
25% left — 1st 25	100	86.17	108.85	73.98
25% left — 2nd 25	100	96.28	52	2.68
25% left — 3rd 25	100	84.84	109	61.16
25% left — 4th 25	100	92.55	80	20.54

These results indicate that the problem is still solvable by Cascor, even if three quarters of the attributes are removed, though not to the same degree as when all the attributes are used. The drop in performance indicates that the data is noisy, and the larger number of attributes is valuable in obtaining a high level of performance.

The most useful groups of attributes appear to be the second and fourth groups — the most frequent words from 26 to 50 and 76 to 100. This is evident when considering the three groups of trials which cover all the data. With one quarter of the attributes missing the training time increases when, in particular, the second and the fourth partitions of the attributes are missing. When half the attributes are missing more training is required when the first 50 attributes are missing — thus excluding the second group of 25 attributes. Finally when only a quarter of the attributes are used it is evident from the cross-validation performance, network size and training times which sets of attributes most aid the training process.

6.2.5 Other methods

Finally other classification methods are considered: the performance of C4.5 (see table 6.21), and then LDA and 5-NN (see table 6.22) are applied to the normalised and reduced attribute data sets, and the classification performance is determined again by full cross-validation.

C4.5 creates a simple decision tree for classifying texts based on the complete data (see figure 6.6). However, the tree produced does not classify all of the training samples. This may be due to the number of training examples being too limited to develop a more sound tree, or the separations between the classes may not be performed by splitting the data on a single attribute value. It should also be noted that the tree developed by C4.5 requires only seven attributes to arrive at its performance. This also indicates, along with the previous pruning and restricted attribute results, that not all of the attributes are required to achieve a reasonably high performance level, if one below the highest possible.

```
that (relative pronoun) <= 0.20202 : romantics (51.0/1.0)
that (relative pronoun) > 0.20202 :
|   who (relative pronoun) <= 0.2 :
|   |   make <= 0.07407 :
|   |   |   you <= 1.26147 : romantics (11.0/1.0)
|   |   |   you > 1.26147 : renaissance (6.0)
|   |   make > 0.07407 :
|   |   |   good > 0.06866 : renaissance (88.0)
|   |   |   good <= 0.06866 :
|   |   |   |   is <= 1.05 : romantics (3.0)
|   |   |   |   is > 1.05 : renaissance (10.0/1.0)
|   who (relative pronoun) > 0.2 :
|   |   come <= 0.24661 : romantics (16.0)
|   |   come > 0.24661 : renaissance (3.0)
```

Figure 6.6 — Tree developed by C4.5 from the normalised tragedy data showing that, for example, 51 cases are correctly classified (and 1 incorrectly) as romantic play sections if the percentage of occurrences of 'that' (relative pronoun) is below 0.20202 percent

In comparison to the performance of Cascor, C4.5, for the reasons stated above, seems to be a relatively poor classifier for this task (see table 6.21). The training set and cross-validation performances are well below those of Cascor: there is a difference of 10 percent between the highest cross-validation performance for Cascor and that of C4.5. There is also evidence that the reduction of the number of attributes affects the final performance of C4.5 through the size of the final tree which has been developed. This reflects the usefulness of some groups of attributes over others, however, unlike Cascor, the final 25 attributes seem to be of more value to the C4.5 classification than the second group.

Table 6.21 — Results of C4.5 cross-validation of 188 data sets: the training and test set percentages, and the nodes in the final tree

Data Set	Training %	Test %	Nodes
Full normalised	98.5	87.2	15.8
1st 25% missing	98.9	85.6	17.3
2nd 25% missing	98	77.1	19.6
3rd 25% missing	98.5	87.8	15.8
4th 25% missing	98.8	75.5	22.9
1st 50% missing	98	80.3	19.5
2nd 50% missing	98.8	80.9	23.2
1st 25% left	98.6	70.7	33.8
2nd 25% left	98.4	82.4	23.1
3rd 25% left	96.4	73.4	29.1
4th 25% left	97.8	87.2	18.8

The performances of LDA and 5-NN are good on these data sets (see table 6.22). Although the performance of both methods does not maintain the high standard achieved with Cascor, these results further highlight how easy it is to solve this problem given the number of attributes and examples.

Table 6.22 — Results of LDA and 5-NN on the various data sets: the training and test set percentages

	LDA		5-NN	
Data Set	Training %	Test %	Training %	Test %
Full normalised	100	96.81	100	93.62
1st 25% missing	100	95.21	100	96.28
2nd 25% missing	98.29	94.15	100	91.49
3rd 25% missing	99.98	95.74	100	93.09
4th 25% missing	100	96.28	100	94.15
1st 50% missing	97.8	91.49	99.37	90.96
2nd 50% missing	99.36	96.28	100	90.96
1st 25% left	91.91	85.11	100	84.04
2nd 25% left	97.68	95.74	99.99	89.89
3rd 25% left	89.31	81.91	99.37	83.51
4th 25% left	94.3	91.49	99.37	91.49

6.2.6 Summary and discussion

The classification of Romantic and Renaissance authors has been examined. Cascor is easily able to build a suitable classifier without the use of hidden nodes, and cross-validation shows that this high performance level is maintained for unseen cases. Hidden nodes may be added only by forcing the network to over-train. Furthermore, a number of the attributes may be removed resulting in a minor degradation in classification performance, indicating that a large number of attributes is required to maintain high levels of classification. Alternative methods are also able to solve the problem, although not to the same performance level as Cascor.

There is little room for improvement by Cascor given that only a linear layer with squashing functions is applied, and further examples are required to examine the non-linear nature of the data set. As mentioned in chapter 5 the complexity of the data set is dependent on the presentation of the data: what is measured and how many examples are available.

The two problems examined in this chapter demonstrate the difficulties present in finding problems of a non-linear nature which are solvable. Insufficient examples will not allow non-linear features to be extracted from the data, and unreliable data may mask complex features. Attributes which are related in a non-linear manner may also avoid the need for complex data features as these are directly obtainable from the inputs. This experience indicates that having sufficient training examples and a concise group of measured attributes which identify non-duplicated features is a method for ensuring complex data. The final necessary feature is a problem of sufficient complexity — as defined by the measured attributes and examples — which may only be determined by close examination of the problem under consideration.

For the process of testing new methods the question becomes whether this is a sensible strategy or whether constructing data is more practical.

7 Constructing data sets — two methods

Looking at real-world problems to aid in the development of new methods, as indicated in the previous chapter, may be a long drawn out method for finding tasks which are difficult, but not impossible, to solve. Thus it is necessary to turn to artificial benchmarks to create problems to test the capabilities of learning systems. Although it will not be possible to develop a single universal benchmark, as all systems have their own biases, it may be possible to develop benchmarks which will at least challenge different learning methods, particularly artificial neural networks, without being overly simplified.

One difficulty with generating tasks is limiting the data sets to be within sensible bounds. For the purposes of the experiments in this chapter the following conditions will be met:

- two real-valued inputs — with values between –1.0 and 1.0, with the benefit that the data sets may be displayed in a two-dimensional graph;
- two classes;
- no missing, redundant or irrelevant classes, attributes or examples; and
- a set number of training and independent test examples — 5000 in each set.

Thus variation in the data sets is performed by changing the underlying theory. Changes in the number of inputs and outputs is a minor extension, and simulating problems encountered in real-world situations — such as noise, redundant data and different numbers of training examples — may also be incorporated. The great advantages of generating data are that as many data sets as required may be produced to test a learning system, and that a large number of examples may be used for testing the accuracy of the final classifier leading to a true and accurate measure of the classifier performance.

The work on Voronoi data sets in §7.1 is published elsewhere [Waugh 1995b].

7.1 Voronoi data sets

This section examines the application of Voronoi diagrams [Okabe, Boots & Sugihara 1992] to the generation of data sets which are more complicated, hence requiring the power of learning methods such as multiple layer neural networks. This stems from Quinlan's concept of P-type problems [Quinlan 1993b; Collier & Waugh 1994]. Okabe et al. give the following informal definition of two dimensional Voronoi diagrams [Okabe, et al. 1992]:

> Given a set of two or more but a finite number of distinct points in the Euclidean plane, we associate all locations in that space with the closest member(s) of the point set with respect to the Euclidean distance. The result is a tessellation of the plane into a set of regions

associated with the members of the point set. We call this tessellation the *planar ordinary Voronoi diagram* generated by the point set, and the regions constituting the Voronoi diagram *ordinary Voronoi polygons.*

In practice the term 'ordinary' may be dropped.

The generation of the data sets is simple: a number of generation points are randomly constructed within the space under consideration, and assigned to a particular class. These are then used to classify further random points in the feature space by their distance from the generating points. These new points are used for the training and test sets. Thus generator points are linked to form regions of a single class (for example see figure 7.1).

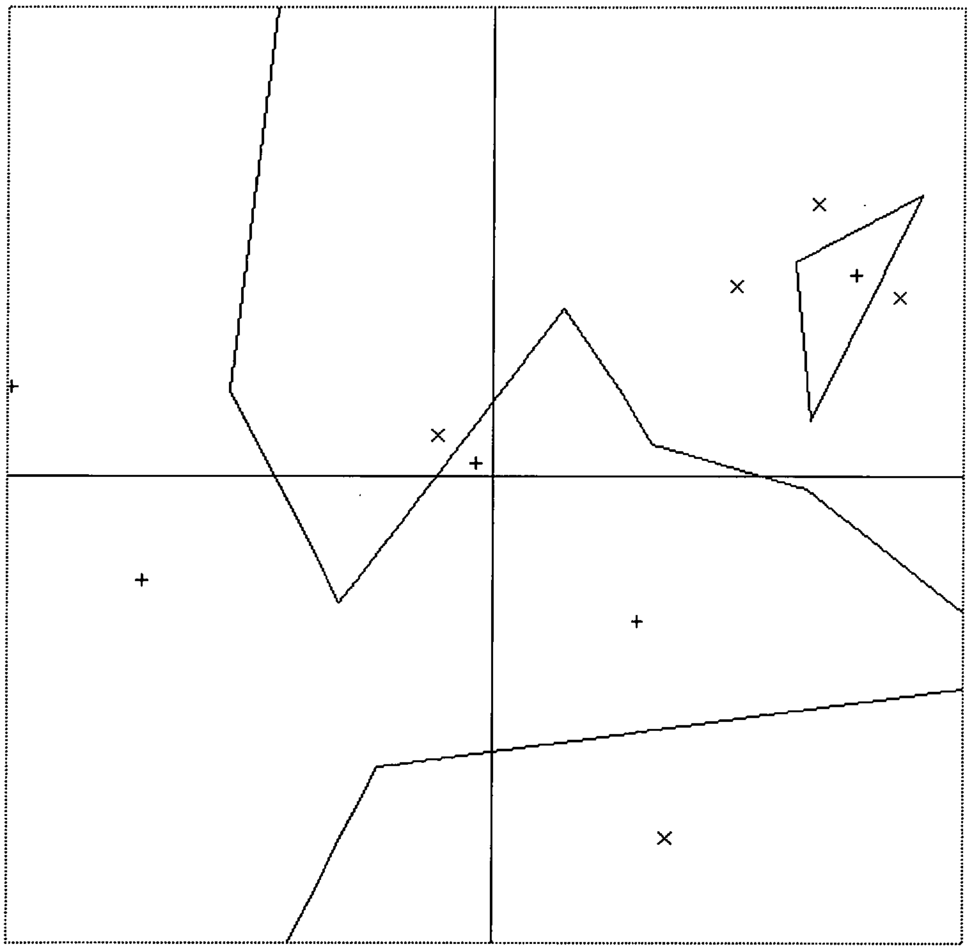
Figure 7.1 — an example of a two class Voronoi data set with five generators in each class

7.1.1 Data set characteristics

There are a number of important features which should be considered:

- the number of generating points actually needed;
- the number of generating points needed in each class;
- the number of edges needed to separate the generating points and hence the classes, which is also proportional to the number of vertices; and

- the total length of the necessary edges.

The number and length of edges is dictated by the number and placement of generators, the placement being random in this situation. If a number of trials are conducted to avoid problems with the placement of generators, complexity in data sets is increased in two ways:

- increasing the total number of generators, as more subregions or extensions of a class are generated; and
- using more equal numbers of generators in each class, increasing the complexity over unequal numbers, corresponding with a greater likelihood of the generators needing to be separated.

This complexity is indicated by the number of divisions within the data set (see §7.1.2).

Figure 7.2 demonstrates the first point by showing the averages over 100 randomly created data sets of the needed generators, edges and edge lengths for each pair of generator numbers. The number of generators is varied from 1 to 50 in each class, thus requiring the generation of 5000 data sets. The number of edges needed grows asymptotically linearly, as does the total length of the edges, which does so at a slower rate indicating in a decreasing average length. This means that each edge has less effect on the final solution, but there are more edges resulting in a higher complexity. Furthermore figure 7.2 shows that the generators required are near the maximum possible in this case where an equal number of generators are used in each class. For example with 50 generators in each class, over 96 generators out of a possible 100 are required on average over the generated data.

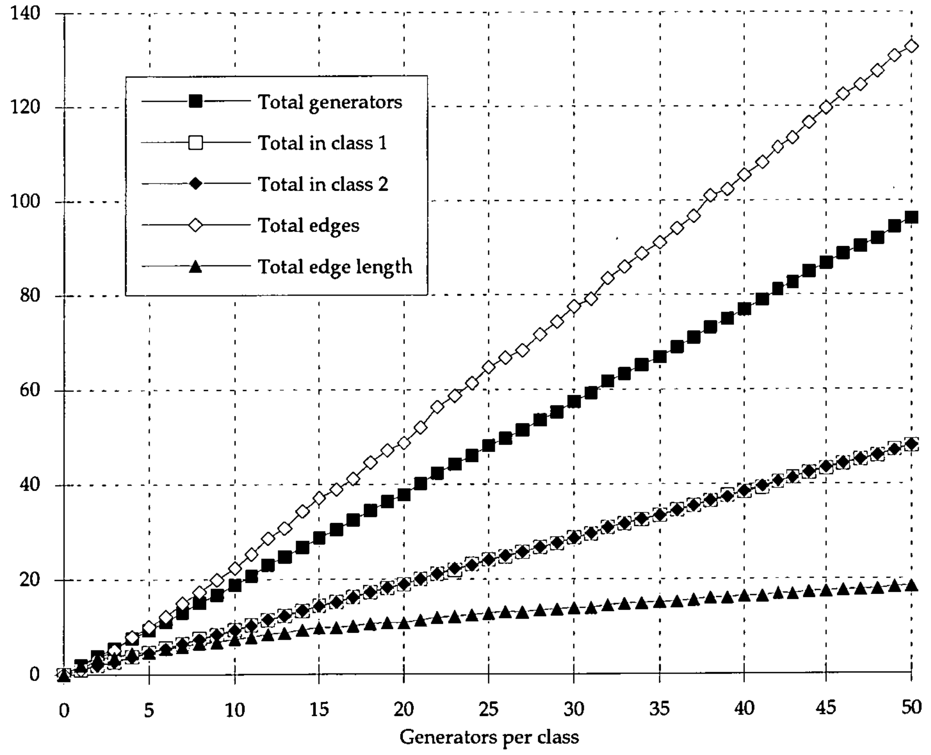

Figure 7.2 — Increase in complexity due to more generators being used for both classes

It should also be noted that the standard deviations of these measurements increase slowly along with the number of generators, as is expected with a larger possible range of numerical values.

Figure 7.3 demonstrates the second point. The data sets plotted have a total of 20 generators, and the number in each class is altered from 1 and 19 to 10 and 10; with again the results being the averages over 100 generated data sets. The shapes of the curves are similar to that of the product of the number of generators used in each class. As the number of generators in each class becomes more even, the total generators and the total edges reach their maximum level. Simultaneously more generators of the lesser class, in this case class one, are being used with a resulting increase in the percentage of the feature space falling under that class, while the number of generators used from the other class falls. Furthermore, when the classes are extremely uneven, few generators of the second class are used at all as the feature space is dominated by that class and a large proportion of them are redundant.

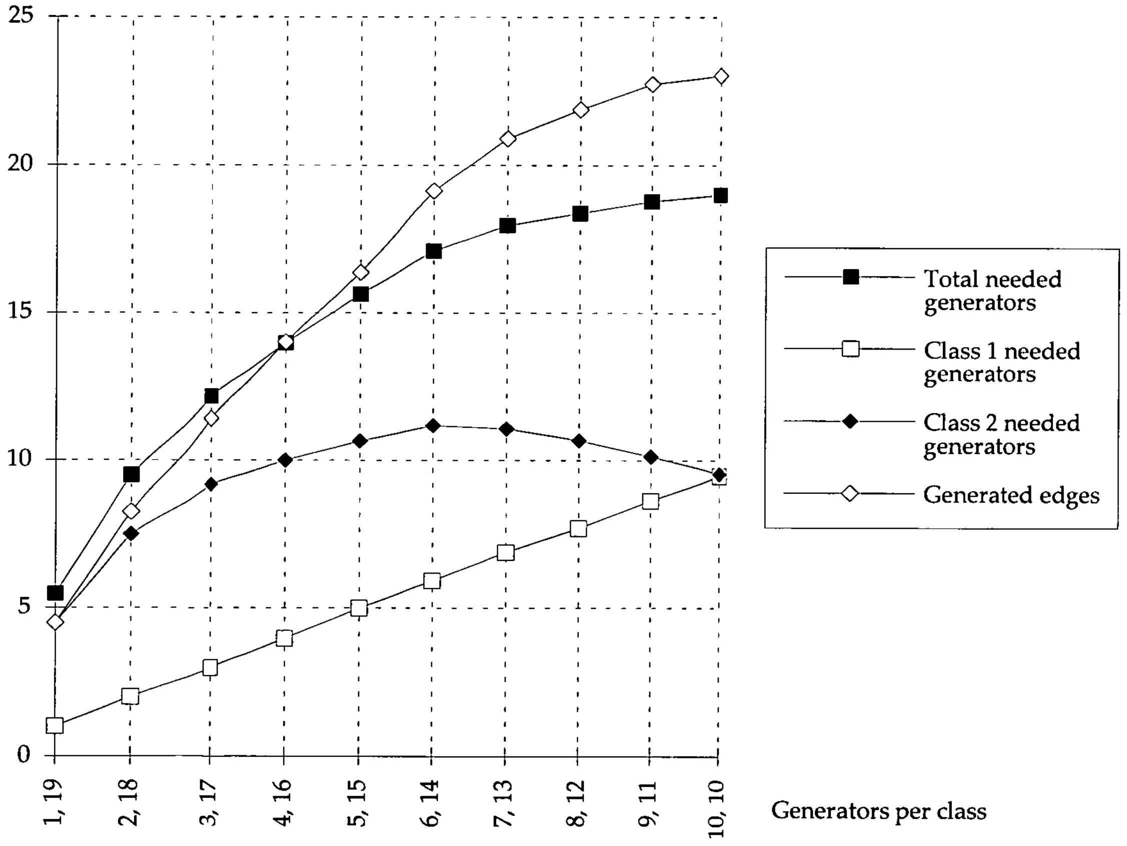

Figure 7.3 — Increase in complexity due to more balanced numbers of generators in each class, with a total of 20 generators

7.1.2 Measuring complexity

It may be possible to determine an exact measure of complexity based on the data set features. However, it is not obvious that this would be a sensible course of action given that complexity is measured in different manners for different learning methods. What is difficult for linear discriminant analysis, for example, may be trivial for artificial neural networks, meaning that it would be of little value to create such a measure.

It is prudent to note at this point that the structure of Voronoi data sets should be easy for perceptron-like architectures to solve, as the regions involved have piece-wise linear boundaries. In comparison C4.5 should have difficulty in solving such problems given that it separates regions by splitting the feature space on the basis of a single attribute value, meaning that non-vertical or horizontal separations in the feature space are harder to classify. Such separations will be common in Voronoi data sets.

Thus there are many possible definitions of complexity depending on the capability of the learning system. Here the focus is on measures which will allow a relative comparison of complexity by measuring features which lead to complexity within a data set, hence the use of the number of generators in total and per class for the Voronoi data sets. This method of measuring complexity is rough and does not take into account features such as convexity and concavity. These are problematic in that some decision needs to be made as to whether a slight concavity should be ranked as being as complex as a large indentation of one class into a region of another within the feature space. Likewise connectivity of edges also needs to be considered, as for example separate regions in feature space are more difficult to isolate using artificial neural networks. A more complete measure of complexity should take these factors into consideration, along with the capabilities of the various learning methods.

Nevertheless, the number of generators does give a relative estimation of complexity. For example a rectangle and a dodecagon are both convex and fully connected, but to model a dodecagon exactly requires more processing power, thus presenting a more complicated problem. Further, considering that in this case the placement of generators is random, the convexity and connectivity of the resulting regions should be averaged over a reasonable number of trials.

Thus on average, it is possible to produce Voronoi data sets of arbitrary complexity by simply increasing the number of generators in all classes and evening up the number of generators between the classes. As it is possible, even though unlikely, to generate a data set with an infinite number of generators which is in turn trivial to solve — namely that the generators are collinear — it is necessary to perform tests over a reasonable number of data sets rather than a single one. These data sets have the added advantages that the maximum and minimum classification rates are known: namely 100 percent and the percentage of the largest class respectively.

7.1.3 Simulation results on Voronoi data sets

Experiments are conducted using LDA, 5-NN, C4.5, back-propagation-style networks using Quickprop (QP) as the update function, and Cascor to test the validity of these data sets for the purpose of generating problems with differing underlying complexity for benchmarking.

Cascor is used with the parameters given in §2.4.1, except that a restricted patience period (20 epochs) is used for both candidate and output training. Node patience is also used with percentage changes in error of 1 and 5 percent over the installation of a single hidden node. The Quickprop trials use the parameters outlined in table 7.1, with patience (1 and 5 percent change) being employed to stop training and either 5 or 10 hidden nodes used. The results of the artificial neural network learning methods on each data set are averaged over 100 machine clock-seeded trials to account for the random nature of the starting points. The back-propagation experiments were also performed using a separate simulator developed by the author.

Table 7.1 — Parameters used for Quickprop trials

Parameter	Value
Total hidden nodes	5 or 10 nodes
Eta	0.1
Mu	1.75
Weight decay	0.0001
Patience percentage	1% or 5%
Patience length	20 epochs
Maximum epochs	1000 epochs
Activation functions	Symmetric sigmoid
Activation function offset	0.1

The results take two forms:

- correctness of the final theory — measured by the percentage correct on the unseen test set; and
- complexity or size of the final theory — measured, where appropriate, by the number of hidden nodes for Cascor or Quickprop trials, or the number of tree nodes for C4.5.

7.1.3.1 Complexity by increasing generator numbers

The effect of increasing the number of generators in each class is considered. The results are on three types of data sets with 4, 10 and 20 generators used for each class. 20 data sets of each type were created, the classification results of which are averaged to give an indication of the complexity of the data sets given the changes in the number of generators used (see table 7.2 and 7.3). The standard deviation is also shown to give an indication of the spread of the results for each data set.

In these experiments more generators lead to relatively less examples per generator, as the number of training examples is fixed. Thus it is expected that the performance will drop slightly, along with a drop due to the increasing complexity of the data sets.

Table 7.2 — Average and standard deviation of the percentage correct results on trials over 20 data sets in each generator category (4 and 4, 10 and 10, and 20 and 20)

	4 and 4		10 and 10		20 and 20	
Method	Average	Stand. Dev.	Average	Stand. Dev.	Average	Stand. Dev.
LDA	75.7	11.2	67.1	10	59.6	6.7
5-NN	99	0.3	98.5	0.3	97.7	0.4
C4.5	98.5	0.5	97.7	0.5	96.4	0.5
Cascor 1%	93.8	6	88.1	4.5	77.4	4.4
Cascor 5%	93.3	6.8	87.3	5.8	77	4
QP 5 hid 1%	70.8	11.7	61.1	7.1	56.2	4.3
QP 5 hid 5%	70.9	12	61.4	7.1	56.1	4.2
QP 10 hid 1%	75.3	13.2	63.5	7.5	57.3	4.4
QP 10 hid 5%	75.6	13.3	63.4	7.3	57.3	4.1

Table 7.3 — Average and standard deviation of the theory size where appropriate on trials over 20 data sets in each generator category (4 and 4, 10 and 10, and 20 and 20)

	4 and 4		10 and 10		20 and 20	
Method	Average	Stand. Dev.	Average	Stand. Dev.	Average	Stand. Dev.
C4.5	111.2	33.1	165.8	36.6	251.6	31.7
Cascor 1%	5.9	1.7	5	1.4	3.9	1
Cascor 5%	5.3	1.8	4.5	1.3	3.5	0.9
QP 5 hid 1%	5	0	5	0	5	0
QP 5 hid 5%	5	0	5	0	5	0
QP 10 hid 1%	10	0	10	0	10	0
QP 10 hid 5%	10	0	10	0	10	0

LDA has a decreasing performance with the increasing complexity of the data sets, as would be expected due to its inability to solve anything more than linearly separable problems. This is also shown by the standard deviations of the LDA results: as the complexity increases the deviation of the LDA results reduce indicating less capability to model the data sets as greater structure is present within them.

5-NN performs very well, though the performance does drop slightly with harder problems which contain more features and more edges where mistakes are likely to occur, with the same number of examples overall to identify them. The high performance levels are due to the algorithms natural bias in favour of these data sets where no noise is involved. This is also shown by the low standard deviation in the data set results as the method is solving the problems to near optimal levels. Note though that the performance is not perfect as the training set does not contain the exact generating points — if it did, then 100 percent accuracy is expected.

C4.5 also performs very well with a slight drop in performance as complexity is increased, and a large increase in the size of the learnt theory as measured by the number of nodes in the induced tree. The large tree indicates that the feature space has been segmented heavily to classify the data sets. The large variation in the size of the induced trees is due to the natural variation in the data sets, as well as complications forced by the inductive bias of the algorithm: namely dividing classes based on a single feature. Although this style of problem is P-type in nature, the large number of training examples allow C4.5 to successfully separate the data set features, resulting in very good classification performance. If less examples are available the performance would probably drop as C4.5 would develop much simpler decision trees.

Cascor also performs well but degrades quickly with increasing complexity, which is further indicated by a decrease in the deviation showing that the extra complexity affects the further addition of hidden nodes: adding nodes becomes more difficult (see table 7.3). There is also a corresponding drop in the size of the final theory as measured by the number of hidden nodes. The drop in performance is accounted for as the trials were conducted using node patience (see §3.1). As the data sets become more complicated the features become smaller, as indicated by the average edge length (see figure 7.2), and correct classification of those features results in less overall performance improvement. Thus Cascor stops training using these node patience levels before all possible improvements have been made. The addition of further hidden nodes may result in better solutions.

The advantage of employing node patience is that it stops the introduction of unnecessary nodes. The obvious disadvantage is that helpful training may not occur. These alternatives cannot be distinguished on the basis of training data only. The differences in the percentage change used for node patience do not seem to have a significant effect on the final classifier, though further variation of the parameters may result in performance improvements. These data sets are difficult to apply node patience to as there is the possibility of very small performance increases later on relative to the initial gains by earlier hidden nodes or the simple perceptron-like output layer. This further points to the value of having a validation set, if enough examples are available.

The performance of Quickprop as a representative of the back-propagation styles of networks is very poor. The trials in most cases did not achieve better results than those gained by LDA, and the standard deviation results are similar in nature, showing the difficulties of increased complexity. Further increasing the number of hidden nodes, with up to 20 hidden nodes being used, resulted in a small performance increase, and a very large increase in the training time. Perhaps the cause of this is what Fahlman terms the 'herding effect' [Fahlman & Lebiere 1989], whereby most of the hidden nodes are covering the major errors as driven by the learning algorithm, without spreading out to cover more areas of the feature space. An alternative explanation is that the architecture, being only a two-processing layer network, may not be able to model the underlying function, although the algorithm should achieve the performance level of LDA. A further explanation may be that the algorithm has not been optimised sufficiently by the setting of parameters; although this argument pales in comparison to C4.5, for example, which requires no parameter adjustments. A final possibility is that there is some problem with the Quickprop algorithm which produces poor results in some cases — this is to be examined in §7.3.1. Whatever the reason, the performance of Quickprop on these problems is inadequate.

Finally the performance difference between the artificial neural network methods and C4.5 and 5-NN needs to be explained. The lack of performance by the network architectures is unexpected given the bias of the methods toward solving Voronoi-style problems. This may be explained by the lack of capacity in the neural network techniques, the natural bias of 5-NN in solving such problems, and the large number of leaves of C4.5 trees indicating a high segmentation of the feature space given the number of generators. Such a classification is not natural, and most likely will not scale up to higher dimensions, and performance on these data sets is likely to drop off more rapidly if the number of training examples is reduced. These conjectures are supported by the results from chapter 6 where Cascor outperformed both 5-NN and C4.5.

7.1.3.2 *Complexity by more even generator numbers*

The change in complexity with differing numbers of generators between the classes is also examined. This is tested by considering a number of generator combinations totalling 20 generators in both classes: 2 and 18, 4 and 16, 6 and 14, 8 and 12, and 10 and 10. The results of trials using the methods mentioned above are outlined in tables 7.4 and 7.5, but only Cascor with 5 percent node patience and one Quickprop trial with 5 hidden nodes and 5 percent patience are considered, as there is little variation in the alternatives tested previously.

Table 7.4 — Average and standard deviation results of test set performance on trials over 20 data sets in each generator category (2 and 18, 4 and 16, 6 and 14, 8 and 12, and 10 and 10)

	2 and 18		4 and 16		6 and 14		8 and 12		10 and 10	
Method	Av	SD	Av	SD	Av	SD	Av	SD	Av	SD
LDA	89.2	4	78.5	4.8	71.9	6.3	64.6	7.4	67.1	10
5-NN	99.3	0.2	98.9	0.3	98.6	0.3	98.5	0.3	98.5	0.3
C4.5	99	0.3	98.2	0.3	97.9	0.5	97.6	0.5	97.7	0.5
Cascor	94	2.9	89	4.6	87.8	5.1	85.5	4.6	87.2	5.6
Quickprop	84.6	7.9	71.1	6.3	64.1	6.8	59.6	6.8	61.4	7.1

The performance of LDA is improved, as would be expected, when the number of generators in the classes are unbalanced. This corresponds to a large area of the feature space being closer to the generators of the majority class, and it becomes simpler to perform well under these circumstances. The performance mirrors closely the expected percentage of the feature space under the largest class.

5-NN performs well again with the percentages correct on the unseen test set remaining about the same high level given the natural bias of the method. The standard deviation of the results does drop slightly as the data sets become more complicated. Whether this is a chance occurrence or whether it indicates more consistent results obtained by nature of the larger number of generator points leading to less erratic data sets is unclear from these figures.

Table 7.5 — Average and standard deviation results of the theory size on trials over 20 data sets in each generator category (2 and 18, 4 and 16, 6 and 14, 8 and 12, and 10 and 10)

	2 and 18		4 and 16		6 and 14		8 and 12		10 and 10	
Method	Av	SD	Av	SD	Av	SD	Av	SD	Av	SD
C4.5	69.7	21.2	119.4	16.8	150.2	29.1	170.4	24.6	165.8	36.6
Cascor	2.9	1	3.7	1.1	4.5	1.6	4.3	0.9	4.5	1.3
Quickprop	5	0	5	0	5	0	5	0	5	0

C4.5 also performs well with results near 100 percent correct. What generally alters is the size of the learnt theory, increasing as the number of generators becomes more even, indicating that the complexity of the data sets is increasing. The standard deviations of the results back up these observations: the result for the test set classification rate does not change greatly, and the spread of the size of the final theories increases with the additional complexity.

The performance of the artificial neural network methods is also consistent with increasing complexity as the performance of the methods decreases, whilst the number of hidden nodes installed by Cascor increases and then drops off with further complexity. The trends are evident, though not perfect, and results may be improved by measuring over more than 20 trials in each case.

7.1.4 Summary

It is possible to generate data sets of increasing complexity to test new inductive learning methods by reporting average performance on specified problems, without biasing results by considering an individual problem. Data set complexity is increased by the addition of greater number of generation points, or by the evening up of the numbers of generators between classes. The suggested method of generating data has been shown to hold over a reasonable number of differing learning methods, although the biases of each method must be taken into account. There is also a large background of Voronoi diagram theory to build on, including areas on generalised [Okabe, et al. 1992] and temporal Voronoi diagrams

[Devillers, Golin, Kedem & Schirra 1994]. Such data sets may be easily created and the experiments replicated.

It would appear that such learning tasks are difficult for artificial neural networks. The higher performance of 5-NN may be accounted for by a natural bias towards problems formulated in this way. The higher performance of C4.5 is more of a surprise, given the previous literature [Quinlan 1993b]. However these previous tests have centred on problems using a total of 5 generators in 5 dimensional space, as opposed to up to 40 generators in 2 dimensional space. The differences between Cascor and C4.5 may alter with alternative learning tasks using more attributes — an interesting area for future investigation. Cascor easily outperforms the Quickprop-trained networks on these more difficult domains. Nevertheless, the performance of the artificial neural network methods is poor, indicating that these problems do effectively test such systems and show that further improvement is possible. Further work may need to be completed to achieve the standards of the other learning methods.

The data sets are biased in that they are all of a similar style which may aid one learning method over another. However, all benchmarks do have some biases regardless of how cleverly they are constructed and these data sets are capable of comparing similar methods such as artificial neural networks. Voronoi data sets are able to represent a very large number of data sets. A final concern is that the data sets are unrealistic in nature as the examples are spread evenly over all the feature space, however this may be a benefit in that matching the exact boundaries of such problems is a very difficult task.

7.2 Normal data sets

A further method has been developed to address the lack of realism exhibited by Voronoi data sets. Instead of generating example points and classifying them due to their distance from the various generation points, the generators are created with a position and a standard deviation. New example points for training and test sets are then created by selecting one of these generators at random and adding to that generator position variations along each axis based on a normal distribution calculated by the generator's standard deviation — thus giving changes around the generator, which becomes the mean of the distribution. This gives the position of the new example in the feature space.

Thus example points are centred in a normal distribution around the generating points, creating a *normal* data set, rather than example points for the training and test sets being spread out evenly over the feature space. This in turn leads to overlap between classes, and areas where no examples fall — a more realistic scenario. The correct classification for each example is given by the generator which is originally selected.

Such data sets, unlike Voronoi data sets, are ill-matched to the capabilities of multi-layer perceptrons as these methods form piece-wise linear boundaries. Data sets based on Gaussian mixtures form quadratic decision boundaries. Rather than being a hindrance, this is a benefit as it allows network training algorithms to be tested on hard problems. As such, it is expected that comparisons to the performance of neural network methods on the data sets will be poorer in relation to the results obtained on the Voronoi data sets.

There are a number of further assumptions with this generation process:

- the generators are within the bounds of –1.0 to 1.0, in two dimensional space;
- the generators are selected with equal probability, meaning that the number of points associated with each generator is equal;
- the points used for training do not have to be within the –1.0 to 1.0 bounds;
- the same standard deviation is used for both axes, giving a round distribution and meaning that the covariances in the multiple-valued normal distributed are set to zero; and
- the standard deviations are generated from the even interval of 0.0 to 0.25.

7.2.1 Optimal classification

It is possible to calculate the optimal Bayesian classification for such data sets, given that there is overlap between the classes. This gives an upper bound on the classification performance, in the same way that the lower bound will again depend on the class with the most examples. Unlike the Voronoi data sets where the maximum classification rate is 100 percent, the overlapping classes in these examples lead to a lower optimal classification rate.

To briefly reiterate the calculation of the Bayesian classification rate, the class of an example is inferred by calculating the probability that the example falls in each class, and choosing the maximum, hence applying Bayes' rule to select a particular class *i*:

$$P(C_i \mid \bar{x}) > P(C_j \mid \bar{x}) \; \forall j \neq i \qquad (7.1)$$

where *P(x)* represents the probability of *x*, *P(x | y)* represents the conditional probability of *x* given *y*, C represents a particular class and $\bar{x}$ represents the inputs to the system. The expression (7.1) may be calculated by employing Bayes' theorem:

$$P(C_i \mid \bar{x}) = \frac{P(\bar{x} \mid C_i) \cdot P(C_i)}{\sum_{\forall j} P(\bar{x} \mid C_j) \cdot P(C_j)} \qquad (7.2)$$

The probability of each class without prior information is proportional to the number of examples in each class, hence in this particular case this is proportional to the number of generators. As is standard the denominator in (7.2) may be dropped when employed within Bayes' rule. Furthermore, given this is a normal distribution, the following formula holds:

$$P(\bar{x} \mid C_i) = \frac{1}{(2\pi)^{\frac{n}{2}} |V_i|^{\frac{1}{2}}} e^{\pm\frac{1}{2}(\bar{x} \pm \mu_i)^T V_i^{\pm 1} (\bar{x} \pm \mu_i)} \qquad (7.3)$$

where n is the number of dimensions, V_i is the covariance matrix of class i, and μ_i is the mean vector of class i. In this case n is two, and the covariance matrix is simplified with the variance in the diagonal, and zeros elsewhere due to the assumption of a circular distribution. A few minor steps gives the following expression from (7.3):

$$P(\bar{x} \mid C_i) = \frac{1}{2\pi\sigma_i^2} e^{\pm\frac{1}{2\sigma_i^2}\left((x \pm \mu_{ix})^2 + (y \pm \mu_{iy})^2\right)} \qquad (7.4)$$

where μ_{ix}, for example, is the mean in the x axis of class i, and σ_i is the standard deviation for class i. This may be used, along with the proportion of generators or number of examples in each class, to calculate the Bayesian classification for each data set generated, which in turn may be compared to the known classifications. The maximum classification rate may thus be given exactly for each training and test set generated. A more detailed discussion on Bayesian classification may be found in statistical texts (for example, [Duda & Hart 1973; James 1985]).

7.2.2 Simulation results on normal data sets

Results are generated on similar data sets for the same methods as mentioned in §7.1.3.1, though only one Quickprop trial — with 5 hidden nodes and a patience stopping percentage of 5 percent — is considered. The results are given in tables 7.6 and 7.7, along with the Bayesian classification rates for the data sets averaged over the 20 sets in each generator grouping.

Table 7.6 — Average and standard deviation results of the test set classification rate on trials over the 20 data sets in each generator category, including the optimal Bayesian classification

	4 and 4		10 and 10		20 and 20	
Method	Average	Stand. Dev.	Average	Stand. Dev.	Average	Stand. Dev.
Bayesian	95.1	4.1	88.6	3.8	83.1	3.5
LDA	73.8	10.7	62.2	7.3	56.4	3.7
5-NN	94.6	4.4	87.6	4.1	81.7	3.6
C4.5	94.4	4.7	86.6	4.4	80.3	3.6
Cascor 1%	91	5.1	79.3	5.9	70.5	3.4
Cascor 5%	91.2	5.1	78.9	6	69.3	4.3
Quickprop	71.5	12.2	56.1	4.2	51.4	1.3

The complexity of the data sets steadily increases as more generators are used (see table 7.6 and 7.7). This is evident from the results of the classification methods as well as the

Bayesian classification which indicates that there is increasing overlap between distributions of examples. LDA again does not perform well as the number of generators increase. The performance when there is only four generators in each class is reasonable, although this is to be expected as there is more likelihood of gaps between the distributions of examples. The spread of the results is also quite large indicating a lack of capability in solving the problems.

Table 7.7 — Average and standard deviation results of the size of the final theory on trials over the 20 data sets in each generator category, where relevant

	4 and 4		10 and 10		20 and 20	
Method	Average	Stand. Dev.	Average	Stand. Dev.	Average	Stand. Dev.
C4.5	131.8	107	365.7	134.9	526	115.4
Cascor 1%	3.3	1.3	4	0.9	3.8	0.5
Cascor 5%	2.9	1.1	3.5	0.7	3.1	0.8
Quickprop	5	0	5	0	5	0

5-NN again performs well, though there is evidence that the results of the method fall off more quickly than those for the Voronoi data sets, as is expected from this data set style. This follows as the overlap between different classes increases. The spread of results is not much larger than for the Bayesian classification. C4.5 also performs well, although again the performance decreases rapidly with increasing complexity. This is accompanied by a massive growth in the size of the final pruned trees necessary to classify the problems. The spread of results here also increases in line with these larger results, indicative of the greater variation possible with the tree size.

The performance of Cascor again is lower than that of 5-NN and C4.5, although the performance drop with increasing data sets complexity is not as great as for the Voronoi data generation method. The results show that there is little difference in the changes to the node patience percentage. Cascor is not able to cope with the additional complexity and the number of hidden nodes installed decreases when there are 40 generators in the feature space. Quickprop also performs badly, not even reaching the performance level of LDA. Initially the variation in classification performance is very high, but this decreases with increasing data set complexity and a decrease in the ability of the network style to solve the problems at hand.

7.2.3 Summary

A second method of generating data for benchmarking artificial neural networks is given — normal data sets. This generates more realistic data sets than the Voronoi data generation method in that the distribution of examples varies across the feature space; but loses some of the properties and correspondence with the Voronoi diagram theory. The results indicate that normal data sets are similar in nature to Voronoi data sets in that complexity may be easily increased by the addition of generator points.

The current generation process is limited by the initial assumptions in that not all possible normal distributions are allowed — only round distributions are produced currently — and that the restriction of the standard deviation is ad hoc in nature. However, as there are so many possibilities for generated data sets, some flexibility has been lost to facilitate comparison. If further variations to the distribution pattern of the generators are required, these may be simply included in the generation process. Note that linear transforms of the generated data will result in data sets with any desired normal distribution.

A continuing problem with this generation of data is that methods with a natural bias towards normally-distributed data may perform better. However, when used in conjunction with the Voronoi data sets, good performance on data sets from both methods would indicate a reasonably robust method, especially for the comparison of different artificial neural network methods.

7.3 Application of benchmarks

Following on from the previous definitions of two artificial benchmarks, it is sensible to apply them to the comparison of different learning methods. Specifically two different groups of experiments are examined. Firstly, a comparison is made between the results obtained by Quickprop and further trials using pattern presentation and batch back-propagation. Here pattern presentation refers to the updating of the weights after the presentation of a single randomly selected example; whereas batch refers to the updating of the weights after the presentation of all the examples in the training set once only. Secondly, a brief examination is made of some of the modifications to Cascor presented in Part I.

7.3.1 Quickprop and back-propagation

Quickprop was developed to speed the learning of back-propagation-style networks (see §2.3.5). It was assumed that the error surface would approximate a quadratic function, and following on from this a number of 'risky' assumptions are made, and changes developed, to the process of updating the network weights [Fahlman 1988a]. The assumption that the error surface fits a quadratic is valid near the minimum, as may be seen from the expansion of the Taylor series. Along with the development of Quickprop, a number of other modifications were introduced, most notably the activation function offset which stops the derivative of the function ever reaching near zero when the squashing function is set hard on or off. A number of papers have critically examined Quickprop [Lister 1994] and the activation function offset [Adams & Lewis 1995] expressing doubts about their effectiveness on all problems. This is a sensible point to test those criticisms using the benchmarks that have been developed.

Experiments are performed in a similar fashion to those given earlier. Six experiments are considered, testing the differences between pattern presentation and batch back-propagation and Quickprop, and the use of activation function offsets or not. A total of 100 trials of each data set are considered, with 5 percent patience over 20 epochs used as the stopping criteria and 5 nodes in the hidden layer. Back-propagation using the steepest descent algorithm, using both pattern presentation and batch methods, is applied with the parameters outlined in table 7.8. The other Quickprop parameters are identical to those given earlier (see table 7.1). The results given are the averages over the 20 data sets for each generator set and for each data set style: the same data sets as used previously to show the changes in complexity. The actual results are the percentage correct on the unseen test set (see table 7.9) and the number of epochs required for training (see table 7.10), given that there are hard limits of 1000 epochs maximum and 20 epochs minimum of training.

Table 7.8 — Parameters used for pattern presentation and batch back-propagation trials

Parameter	Value
Total hidden nodes	5 nodes
Eta	0.1
Alpha	0
Weight decay	0
Patience percentage	5%
Patience length	20 epochs
Maximum epochs	1000 epochs
Activation functions	Symmetric sigmoid

Generally the activation function offset is not beneficial (see table 7.9). Although it occasionally aids batch back-propagation in finding a better solution, the results for pattern presentation back-propagation and for Quickprop indicate that the inclusion of the offset results in an inferior classifier. These results are consistent with previous theoretical work [Adams & Lewis 1995].

The effectiveness of Quickprop is also called into question. Quickprop is on a par with batch back-propagation when the activation function offset is used, and easily outperforms the batch method when no offsets are used. However, at no stage did Quickprop outperform the pattern presentation back-propagation algorithm, and the results appear to be significantly worse (see §7.3.1.1). Performance differences of around 20 percent are not uncommon, which is not encouraging given that the generalisation performance is the prime goal of training. The stochastic nature of pattern presentation does assist in escaping local minima and plateaus in the feature space, which is an advantage over the batch methods.

Table 7.9 — Comparison of Quickprop (QP), and pattern presentation (PP) and batch (B) back-propagation (BP) on the generated benchmark data sets: results on the unseen test set showing average (Av) and standard deviation (SD) over 20 different data set trials

Data style and gen's	PPBP with offsets		PPBP with no offsets		BBP with offsets		BBP with no offsets		QP with offsets		QP with no offsets	
	Av	SD	Av	SD	Av	SD	Av	SD	Av	SD	Av	SD
Voronoi												
4 & 4	91.1	9.1	96.4	2.7	71.7	13	60.9	6	70.9	12	87.9	8.2
10 & 10	85.1	5	89.3	4.2	61.7	8.5	56.3	4	61.4	7.1	76.3	9.2
20 & 20	77.9	3.3	80.3	3.1	55.6	5.2	53.6	3.6	56.1	4.2	67.3	7.3
Voronoi												
2 & 18	97	1.9	97.1	2.2	88.4	5.3	88.9	4.2	84.6	7.9	93.1	2.8
4 & 16	90.3	3.8	93.9	1.9	75.2	5.7	77.6	4.9	71.1	6.3	86.3	4.1
6 & 14	87	4	90.8	3.4	66.3	6.9	68.43	5.6	64.1	6.8	81.8	4.8
8 & 12	84.6	5.2	88.9	3.7	60.5	7.9	59.1	5.6	59.6	6.8	76	7.1
Normal												
4 & 4	93.8	5.1	93.7	4.7	71.5	14.1	54.2	4.6	71.5	12.2	72.7	10.9
10 & 10	80.7	6.1	82	6.2	56.1	5.6	52	2.5	56.1	4.2	61.8	7.5
20 & 20	69.5	4.3	69.9	4.9	51	1	50.2	0.3	51.4	1.3	53.3	3.2

Table 7.10 — Comparison of Quickprop (QP), and pattern presentation (PP) and batch (B) back-propagation (BP) on the generated benchmark data sets: epoch results showing average (Av) and standard deviation (SD) over 20 different data set trials

Data style and gen's	PPBP with offsets		PPBP with no offsets		BBP with offsets		BBP with no offsets		QP with offsets		QP with no offsets	
	Av	SD	Av	SD	Av	SD	Av	SD	Av	SD	Av	SD
Voronoi												
4 & 4	713.1	302.9	577.5	318.3	999.7	1.6	58	40.4	947.9	155.9	303.3	107.5
10 & 10	269.2	213.4	182.7	201.9	937.7	176.7	42.2	25.2	990.7	21.5	296.9	88.9
20 & 20	81.3	24.3	67.1	21	895.6	244.8	38.1	24.8	980.9	29.8	317.5	104.6
Voronoi												
2 & 18	801.5	227.5	440.7	270.1	947.3	154.3	20.3	0.9	880.6	225.4	488.1	142.8
4 & 16	450.6	245.8	232	189.9	1000	0	21.7	2	1000	0	442.5	122.1
6 & 14	315.3	237.7	150.7	99.2	1000	0	27.6	9.1	1000	0	370.9	121.1
8 & 12	280.3	217.2	133.9	109.1	978.7	91	36	19.2	987.4	31.6	367.2	116.1
Normal												
4 & 4	212.1	235	120.9	207.4	841.5	296.6	38.2	30.1	861.1	227.7	162.9	78.4
10 & 10	103.5	63	64.9	24.7	751.6	318.4	32.5	11	889.6	177.8	154.9	37.2
20 & 20	47.2	7.2	46.6	9.1	317.3	314.9	25.5	6.3	605.6	344.8	135.2	33.2

The speed of training should also be considered (see table 7.10), especially as improving this is the purpose of both Quickprop and the activation function offset. The activation function offset, considering pattern presentation, batch and Quickprop learning, slows training down. The pattern presentation back-propagation and Quickprop trials without the offset are not only faster but, as mentioned previously, the results are much better. Although the batch back-propagation trials with the activation function offset are better in terms of generalisation ability, it is dubious whether the large amount of extra training required is worth the performance increase.

The training speed performance of Quickprop in relation to the back-propagation methods is also comparatively poor. Though the comparison with batch back-propagation is favourable, the performance compared with pattern presentation back-propagation is not good — far from being a major speed improvement. This poor performance may be partially explained by the large size of the data sets. If a training set is doubled by duplication of the examples then batch methods will require double the amount of training presentations to achieve the previous performance level, whereas pattern presentation learning will train with the same effort as before. This does not fully apply in this situation as the examples in the data sets are not duplicated, although they may be clustered in the feature space. The generalisation performance also has to be taken into account, which does not help Quickprop in these comparisons.

7.3.1.1 Significance of results

There are only a limited number of trials presented here on which to base the observations given above, and as such the results should be interpreted carefully. To obtain a measure of confidence in the results presented above statistical tests have been performed to give some indication of whether the means of the samples gathered about each method are distinct. To do this a homoscedastic two-tailed T-tests have been performed on the results of the three distinct Voronoi data set groups involving even numbers of generators: namely 4 and 4, 10 and 10, and 20 and 20 (see table 7.11).

The main feature of the information displayed in table 7.11 is the large number of significant differences between the data set results, on both the number of epochs and the percentage correct on the unseen test set. In examining the percentage correct results it is evident that the QP trials with activation function offsets consistently achieve similar results to that of BBP with offsets. There is minor similarity between QP and BBP with no offsets on the 20 and 20 generator data sets, and between QP with no offsets and PPBP using the 4 and 4 generator data sets. Other than that the probabilities that the corresponding distributions are the same are very small indicating that the results obtained are significant. The results showing similarity are not strong indicators that the results are from the same distribution, apart from the QP and BBP using the activation function offset.

There are also some similarities with regard to the number of epochs, but taking into account the differences shown by the percentages on the unseen test cases, the only significant and consistent difference is again with the QP and BBP results. There are some similarities between PPBP styles of training on this epoch measure, and there is an interesting strong similarity between PPBP with offsets and QP without offsets with regard to training time on the 10 and 10 generator data sets even though the classification performance levels are remarkably different.

Table 7.11 — Results of T-test comparison of means for the 4 and 4, 10 and 10, and 20 and 20 Voronoi generator pair data sets: shown are the values for the test set performance comparison (lower triangular) and the number of epochs of training (upper triangular), where each figure is the probability (to 4 decimal places) that the distributions are the same

	PPBP with offsets	PPBP with no offsets	BBP with offsets	BBP with no offsets	QP with offsets	QP with no offsets
PPBP	N/A	0.1756	0.0001	0	0.0038	0
PPBP NO	0.0175	N/A	0	0	0	0.0008
BBP	0	0	N/A	0	0.1459	0
BBP NO	0	0	0.0018	N/A	0	0
QP	0	0	0.8364	0.0021	N/A	0
QP NO	0.2405	0.0001	0	0	0	N/A
PPBP	N/A	0.1957	0	0	0	0.5958
PPBP NO	0.0062	N/A	0	0.0037	0	0.0261
BBP	0	0	N/A	0	0.1094	0
BBP NO	0	0	0.0141	N/A	0	0
QP	0	0	0.9077	0.0078	N/A	0
QP NO	0.0006	0	0	0	0	N/A
PPBP	N/A	0.0561	0	0	0	0
PPBP NO	0.0269	N/A	0	0.0003	0	0
BBP	0	0	N/A	0	0.1299	0
BBP NO	0	0	0.1568	N/A	0	0
QP	0	0	0.7589	0.0518	N/A	0
QP NO	0	0	0	0	0	N/A

Overall the results obtained, from this indication, are sufficiently significant.

7.3.2 Cascade-Correlation and modifications

Three modifications which are examined are the performance of the independent candidate training, the application of node patience, and the application of pruning within Cascor. Independent candidate training is examined as it is recommended as the best method for training candidate nodes within Cascor (see §3.2). The use of node patience is also examined, as up until now it has been used within the benchmarking experiments presented in this chapter without examining its effect on the training results. It is expected that better classification performance will be achieved with a cost of extra training (see §7.1.3.1). Pruning of connections within Cascor (see §4.2) is also examined as a sensible way of reducing connections.

Three experiments are conducted with the application of Cascor to the generated data sets (see table 7.12): Cascor with node patience; Cascor with node patience and independent candidate training; Cascor with node patience, independent candidate training and connection pruning; and Cascor without node patience and with independent candidate training. In the latter case a maximum of 15 candidate nodes are installed. The rest of the parameters are the same as those outlined in §2.4.1. The pruning performed is absolute level pruning using a saliency level of 0.01 for both candidate and output layer connections.

Table 7.12 — Comparison of Cascor modifications: the average and standard deviation (SD) of the percentage correct are given

Data style and gen's	Standard		Independent		Independent and pruning		No patience	
	Average	SD	Average	SD	Average	SD	Average	SD
Voronoi								
4 and 4	93.3	6.8	97.2	3.7	97.1	3.5	98.3	1.4
10 and 10	87.2	5.8	92.8	5.6	92.3	5.7	95.8	1.7
20 and 20	77	4	84.6	4.1	84	3.9	91.2	2.2
Voronoi								
2 and 18	94	2.9	97.4	1.3	97.5	1.3	98.2	1.3
4 and 16	89	4.6	94.4	2.5	94.3	2.7	96.4	1.7
6 and 14	87.8	5.1	93.7	3.2	93.4	3.3	93.4	11.2
8 and 12	85.5	4.6	92.8	3.1	92.5	3.6	95.5	1.5
Normal								
4 and 4	91.2	5.1	93.4	5.2	93.4	5.2	94.1	4.8
10 and 10	78.9	6	83.8	4.6	83.7	4.6	86.7	3.6
20 and 20	69.3	4.3	71.6	5.1	71.1	5.1	80.1	3.3

The comparison of standard candidate training and independent candidate training confirms the results from §3.2 (see table 7.12). Independent candidate training results in more effective training which ensures that better and more stable candidate nodes are produced; and, with the application of node patience, enables training to continue for longer resulting in a much more complex and effective classifier. The use of pruning does not appear to have a detrimental effect on the classification performance of the final classifier, with most of the results being very similar.

Comparing the application of node patience, it is evident that no node patience and training to install 15 hidden nodes results in better generalisation performance. Further training is conducted without the node patience parameters, and as is expected better results are achieved. Without node patience large amounts of unnecessary overtraining may be performed. Indeed here the choice of limiting the number of candidates installed was made after seeing the results of the node patience trials. If the limit which has been applied throughout this thesis had been used, namely 25 hidden nodes, an extremely large amount of unnecessary training may have been performed.

The results of the independent candidate training are encouraging as they show that Cascor may solve these problems with accuracies closer to the performance levels of C4.5 and 5-NN.

The number of connections required to obtain the final solutions also presents an interesting picture (see figure 7.13). Standard Cascor, which performs less effective training than independent candidate training, requires far fewer connections in the solutions developed, as is expected. Pruning is very effective at reducing the number of connections, and, as mentioned previously, does so without any drop in training performance. The standard deviation results also show that the final number of connections used is much more stable using pruning, indicating more consistent networks. Not using node patience results in usually very large networks which, although better classification performance is obtained, are much more complicated than those obtained by using node patience.

Table 7.13 — Comparison of Cascor methods: the average and standard deviation (SD) of the size of the final theory in connections are given

Data style and gen's	Standard		Independent		Independent and pruning		No patience	
	Average	SD	Average	SD	Average	SD	Average	SD
Voronoi								
4 and 4	50.2	19.7	138.2	60.4	101.8	37.2	183.9	5.4
10 and 10	38.8	11.6	123.4	50.5	82.5	30.3	186	0
20 and 20	29	7.5	75.6	29.4	55.8	17.5	186	0
Voronoi								
2 and 18	25	8.2	68.7	32.1	58.2	21	185.1	2.9
4 and 16	31.5	9.4	96	33.9	74.3	26.7	186	0
6 and 14	40.2	15.7	131.9	56.9	89.7	32.6	180.9	22.8
8 and 12	37.2	8.1	127	42.8	87	27.1	186	0
Normal								
4 and 4	24.6	9.7	32.2	14.8	29	14	185.8	1.5
10 and 10	28.9	5.8	46.3	9	40.9	8	186	0
20 and 20	25.3	6.1	31	9.5	26.9	7.7	186	0

7.3.2.1 *Significance of results*

It is again worth checking the significance of these results. T-tests have been applied to the results of the Cascor trials on the Voronoi data sets with an even number of generators. These calculations are presented below (see table 7.14).

The immediately evident point is the high relationship between the independent patience methods with and without pruning. The figures indicate that the results on the percentage correct are very close, where as the number of connections show only a loose matching. There are a number of similarities between the classification rates, though only very weak, on the 4 and 4 generator data sets. These relationships tend to diminish as the number of generators increase as would be expected, given that there is much more possible variation within the data sets.

The important point is that these relationships — apart from that between the pruned and unpruned independent node patience trials — are very weak. The remainder of the figures suggest a good separation between the distributions of the results around each data set style, and that the results obtained are significant.

Table 7.14 — Results of T-test comparison of means for the 4 and 4, 10 and 10, and 20 and 20 Voronoi generator pair data sets: shown are the values for the test set performance comparison (lower triangular) and the number of connections (upper triangular), where each figure is the probability (to 4 decimal places) that the distributions are the same

	Standard	Independent	Ind Prune	Ind No Pat
Standard	N/A	0	0	0
Independent	0.0284	N/A	0.0017	0.0272
Ind Prune	0.0299	0.9530	N/A	0
Ind No Pat	0.0025	0.2185	0.1796	N/A
Standard	N/A	0	0	0
Independent	0.0035	N/A	0.0035	0
Ind Prune	0.0075	0.7807	N/A	0
Ind No Pat	0	0.0294	0.0127	N/A
Standard	N/A	0	0	0
Independent	0	N/A	0.0137	0
Ind Prune	0	0.6656	N/A	0
Ind No Pat	0	0	0	N/A

7.3.3 Summary

Pattern presentation and batch back-propagation have been compared with the Quickprop algorithm, and the Quickprop update does not seem to yield the expected performance improvements. Likewise the trials performed compared the application of an activation offset, and this is more likely to result in worse generalisation and slower convergence times than the use of standard activation functions without the modification.

The trials conducted here may be criticised in that only simplistic algorithm parameters have been used, with no search for the best possible settings. This may produce effects which are larger than necessary, which is a valid criticism. However, through experience it is assumed that the parameters selected will give reasonable results. Regardless, it is unlikely that such large differences as were shown above could be accounted for by the modification of the algorithm parameters. Indeed, an algorithm which requires such extensive parameter modification is not generally practical.

The modifications to Cascor have also been shown to be effective. The independent candidate training is a substantial improvement over the standard method of candidate training. Although node patience does not necessarily produce better results than training to a set number of hidden nodes, the results indicate that the method is able to stop substantial overtraining and give an indication of how many hidden nodes are actually required. Pruning of connections as developed in §4.2 results in significantly smaller networks without degrading classification performance.

The results for both sets of experiments have also been shown to be statistically significant, often to a very high level, giving confidence to the results presented here.

The methods for generating data sets are shown to be effective for the benchmarking of different artificial neural network methods, including Cascor. They are especially useful as complicated problems to test the need for hidden nodes which are required to solve these problems.

8 Conclusion

The work of this thesis is summarised, and conclusions presented, followed by details of further work to be performed.

Part one of the thesis examines various methods for changing network topology during training. This is motivated by two concerns. It is a non-trivial choice to decide how a network should be structured — if this can be automated, the use of neural networks would become greatly simplified. Furthermore, it is not obvious that a static network will find a solution — allowing a network to change its capacity during training gives an extra degree of freedom. A survey of current literature indicates that increasing the number of hidden nodes and reducing the number of connections between nodes are the most commonly used methods of dynamically altering neural networks structure. Cascor is one algorithm for growing hidden nodes which stands out as being effective and practical. Pruning connections by the use of saliency measures also is effective in reducing network size.

Modifications to Cascor to produce faster training times are examined: in particular, node patience and different methods of training the candidate pool. The effect of these modifications is measured by comparison to standard Cascor on a number of benchmark data sets. Chapter 3 introduces these changes made to the Cascor training mechanism. Node patience is shown to be an effective method for controlling the size of Cascor network. On tasks, such as the LED and Diabetes1 problems, node patience greatly improves the speed of training, reducing the classifier size, and often increasing the generalisation performance. Rollback — the removal of the last few hidden nodes added, after the application of node patience to halt training — is also an effective addition which reduces the size of the final network. Node patience is not a substitute for the use of a validation set of examples used to halt training, but it is an effective technique for halting training when no such validation set exists.

Chapter 3 concludes by introducing modifications to the training of the candidate pool. The independent training of candidates often leads to more effective training resulting in less hidden nodes being required overall. The benefits of using independent candidate training outweigh the chances that a minimal amount of extra training may be performed. Subgroup candidate training also exhibits these benefits to a lesser extent. Summing the candidate correlations, as opposed to finding the maximum, to determine when to stop training the candidate pool, may also perform more effective training but results in greater training times.

Further modifications made to Cascor allow for the reduction of the number of connections used within a network, as examined in Chapter 4. The first section of Chapter 4 shows that the introduction of hidden nodes with limited connection strategies are able to modify the network connection structure by guiding the development of the network. The second section examines the use of connection pruning within Cascor networks, a more principled approach than selecting an arbitrary hidden node connection strategy, whereby many connections may be removed with no detrimental effect to the generalisation performance. The methods of stopping the pruning process are also shown to be effective under these conditions.

The main point which becomes evident through the work presented here is that few of the problems used for benchmarking require the power of the Cascor algorithm to solve them, whether this is due to the classes being separate or overlapping is unclear. This makes comparison of the newly developed methods with standard Cascor very difficult. The second part of this thesis examines the area of benchmarking in more detail, to determine how Cascor, and other artificial neural network methods, may be more effectively tested.

In this part, two aspects of benchmarking are examined: benchmarking by performance on real-world problems, and benchmarking by constructing new problems. It is noted that there is wide scope for the examination of data sets, with a large number of modifications to data sets possible in modelling real-world situations. There is also an obvious trade-off between the two methods: on the one hand realistic data sets are required, and on the other data sets which are complex are required to test learning methods. Chapter 5 started by examining the features of data sets, details of previous work, and the results of some previous benchmarking of Cascor.

Two new real-world problems are then examined in Chapter 6. The aim of this is two-fold: to apply Cascor to the problems as a practical tool for finding solutions, and to see if these problems present features which require the higher-order feature detectors available in multi-layer perceptron styles of networks. The first data set, which involves determining the age of abalone, was not solvable given the information available. What was solvable may be done with very few hidden nodes being introduced into a Cascor network, implying that the data set involves overlapping classes. Only minor improvements are made by the addition of a few hidden nodes. The second data set involves the classification of plays as either Renaissance or Romantic tragedies. This is different from the first data set in that there are a large number of attributes compared with the number of examples available for training, and the number of classes is much smaller. The data set is solvable from the examples available, with a high cross-validation performance level, and this may be done using no hidden nodes in an artificial neural network. A final point illustrated by Chapter 6 is that the classification process is aided greatly by sensible pre-processing. Scaling of the data particularly helps learning in artificial neural networks.

Following this examination of real-world problems, two new methods for generating artificial data sets are examined in Chapter 7, where the complexity of those data sets may be increased as required and as much data as is needed may be generated. Voronoi data sets are shown to be capable of creating data sets of differing complexity. The second method, normal data sets, creates a more realistic setting where examples are not distributed evenly throughout the feature space. This method is also capable of generating data sets of differing complexity where the maximum classification rate may be calculated.

Examples of these data generation methods are then used to compare standard artificial neural network methods, as well as to briefly examine some of the modifications to Cascor developed earlier. Specifically, pattern presentation and batch back-propagation, and Quickprop are examined, and pattern presentation back-propagation is shown to be the fastest and most effective training method on these problems. The use of activation function offsets was also examined and the results indicate that their use is detrimental to learning. Furthermore, the experiments on the Cascor modifications show that independent candidate training produces much more effective networks and that pruning is able to reduce the network sizes without affecting the classification performance. The application of node patience is also examined and, although the results are not as good as no node patience, the use of the method does greatly decrease the training time. At the very least, node patience may be used to obtain an indication of how many hidden nodes are required to solve a problem.

Thus, these methods of generating data sets result in problems which are difficult for Cascor and other artificial neural networks to solve, without being completely unsolvable. They are effective in benchmarking existing and new artificial neural network algorithms.

8.1 Further work

The results of the node patience experiments in Chapter 3 indicate that it would be valuable to evaluate the use of validation sets in the stopping of training through all phases within Cascor — stopping candidate and output layer training as well as the overall network training. Improvements may also be possible by the use of more efficient algorithms within Cascor, such as conjugate gradient methods [Møller 1993; Stone & Lister 1994] for weight training, and OBS [Hassibi & Stork 1992] or PCP [Levin, et al. 1994] for pruning. In fact there has been little work on benchmark comparisons of pruning methods, and such a study could also include comparisons of the methods outlined in §4.2 for controlling the pruning of connections. A detailed study of the performance of weight decay in removing connections should also be included. Further investigation of error functions for classification or regression may also be considered [Lister & Stone 1995].

It has been noted that Cascor has difficulties in solving function interpolation problems [Fahlman 1993; Freeman 1994; Adams & Waugh 1995]. Prior to the commencement of the work undertaken in this thesis there was an expectation that Cascade2 would become available, an algorithm which trains candidates with the network error directly, similar to previous work [Littmann & Ritter 1992], and which solves the difficulties experienced by Cascor in function approximation problems. However, there has been no public release of any such software or any description of the algorithm. Nevertheless, the changes to Cascor presented in this thesis may be easily applied to any architecture which is similar to Cascor in the way it structures networks, even if the training mechanisms differ slightly. Further work may be performed on problems which have been identified, such as Quickprop training [Squires & Shavlik 1991] and the freezing of weights [Ash 1989; Adams & Waugh 1995].

The work on real-world data sets may be continued. Specifically, there is a large scope for the addition of further attributes to the abalone data, such as site information, to aid the classification process. In particular, information about the site giving general characteristics and information about ocean currents may play an important role — thus requiring information about Tasmania's weather systems. Further work may be performed on the authorship data set with regard to, for example, trying to separate Shakespeare from his contemporaries. However this may be difficult without further examples to train on.

Extension of current work on artificial benchmarks may also be continued: firstly including the generation of multiple classes, multiple attribute data sets and altering the number of examples available for training. This may then be further expanded by the examination of the effect of noise, redundancy, irrelevancy and so on. Furthermore, with the normal data set construction, the generation of different variance matrices may be examined — including different methods of generating the standard deviations. The generation of further data sets more suitable for solution with neural networks may also be examined. In particular the random generation of neural network structures may be used to generate random problems solvable by neural networks. Again this needs to be constrained to stop a combinatorial explosion of possible data sets.

Finally, applying the theory of Voronoi diagrams to generating other styles of data sets may be an interesting area for further study. For example, function approximation problems and temporal data may be developed. Existing real-world data sets may also be examined, using Voronoi diagrams to reverse engineer a data set to find features and to estimate the data set complexity.

Appendices

A Node patience results

This appendix presents the full results of the experiments on node patience. The details presented here include the percentage correct on the unseen test set, the number of hidden nodes required and the amount of training required as measured by the the number of connection crossings. These results are the median results of 100 trials.

Table A.1.1 — Percentage correct on the unseen test set for the Monks1 data set

Patience length		1	2	3	4	5
Patience percentage	1	97.92	97.69	97.69	97.45	97.92
	2	97.57	97.92	97.69	97.92	97.69
	3	97.69	97.92	97.92	97.92	97.92
	4	97.92	97.69	97.69	97.69	97.92
	5	97.92	97.69	97.69	97.92	97.69

Table A.1.2 — Hidden nodes required for the Monks1 data set

Patience length		1	2	3	4	5
Patience percentage	1	1	1	1	1	1
	2	1	1	1	1	1
	3	1	1	1	1	1
	4	1	1	1	1	1
	5	1	1	1	1	1

Table A.1.3 — Connection crossings (millions) required for the Monks1 data set

Patience length		1	2	3	4	5
Patience percentage	1	4.5	4.4	4.5	4.5	4.4
	2	4.5	4.5	4.5	4.4	4.4
	3	4.5	4.5	4.5	4.5	4.4
	4	4.5	4.5	4.5	4.5	4.4
	5	4.5	4.5	4.5	4.4	4.2

Table A.2.1 — Percentage correct on the unseen test set for the Monks2 data set

Patience length		1	2	3	4	5
Patience percentage	1	99.77	99.77	99.77	99.77	99.77
	2	99.77	99.77	99.77	99.77	99.77
	3	99.77	99.77	99.77	99.77	99.77
	4	99.77	99.77	99.77	99.77	99.77
	5	99.77	99.77	99.77	99.77	99.77

Table A.2.2 — Hidden nodes required for the Monks2 data set

Patience length		1	2	3	4	5
	1	1	1	1	1	1
Patience	2	1	1	1	1	1
percentage	3	1	1	1	1	1
	4	1	1	1	1	1
	5	1	1	1	1	1

Table A.2.3 — Connection crossings (millions) required for the Monks2 data set

Patience length		1	2	3	4	5
	1	5.8	5.8	6.0	6.0	5.9
Patience	2	6.0	5.8	5.9	5.9	5.9
percentage	3	5.8	5.9	5.9	5.9	5.9
	4	6.0	5.8	5.9	5.9	5.8
	5	5.9	5.9	5.9	5.8	5.8

Table A.3.1 — Percentage correct on the unseen test set for the Monks3 data set

Patience length		1	2	3	4	5
	1	88.77	88.16	88.31	89	88.89
Patience	2	89	88.66	88.19	88.66	88.08
percentage	3	88.19	88.19	88.77	88.1	88.54
	4	88.13	88.54	88.08	88.66	88.43
	5	88.43	88.54	88.31	87.96	88.19

Table A.3.2 — Hidden nodes required for the Monks3 data set

Patience length		1	2	3	4	5
	1	2	2	2	2	2
Patience	2	2	2	2	2	2
percentage	3	2	2	2	2	2
	4	2	2	2	2	2
	5	2	2	2	2	2

Table A.3.3 — Connection crossings (millions) required for the Monks3 data set

Patience length		1	2	3	4	5
	1	16.3	17.2	16.5	16.4	16.5
Patience	2	15.9	16.2	16.4	16.6	16.1
percentage	3	16	16.8	15.9	15.8	16.2
	4	16.7	16.1	16.7	16.9	16.5
	5	17.5	16.8	16.1	17.3	16.1

Table A.4.1 — Percentage correct on the unseen test set for the Two Spirals data set

Patience length		1	2	3	4	5
Patience percentage	1	95.83	95.83	95.83	95.83	95.83
	2	95.83	95.57	95.83	95.83	96.09
	3	53.12	95.83	95.83	95.83	95.83
	4	53.12	95.83	96.35	96.35	95.83
	5	53.12	95.83	95.83	95.83	95.83
	6	53.12	96.09	95.83	95.83	95.83
	7	53.12	93.75	95.83	96.35	96.09
	8	53.12	61.98	95.83	96.35	95.83
	9	53.12	61.98	95.83	95.83	96.35
	10	53.12	61.98	95.31	96.35	96.09

Table A.4.2 — Hidden nodes required for the Two Spirals data set

Patience length		1	2	3	4	5
Patience percentage	1	12	12	12	12	12
	2	12	12	12	12	12
	3	1	12	12	12	13
	4	1	12	12	12.5	12
	5	1	12	13	12	13
	6	1	12	12	12	12
	7	1	12	12	12	12
	8	1	2	13	12	12
	9	1	2	12	12	12
	10	1	2	12	12	12

Table A.4.3 — Connection crossings (millions) required for the Two Spirals data set

Patience length		1	2	3	4	5
Patience percentage	1	126.5	124	124.8	127.7	126.2
	2	125.4	126.4	122.6	122.3	125.5
	3	1.7	126	125.1	125.9	131.1
	4	1.7	125.7	126.9	130.5	125.6
	5	1.7	126.4	128.1	126.4	131.6
	6	1.7	123.6	127.3	126.3	122.1
	7	1.7	114.3	127.3	124.5	120.5
	8	1.7	4.3	129.3	127.2	120.9
	9	1.7	4.4	127.1	121.1	126
	10	1.6	4.5	128.9	125.4	120.9

Table A.5.1 — Percentage correct on the unseen test set for the Double Helix data set

Patience length		1	2	3	4	5
	1	100	100	100	100	100
Patience	2	100	100	100	100	100
percentage	3	100	100	100	100	100
	4	100	100	100	100	100
	5	100	100	100	100	100

Table A.5.2 — Hidden nodes required for the Double Helix data set

Patience length		1	2	3	4	5
	1	6	6	6	6	6
Patience	2	6	5	6	6	6
percentage	3	6	6	6	6	5
	4	6	6	5	6	6
	5	6	6	6	6	6

Table A.5.3 — Connection crossings (millions) required for the Double Helix data set

Patience length		1	2	3	4	5
	1	66.4	67.1	67.7	68.2	63.3
Patience	2	69	59.4	68.6	65.3	67.1
percentage	3	66	67.1	68.7	66.2	62.4
	4	68.7	65.1	61.8	66.4	64.1
	5	65.2	63.3	65.2	62.7	63.6

Table A.6.1— Percentage correct on the unseen test set for the LED data set

Patience length		1	2	3	4	5
	1	72	72.1	72	71.8	72
Patience	2	72	72.2	72.2	72.2	72.2
percentage	3	72	72.2	72.2	72	72
	4	72	72.2	72.2	72	72
	5	72	72.2	72.2	72.2	72

Table A.6.2 — Hidden nodes required for the LED data set

Patience length		1	2	3	4	5
	1	1	6	10	15	20
Patience	2	1	2	6	7	12
percentage	3	1	2	3	4	10
	4	1	2	3	4	5
	5	1	2	3	4	5

Table A.6.3 — Connection crossings (millions) required for the LED data set

Patience length		1	2	3	4	5
	1	108.8	605.2	1254.8	2225.9	3279
Patience	2	108.1	199.3	615.4	768.4	1552.5
percentage	3	109.7	196.2	284.8	398.4	1176.1
	4	107	196.6	288.5	390.8	511.3
	5	108.9	194.9	291.4	391	503.7

Table A.7.1 — Percentage correct on the unseen test set for the Cancerl data set

Patience length		1	2	3	4	5
	1	95.98	95.98	95.98	95.98	96.55
Patience	2	95.98	96.55	95.4	95.4	95.98
percentage	3	95.98	95.98	95.98	95.98	95.98
	4	95.98	95.98	95.98	95.98	95.98
	5	95.98	95.4	95.98	96.55	95.98

Table A.7.2 — Hidden nodes required for the Cancerl data set

Patience length		1	2	3	4	5
	1	5	5	5	5	5
Patience	2	5	5	5	5	5
percentage	3	5	5	5	5	5
	4	5	5	5	5	5
	5	5	5	5	5	5

Table A.7.3 — Connection crossings (millions) required for the Cancerl data set

Patience length		1	2	3	4	5
	1	176.2	175.4	178.1	166.9	174.9
Patience	2	174.2	176.5	172.9	171.8	181.7
percentage	3	176.2	184.4	173.4	175.5	176.8
	4	176.5	173.7	173.2	174.5	173.8
	5	174.9	172.8	167.7	181.5	173.5

Table A.8.1 — Percentage correct on the test set for the Diabetes1 data set

Patience length		1	2	3	4	5
	1	69.27	69.27	68.75	69.27	68.75
Patience	2	71.61	68.75	68.75	69.27	68.75
percentage	3	75.52	68.75	69.27	69.27	69.27
	4	76.04	68.75	69.79	69.79	68.23
	5	76.04	69.79	68.75	69.27	69.27

Table A.8.2 — Hidden nodes required for the Diabetes1 data set

Patience length		1	2	3	4	5
	1	25	25	25	25	25
Patience	2	24.5	25	25	25	25
percentage	3	3	25	25	25	25
	4	2	25	25	25	25
	5	2	25	25	25	25

Table A.8.3 — Connection crossings (millions) required for the Diabetes1 data set

Patience length		1	2	3	4	5
	1	1880.8	1955.3	1959	1959.7	1969.8
Patience	2	1798.5	1975.8	1941	1979.8	1941.5
percentage	3	104.2	1946.1	1944.6	1946.1	1956.7
	4	63.3	1903.4	1963.8	1932.8	1964.9
	5	62.2	1898.8	1972.4	1964.8	1960

Table A.9.1 — Percentage correct on the unseen test set for the Glass1 data set

Patience length		1	2	3	4	5
	1	66.04	66.04	66.04	66.04	66.04
Patience	2	66.04	66.98	66.04	66.04	66.04
percentage	3	66.04	66.04	67.92	66.98	66.04
	4	66.04	66.04	66.04	66.04	64.15
	5	66.04	66.04	66.04	66.04	64.15

Table A.9.2 — Hidden nodes required for the Glass1 data set

Patience length		1	2	3	4	5
	1	15	16	16	16	17
Patience	2	15	17	16	17	16
percentage	3	14.5	16	16	17	16
	4	14	15	16	16	16
	5	13	16	16	17	18

Table A.9.3 — Connection crossings (millions) required for the Glass1 data set

Patience length		1	2	3	4	5
	1	379.0	414.9	411.29	403.0	444.7
Patience	2	373.7	412.5	392.1	403.9	421.0
percentage	3	336.9	399.6	385.3	416.8	402.1
	4	339.3	363.6	396.9	413.9	403.8
	5	314.6	385.7	396.8	432	426.8

B Candidate training results

This appendix presents the full results of the experiments on candidate training methods. The details presented here include the percentage correct on the unseen test set, the number of hidden nodes required and the amount of training required as measured by the the number of connection crossings. These results are the median results of 100 trials.

A number of abbreviations are used within the tables. 'Cand' refers to the number of candidates; 'Cand training' refers to the candidate training style; and 'HL patience' refers to the hidden layer patience period. The four candidate training methods are referred using 'Stand' for standard, 'Ind' for independent, 'Sum' for summation, and 'Sub' for subgroup candidate training methods.

B.1 Single activation function

Table B.1.1.1 — Percentage correct on the unseen test set for the Monks1 data set

HL patience		10			20			50		
Cand training		Stand	Ind	Sum	Stand	Ind	Sum	Stand	Ind	Sum
Cand	4	98.61	98.61	98.38	97.92	98.38	98.15	97.92	98.15	98.38
	10	97.69	97.92	97.92	97.69	97.92	97.69	97.92	97.92	97.92
	20	97.92	97.69	97.69	97.92	97.69	97.69	97.69	97.92	97.92

Table B.1.1.2 — Hidden nodes required for the Monks1 data set

HL patience		10			20			50		
Cand training		Stand	Ind	Sum	Stand	Ind	Sum	Stand	Ind	Sum
Cand	4	1	1	1	1	1	1	1	1	1
	10	2	1	1	1	1	1	1	1	1
	20	2	1	1	1	1	1	1	1	1

Table B.1.1.3 — Connection crossings (millions) required for the Monks1 data set

HL patience		10			20			50		
Cand training		Stand	Ind	Sum	Stand	Ind	Sum	Stand	Ind	Sum
Cand	4	1.89	1.52	1.67	1.85	1.9	1.96	2.36	2.42	2.5
	10	3.83	2.46	3	3.23	3.38	3.57	4.52	4.78	5.01
	20	5.38	3.98	5.13	5.49	6	6.47	7.91	8.71	9.35

Table B.1.2.1 — Percentage correct on the unseen test set for the Monks2 data set

HL patience		10			20			50		
Cand training		Stand	Ind	Sum	Stand	Ind	Sum	Stand	Ind	Sum
Cand	4	99.77	99.77	99.54	99.54	99.54	99.77	99.77	99.77	99.77
	10	99.19	99.77	99.54	99.54	99.54	99.77	99.77	99.77	99.77
	20	99.07	99.77	99.54	99.54	99.54	99.77	99.77	99.77	99.77

Table B.1.2.2 — Hidden nodes required for the Monks2 data set

HL patience		10			20			50		
Cand training		Stand	Ind	Sum	Stand	Ind	Sum	Stand	Ind	Sum
Cand	4	3	1	1	1	1	1	1	1	1
	10	3	1	1	1	1	1	1	1	1
	20	4	1	1	1	1	1	1	1	1

Table B.1.2.3 — Connection crossings (millions) required for the Monks2 data set

HL patience		10			20			50		
Cand training		Stand	Ind	Sum	Stand	Ind	Sum	Stand	Ind	Sum
Cand	4	4.08	2.3	2.43	2.37	2.84	2.75	3.01	3.92	4.02
	10	7.6	3.42	4.67	4.15	5.71	5.71	5.9	8.29	8.49
	20	13.69	5.76	8.32	7.35	10.59	10.58	10.63	15.87	15.73

Table B.1.3.1 — Percentage correct on the unseen test set for the Monks3 data set

HL patience		10			20			50		
Cand training		Stand	Ind	Sum	Stand	Ind	Sum	Stand	Ind	Sum
Cand	4	88.54	88.43	88.89	88.19	87.96	88.19	88.66	88.66	88.43
	10	88.66	88.89	89.12	88.43	88.89	88.43	88.31	87.96	88.43
	20	88.66	89	89.35	88.89	88.54	89.35	88.19	88.43	88.43

Table B.1.3.2 — Hidden nodes required for the Monks3 data set

HL patience		10			20			50		
Cand training		Stand	Ind	Sum	Stand	Ind	Sum	Stand	Ind	Sum
Cand	4	3	3	3	3	2.5	2	2	2	2
	10	4	2	2	2	2	2	2	2	2
	20	4	2	2	2	2	2	2	2	2

Table B.1.3.3 — Connection crossings (millions) required for the Monks3 data set

HL patience		10			20			50		
Cand training		Stand	Ind	Sum	Stand	Ind	Sum	Stand	Ind	Sum
Cand	4	9.03	7.57	7.48	8.57	7.71	8.3	9.47	9.21	9.83
	10	13.61	9.36	9.64	11.81	10.71	11.34	16.49	14.98	16.41
	20	18.46	12.01	14.02	17.88	16.4	18.01	28.66	24.61	27.24

Table B.1.4.1 — Percentage correct on the unseen test set for the Two Spirals data set

HL patience		10			20			50		
Cand training		Stand	Ind	Sum	Stand	Ind	Sum	Stand	Ind	Sum
Cand	4	59.9	85.94	64.06	95.31	94.79	94.79	95.83	95.31	95.31
	10	59.38	95.31	63.02	95.31	94.79	95.31	96.09	95.83	95.83
	20	60.16	95.31	65.63	95.31	95.31	95.57	95.31	95.31	95.57

Table B.1.4.2— Hidden nodes required for the Two Spirals data set

HL patience		10			20			50		
Cand training		Stand	Ind	Sum	Stand	Ind	Sum	Stand	Ind	Sum
Cand	4	25	25	25	16	16	16	14	14	14
	10	25	22	25	14	14	14	12	12	12
	20	25	16	25	13.5	13	13	12	11	11

Table B.1.4.3— Connection crossings (millions) required for the Two Spirals data set

HL patience		10			20			50		
Cand training		Stand	Ind	Sum	Stand	Ind	Sum	Stand	Ind	Sum
Cand	4	22.1	41.7	24.1	62.7	55.7	64.7	76.4	70.5	81.2
	10	29.9	71	33.3	92.2	77.2	96.9	125.7	112.8	131.4
	20	43.2	75.3	50.9	152.5	120.3	156.8	207.9	182.1	221.8

Table B.1.5.1 — Percentage correct on the unseen test set for the Double Helix data set

HL patience		10			20			50		
Cand training		Stand	Ind	Sum	Stand	Ind	Sum	Stand	Ind	Sum
Cand	4	49.25	50	48.88	100	100	100	100	100	100
	10	50	82.75	50	100	100	100	100	100	100
	20	50	100	49.75	100	100	100	100	100	100

Table B.1.5.2 — Hidden nodes required for the Double Helix data set

HL patience		10			20			50		
Cand training		Stand	Ind	Sum	Stand	Ind	Sum	Stand	Ind	Sum
Cand	4	25	25	25	6	6	6	6	6	6
	10	25	25	25	6	6	6	6	6	6
	20	25	12	25	6	6	6	6	6	6

Table B.1.5.3 — Connection crossings (millions) required for the Double Helix data set

HL patience		10			20			50		
Cand training		Stand	Ind	Sum	Stand	Ind	Sum	Stand	Ind	Sum
Cand	4	49.86	50.96	49.85	28.82	31.81	33.54	38.96	39.7	41.41
	10	67.98	74.36	68.21	47.43	49.08	54.05	65.38	71.51	72.44
	20	98.8	72.62	98.91	74.18	83.23	92.52	111.14	116.06	133.11

Table B.1.6.1 — Percentage correct on the unseen test set for the Cancer1 data set

HL patience		10			20			50		
Cand training		Stand	Ind	Sum	Stand	Ind	Sum	Stand	Ind	Sum
Cand	4	98.28	98.28	98.28	96.26	95.40	95.98	95.98	95.98	95.98
	10	98.28	97.13	98.28	95.98	96.55	95.98	95.98	96.55	95.98
	20	98.28	96.55	98.28	96.55	96.55	95.98	95.98	95.98	95.98

Table B.1.6.2 — Hidden nodes required for the Cancer1 data set

HL patience		10			20			50		
Cand training		Stand	Ind	Sum	Stand	Ind	Sum	Stand	Ind	Sum
Cand	4	25	25	25	7	6	6	5	5	5
	10	25	25	25	6	5	5	5	5	5
	20	25	18	25	6	5	5	5	4	5

Table B.1.6.3 — Connection crossings (millions) required for the Cancer1 data set

HL patience		10			20			50		
Cand training		Stand	Ind	Sum	Stand	Ind	Sum	Stand	Ind	Sum
Cand	4	94.2	97.1	95	96.3	81.5	87.5	99.5	92.8	100.6
	10	134.1	146.3	134.4	142.1	115.9	133.8	170.6	156.3	182.3
	20	192.3	208.2	192.2	231.2	184	235.9	336.8	269.9	323.6

B.2 Multiple activation functions

Table B.2.1.1 — Percentage correct on the unseen test set for the Monks1 data set

HL patience		10			20			50		
Cand training		Stand	Sub	Ind	Stand	Sub	Ind	Stand	Sub	Ind
Cand	4	99.77	100	100	100	100	100	100	99.65	99.77
	10	99.31	99.42	99.77	99.54	99.31	99.88	100	99.77	100
	20	98.61	98.38	98.61	98.61	98.73	98.84	98.15	98.61	98.84

Table B.2.1.2 — Hidden nodes required for the Monks1 data set

HL patience		10			20			50		
Cand training		Stand	Sub	Ind	Stand	Sub	Ind	Stand	Sub	Ind
Cand	4	1	1	1	1	1	1	1	1	1
	10	1	1	1	1	1	1	1	1	1
	20	1	1	1	1	1	1	1	1	1

Table B.2.1.3 — Connection crossings (millions) required for the Monks1 data set

HL patience		10			20			50		
Cand training		Stand	Sub	Ind	Stand	Sub	Ind	Stand	Sub	Ind
Cand	4	1.58	1.57	1.52	1.72	1.86	1.92	2.27	2.54	2.58
	10	2.53	2.29	2.56	3.1	3.19	3.46	4.31	4.47	4.99
	20	4.15	3.45	4.26	5.13	5.34	5.97	7.83	8.05	9.17

Table B.2.2.1 — Percentage correct on the unseen test set for the Monks2 data set

HL patience		10			20			50		
Cand training		Stand	Sub	Ind	Stand	Sub	Ind	Stand	Sub	Ind
Cand	4	99.54	99.77	98.61	99.54	99.77	98.61	99.07	99.54	99.77
	10	100	99.77	99.77	99.77	99.77	99.77	99.77	99.77	99.77
	20	99.77	99.77	99.77	99.77	99.77	99.77	99.77	99.77	99.77

Table B.2.2.2 — Hidden nodes required for the Monks2 data set

HL patience		10			20			50		
Cand training		Stand	Sub	Ind	Stand	Sub	Ind	Stand	Sub	Ind
Cand	4	2	1	2	2	2	2	2	2	1
	10	1	1	1	1	1	1	1	1	1
	20	1	1	1	1	1	1	1	1	1

Table B.2.2.3 — Connection crossings (millions) required for the Monks2 data set

HL patience		10			20			50		
Cand training		Stand	Sub	Ind	Stand	Sub	Ind	Stand	Sub	Ind
Cand	4	4.4	2.38	5	4.76	5.11	6.17	8.01	7.95	4.48
	10	3.54	3.19	3.91	4.39	4.94	5.66	6.17	7.14	8.24
	20	5.67	4.26	6.64	7.14	7.97	10.23	10.59	11.77	15.46

Table B.2.3.1 — Percentage correct on the unseen test set for the Monks3 data set

HL patience		10			20			50		
Cand training		Stand	Sub	Ind	Stand	Sub	Ind	Stand	Sub	Ind
Cand	4	88.31	89.12	89	88.77	88.89	88.66	88.19	88.19	88.66
	10	88.54	88.31	88.31	87.73	88.54	88.54	87.96	88.89	88.77
	20	88.77	88.89	88.19	88.19	89.12	88.89	88.77	87.96	88.43

Table B.2.3.2 — Hidden nodes required for the Monks3 data set

HL patience		10			20			50		
Cand training		Stand	Sub	Ind	Stand	Sub	Ind	Stand	Sub	Ind
Cand	4	3	3	3	3	3	3	3	3	2
	10	3	2	3	3	3	2	2	3	2
	20	3	3	2	2	2	2	2	2	2

Table B.2.3.3 — Connection crossings (millions) required for the Monks3 data set

HL patience		10			20			50		
Cand training		Stand	Sub	Ind	Stand	Sub	Ind	Stand	Sub	Ind
Cand	4	7.76	7.91	7.48	7.94	8.79	8.1	11	10.68	9.27
	10	11.62	8.94	10.31	12.72	13.13	11.77	17.69	19.15	16.17
	20	15.19	13.64	13.4	19.01	16.61	16.97	27.42	27.41	25.2

Table B.2.4.1 — Percentage correct on the unseen test set for the Two Spirals data set

HL patience		10			20			50		
Cand training		Stand	Sub	Ind	Stand	Sub	Ind	Stand	Sub	Ind
Cand	4	92.71	94.79	94.27	94.53	95.31	94.79	95.05	94.79	94.79
	10	95.83	95.31	94.79	94.79	95.57	95.83	95.31	95.05	95.31
	20	96.09	95.31	95.83	95.83	95.83	95.83	95.83	95.83	94.79

Table B.2.4.2 — Hidden nodes required for the Two Spirals data set

HL patience		10			20			50		
Cand training		Stand	Sub	Ind	Stand	Sub	Ind	Stand	Sub	Ind
Cand	4	25	22	22	19	18	18	17	16	16
	10	24	19	18.5	17	16	15.5	14	14	14
	20	20	17	15	13	14	13	12	12	12

Table B.2.4.3 — Connection crossings (millions) required for the Two Spirals data set

HL patience		10			20			50		
Cand training		Stand	Sub	Ind	Stand	Sub	Ind	Stand	Sub	Ind
Cand	4	74	67.1	71	82.8	74.3	76.1	97.2	84.6	92.8
	10	114.6	76.9	79.1	116.7	106.7	98.9	157.9	151.5	139.9
	20	151.8	95.4	84.6	146	151	130.3	136.9	206.3	205.8

Table B.2.5.1 — Percentage correct on the unseen test set for the Double Helix data set

HL patience		10			20			50		
Cand training		Stand	Sub	Ind	Stand	Sub	Ind	Stand	Sub	Ind
Cand	4	100	100	100	100	100	100	100	100	100
	10	100	100	100	100	100	100	100	100	100
	20	100	100	100	100	100	100	100	100	100

Table B.2.5.2 — Hidden nodes required for the Double Helix data set

HL patience		10			20			50		
Cand training		Stand	Sub	Ind	Stand	Sub	Ind	Stand	Sub	Ind
Cand	4	13	9	9	7	7	7	7	7	7
	10	10	8	7	7	6	6	6	6	6
	20	9	7	6	6	6	6	6	6	6

Table B.2.5.3 — Connection crossings (millions) required for the Double Helix data set

HL patience		10			20			50		
Cand training		Stand	Sub	Ind	Stand	Sub	Ind	Stand	Sub	Ind
Cand	4	54.84	39.92	35.67	35.25	37.18	38.08	47.29	46.41	48.05
	10	64.91	36.83	37.24	54.94	50.91	50.43	74.51	70.93	75.91
	20	84.78	43.72	42.32	74.77	75.72	78.44	117.95	110.3	121.2

Table B.2.6.1 — Percentage correct on the unseen test set for the Cancer1 data set

HL patience		10			20			50		
Cand training		Stand	Sub	Ind	Stand	Sub	Ind	Stand	Sub	Ind
Cand	4	97.70	96.55	96.55	96.55	96.55	96.55	96.55	96.26	96.55
	10	97.7	97.13	96.55	96.55	97.13	96.55	95.98	96.55	96.55
	20	97.7	97.13	97.13	96.55	96.55	96.55	96.55	96.55	96.55

Table B.2.6.2 — Hidden nodes required for the Cancer1 data set

HL patience		10			20			50		
Cand training		Stand	Sub	Ind	Stand	Sub	Ind	Stand	Sub	Ind
Cand	4	25	14	13	6	6	6	5	5	5
	10	25	23	9	6	5	5	5	5	5
	20	25	25	7	5	5	5	5	5	5

Table B.2.6.3 — Connection crossings (millions) required for the Cancer1 data set

HL patience		10			20			50		
Cand training		Stand	Sub	Ind	Stand	Sub	Ind	Stand	Sub	Ind
Cand	4	122.7	112.1	116.5	86.1	77.6	78.9	99.6	97.3	92.9
	10	163.6	188	85.3	128.7	115.6	116.3	171.7	167.8	169.6
	20	226.7	247.4	89.6	201.2	192	178.8	305.4	307.7	275.4

C Limited candidate node results

This appendix details the full results from the experiments conducted on the introduction of limited candidate nodes. Six of the nine problems are used: the Monks problems, Two Spirals, Double Helix and Cancer1. The details presented here include the percentage correct on the unseen test set, the total and number of limited hidden nodes required, the total number of connections needed, and the amount of training required as measured by the the number of connection crossings. These results are the median results of 100 trials. Where there is no applicable result, the abbreviation 'N/A' is given.

Table C.1.1 — Percentage correct on the unseen test set for the Monks1 data set

Connection strategy	Half pool	Forcing 10%	Forcing 50%	Full pool
Full connection	N/A	N/A	N/A	97.92
Layered	97.8	97.92	97.92	97.92
Minimal shortcuts	97.69	97.69	97.69	97.69
Two random connections	97.92	98.38	97.69	98.38
Completely random	97.92	98.26	99.42	98.15

Table C.1.2 — Hidden nodes added (and limited hidden nodes added) for the Monks1 data set

Connection strategy	Half pool	Forcing 10%	Forcing 50%	Full pool
Full connection	N/A	N/A	N/A	1 (0)
Layered	1 (0)	1 (0)	1 (0)	1 (0)
Minimal shortcuts	1 (0)	1 (0)	1 (0)	1 (0)
Two random connections	1 (0)	1 (0)	1 (0)	5 (5)
Completely random	1 (0)	1 (1)	1 (1)	1 (1)

Table C.1.3 — Total network connections for the Monks1 data set

Connection strategy	Half pool	Forcing 10%	Forcing 50%	Full pool
Full connection	N/A	N/A	N/A	50
Layered	50	50	50	50
Minimal shortcuts	50	50	50	50
Two random connections	50	50	50	57
Completely random	50	48	48	48

Table C.1.4 — Connection crossings (millions) for the Monks1 data set

Connection strategy	Half pool	Forcing 10%	Forcing 50%	Full pool
Full connection	N/A	N/A	N/A	4.8
Layered	4.8	4.8	4.8	4.8
Minimal shortcuts	4.7	4.8	4.9	4.8
Two random connections	3.1	3.1	3.2	8.4
Completely random	4.0	4.1	4.3	3.3

Table C.2.1 — Percentage correct on the unseen test set for the Monks2 data set

Connection strategy	Half pool	Forcing 10%	Forcing 50%	Full pool
Full connection	N/A	N/A	N/A	99.77
Layered	99.77	99.77	99.77	99.77
Minimal shortcuts	99.77	99.77	99.77	99.77
Two random connections	99.77	99.77	99.77	75.23
Completely random	99.77	99.77	99.77	99.77

Table C.2.2 — Hidden nodes added (and limited hidden nodes added) for the Monks2 data set

Connection strategy	Half pool	Forcing 10%	Forcing 50%	Full pool
Full connection	N/A	N/A	N/A	1 (0)
Layered	1 (0)	1 (0)	1 (0)	1 (0)
Minimal shortcuts	1 (0)	1 (0)	1 (0)	1 (0)
Two random connections	1 (0)	1 (0)	1 (0)	25 (25)
Completely random	1 (0)	1 (0)	1 (1)	1 (1)

Table C.2.3 — Total network connections for the Monks2 data set

Connection strategy	Half pool	Forcing 10%	Forcing 50%	Full pool
Full connection	N/A	N/A	N/A	50
Layered	50	50	50	50
Minimal shortcuts	50	50	50	50
Two random connections	50	50	50	157
Completely random	50	50	50	50

Table C.2.4 — Connection crossings (millions) for the Monks2 data set

Connection strategy	Half pool	Forcing 10%	Forcing 50%	Full pool
Full connection	N/A	N/A	N/A	8.3
Layered	8.4	8.3	8.3	8.3
Minimal shortcuts	8.3	8.4	8.1	8.2
Two random connections	5.0	4.9	5.0	66.6
Completely random	7.0	7.1	7.6	7.3

Table C.3.1 — Percentage correct on the unseen test set for the Monks3 data set

Connection strategy	Half pool	Forcing 10%	Forcing 50%	Full pool
Full connection	N/A	N/A	N/A	87.96
Layered	88.31	87.96	88.43	88.31
Minimal shortcuts	87.96	88.19	88.43	88.19
Two random connections	88.89	88.89	88.43	89.81
Completely random	88.43	88.31	89.12	88.77

Table C.3.2 — Hidden nodes added (and limited hidden nodes added) for the Monks3 data set

Connection strategy	Half pool	Forcing 10%	Forcing 50%	Full pool
Full connection	N/A	N/A	N/A	2 (0)
Layered	2 (1)	2 (1)	2 (1)	2 (1)
Minimal shortcuts	2 (0)	2 (0)	2 (0)	2 (0)
Two random connections	2 (0)	2 (0)	2 (0)	11.5 (11.5)
Completely random	2 (0)	2 (1)	3 (2.5)	2.5 (2)

Table C.3.3 — Total network connections for the Monks3 data set

Connection strategy	Half pool	Forcing 10%	Forcing 50%	Full pool
Full connection	N/A	N/A	N/A	69
Layered	69	68	68	68
Minimal shortcuts	69	69	69	69
Two random connections	69	69	69	89.5
Completely random	69	69	69	69

Table C.3.4 — Connection crossings (millions) for the Monks3 data set

Connection strategy	Half pool	Forcing 10%	Forcing 50%	Full pool
Full connection	N/A	N/A	N/A	15.0
Layered	14.6	14.5	14.6	14.2
Minimal shortcuts	15.1	14.7	14.7	14.8
Two random connections	10.8	10.6	11.1	28.4
Completely random	12.3	13.3	16.7	11.7

Table C.4.1 — Percentage correct on the unseen test set for the Two Spirals data set

Connection strategy	Half pool	Forcing 10%	Forcing 50%	Full pool
Full connection	N/A	N/A	N/A	95.83
Layered	95.83	96.09	95.31	78.13
Minimal shortcuts	95.83	95.83	96.35	96.88
Two random connections	95.83	95.83	95.83	83.33
Completely random	95.83	95.05	94.79	95.57

Table C.4.2 — Hidden nodes added (and limited hidden nodes added) for the Two Spirals data set

Connection strategy	Half pool	Forcing 10%	Forcing 50%	Full pool
Full connection	N/A	N/A	N/A	12 (0)
Layered	12 (2)	13 (4)	14 (9)	25 (24)
Minimal shortcuts	13 (1)	13 (2)	14 (7)	19.5 (17.5)
Two random connections	14 (1)	13 (1)	14 (3)	25 (24)
Completely random	13 (2)	13 (4)	15 (9)	15 (10)

Table C.4.3 — Total network connections for the Two Spirals data set

Connection strategy	Half pool	Forcing 10%	Forcing 50%	Full pool
Full connection	N/A	N/A	N/A	132
Layered	132	143	151.5	131
Minimal shortcuts	149	139	127	122
Two random connections	163	148.5	149	131
Completely random	145	139.5	153	157.5

Table C.4.4 — Connection crossings (millions) for the Two Spirals data set

Connection strategy	Half pool	Forcing 10%	Forcing 50%	Full pool
Full connection	N/A	N/A	N/A	112.8
Layered	106.9	111.2	124.9	88.9
Minimal shortcuts	95.3	92.4	97.8	95.3
Two random connections	87.6	87.2	90	71.8
Completely random	98.2	97.5	119.3	96.1

Table C.5.1 — Percentage correct on the unseen test set for the Double Helix data set

Connection strategy	Half pool	Forcing 10%	Forcing 50%	Full pool
Full connection	N/A	N/A	N/A	100
Layered	100	100	100	100
Minimal shortcuts	100	100	100	100
Two random connections	100	100	100	100
Completely random	100	100	100	100

Table C.5.2 — Hidden nodes added (and limited hidden nodes added) for the Double Helix data set

Connection strategy	Half pool	Forcing 10%	Forcing 50%	Full pool
Full connection	N/A	N/A	N/A	6 (0)
Layered	5 (1)	5 (3)	6 (4)	6 (5)
Minimal shortcuts	6 (1)	6 (3)	6 (3.5)	6 (4)
Two random connections	5 (0)	5 (2)	6 (3)	10 (10)
Completely random	5 (0)	5 (2)	6 (4)	6 (2)

Table C.5.3 — Total network connections for the Double Helix data set

Connection strategy	Half pool	Forcing 10%	Forcing 50%	Full pool
Full connection	N/A	N/A	N/A	59
Layered	48	44	44	44
Minimal shortcuts	55	49	49	49
Two random connections	48	47	48	58
Completely random	48	47	52	50.5

Table C.5.4 — Connection crossings (millions) for the Double Helix data set

Connection strategy	Half pool	Forcing 10%	Forcing 50%	Full pool
Full connection	N/A	N/A	N/A	71.5
Layered	60.2	54.4	61.2	52.9
Minimal shortcuts	65.5	61.5	69.2	61.2
Two random connections	50.8	47.9	58.5	77.2
Completely random	55.9	54.9	67.7	48.4

Table C.6.1 — Percentage correct on the unseen test set for the Cancer1 data set

Connection strategy	Half pool	Forcing 10%	Forcing 50%	Full pool
Full connection	N/A	N/A	N/A	96.55
Layered	95.98	96.55	96.55	96.55
Minimal shortcuts	96.55	95.98	95.98	96.55
Two random connections	95.98	95.98	95.98	95.4
Completely random	95.98	96.55	95.98	95.98

Table C.6.2 — Hidden nodes added (and limited hidden nodes added) for the Cancer1 data set

Connection strategy	Half pool	Forcing 10%	Forcing 50%	Full pool
Full connection	N/A	N/A	N/A	5 (0)
Layered	5 (1)	5 (3)	5 (4)	5 (4)
Minimal shortcuts	5 (1)	5 (2)	5 (3)	5 (3)
Two random connections	5 (0)	5 (0)	5.5 (2)	12 (12)
Completely random	5 (1)	5 (2)	6 (5)	5 (4)

Table C.6.3 — Total network connections for the Cancer1 data set

Connection strategy	Half pool	Forcing 10%	Forcing 50%	Full pool
Full connection	N/A	N/A	N/A	90
Layered	87	86	80	80
Minimal shortcuts	87	85	84	84
Two random connections	90	90	80	80
Completely random	87	83.5	85	81

Table C.6.4 — Connection crossings (millions) for the Cancer1 data set

Connection strategy	Half pool	Forcing 10%	Forcing 50%	Full pool
Full connection	N/A	N/A	N/A	156.3
Layered	160.8	168.4	169.6	159.4
Minimal shortcuts	160.8	172.3	169	162.4
Two random connections	123	122.2	139	174.9
Completely random	143.9	148.6	174.8	125.1

D Pruning results

The following section details tests on the absolute, percentage and relative methods of pruning connections from Cascor networks. When required pruning is performed separately on the candidate and output layers: Monks3, Cancer1, Diabetes1, Glass1 and LED problems do not require the addition of candidate nodes. The details presented here include the percentage correct on the unseen test set ('Tst %'), the total number of connections needed ('Con'), and the amount of training required as measured by the the number of connection crossings ('CCs') measured in millions. These results are the median results of 100 trials. Where there is no applicable result, the abbreviation 'N/A' is given.

Table D.1.1 — Results of Monks1 problem on candidate node pruning

	Absolute			Percentage			Relative		
Prune	Tst %	Con	CCs	Tst %	Con	CCs	Tst %	Con	CCs
None	97.92	50	4.78	N/A	N/A	N/A	N/A	N/A	N/A
0.0	97.92	48	6.61	N/A	N/A	N/A	N/A	N/A	N/A
0.01	97.92	45	6.2	97.69	46	6.3	98.15	43	10.31
0.02	97.80	43	5.98	97.69	45	5.98	95.72	81	81.53
0.03	97.92	42	5.97	97.69	43	5.93	75.23	116	192.30
0.04	98.15	40	5.64	97.92	42	6.0	75.23	114	194.07
0.05	97.92	40	5.57	97.92	41	5.82	75.23	113	193.08
0.06	97.92	39	5.56	97.69	41	5.76	75.23	113	194.03
0.07	98.03	39	5.58	98.15	40	5.74	75.23	112	193.72
0.08	98.38	39	5.51	97.92	40	5.69	75.23	111	194.3
0.09	98.38	40	5.97	98.15	40	5.58	75.23	111	194.2
0.1	97.8	43.5	11.13	98.15	39	5.52	75.23	111	194.8

Table D.1.2 — Results of Monks1 problem on output layer pruning

	Absolute			Percentage			Relative		
Prune	Tst %	Con	CCs	Tst %	Con	CCs	Tst %	Con	CCs
None	97.92	50	4.78	N/A	N/A	N/A	N/A	N/A	N/A
0.0	99.42	44	4.94	N/A	N/A	N/A	N/A	N/A	N/A
0.01	99.54	44	4.91	99.31	44	4.95	99.54	43.5	4.9
0.02	99.77	43.5	4.86	99.31	44	4.96	99.54	44	4.91
0.03	99.54	44	4.89	99.54	44	4.91	99.54	44	4.92
0.04	99.54	44	4.96	99.54	44	4.9	99.07	44	4.89
0.05	99.54	44	4.86	99.54	44	4.84	99.54	44	4.98
0.06	99.77	43	4.88	99.54	44	4.96	99.54	44	4.87
0.07	99.54	43	4.85	99.54	44	4.88	99.54	44	4.94
0.08	99.54	43	4.92	99.77	44	4.85	99.54	44	4.8
0.09	99.77	43	4.98	99.31	44	4.92	99.54	44	4.93
0.1	99.54	43	5.01	99.54	44	4.97	99.54	44	4.91

Table D.2.1 — Results of Monks2 problem on candidate node pruning

	Absolute			Percentage			Relative		
Prune	Tst %	Con	CCs	Tst %	Con	CCs	Tst %	Con	CCs
None	99.77	50	8.29	N/A	N/A	N/A	N/A	N/A	N/A
0.0	99.77	50	10.95	N/A	N/A	N/A	N/A	N/A	N/A
0.01	99.77	47	10.77	99.77	49	10.70	86.69	143	386.57
0.02	99.77	44	10.83	99.77	47	10.94	63.43	119	354.19
0.03	99.77	43	10.74	99.77	45	10.75	62.27	112	348.78
0.04	98.61	43	11.43	99.77	44	10.70	62.27	111	349.53
0.05	98.38	52.5	21.95	99.77	43	10.79	62.27	110	350.57
0.06	94.68	67	46.65	99.77	43	10.54	62.27	110	350.80
0.07	91.90	87	91.42	99.77	43	10.90	62.27	109	352.20
0.08	89.58	112	186.51	98.61	50	21.24	62.27	109	352.71
0.09	80.67	136	377.53	98.38	49.5	20.69	62.27	109	353.68
0.1	69.44	128	371.58	96.88	60.5	34.48	62.27	109	352.48

Table D.2.2 — Results of Monks2 problem on output layer pruning

	Absolute			Percentage			Relative		
Prune	Tst %	Con	CCs	Tst %	Con	CCs	Tst %	Con	CCs
None	99.77	50	8.29	N/A	N/A	N/A	N/A	N/A	N/A
0.0	99.77	40	8.43	N/A	N/A	N/A	N/A	N/A	N/A
0.01	99.77	40	8.64	99.77	40	8.87	99.77	40	8.89
0.02	99.77	40	8.71	99.77	40	8.68	99.77	40	8.50
0.03	99.77	40	8.65	99.77	40.5	8.82	99.77	40	8.78
0.04	99.77	40	8.65	99.77	40.5	8.75	99.77	40	8.85
0.05	99.77	40	8.66	99.77	40	8.70	99.77	40	8.75
0.06	99.77	40	8.77	99.77	40	8.96	99.77	40	8.57
0.07	99.77	40	8.72	99.77	40	8.74	99.77	40	8.68
0.08	99.77	40	8.83	99.77	40	8.92	99.77	40	8.89
0.09	99.77	39	8.79	99.77	40	8.76	99.77	40	8.57
0.1	99.77	39	8.77	99.77	40	8.55	99.77	40	8.76

Table D.3 — Results of Monks3 problem on output layer pruning

	Absolute			Percentage			Relative		
Prune	Tst %	Con	CCs	Tst %	Con	CCs	Tst %	Con	CCs
None	96.76	32	0.73	N/A	N/A	N/A	N/A	N/A	N/A
0.0	96.99	25	1.13	N/A	N/A	N/A	N/A	N/A	N/A
0.01	96.99	25	1.10	96.99	25	1.13	97.22	25	1.11
0.02	96.99	25	1.15	97.22	25	1.08	96.99	25	1.12
0.03	96.99	25	1.12	97.22	25	1.11	96.99	25	1.11
0.04	97.11	25	1.10	96.99	25	1.14	96.99	25	1.12
0.05	96.99	24	1.11	97.22	24.5	1.11	97.11	25	1.12
0.06	96.99	25	1.12	96.99	25	1.16	96.99	25	1.14
0.07	96.99	24	1.10	96.99	25	1.13	96.99	25	1.10
0.08	97.22	24	1.14	97.11	25	1.15	96.99	25	1.11
0.09	97.22	24	1.10	96.99	25	1.11	96.99	25	1.14
0.1	97.22	24	1.10	96.99	25	1.09	97.22	24	1.11

Table D.4.1 — Results of Two Spirals problem on candidate node pruning

	Absolute			Percentage			Relative		
Prune	Tst %	Con	CCs	Tst %	Con	CCs	Tst %	Con	CCs
None	95.83	132	112.8	N/A	N/A	N/A	N/A	N/A	N/A
0.0	95.83	124.5	130.0	N/A	N/A	N/A	N/A	N/A	N/A
0.01	95.83	104	120.0	95.83	115	129.5	95.31	99	124.9
0.02	95.83	100	128.6	95.31	107.5	115.6	95.31	96.5	163.4
0.03	95.83	96.5	133.2	95.57	106.5	124.7	94.79	106	256.5
0.04	96.61	93.5	139.7	95.83	102.5	121.6	93.23	111	338.6
0.05	95.83	95	151.3	95.31	101.5	122.1	87.24	104.5	361.8
0.06	95.83	92	149.7	95.83	102	125.3	82.29	100	368.1
0.07	95.83	93.5	177.6	95.83	96	123.9	79.17	98	355.6
0.08	94.79	94	189.9	95.57	96	126.2	72.14	93	365.6
0.09	94.79	98.5	219.3	95.83	95	126.3	70.83	91	358.0
0.1	94.79	101.5	260.1	95.83	99.5	134.4	66.15	88	354

Table D.4.2 — Results of Two Spirals problem on output layer pruning

	Absolute			Percentage			Relative		
Prune	Tst %	Con	CCs	Tst %	Con	CCs	Tst %	Con	CCs
None	95.83	132	112.8	N/A	N/A	N/A	N/A	N/A	N/A
0.0	95.31	131	104.4	N/A	N/A	N/A	N/A	N/A	N/A
0.01	95.83	131	109.8	95.31	147	116.4	95.83	139	113.7
0.02	95.83	131	110.7	95.83	132	113.6	95.31	132	116.3
0.03	95.31	131	110.7	95.83	131	108.4	95.31	131	110.3
0.04	95.31	131	111.9	95.83	132	107.5	95.31	131	106.8
0.05	95.83	132	112.8	95.57	131.5	110.1	95.57	131	112.3
0.06	95.83	131	111.8	95.83	132	108.5	95.31	146.5	117.0
0.07	95.83	132	110.6	95.83	146.5	115.9	95.31	132	114.0
0.08	95.83	138.5	116.9	95.83	131	106.0	95.83	132	113.0
0.09	95.31	130	110.1	95.31	131	109.2	95.83	131.5	111.1
0.1	95.83	130.5	111.1	95.31	130	100.5	95.83	131	110.8

Table D.5.1 — Results of Double Helix problem on candidate node pruning

	Absolute			Percentage			Relative		
Prune	Tst %	Con	CCs	Tst %	Con	CCs	Tst %	Con	CCs
None	100	59	71.5	N/A	N/A	N/A	N/A	N/A	N/A
0.0	100	48	75.19	N/A	N/A	N/A	N/A	N/A	N/A
0.01	99.5	46.5	76.41	100	48	74.65	100	49.5	81.86
0.02	99.5	50	82.95	100	48	76.11	99.5	46.5	85.25
0.03	100	50	83.93	100	50	77.08	100	48	112.84
0.04	100	48	84.85	99.75	50	80.02	100	57	190.74
0.05	100	48	85.49	99.13	49	81.84	99.5	84	456.30
0.06	100	47	86.25	99.63	50	82.65	98.88	95	543.24
0.07	100	47	88.28	100	48	82.83	95.75	95	566.72
0.08	100	47	91.37	99.5	46	80.98	97.38	94	583.94
0.09	100	47	93.47	99.25	47.5	86.46	87.63	92	594.75
0.1	100	52	106.58	100	48.5	86.08	80.5	91	607.40

Table D.5.2 — Results of Double Helix problem on output layer pruning

	Absolute			Percentage			Relative		
Prune	Tst %	Con	CCs	Tst %	Con	CCs	Tst %	Con	CCs
None	100	59	71.5	N/A	N/A	N/A	N/A	N/A	N/A
0.0	100	58	67.68	N/A	N/A	N/A	N/A	N/A	N/A
0.01	100	48	64.17	100	58.5	68.24	100	59	69.43
0.02	100	59	69.74	100	59	69.52	100	48	63.21
0.03	100	59	70.00	100	59	70.20	100	59	70.63
0.04	100	57	66.70	100	59	70.00	100	59	70.37
0.05	100	59	68.34	100	58	69.86	100	59	69.76
0.06	100	59	68.45	100	59	68.51	100	59	72.86
0.07	100	48	62.08	100	58	69.72	100	56.5	68.63
0.08	100	48	65.99	100	59	70.54	100	59	69.17
0.09	100	59	69.44	100	58	68.06	100	59	70.36
0.1	100	58	68.91	100	59	69.67	100	59	69.16

Table D.6 — Results of LED problem on output layer pruning

	Absolute			Percentage			Relative		
Prune	Tst %	Con	CCs	Tst %	Con	CCs	Tst %	Con	CCs
None	72	80	31.7	N/A	N/A	N/A	N/A	N/A	N/A
0.0	72.4	71.5	47.4	N/A	N/A	N/A	N/A	N/A	N/A
0.01	72.4	71	47.0	72.2	71	47.2	72.4	72	47.9
0.02	72.4	71	46.7	72.3	71	46.8	72.2	71	46.8
0.03	72.4	71	47.2	72.2	71	46.8	72.4	71	47.1
0.04	72.3	72	46.4	72.2	72	46.8	72.4	71	47.6
0.05	72.4	71	46.7	72.2	71	47.1	72.4	72	47.1
0.06	72.2	72	47.3	72.4	72	47.5	72.2	71	46.9
0.07	72.2	71	47.0	72.3	72	47.3	72.4	71	47.1
0.08	72.4	71	46.6	72.4	71	47.0	72.4	71	46.8
0.09	72.4	72	48.2	72.2	72	46.8	72.4	71	47.1
0.1	72.3	71	47.3	72.4	71	47.1	72.4	71	46.8

Table D.7 — Results of Cancer1 problem on output layer pruning

	Absolute			Percentage			Relative		
Prune	Tst %	Con	CCs	Tst %	Con	CCs	Tst %	Con	CCs
None	98.28	20	2.99	N/A	N/A	N/A	N/A	N/A	N/A
0.0	98.28	20	4.00	N/A	N/A	N/A	N/A	N/A	N/A
0.01	98.28	20	3.97	98.28	20	3.98	98.28	20	3.93
0.02	98.28	20	3.95	98.28	20	3.97	98.28	20	4.08
0.03	98.28	20	4.03	98.28	20	4.01	98.28	20	4.03
0.04	98.28	20	3.97	98.28	20	4.04	98.28	20	4.03
0.05	98.28	20	4.05	98.28	20	4.01	98.28	20	3.96
0.06	98.28	20	3.88	98.28	20	3.95	98.28	20	3.97
0.07	98.28	20	4.01	98.28	20	4.03	98.28	20	4.07
0.08	98.28	20	4.05	98.28	20	3.96	98.28	20	4.02
0.09	98.28	20	4.07	98.28	20	4.07	98.28	20	3.99
0.1	98.28	20	4.09	98.28	20	4.02	98.28	20	4.02

Table D.8 — Results of Diabetes1 problem on output layer pruning

	Absolute			Percentage			Relative		
Prune	Tst %	Con	CCs	Tst %	Con	CCs	Tst %	Con	CCs
None	77.08	18	3.40	N/A	N/A	N/A	N/A	N/A	N/A
0.0	77.08	18	4.49	N/A	N/A	N/A	N/A	N/A	N/A
0.01	77.08	18	4.55	77.08	18	4.67	77.08	18	4.59
0.02	77.08	18	4.53	77.08	18	4.62	77.08	18	4.59
0.03	77.08	18	4.58	77.08	18	4.55	77.08	18	4.59
0.04	77.08	18	4.51	77.08	18	4.42	77.08	18	4.58
0.05	77.08	18	4.6	77.08	18	4.4	77.08	18	4.45
0.06	77.08	18	4.48	77.08	18	4.62	77.08	18	4.47
0.07	77.08	18	4.5	77.08	18	4.53	77.08	18	4.57
0.08	77.08	18	4.51	77.08	18	4.53	77.08	18	4.58
0.09	77.08	18	4.62	77.08	18	4.57	77.08	18	4.52
0.1	77.08	18	4.59	77.08	18	4.54	77.08	18	4.56

Table D.9 — Results of Glass1 problem on output layer pruning

	Absolute			Percentage			Relative		
Prune	Tst %	Con	CCs	Tst %	Con	CCs	Tst %	Con	CCs
None	66.04	60	3.13	N/A	N/A	N/A	N/A	N/A	N/A
0.0	64.15	58	4.33	N/A	N/A	N/A	N/A	N/A	N/A
0.01	66.04	58	4.34	65.09	58	4.21	66.04	58	4.23
0.02	66.04	58	4.37	66.04	58	4.18	66.04	58	4.39
0.03	66.04	58	4.3	66.04	58	4.3	66.04	58	4.28
0.04	66.04	57	4.36	66.04	58	4.27	64.15	57	4.21
0.05	66.04	57	4.28	66.04	57.5	4.29	66.04	57	4.33
0.06	65.09	57	4.11	66.04	58	4.28	66.04	58	4.28
0.07	66.04	57	4.22	66.04	58	4.33	64.15	58	4.23
0.08	65.09	57	4.38	66.04	57	4.3	66.04	57.5	4.21
0.09	66.04	57	4.26	66.04	58	4.34	66.04	58	4.18
0.1	66.04	57	4.31	66.04	58	4.25	66.04	58	4.25

E TasCas — a Cascade-Correlation simulator

This is the User and System Manual for the TasCas Cascade-Correlation artificial neural network simulator. The system options, data format, output and error are described, along with details relevant to the code structure and assumptions made in the development of the package. Examples are presented throughout.

This is a slightly abridged version of the full technical report TR95-9, from the Department of Computer Science at the University of Tasmania. Part one of the technical report, the User Manual, entails §E.2 to §E.5, part two entails §E.6 to §E.8, and the appendices to the technical report are presented in §E.A to §E.D.

E.1 Introduction

This document outlines the various facilities and structure of the TasCas Cascade-Correlation (Cascor) simulator, version 4.0, developed at the University of Tasmania. In writing this manual it was assumed that the reader has a fair understanding of the Cascor algorithm [Fahlman & Lebiere 1989] and artificial neural networks in general. The details in this text specifically relate to the simulator, and where required, references are given to relevant literature.

TasCas implements the Cascor algorithm, relying on Quickprop [Fahlman 1988a] for the actual weight training, and using the C4.5 data set format [Quinlan 1993a] for training and test sets. This format has been extended to allow for continuous-valued outputs (see §E.A). The simulator has had many features added to the original algorithm [Fahlman & Lebiere 1989].

This document is divided into two major parts. Part one is the User Manual which details information necessary to use the system. This includes an overview of the data format, the possible inputs to the system, the simulator output, and possible simulator errors. Part two is the System Manual detailing how the code is structured, any assumptions made during development, and planned future improvements to the system. The first two sections of this part should be consulted before making any modifications to the code.

The code is written in ANSI C with few assumptions beyond the standard libraries (any non-ANSI C code is detailed in §E.7). There are no requirements for special path names. So far the TasCas system has been successfully compiled on an IBM RS/6000, Sun system and DEC Alpha machine, using IBM's xlc, Sun's acc, and DEC's cc compilers respectively, as well as Gnu's gcc. Note, however, that this package has been developed as a by-product of thesis work. It is stable, but not polished or complete code.

E.2 Network input I — data file

The program uses Quinlan's C4.5 data format [Quinlan 1993a] which requires the files '<filestem>.names', '<filestem>.data' and '<filestem>.test' for the data information file, the training set file and the test set file respectively. The name of the data files (the filestem) is given via the command line directly after calling the executable. Data sets are read in from standard text files. It is not necessary to provide a test set — it is only required to give a measure of the network's generalisation on unseen cases. If the file is not present, no error will occur.

The '.names' file contains the names of the final classifications, or an indication of a regression problem, and details about the attribute values. For example, consider the following contents of a '.names' file:

```
Red, Green, Blue.

Length: continuous.
Size: small, medium, large.
```

This indicates that the data files '.data' and '.test' — if the latter exists — contain examples with two attributes — Length and Size — being classified into three classes — Red, Green and Blue. The three classes are encoded as three output nodes — the network is trained to give a high value for an output node when an example of the corresponding class is given. If there are only two classes, they are still encoded as two output nodes.

The first attribute, Length, is a continuous numeric value and so is encoded as one input to the network. The second attribute, Size, is an unordered discrete variable and is encoded as three separate inputs, each input corresponding to an individual attribute value. When a particular input value is received the corresponding node is set high and the rest are set low. The only exception is when the attribute is binary-valued — then only one input to the network is used, whereby a high node value represents one attribute value, and a low node value represents the other. A high value is encoded as a 1, and a low value as a –1, for discrete attribute values.

There are two further styles of attributes, for example:

```
Width: ignore.
Height: discrete 8.
```

The first may be simply used to avoid having to remove information from the data sets: by setting the attribute to 'ignore' the information is read and ignored. The second is also for unordered discrete variables, whereby the actual attribute values are not specified, and are simply read from the data.

Examples in the '.data' and '.test' files have the same format: a list of comma separated attribute values followed by the actual classification. For example, from the above information, examples may have the following form:

```
2.0,        Small,      Red
5.43,       Medium,     Green
-1.0,       Medium,     Blue
```

Any white space is acceptable between values, and comments may be added to the end of a line by placing a vertical bar ' | ' before the comment. Everything following the bar on that line will be ignored. This applies to all the data files.

Regression problems have a slightly different format. The term 'continuous' is used in place of the classes to indicate that a regression problem with one output is being described. At this stage only one regression output is allowed under this implementation. This is a partial implementation of the full Extended Quinlan format as outlined in §E.A.

E.3 Network input II — simulator options

TasCas uses command line options for the setting of the network parameters. The output of the simulator is directed to standard output, apart from errors (directed to standard error) and the final network weights. There are default settings for all options. Currently the standard default values ensure that all Boolean options are false, hence they will not be used without being set. For example, by default no output is produced unless specifically requested by the user. The standard numeric default values for the options outlined in §E.3.1 to E.3.7 are listed in tables E.1 and E.2 (see the header information when running the system for completely up-to-date information).

Table E.1 — Default values for candidate and output layer training parameters

Parameter	Candidate Value	Output Value
Eta	1.0	0.35
Mu	1.75	1.75
Weight decay	0	0
Minimum pruning sensitivity	0.01	0.01
Pruning patience percentage	0.03	0.01
Patience percentage	0.03	0.01
Patience length	50	50
Epoch limit	500	500
Activation function offset	0.0	0.1

Table E.2 — Default values for network training parameters.

Parameter	Value	Parameter	Value
Number of candidates	4	Network patience percentage	0.02
Candidate node limit	25	Network patience length	1
Special node force factor	1.1	Network trials	1
Number of random connections	2	Percentage allowable error bits	0.0
Default activation function	Sigmoid	Error threshold	0.4
Default connection strategy	Full	Expected value buffer	0.0
		Allowable regression error	0.001

The command to run the simulator is then as follows:

```
tascas <filestem> [options]
```

Substitute the name of the simulator executable for 'tascas'. The filestem is the name of the data files (see §E.2) and the network weight file (see §E.4.3).

The following points refer to the options listed below:

- when two options are in conflict, the latter option has priority;
- the order of the options may have an effect on the values given to a trial (see §E.3.3.3);
- the actual option flags are given below within parentheses where '#' represents an integer value, '#.#' represents a floating point number and brackets indicate an optional input;
- when there are options which may be employed on both candidate and output training, the convention is to use the same letters with the output training flags in lower case and the candidate training flags in upper case;
- with the output options an upper case flag provides more information as opposed to a lower case flag; and
- percentages given to flags are in decimal point form, for example a value of 0.1 is regarded as ten percent.

The output layer is fixed to use symmetric sigmoid functions as the activation functions for each output node when performing classification tasks, and linear activation functions for regression problems. Linear activation functions are not allowed in the hidden layer.

A tabulated summary of the options is given in §E.B.

E.3.1 Weight training options (Quickprop)

The following options alter the standard training parameters:

eta	learning rate for candidate (-E#.#) and output (-e#.#) nodes;

mu	maximum growth factor (-M#.# and -m#.#) — no weight step is allowed to be greater in magnitude than mu times the previous step for that weight; and
offset	offset given to activation functions (-O#.# and -o#.#) — the derivatives for the candidate activation functions do not usually include an offset as this confuses the correlation machinery, but an offset is included on the output layer by default.

The weight updates are performed using the Quickprop algorithm with activation function offsets where required [Fahlman 1988a]. There is no particular reason for using Quickprop other than that historically Cascor has used this algorithm for weight updates.

E.3.2 Stopping training

There are two distinct levels of stopping training within Cascor: stopping the training of a candidate or output layer, and stopping training of the entire network.

E.3.2.1 Stopping layer training

The following (-s) options are used to stop training of network layers:

patience percentage	if the percentage error improvement has been less than this value over patience length epochs then stop training (-sP#.# and -sp#.#);
patience length	length allowed for percentage change to occur (-sL# and -sl#); and
maximum epochs	maximum epochs during layer training phase (-sM# and -sm#).

The above options are described more fully in [Fahlman & Lebiere 1989]. Note that the minimum number of epochs for candidate training is two epochs rather than one — allowing for an initial epoch which simply generates the correlations.

E.3.2.2 Stopping network training

The following (-S) options are used to stop overall network training either directly, or indirectly by modifying the expected outputs and error bit threshold:

node maximum	the maximum number of candidate nodes which can be installed (-Sm#) — setting this does not force the use of node patience;

node percentage	use and possibly set node based patience percentage (-Sp[#.#]) — halts overall network training by using patience with the number of nodes installed as the time period;
node length	node patience length in the number of hidden nodes used (-Sl#);
rollback	remove redundant hidden nodes after training stopped by node patience (-Sr) — simply done by removing the nodes and retraining the output layer;
errors	the maximum allowable error for regression problems (-Se#.#);
error bits	set the percentage of allowable error bits (-Sb#.#) where the total error bits are the number of outputs over all examples incorrectly classified — this is a measure of correctness used to halt the training of classification networks;
error threshold	this sets the allowable distance away from the required result that a training example output value can be without being recorded as an error (-St#.#) when using the number of error bits to stop training of classification networks; and
expected value	ability to change the expected value range or buffer of the sigmoids (-Sx#.#) — for example a value of 0.1 would change the expected values from the output layer sigmoids from -0.5 and 0.5 to -0.4 and 0.4 (note the error threshold is adjusted so that the threshold remains the same regardless of the expected value).

The node patience options are described in detail in §3.1. The errors option is only used for regression style problems, and the error bits, error threshold and expected value options are only used in classification problems.

E.3.3 Candidate training controls and options

The following (-c) options are available for candidate training — most are additions to standard Cascor. The first set involve general candidate training:

candidate total	the number of nodes in the candidate pool (-cn#)
individual patience	train candidate nodes using patience on each individual node, rather than all candidates in the pool (-cI);

subgroup patience	(homogeneous patience) train candidate nodes using patience on each subgroup of similar nodes, rather than all candidates (-cH);
summation	the candidate pool or sub-pool is trained on the summation of the correlation scores rather than the maximum (-cS); and
force usage	non-default nodes (see below) are forced by a percentage factor (-cF#.#), for example -cF1.1 adds an extra ten percent of their correlation to the non-default nodes.

Standard, individual and subgroup training are alternatives, with standard candidate training being the default and individual candidate training having the highest priority.

The following two sections detail the options for changing the activation functions and the connection strategy of the candidate nodes. Most of the options allow for an optional integer to be included to specify the number of nodes of that particular style that are required. If the specified node total is greater than the number of nodes in the pool, the pool size is increased. If the number is less than the pool size the rest of the nodes will be of the default (connection and activation) style. If the optional number is not included the default node style is altered. Examples will be presented in §E.3.3.3.

E.3.3.1 *Setting candidate activation functions*

These options are used to set the candidate pool activation functions:

Gaussian	add Gaussian nodes (-cg[#]);
sigmoid	add symmetric sigmoids (-cs[#]);
tanh	add tanh functions (-ct[#]);
asymmetric	add asymmetric sigmoids (-ca[#]); and
distributed	distributes the activation varieties about the candidates (-cD[#]) — the order of preference is Gaussian, sigmoid, tanh and asymmetric sigmoid.

E.3.3.2 *Setting candidates with limited connections*

These options allow the setting of the connection strategies of the candidate pool:

full connections	nodes with full connections (-cf[#]);
form layers	nodes with no connection to the previous layer (-cl[#]);
minimum shortcuts	nodes with minimal shortcuts — the only connections to the hidden node are those from immediately the inputs and the immediately previous hidden node (-cm[#]);
random weights	randomly connected nodes (-cr[#]);

total random	number of random connections (-cR# where # equalling zero means a random number of connections); and
distributed	distributes the connection varieties about the candidates (-cd[#]) — the order of preference is full, layered, minimal shortcuts, and random connection nodes.

The above options are described more fully in §4.1. The distribution of activation functions and connection strategies means that, for example, if a candidate pool of ten nodes with distributed activation functions is required, three will be Gaussian, three will be symmetric sigmoids, two will be tanh functions and two will be asymmetric sigmoids.

E.3.3.3 Examples and notes

Consider the following example:

```
-cn20 -cl -cf5
```

This sets the candidate pool to contain twenty nodes, the default connection strategy to layered and five of the nodes have full connections. Consider another example:

```
-cs -cD -cn20
```

This sets the default node style to be sigmoids, distributes the activation functions within the candidate pool — the default size is assumed to be four nodes in the pool — leading to one node of each type, then increases the number of default nodes to total seventeen. The following example distributes the node activation types evenly among the twenty candidate nodes:

```
-cs -cn20 -cD
```

This illustrates how the ordering of the options is important. The system is slightly more difficult to use than others which could have been devised, but it is extremely flexible.

Finally there is no intelligent distribution of activation functions and connection strategies implemented. So if both features are distributed, the result will not be a mixture of all connection strategies with all activation functions. Rather, a quarter of the nodes will have the same activation function and connection strategy, and these will change at the same time to differing activation functions and connection strategies. For example

```
-cn20 -cd -cD
```

will give five nodes with Gaussian activation functions and with full connections, five with symmetric sigmoids and a layered connection strategy, and so on.

E.3.4 Pruning and weight reduction

The following (-p) options are additions to standard Cascor for both candidate and output layer training. Standard weight decay is also included (even though it is not strictly a pruning method):

Karnin pruning	use absolute Karnin [Karnin 1990] pruning for connections, removing connections with an estimated error (or sensitivity) less than the pruning level (-pK#.# and -pk#.#);
percentage change	rather than absolute values when used in conjunction with the above options, prune on the percentage change in the error (-pC and -pc);
patience pruning	use patience percentage change in the error value (the patience length is not needed) to control pruning, after pruning all zero and negative saliency connections (-pP#.# and -pp#.#);
every output	prune the output layer at the end of every output layer training phase as opposed to at the end of network training (-pe);
weight decay	reduce weights by adding a term to the error function (-pD#.# and -pd#.#); and
small decay	use decay term which reduces smaller weights more than larger weights relative to the standard weight decay (-pS and -ps).

The weight decay terms are added to the slope during the Quickprop update. Standard weight decay is:

$$\text{decay} \cdot w_{ji}$$

where w_{ji} is the layer weight. The small weight decay term is:

$$\frac{\text{decay} \cdot w_{ji}}{\left(1 + w_{ji}^2\right)^2}$$

This term gives a smaller decay for larger weights relative to the previous decay term. See [Fahlman & Lebiere 1989] for more details on weight decay, [Hertz, et al. 1991] for more details on the small weight decay term, and §4.2 for more details on Karnin pruning within Cascor.

E.3.5 Obtaining network results

The following (-w) options are also available for obtaining results from the simulator (note the term 'output' here refers to writing to 'stdout', and 'write' refers to writing to a text file):

header information	output training parameters (-wh);
final	output final network results (-wf);
summary	output summary statistics of all runs (-ws);
full summary	output the full results of each network in (tab separated) tabular form (-wS) for multiple trials;
weights	write out the weights to '<filestem>.wei' (-ww);
connections	output which network connections exist (-wn);
examples	output final results for each data set example (-we);
matrix	output the final confusion matrix for a single network (-wm);
vectors	output the final output layer vectors for each example (-wv);
best vectors	output the best output layer vectors for each example, obtained from the network at the end of an output layer training phase where the error on the training set is smallest (-wb);
epoch training	output progress after each epoch of training (-wT);
node training	output progress after each training phase (-wt);
epoch correlation	output correlations of pool after each epoch during training (-wC); and
node correlation	output correlations after hidden node training is completed (-wc).

The above options are described more fully in §E.4. Note that the result options are slightly different from other TasCas options in that the choices, which are all Boolean, can be made using the one flag. For example:

```
-wh -wt -wf -wn -wc
```

can be expressed as:

```
-whtfnc
```

The options can be in any order, duplicated or in two or more separate -w flags. They just switch on the appropriate reporting.

E.3.6 Trial options

The following options alter the number of trials and the random weight seeding:

trials (runs)	number of different trials (-t#); and
completely random	seeds the random number generator off the clock (-R), used to initialise the network weights and generate random connections to nodes (see note immediately below).

Note that differences in the clock seeding only occur every second. Hence if an individual trial is shorter than one second wall-clock time, the same seed will be used.

E.3.7 Checkpointing and file recovery

The current version of the TasCas simulator performs simple checkpointing of multiple trials. All the results of completed networks are saved for later summarisation.

If a run is killed by whatever means, it may be restarted simply by typing:

```
tascas -R<filestem>.<process number>
```

where 'tascas' refers to the executable name, and where each checkpointing file is stored as the name of the data set filestem followed by the process identification number of the process which was performing the initial simulation. The simulator picks up from the last completed network. If the process is interrupted again, the same process number of the original process will be used as the recovery file suffix, meaning that the same recovery command may be reused.

Three points to note: firstly, if there is a checkpoint file present with the same number as the newly started process (a completely separate trial) the simulator will exit with an error to that effect, and it will not attempt to overwrite the previous checkpoint file. Secondly, if the checkpoint file is deleted during a simulation, the simulator will fail to give summary results and, of course, further recovery of results will not be possible. Finally, no checkpoint file is produced when only one trial is being performed.

E.4 Network output

This section describes in more detail the output which can be expected from the various reporting options. Complete outputs of the major examples are given in §E.D.

E.4.1 Header Information

The header information (option -wh) gives details about the particular run, which is useful to document experiments. There are six lines in the header which are produced with most simulations, plus a number of other lines of option details if required. The exception being when no hidden nodes are added, the lines containing candidate training information are not included, leaving a minimum of four lines. For example the call:

```
tascas spiral -whtf -cn10 -Sm20
```

produces the following header:

```
tascas spiral (v4.0)
Train 192  Test 192  Inputs 2  Outputs 2
OEta 0.35  OMu 1.75  OLen 50  OPerc 0.01  OEp 500  OOff 0.10
IEta 1.00  IMu 1.75  ILen 50  IPerc 0.03  IEp 500
MaxCand 20  PErrBits 0.00  ErrTh 0.40
Pool 10  Sigmoid 10  Full 10
```

The first line shows the simulator name, the filestem of the data set and the current simulator version being used. The second line shows information about the data set (number of training and test examples, and the number of network inputs and outputs). The third and fourth lines show the training parameters for both output and candidate (input) training respectively: the learning rate, the maximum growth factor, patience period and percentage, maximum number of training epochs per training phase, and the activation offset where used. The fifth shows network training parameters (in this case the maximum number of candidates which may be installed, the number of allowable error bits, and the current error threshold). The sixth line contains candidate information: the size of the candidate pool and the number of different candidates with different activation functions and connection strategies. This is all interspersed and followed by other information about optional settings: such as the number of trials, whether a random seed is used for setting the network weights, or pruning parameters. Information referring to the candidate or input layer training is always prefixed with 'I', and that which refers to the output layer is always prefixed with 'O'.

Section E.C details all the different header outputs, including the line of appearance. Remember that information about candidate training parameters is only produced when candidates are trained — if the network is limited to adding no candidates the information is not provided.

E.4.2 Final and summary results

The final results (option -wf) are of a network after training has been completed. This information includes the percentage correct on the training and test sets, the number of hidden nodes installed, the total number of connection crossings and the total number of connections. If the entire candidate pool is trained together with the one set of patience parameters, then the number of epochs is also shown. If some form of connection limitation is used, the number of limited connection nodes, the number of layers and the maximum number of possible connections are also shown. For example, the following command (example 1 in §E.D):

```
tascas spiral -whtf -pk -pK0.05 -Sm20 -cn10
```

produces this final report:

```
Final Network Results :
training %correct        : 100.00
testing %correct         : 92.71
hidden nodes             : 16
limited hidden nodes     : 16
layers                   : 11
total epochs             : 8811
total conn. cross.       : 239094720
total connections        : 109
maximum connections      : 206
```

Of course the actual details of the results will differ from machine to machine, as all the results in this report depend on the seed given to the random number generator as well as the precision of the machine.

The summary option (option -ws) produces the same information as the final results but is used with multiple trials to give the mean, mean absolute deviation, standard deviation, coefficient of variation, skew, kurtosis, confidence interval, median, minimum, maximum and inter-quartile range values for each field. For example, the command (example 2 in §E.D):

```
tascas spiral -whs -t50 -R -cn10 -c15 -cI -cF1.1
```

gives the following summary:

```
Summary Statistics :
          Trn%     Tst%    Hid   LimH    Lay    TEps        TCC    TCn    MxC
Mean     99.95    95.45   14.4    5.5    9.9   32826  129383163  170.5  180.4
MAD       0.10     1.25   2.38   2.16   1.03  3568.6   26014314  44.91  50.29
SD        0.22     1.63   3.46   3.04   1.41  4436.8   34919506  68.20  77.04
CoV       0.22     1.71  24.06  55.45  14.23   13.52      26.99  40.00  42.71
Skew     -3.99    -0.53   1.83   1.34   0.14    0.68       1.23   2.17   2.17
Kurt     14.76     0.19   2.87   1.74   0.53   -0.29       0.99   4.19   4.17
CI +/-    0.06     0.45   0.96   0.84   0.39 1229.82 9679193.24  18.90  21.35

Median  100.00    95.31   13.5    5.0   10.0   31443  120590016  148.5  158.0
Min      98.96    90.62     11      1      7   25544   83701824    111    116
Max     100.00    97.92     25     15     14   43029  232353408    403    431
IQR       0.00     1.56   3.00   3.00   2.00  4978.0   37061568  49.00  54.00
```

A result of 'na' is given for values of the coefficient of variation, the skewness and the kurtosis when there is no valid result.

The other final report is the full summary (produced by -wS) which simply gives a tab-separated list of all the final results of all the networks, in case this is required.

E.4.2.1 *Formulas used in statistical summary results*

For completeness, the following are the formulas used to calculate the summary results where *t* is the number of trials, and x_i is the result of trial *i*:

Mean
$$\bar{x} = \frac{\sum_{i=1}^{t} x_i}{t}$$

Mean absolute deviation
$$\mathrm{MAD} = \frac{\sum_{i=1}^{t} \left| x_i \pm \bar{x} \right|}{t}$$

Standard deviation
$$s = \sqrt{\frac{\sum_{i=1}^{t} (x_i \pm \bar{x})^2}{t \pm 1}}$$

Skewness
$$sk = \frac{\sum_{i=1}^{t} (x_i \pm \bar{x})^3}{t \cdot s^3}$$

Kurtosis
$$ku = \frac{\sum_{i=1}^{t} (x_i \pm \bar{x})^4}{t \cdot s^4} \pm 3$$

Coefficient of variance
$$\mathrm{CoV} = \frac{100 \cdot s}{\bar{x}}$$

Confidence interval
$$\mathrm{CI} = \frac{1.96 \cdot s}{\sqrt{t}}$$

Note that the confidence interval is only valid for trials of greater than thirty networks, and is for ninety five percent confidence.

E.4.3 Other outputs for completed training of a single trial

Writing weights (option -ww) writes out the weights of a particular run to a file '<filestem>.wei'. If this option is used during a multiple trial, all the weight sets are sent to the one weight file. Writing network connections (option -wn) shows which connections are present, and what the activation function is on each node. This is produced for each network after the final results of that network. The following letters are used to represent candidate node activation functions: 'A' for asymmetric sigmoids, 'S' for symmetric sigmoids, 'T' for tanh functions, and 'G' for Gaussian functions.

Writing examples (option -we) produces the actual and expected outputs of the network for each example in the training and test sets. Writing vectors (option -wv) writes out the actual and expected output vectors for each example in the training and test sets. Writing the best vectors (option -wb) produces the vectors for when the best result on the training error is reached. These vector results are displayed separately from the other vectors, and are distinguished by the tags 'tr-v' and 'ts-v' at the beginning of each vector depending on whether the example is from the training or the test set respectively. Writing the confusion matrix (option -wm) produces a totals breakdown of what examples are correctly classified, and what class is given to incorrectly classified examples. Note that the predicted values are listed across the matrix, hence the total examples predicted in a particular class are obtained by summing the column. Similarly the actual value totals for each class are obtained by summing the row — the actual class labels are given in column format. The example results and confusion matrices are not produced when trials are performed on regression data sets.

The options for producing the network connections, the examples, the confusion matrix and the output vectors are illustrated by the following trial on the simple xor problem:

```
tascas xor -whfnevm
```

which gives the following output:

```
tascas xor (v4.0)
Train 4  Test 0  Inputs 2  Outputs 2
OEta 0.35  OMu 1.75  OLen 50  OPerc 0.01  OEp 500  OOff 0.10
IEta 1.00  IMu 1.75  ILen 50  IPerc 0.03  IEp 500
MaxCand 25  PErrBits 0.00  ErrTh 0.40
Pool 4  Sigmoid 4  Full 4

Final Training Examples & Output Vectors :
  1     1   0.24264  -0.24264     0.50000  -0.50000
  2     2  -0.24264   0.24264    -0.50000   0.50000
  2     2  -0.21852   0.21852    -0.50000   0.50000
  1     1   0.24264  -0.24264     0.50000  -0.50000

Confusion Matrix (rows predicted values, columns actual) :
  2    0
  0    2

Final Network Results :
training %correct       : 100.00
hidden nodes            : 1
total epochs            : 131
total conn. cross.      : 9620
total connections       : 11

Network Connections and Activation Functions :
1 1 1 S
1 1 1 1 S
1 1 1 1 S
```

The training examples are shown with six columns: the first two are the actual and expected example output classes, followed by the actual and expected output vectors. The final section of the output shows the network connections: there is one hidden node followed by the two output nodes, all of which are fully connected with symmetric sigmoid functions. Not only are the output nodes connected to the bias node and the two inputs, but also the connection to the hidden node is shown.

E.4.4 Progress during training

Epoch training (option -wT) produces the mean squared error (MSE) and percentage correct for both the training and test sets at the end of each epoch of output layer training. It also produces the current maximum correlation score (for normal or subgroup candidate training), the current hidden node correlation (for individual candidate training) or the total correlation of all the nodes (for summation candidate training) plus a letter representing the selected node activation function during the candidate node training phase.

Node training (option -wt) produces the same information at the end of the candidate node and the output layer training phases. This is prefixed with the number of hidden nodes installed, a cumulative number of epochs — if appropriate — and connection crossings for the entire network training. The tag 'best' is also given at the end of output layer training if the writing of best vectors is required (option -wb) and the output phase produces the lowest error for that network on the training set.

Epoch and node correlation reporting (options -wC and -wc respectively) produce the correlation scores for all candidate nodes after each epoch or each candidate node training phase. Note that options -wC and -wT produce the same results under individual candidate training, hence only one is given.

These are illustrated with the following example:

```
tascas xor -whftTcC -sL5 -sl5
```

which gives:

```
tascas xor (v4.0)
Train 4  Test 0  Inputs 2  Outputs 2
OEta 0.35  OMu 1.75  OLen 5  OPerc 0.01  OEp 500  OOff 0.10
IEta 1.00  IMu 1.75  ILen 5  IPerc 0.03  IEp 500
MaxCand 25  PErrBits 0.00  ErrTh 0.40
Pool 4  Sigmoid 4  Full 4

0.30443 50.00
0.29903 50.00
...
0.25000 50.00
0.25000 50.00
  0       11        552      0.25000 50.00
 0.04890   0.00275   0.08276   0.01291
 0.04994   0.00279   0.08404   0.01343
0.08404 S
 0.05284   0.00291   0.08759   0.01487
0.08759 S
...
 0.49369   0.49335   0.49801   0.49365
0.49801 S
 0.49369   0.49335   0.49801   0.49365
```

```
  1        32         2520       0.49801 S
0.23472 75.00
...
0.20594 75.00
0.18290 75.00
0.14942 75.00
0.11019 100.00
0.07084 100.00
  1        41         3140       0.07084 100.00

Final Network Results :
training %correct       : 100.00
hidden nodes            : 1
total epochs            : 41
total conn. cross.      : 3140
total connections       : 11
```

Note that a number of the lines in the example are deleted (as shown by the ellipsis). The example shows the training information (MSE and percentage correct) for the output layer after each epoch (option -wT), followed by the training results for that layer (hidden nodes installed, epochs, connection crossing, MSE and percentage correct) (option -wt) which shows the number of epochs completed as being 11. This is followed by the correlation results (correlation of each candidate) (option -wC) for the initial random weights of the candidates, and then after each epoch of training. Interleaved is the selected candidate as shown by its (maximum) correlation and the activation function of the candidate (option -wT again). The training of candidates is completed and the final candidate results are shown (option -wc), although this output is the same as the last candidate results of epoch training. This is followed by the summary of the candidate training (hidden nodes installed, epochs, connection crossings, maximum correlation and the hidden node activation function) (option -wt again) showing that 32 epochs of training have been completed. This is further followed by the output layer training results until all the examples are classified correctly.

E.4.5 Regression results

Regression results are slightly different in that it is not possible to calculate a percentage correct, hence in all the training results this is not given, and in the final and summary results the final MSE is used to indicate the strength of the learnt theory.

E.5 Possible errors

The TasCas system produces a number of errors in exceptional cases. These are grouped by their type and the return code from the system will give an indication of the error type, as will the actual error message. When the system completes the required task with no errors, a return code of zero is given. The errors are as follows:

- data file reading (return code 1) — an error has occurred while the data file is being read in, with a maximum of ten of errors displayed;
- memory allocation (return code 2) — not enough memory available for allocation;
- major command line error (return code 3) — if a command line error is major enough to halt the simulator from sensibly continuing; and
- output errors (return code 4) — when the required file, such as the checkpointing file, is not available.

Note that minor command line error warnings are also displayed for unknown options or invalid options values.

E.6 Code structure

This section outlines the structure of the TasCas code. The module *tascas* details the main code for the TasCas system. The other modules (*basic, data, inp, out, eval, train*) provide routines used by the *tascas* code. For a deeper understanding of the workings of the system than is presented here, it is probably best to go to the code itself.

E.6.1 Module overview

The following is an overview of the various modules:

- *basic* contains simple procedures for neural network calculations which are used throughout the entire code, and some data structures;
- *data* defines the data structure for examples to be used for training and testing and routines for reading in Quinlan style data sets, and the data reading function itself;
- *eval* contains routines that evaluate Cascor networks to determine the result of the network given a particular input, cache training results and to give test set estimates of accuracy;
- *train* contains routines to perform generic weight training, pruning and patience calculations;
- *inp* contains routines for the remaining input (other than data reading) which involves the command line options, the modification of options as required and the setting up of checkpointing; and
- *out* contains all output including the writing of network weights and the reporting of the training process, the completion of training and result summaries.

Note that *eval.h* contains the definitions of network structure, *train.h* contains the layer training parameters, and *inp.h* contains definitions of the reporting structures and the other parameters used for training the network and writing out the results. The code in *basic* and *data* is not specific to the TasCas simulator and has been used in other simulators. The data reader may be altered as long as it produces the information in the form of the data types shown in *data.h*.

E.6.2 Main training mechanism

Training Cascor networks is a two stage process. Firstly the output layer is trained until patience has run out, and then if the desired result has not been achieved (namely the stopping criteria on the training set have not been met) a hidden node is added. This is done by training candidate nodes with connections to the inputs and previous hidden nodes, adding the best candidate to the network, and then retraining the output layer with extra connections to the added node. This process cycles until training is complete or the maximum number of hidden nodes has been installed. Options are available to use node patience as well to halt the training process.

The code reflects this structure. The *trnout* function trains the output layer determining whether the training is complete. If that has not occurred, *trncand* is called to train the candidate nodes. Both these functions complete a single training phase — namely training until loss of patience on one layer, whether that be training the candidate nodes or the output layer, or pruning the connections afterward. These functions call *trnoutperiod*, *trncandperiod* to train the weights for a single patience period which in turn call *trnoutepoch* and *trncandepoch* which, as their names suggest, train the output and candidate nodes for a single epoch. The *trnout* and *trncand* functions act as shells to cope with the possibility of pruning the network.

Note *trnoutepoch* trains all of the output nodes, unlike *trncandepoch* which trains only one node. This is due to the different set up for training candidates with separate patience parameters. The candidate training functions also include *trncandsub* for training a subgroup of candidate nodes.

E.6.3 Other code groups

The candidate node training has a lot of associated machinery which is absent from the output layer training. These extra procedures include *initcand* which sets up the candidates for training (including limiting connections where required), *addcand* which adds a selected candidate node to the network ready for output layer training, *selectcand* which selects the candidate node for inclusion in the network, and *calccorr* which calculates the correlation of the candidate nodes with the network output.

The main program also has various initialisation and post-training functions associated with it as well as procedures to produce the correct number of trials and calls to the reporting functions. From the code it should be evident which functions are called from modules throughout the training process and which are only required before or after training.

E.7 Special considerations

These are more detailed comments regarding certain features of the code. Assumptions have been made at various points during the development of the system, and it is the aim of this section to detail the more important ones.

E.7.1 Standard notation and indexing

The inputs are numbered 0 to n (d->n in *main*) with i used as index and element zero being the bias node, the outputs are numbered 1 to m (d->m in *main*) with k as the index, the hidden nodes are numbered 1 to h (net->h in *main*) with j as the index, and the candidate nodes are numbered 0 to $c - 1$ (c->c) with u as the index variable.

E.7.2 Module specific considerations

The following are specific considerations which need to be taken into account when modifying the code. They are prefixed by the name of the module where the feature occurs:

- Basic — the memory allocation functions simply exit when there is not enough memory available. All exits caused by memory problems return a value of 2.
- Basic — the inputs to the activation functions are bounded (in *fun*) to prevent over and underflow errors occurring. The bounds may need to be altered depending on the precision of the floating point processing used.
- Data — discrete attribute values are encoded as separate nodes. If the particular value is set, the corresponding node value is set to 1.0, otherwise it is set to -1.0. However two-valued discrete attributes are encoded as one input node.
- Data — in *GetNames* storing the number of expected values for the 'discrete' option has one too many type coercions.
- Eval — *classeg* is not used during any training calculations, so does not pass back the number of connection crossings.
- Train — with *nodeactiv* from the *eval* module, *backprop* and *prune* only perform calculations when there is a connection present. Checking first whether there is a connection present should not be a detriment when there are no missing connections (needing a truth check followed by an increment, as opposed to a straight multiplication of a value).

- Inp — the checkpointing uses the process id to determine a unique file name.
- Inp — the default options are set in the function *setup* in *inp.c*. The following options should not be altered: 'trp->completed', any Boolean options such as the writing options (all Booleans are set to false, and using the flag will not flip the value — only set it to true), and the initial counter for the candidate activations and connection strategies.
- Out — the summary functions have a bad case of magic numbers.
- TasCas — as an informed guess, the output weights to a newly added candidate node are set to minus the previous correlation at the output. This seems to be better than just setting random weights [Fahlman 1993]
- TasCas — it is assumed that the output layer will have symmetric sigmoid activation functions for classification problems, and linear activation functions for regression problems.
- TasCas — the 'eta' values are normalised within the calls to the *update* (Quickprop) function. The output eta is divided by the number of training examples, while the input eta is divided by the total training examples multiplied by the maximum number of inputs to each candidate. The input eta normalisation should possibly be changed when a limited connected hidden node is being trained — this has not been examined.

E.7.3 Error and correlation formulas

This section refers to differences between Fahlman's publicly released code [Crowder & Fahlman 1991] and this simulator.

- The error being used at the moment is

$$e_k = y_k - t_k \tag{E.1}$$

  in *classout*, where *e* is the error, *y* is the actual output, *t* the expected output, and *k* the output layer index. Fahlman often uses

$$e_k = (y_k - t_k) \,.\, \text{derivactprime}(y_k) \tag{E.2}$$

  for the error, sum of errors and sum of squared error in the calculation of the correlation and derivative of the correlation, where *derivactprime* is the derivative of the activation function with 0.1 offset. This changes the 'true' error — effectively removing the sigmoid in classification problems, and so both would have to be stored for patience calculations. Fahlman refers to the error criteria used in this simulator as the raw error. These differences should be taken into account when examining formulas E.4 and E.5 below.
- The mean squared error (MSE) used in this simulator is:

$$\mathrm{MSE} = \frac{\sum_{k=1}^{m} \sum_{p=1}^{d} \left(y_{kp} \pm t_{kp}\right)^2}{m \cdot d} \qquad \text{(E.3)}$$

where *p* is the index to *d* the number of patterns, and *k* is the index to *m* the number of outputs. Fahlman's simulator does not divide the MSE by the number of training examples and by the number of outputs. This should not have an effect, though, as the patience calculations are performed with reference to the percentage change. The changes made to the output weights are the same in both simulators.

- Fahlman uses an error index for determining when training has been completed in regression problems, rather than the simple MSE used in this simulator. The MSE is normalised to give relatively the same error for different training sets. The formula for this (using E.3 above) is:

$$\text{Error Index} = \frac{\sqrt{\mathrm{MSE}}}{\mathrm{sdtr}} \qquad \text{(E.4)}$$

where *sdtr* is the standard deviation of the training set as defined by:

$$\mathrm{sdtr} = \sqrt{\frac{\sum_{k=1}^{m} \sum_{p=1}^{d} t_{kp}^2 \cdot m \cdot d \pm \left(\sum_{k=1}^{m} \sum_{p=1}^{d} t_{kp}\right)^2}{m \cdot d\,(m \cdot d \pm 1)}} \qquad \text{(E.5)}$$

- Error normalisation is implemented for correlation values. This amounts to having the following formulas instead of those given in Fahlman's Cascor paper [Fahlman & Lebiere 1989]:

$$S = \sum_{k=1}^{m} \left| \frac{\sum_{p=1}^{d} V_p \,.\, E_{kp} \pm \bar{V} \,.\, \bar{E_k}}{\sum_{k=1}^{m} \sum_{p=1}^{d} E_{kp}^2} \right| \qquad \text{(E.6)}$$

$$\frac{\partial S}{\partial w_i} = \sum_{k=1}^{m} \sum_{p=1}^{d} \frac{\pm \sigma_k \,.\, \left(E_{kp} \pm \bar{E_k}\right) . f'_p \,.\, x_{ip}}{\sum_{k=1}^{m} \sum_{p=1}^{d} E_{kp}^2} \qquad \text{(E.7)}$$

where *i* is the index for the input x_i, w is the weight to the candidate from the input layer. This is the same as the publicly released code.

E.8 Planned improvements

The following are possible future improvements to the code. Generally they will not be implemented until they are needed:

- full implementation of the Extended Quinlan format;
- cross-validation of networks;
- proper handling of validation sets;

- even class selection;
- allow for separate training and test set results;
- reading in previously generated networks;
- allow a configuration file to be read in as well as having command line options;
- update with improved algorithms for training and pruning;
- *getopt* code is ad hoc is some respects — need a more consistent system;
- change *initcand* so that different activation functions are associated with different limited connection strategies — more than one of the techniques is used at the one time (needs priorities for candidate groups: eg train activation types in sub-pools regardless of connections);
- printing of doubles in header to the appropriate number of decimal places;
- add summary reporting on different activation functions and connection strategies when used; and
- more consistent output and graphical output of results.

E.A Extended Quinlan format

One of the most systematic formats for inductive learning data is Ross Quinlan's C4.5 data format [Quinlan 1993a]. This is a user friendly way of expressing and documenting data, which has the additional benefit of being a style that is independent of the learning system. Consider a problem with three classes (red, blue and green) separated by two attributes (height, which is a numerical value; and size, which is an unordered discrete attribute with two possible values small and large). The description of this problem would be expressed as the following '.names' information file:

```
red, blue, green.

height: continuous.
size: small, large.
```

Both training and test examples (extensions '.data' and '.test' respectively) are then expressed in the following form:

```
-3.67, small, red
```

The major problem with the format is that there is no method for expressing regression style problems — Quinlan's format is specifically for classification problems. This has lead to the proposed Extended Quinlan Format for the system output which encompasses the classification style:

```
Problem ::= Subproblem { ";" Subproblem } "."
Subproblem ::= [Subprobname] Values
Subprobname ::= ident ":"
Values ::= "continuous" | ident ( "," ident )
```

This defines additions to the Quinlan format to allow for regression problems as well as multiple classifications or regressions stemming from the same data, where that may be required:

```
continuous.                                        (simple regression)
depth: continuous.                                 (named regression)
depth: continuous; height: continuous.             (multiple regression)
depth: continuous; colour: red, green, blue.       (classification & regression)
```

As can be seen, the labelling of the sub-problems would be optional, but it would aid in the data documentation. The attribute descriptions would have the same form, and examples would be appended with specific results for each sub-problem. It is then up to each learning system how the data is handled, and the beauty of this system is that all prior Quinlan format files remain valid.

This may be further expanded to account for time series problems by replicating the attributes required for each time frame in the example, separating this information by a semicolon as opposed to a comma. Consider the previous example, if this were a time series problem, the header file would remain the same but examples would be in the following form:

```
-3.67, small; 0.45, small; 4.78, large, green
```

This last example has three time frames ending in the classification green.

A final extension would be to allow for the explicit definition of a validation set [Prechelt 1994a]. Though it is quite possible to randomly select a validation set from the training set, in some cases it may be preferable to have an explicitly defined validation set, hence it is proposed to reserve the extension '.valid' for this purpose, using the same format as examples from the training and test sets.

These extensions do not include those necessary for ordered discrete attributes and classes or for partial orderings. These need to be examined in the future.

E.B Options summary

This appendix summarises all the possible options. The headings 'Option', 'Subopt' and 'Params' refer to the main option letter, the sub-option letter and the format of any required parameters respectively. A dash indicates that no value is required, brackets indicate an optional part of the flag, and the symbols '#' and '#.#' represent integer and floating point number parameters respectively. Note that upper case letters refer to the candidate options, whereas lower case letters refer to the output layer training options. With output options upper case letters mean more information is produced than when the lower case option is used.

Option	Subopt	Params	Description
e/E	–	#.#	eta values
m/M	–	#.#	mu values
o/O	–	#.#	activation function offsets
s			stopping layer training
	p/P	#.#	patience percentage change
	l/L	#	patience period length (epochs)
	m/M	#	maximum epochs for layer training
S			stopping network training
	m	#	maximum number of hidden nodes installed
	p	#.#	node patience percentage change
	l	#	node patience period length (hidden nodes)
	r	–	node patience rollback of unneeded nodes
	e	#.#	maximum allowable error for regression
	b	#.#	percentage allowable error bits (class.)
	t	#.#	error threshold for error bits
	x	#.#	change to expected value range
c			candidate training options
	n	#	number of nodes in candidate pool
	I	–	individual candidate training
	H	–	subgroup (homogeneous) candidate training
	S	–	summation rather maximum candidate selection
	F	#.#	percentage forcing usage of non-default nodes
	g	[#]	Gaussian activation functions
	s	[#]	sigmoid activation functions
	a	[#]	asymmetric sigmoid activation functions
	t	[#]	tanh activation functions
	D	[#]	all activation functions distributed
	f	[#]	fully connected candidate nodes
	l	[#]	layered candidate nodes
	m	[#]	minimum shortcut candidate nodes
	r	[#]	randomly connected candidate nodes
	R	#	number of random connections (see -cr)
	d	[#]	all connection strategies distributed
p	k/K	#.#	absolute Karnin pruning
	c/C	–	percentage pruning when used with -pk/K
	p/P	#.#	patience percentage pruning
	e	–	prune output layer after every training phase
	d/D	#.#	weight decay
	s/S	–	use weight decay to decay smaller terms more
w			output and writing options
	h	–	header information
	f	–	final network information
	s	–	summary information over trials
	S	–	final information for all trials (tab spaced)
	w	–	write weights to file
	n	–	output table of connections
	e	–	output final results for each example
	m	–	output final confusion matrices sets
	v	–	output actual and expected output vectors
	b	–	output best vectors of network
	t/T	–	output node/epoch training information
	c/C	–	output node/epoch candidate training info.

Option	Subopt	Params	Description
t	–	#	number of trials required
R	–	–	clock-seeded random numbers for trials

E.C Full header information

This appendix details the header information: italics indicates that the detail may vary, '#' indicates a integer value, '#.#' indicates a floating point number, brackets indicate options, and 'a' and 'b' lines may not appear.

Header Details	Line	Description
tascas	1	executable name
spiral	1	data set name
(v4.0)	1	version number
Train #	2	total training examples
Test #	2	total testing examples
Inputs #	2	total number of network inputs
Outputs #	2	total number of network outputs
[OI]Eta #.#	3/3a	learning rate
[OI]Mu #.#	3/3a	growth factor
[OI]Len #	3/3a	patience period (epochs)
[OI]Perc #.#	3/3a	patience percentage change
[OI]Ep #	3/3a	maximum number of epochs per training phase
[OI]Off #.#	3/3a	activation function offset
[OI]Dcy #.#	3b	standard weight decay parameter
[OI]SmDcy #.#	3b	small weight decay parameter
[OI]Prn #.#	3b	Karnin pruning level
[OI]PrnPerc #.#	3b	Karnin percentage pruning level
[OI]PPerc #.#	3b	Patience pruning percentage
NLen #	4	node patience period (nodes)
NPerc #.#	4	node patience percentage change
Rollback	4	node patience rollback used
MaxCand #	4	maximum number of candidates
MinError #.#	4	minimum error for regression problems
PErrBits #.#	4	percentage of allowable error bits
ErrTh #.#	4	error threshold
ExpVBuff #.#	4	expected value buffer
PrnEveryO	4	prune output layer after every training phase
Trials #	4	total trials to be conducted
Clock seed	4	whether trial or trials clock-seeded
Pool #	4a	candidate pool size
Gaussian #	4a	number of Gaussian nodes
Sigmoid #	4a	number of symmetric sigmoid nodes
TanH #	4a	number of tanh nodes
ASymSig #	4a	number of asymmetric sigmoid nodes
Full #	4a	number of fully connected nodes
Layered #	4a	number of layered nodes
MinShort #	4a	number of nodes with minimal shortcuts
RanConn #	4a	number of randomly connected nodes
(# links)	4a	number of connections per randomly connected node
(rand)	4a	random connections per randomly connected node

Header Details	Line	Description
SumCorr	4b	summation (not max) of candidate correlations
IndCandPat	4b	independent candidate training
SubCandPat	4b	subgroup candidate training
Forcing #.#	4b	forcing level of non-default candidates
(def *string*)	4b	default node features

E.D Complete examples

Here two runs are detailed using the simulator on the Two Spirals data set [Fahlman & Lebiere 1989].

E.D.1 Example one

For an example of how the program is used, consider the following training command to recognise the Two Spirals data set:

```
tascas spiral -whtf -pk -pK0.05 -Sm20 -cn10
```

This trains a network on the Two Spirals data set whilst producing the header information, the training progress after each layer training phase is completed, and the final results. It uses Karnin pruning on the output layer with the default setting, prunes the input (hidden node) layer with setting 0.05, can install a maximum of twenty hidden nodes, and uses a candidate pool of ten nodes.

The output is as follows:

```
tascas spiral (v4.0)
Train 192  Test 192  Inputs 2  Outputs 2
OEta 0.35  OMu 1.75  OLen 50  OPerc 0.01  OEp 500  OOff 0.10
IEta 1.00  IMu 1.75  ILen 50  IPerc 0.03  IEp 500
OPrn 0.00  IPrn 0.05
MaxCand 20  PErrBits 0.00  ErrTh 0.40
Pool 10  Sigmoid 10  Full 10

  0       56      130176      0.24673 50.00   0.24672 50.00
  1      186     1622016      0.10293 S
  1      236     1812096      0.09293 S
  1      296     1998144      0.23948 55.21   0.24084 54.17
  2      539     5722944      0.13779 S
  2      589     6141120      0.14147 S
  2      654     6393024      0.22662 62.50   0.22748 63.54
  3      917    11433024      0.17465 S
  3      967    11965248      0.17541 S
  3     1044    12323136      0.20485 64.58   0.21056 64.58
  4     1382    20099136      0.18554 S
  4     1432    20688384      0.16852 S
  4     1507    21095232      0.18879 67.71   0.19856 65.62
  5     1744    27452352      0.15802 S
  5     1835    28112640      0.11091 S
```

```
• 5      1915     28607808     0.18091 67.71    0.19109 67.71
  6      2226     38146368     0.21388 S
  6      2301     38918784     0.18378 S
  6      2368     39385728     0.16850 70.83    0.17549 68.75
  7      2534     45105408     0.28347 S
  7      2686     46327104     0.25419 S
  7      2801     47215104     0.13641 75.00    0.14902 71.88
  8      2927     52034304     0.16445 S
  8      3019     52772160     0.16733 S
  8      3105     53503872     0.12508 76.04    0.14417 73.96
  9      3424     66957312     0.19930 S
  9      3513     67806912     0.15677 S
  9      3606     68669376     0.11614 79.17    0.13453 75.00
 10      3987     86202816     0.24180 S
 10      4126     88011072     0.22100 S
 10      4263     89384640     0.10069 85.42    0.12006 81.25
 11      4497    101040960     0.17079 S
 11      4605    101701440     0.11558 S
 11      4695    102675072     0.09554 84.38    0.11709 82.29
 12      5185    128990592     0.32415 S
 12      5311    130677312     0.29859 S
 12      5572    133691520     0.06989 92.71    0.10454 86.46
 13      5936    154629120     0.45196 S
 13      6039    156636480     0.45693 S
 13      6516    162505536     0.02738 95.83    0.09921 89.58
 14      6727    175438656     0.29196 S
 14      6777    176065920     0.29349 S
 14      7210    181726464     0.01881 97.92    0.09266 89.58
 15      7363    191681664     0.39839 S
 15      7429    192436224     0.39961 S
 15      7929    199355904     0.00859 98.96    0.08356 91.67
 16      8429    233881344     0.72292 S
 16      8517    234788544     0.74221 S
 16      8811    239087616     0.00298 100.00   0.07897 92.71
 16      8811    239094720     0.00298 100.00   0.07897 92.71

Final Network Results :
training %correct        : 100.00
testing %correct         : 92.71
hidden nodes             : 16
limited hidden nodes     : 16
layers                   : 11
total epochs             : 8811
total conn. cross.       : 239094720
total connections        : 109
maximum connections      : 206
```

The header information is followed by a blank line, then this is followed by the actual training information — firstly the output layer training without any hidden nodes, then two phases of candidate training, the first being training before pruning occurs, the second after pruning has been completed. This is followed by the results of training the output layer with the added hidden unit — selected from the candidate pool based on its performance.

This process cycles until one hundred percent is achieved on the training set, whereby the output layer is pruned and retrained. Finally, after another blank line, the final report of the training run is produced.

E.D.2 Example two

Another run on the Two Spirals data set may look like the following:

```
tascas spiral -whs -t50 -R -cn10 -cl5 -cI -cF1.1
```

This writes to the screen the header information and summary information of the fifty clock-seeded trials, which involve using a pool of ten candidates, individual candidate patience, forming layers with half the nodes and forcing the usage of layered nodes by an extra ten percent.

The output produced by this command is as follows:

```
tascas spiral (v4.0)
Train 192  Test 192  Inputs 2  Outputs 2
OEta 0.35  OMu 1.75  OLen 50  OPerc 0.01  OEp 500  OOff 0.10
IEta 1.00  IMu 1.75  ILen 50  IPerc 0.03  IEp 500
MaxCand 25  PErrBits 0.00  ErrTh 0.40  Trials 50  Clock seed
Pool 10  Sigmoid 10  Full 5  Layered 5
IndCandPat  Forcing 1.1 (def full symsig)

Summary Statistics :
         Trn%    Tst%    Hid   LimH    Lay    TEps         TCC    TCn    MxC
Mean    99.95   95.45   14.4    5.5    9.9   32826   129383163  170.5  180.4
MAD      0.10    1.25   2.38   2.16   1.03  3568.6    26014314  44.91  50.29
SD       0.22    1.63   3.46   3.04   1.41  4436.8    34919506  68.20  77.04
CoV      0.22    1.71  24.06  55.45  14.23   13.52       26.99  40.00  42.71
Skew    -3.99   -0.53   1.83   1.34   0.14    0.68        1.23   2.17   2.17
Kurt    14.76    0.19   2.87   1.74   0.53   -0.29        0.99   4.19   4.17
CI +/-   0.06    0.45   0.96   0.84   0.39 1229.82  9679193.24  18.90  21.35

Median 100.00   95.31   13.5    5.0   10.0   31443   120590016  148.5  158.0
Min     98.96   90.62     11      1      7   25544    83701824    111    116
Max    100.00   97.92     25     15     14   43029   232353408    403    431
IQR      0.00    1.56   3.00   3.00   2.00  4978.0    37061568  49.00  54.00
```

F References

Adams A. (1994) A neural network local minimum testbed, in *1994 International Symposium on Artificial Neural Networks*, IEEE, pp57–62.

Adams A and Jones P. (1992) Function evaluation and interpolation using a backpropagation artificial neural network, in *The Fifteenth Australian Computer Science Conference*, Gupta G and Keen C, Editors, Department of Computer Science, University of Tasmania, pp13–25.

Adams A and Lewis C. (1995) Neural network function evaluation and the sigmoid prime offset, in *The Sixth Australian Conference on Neural Networks*, Charles M and Latimer C, Editors, University of Sydney, Electrical Engineering, pp229–233.

Adams A and Waugh S. (1995) Function evaluation and the Cascade-Correlation architecture, in *IEEE International Conference on Neural Networks*, IEEE and Causal Productions, pp942–946.

Ash T. (1989) Dynamic node creation in backpropagation networks, *Connection Science*, **1** (4): 365–375.

Baffes PT and Zelle JM. (1992) Growing layers of perceptrons: introducing the extentron algorithm, in *International Joint Conference on Neural Networks*, IEEE, pp392–397.

Baluja S and Fahlman SE. (1994) *Reducing network depth in the cascade-correlation learning architecture*, School of Computer Science, Carnegie Mellon University, TR CMU-CS-94-209.

Baum EB. (1989) A proposal for more powerful learning algorithms, *Neural Computation*, **1**: 201–207.

Bolt GR. (1992) *Fault tolerance in artificial neural networks: are neural networks inherently fault tolerant?*, University of York, D.Phil. thesis.

Bratko I. (1990) *Prolog programming for artificial intelligence*, 2nd ed., Addison-Wesley.

Breiman L, Friedman JE, Olshen RA and Stone CJ. (1984) *Classification and regression trees*, Wadsworth International Group: Belmont, California.

Burkitt AN and Ueberholz P. (1993) Pruning feed-forward neural networks, in *The Fourth Australian Conference on Neural Networks*, Leong P and Jabri M, Editors, Sydney University Electrical Engineering, pp185–188.

Burrows JF and Craig DH. (1994) Lyrical drama and the "Turbid Mountebanks": styles of dialogue in Romantic and Renaissance tragedy, *Computers and the Humanities*, **28**: 63–86.

Caruana R and Freitag D. (1994) Greedy attribute selection, in *The Eleventh International Conference on Machine Learning*, Cohen WW and Hirsh H, Editors, Morgan Kaufmann, pp28–36.

Catlett J. (1992) Peepholing: choosing attributes efficiently for megainduction, in *The Ninth International Workshop on Machine Learning*, Sleeman D and Edwards P, Editors, Morgan Kaufmann, pp49–54.

Chauvin Y. (1988) A back-propagation algorithm with optimal use of hidden units, in *Advances in Neural Information Processing Systems 1*, Touretzky DS, Editor, Morgan Kaufmann, pp519–526.

Chung FL and Lee T. (1992) A node pruning algorithm for backpropagation networks, *International Journal of Neural Systems*, **3** (3): 301–314.

Collier PA. (1995) Choosing between back propagation neural networks and C4.5 for different types of data, *ACS AISIG (Vic) Newsletter*, **8** (3): 3–10.

Collier PA and Waugh S. (1994) Characteristics of data suitable for learning with connectionist and symbolic methods, in *The 7th Australian Joint Conference of Artificial Intelligence*, Zhang C, Debenham J and Lukose D, Editors, World Scientific, pp116–123.

Cortes C, Jackel LD and Chiang W-P. (1995) Limits on learning machine accuracy imposed by data quality, in *Advances in Neural Information Processing Systems 7*.

Crowder RS and Fahlman SE. (1991) *C implementation of the Cascade-Correlation learning algorithm*, Carnegie Mellon University, 1.32, URL: ftp://pt.cs.cmu.edu/afs/cs/project/connect/code.

de le Maza M. (1991) SPLITnet: dynamically adjusting the number of hidden units in a neural network, in *Artificial Neural Networks*, Kohonen T, Mäkisara K, Simula O and Kangas J, Editors, North-Holland, pp647–651.

Deffuant G. (1995) An algorithm for building regularized piecewise linear discrimination surfaces: the perceptron membrane, *Neural Computation*, **7**: 380–398.

Devillers O, Golin M, Kedem K and Schirra S. (1994) *Revenge of the dog: queries on Voronoi diagrams of moving points*, Institut National de Recherche en Informatique et en Automatique, TR 2329.

Duda RO and Hart PE. (1973) *Pattern classification and scene analysis*, Wiley-Interscience.

Dunne RA, Campbell NA and Kiiveri HT. (1992) Task based pruning, in *The Third Australian Conference on Neural Networks,* Leong P and Jabri M, Editors, Sydney University Electrical Engineering, pp166–169.

Elomaa T. (1994) In defense of C4.5: notes on learning one-level decision trees, in *The Eleventh International Conference on Machine Learning,* Cohen WW and Hirsh H, Editors, Morgan Kaufmann, pp62–69.

Fahlman SE. (1988a) *An empirical study of learning speed in back-propagation networks,* Carnegie Mellon University, TR CMU-CS-88-162.

Fahlman SE. (1988b) Fast-learning variations on back-propagation: an empirical study, in *Proceedings of the 1988 Connectionist Models Summer School,* Touretzky D, Hinton G and Sejnowksi T, Editors, Morgan Kaufmann, pp38–51.

Fahlman SE. (1990) *Summary of NIPS-90 workshop: constructive and destructive learning algorithms,* Carnegie-Mellon University, neuroprose archive.

Fahlman SE. (1993) Private communication.

Fahlman SE. (1994) Private communication.

Fahlman SE and Lebiere C. (1989) The cascade-correlation learning architecture, in *Advances in Neural Information Processing Systems* 2, Touretzky DS, Editor, Morgan Kaufmann, pp525–532.

Fiesler E. (1994) Comparative bibliography of ontogenic neural networks, in *International Conference on Artificial Neural Networks.*

Frean M. (1990) The Upstart algorithm: a method for constructing and training feedforward neural networks, *Neural Computation,* **2**: 198–209.

Freeman T. (1994) Private communication.

Gallant SI. (1986) Three constructive algorithms for network learning, in *The Eighth Annual Conference of the Cognitive Science Society,* Lawrence Erlbaum Associates, pp652–660.

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

Gorodkin J, Hansen LK, Krogh A, Svarer C and Winther O. (1993) A quantitative study of pruning by optimal brain damage, *International Journal of Neural Systems,* **4** (2): 159–169.

Hamamoto M, Kamruzzaman J and Kumagai Y. (1992) A study on generalization properties of artificial neural network using Fahlman and Lebiere's learning algorithm, in

Artificial Neural Networks 2, Aleksander I and Taylor J, Editors, North-Holland, pp1067–1070.

Hamey LGC. (1991) Benchmarking feed-forward neural networks: models and measures, in *Advances in Neural Information Processing Systems 4,* Moody JE, Hanson SJ and Lippmann R, Editors, Morgan Kaufmann, pp1167–1174.

Hancock PJB. (1992) Pruning neural nets by genetic algorithm, in *Artificial Neural Networks 2,* Aleksander I and Taylor J, Editors, North-Holland, pp991–994.

Hanson SJ. (1989) Meiosis networks, in *Advances in Neural Information Processing Systems 2,* Touretzky DS, Editor, Morgan Kaufmann, pp533–541.

Hanson SJ and Pratt LY. (1988) Comparing biases for minimal network construction with back-propagation, in *Advances in Neural Information Processing Systems 1,* Touretzky DS, Editor, Morgan Kaufmann, pp177–185.

Hassibi B and Stork DG. (1992) Second order derivatives for network pruning: Optimal Brain Surgeon, in *Advances in Neural Information Processing Systems 5,* Morgan Kaufmann, pp164–171.

Hassibi B, Stork DG and Wolff G. (1993) Optimal Brain Surgeon: extensions and performance comaprisons, in *Advances in Neural Information Processing Systems 6,* Cowan JD, Tesauro G and Alspector J, Editors, Morgan Kaufmann, pp263–270.

Hertz J, Krogh A and Palmer RG. (1991) *Introduction to the theory of neural computation,* Addison-Wesley.

Hickey RJ. (1992) Artificial universes — towards a systematic approach to evaluating algorithms which learn from examples, in *The Ninth International Workshop on Machine Learning,* Sleeman D and Edwards P, Editors, Morgan Kaufmann, pp196–205.

Holte RC. (1993) Very simple classification rules perform well on most commonly used datasets, *Machine Learning,* **11**: 63–91.

Hwang J-N, Lay S-R, Maechler M, Martin RD and Schimert J. (1994) Regression modeling in back-propagation and projection pursuit learning, *IEEE Transactions on Neural Networks,* **5** (3): 342–353.

Hwang J-N, You S-S, Lay S-R and Jou I-C. (1993) *What's wrong with a cascaded correlation learning network: a projection pursuit learning perspective,* neuroprose archive.

Izui Y and Pentland A. (1990) Analysis of neural networks with redundancy, *Neural Computation,* **2**: 226–238.

James M. (1985) *Classification algorithms,* Collins: London.

Ji C, Snapp RR and Psaltis D. (1990) Generalizing smoothness constraints from discrete samples, *Neural Computation*, **2**: 188–197.

John GH. (1995) Cascade Correlation: derivation of a more numerically stable update rule, in *IEEE International Conference on Neural Networks*, IEEE and Causal Productions, pp1126–1129.

Karnin ED. (1990) A simple procedure for pruning back-propagation trained neural networks, *IEEE Transactions on Neural Networks*, **1** (2): 239–242.

Kendall GD and Hall TJ. (1992) Ockham's nets: self-adaptive minimal neural networks, in *Artificial Neural Networks* 2, Aleksander I and Taylor J, Editors, North-Holland, pp183–186.

Kendall GD and Hall TJ. (1993) Optimal network construction by minimum description length, *Neural Computation*, **5**: 210–212.

Kira K and Rendell LA. (1992) A practical approach to feature selection, in *The Ninth International Workshop on Machine Learning*, Sleeman D and Edwards P, Editors, Morgan Kaufmann, pp249–256.

Klagges H and Soegtrop M. (1992) *Limited fan-in random wired cascade-correlation*, IBM Research Division, Physics Group Munich, neuroprose archive.

Kohonen T, Chrisley R and Barna G. (1988) Statistical pattern recognition with neural networks: benchmarking studies, in *Neural networks from models to applications*, Personnaz L and Dreyfus G, Editors, IDSET, Paris, pp160–167.

Krogh A and Hertz JA. (1991) A simple weight decay can improve generalisation, in *Advances in Neural Information Processing Systems 4*, Moody JE, Hanson SJ and Lippmann RP, Editors, Morgan Kaufmann, pp950–957.

Le Cun Y, Denker JS and Solla SA. (1989) Optimal Brain Damage, in *Advances in Neural Information Processing Systems* 2, Touretzky DS, Editor, Morgan Kaufmann, pp598–605.

Lee Y and Lippmann RP. (1989) Practical characteristics of neural network and conventional pattern classifiers on artificial and speech problems, in *Advances in Neural Information Processing Systems 2*, Touretzky DS, Editor, Morgan Kaufmann, pp168–177.

Levin A, Leen TK and Moody JE. (1994) Fast pruning using principle components, in *Advances in Neural Information Processing Systems 6*, Cowan J, Tesauro G and Alspector J, Editors, Morgan Kaufmann, pp35–42.

Lippmann RP. (1987) An introduction to computing with neural nets, *IEEE Transactions on Acoustics, Speech and Signal Processing*, **4** (2): 4–22.

Lister R. (1994) The problem with Quickprop's weight independence assumption, in *The Fifth Australian Conference on Neural Networks,* Tsoi AC and Downs T, Editors, University of Queensland Electrical and Computer Engineering, pp5–8.

Lister R and Stone JV. (1995) Error functions and conjugate gradient back propagation, in *The Sixth Australian Conference on Neural Networks,* Charles M and Latimer C, Editors, University of Sydney, Electrical Engineering, pp130–133.

Littmann E and Ritter H. (1992) Cascade networks architectures, in *International Joint Conference on Neural Networks,* IEEE, pp398–404.

Lounis H and Bisson G. (1991) Evaluation of learning systems: an artificial data-based approach, in *European Working Session on Learning,* Kodratoff Y, Editor, Springer-Verlag Lecture Notes in Artificial Intelligence, 482, pp463–481.

Marchand M, Golea M and Rujian P. (1990) A convergence theorem for sequential learning in two-layer perceptrons, *Europhysics Letters,* **11**: 487–492.

Matthews RAJ and Merriam TVN. (1993) Neural computation in Stylometry I: an application to the works of Shakespeare and Fletcher, *Literary and Linguistic Computing,* **8** (4): 203–209.

Merriam TVN and Matthews RAJ. (1994) Neural computation in Stylometry II: an application to the works of Shakespeare and Marlowe, *Literary and Linguistic Computing,* **9** (1): 1–6.

Mézard M and Nadal J-P. (1989) Learning in feedforward layered networks: the tiling algorithm, *Journal of Physics A: Mathematical and General,* **22**: 2191–2203.

Møller MF. (1993) A scaled conjugate gradient algorithm for fast supervised learning, *Neural Networks,* **6**: 525–533.

Mozer MC and Smolensky P. (1988) Skeletonization: a technique for trimming the fat from a network via relevance assessment, in *Advances in Neural Information Processing Systems 1,* Touretzky DS, Editor, Morgan Kaufmann, pp107–115.

Mozer MC and Smolensky P. (1989) Using relevance to reduce network size automatically, *Connection Science,* **1** (1): 3–16.

Murase K, Matsunaga Y and Nakade Y. (1991) A back-propagation algorithm which automatically determines the number of association units, in *International Joint Conference on Neural Networks,* IEEE, pp783–788.

Murphy PM and Aha DW. (1994) *UCI Repository of machine learning databases*, University of California, Irvine, Department of Information and Computer Science, URL: ftp://ics.uci.edu/pub/machine-learning-databases/.

Nash WJ, Sellers TL, Talbot SR, Cawthorn AJ and Ford WB. (1994) *The population biology of Abalone (Haliotis species) in Tasmania. I. Blacklip Abalone (H. rubra) from the North Coast and the Islands of Bass Strait*, Sea Fisheries Division, Marine Research Laboratories — Taroona, Department of Primary Industry and Fisheries, Tasmania, TR 48.

Nolfi S and Parisi D. (1991) *Growing neural networks*, Department of Cognitive Processes and Artificial Intelligence, Institute of Psychology, National Research Council, Rome, TR PCIA-91-15, neuroprose archive.

Nowlan SJ and Hinton GE. (1992) Simplifying neural networks by soft weight-sharing, *Neural Computation*, **4**: 473–493.

Okabe A, Boots B and Sugihara K. (1992) *Spatial tessellations: concepts and applications of Voronoi diagrams*, John Wiley and Sons: Chichester.

Platt J. (1991) A resource-allocating network for function interpolation, *Neural Computation*, **3**: 213–225.

Prechelt L. (1994a) *PROBEN1 — a set of neural network benchmark problems and benchmarking rules*, Fakultät für Informatik, Universität Karlsruhe, TR 21/94.

Prechelt L. (1994b) *A study of experimental evaluations of neural network learning algorithms: current research practice*, Fakultät für Informatik, Universität Karlsruhe, TR 19/94.

Quinlan J. (1987) Simplifying decision trees, *International Journal of Man-Machine Studies*, **26**: .

Quinlan JR. (1986a) The effect of noise on concept learning, in *Machine learning: an artificial intelligence approach*, Michalski RS, Carbonell JG and Mitchell TM, Editors, Morgan Kaufmann: Los Altos, California, Vol. 2, pp149–166.

Quinlan JR. (1986b) Induction of decision trees, *Machine Learning*, **1** (1): 81–106.

Quinlan JR. (1993a) *C4.5: programs for machine learning*, Morgan Kaufmann: San Mateo, California.

Quinlan JR. (1993b) Comparing connectionist and symbolic learning methods, in *Computational learning and natural learning systems: constraints and prospects*, Hanson S, Drastal G and Rivest R, Editors, MIT Press: Cambridge, Massachusetts.

Ramachandran S and Pratt LY. (1991) Information measure based skeletonisation, in *Advances in Neural Information Processing Systems 4*, Moody JE, Hanson SJ and Lippmann RP, Editors, Morgan Kaufmann, pp1080–1087.

Reed R. (1993) Pruning algorithms — a survey, *IEEE Transactions on Neural Networks*, **4** (5): 3–16.

Refenes AN and Vithlani S. (1991) Constructive learning by specialisation, in *Artificial Neural Networks*, Kohonen T, Mäkisara K, Simula O and Kangas J, Editors, North-Holland, pp923–929.

Rendell L and Cho H. (1990) Empirical learning as a function of concept character, *Machine Learning*, **5**: 267–298.

Rögnvaldsson T. (1993) Pattern discrimination using feedforward networks: a benchmark study of scaling behaviour, *Neural Computation*, **5**: 483–491.

Rumelhart DE, Hinton GE and Williams RJ. (1986) Learning internal representations by error propagation, in *Parallel Distributed Processing*, Rumelhart DE and McClelland JL, Editors, MIT Press: Cambridge, Massachusetts, Vol. 1, pp318–362.

Segee BE and Carter MJ. (1991) Fault tolerance of pruned multilayer networks, in *International Joint Conference on Neural Networks*, IEEE, pp447–452.

Shamir N, Saad D and Marom E. (1993) Neural net pruning based on functional behaviour of neurons, *International Journal of Neural Systems*, **4** (2): 143–158.

Sietsma J and Dow RJF. (1988) Neural net pruning — why and how, in *IEEE International Conference on Neural Networks*, IEEE, pp325–333.

Sietsma J and Dow RJF. (1991) Creating artificial neural networks that generalise, *Neural Networks*, **4**: 67–79.

Simon N. (1993) *Constructive supervised learning algorithms for artificial neural networks*, Delft University of Technology, Masters thesis.

Simon N, Corporaal H and Kerckhoffs E. (1992) *Variations on the cascade-correlation learning architecture for fast convergence in robot control*, Delft University of Technology, neuroprose archive.

Singh S and Tweedie FJ. (1995) Neural networks and disputed authorship: new challenges, in *4th International Conference on Artificial Neural Networks*.

Sjogaard S. (1991) *A conceptual approach to generalisation in dynamic neural networks*, Aarhus University, Denmark, Masters thesis.

Solla SA. (1988) Learning and generalization in layered neural networks: the contiguity problem, in *Neural networks from models to applications*, Personnaz L and Dreyfus G, Editors, IDSET, Paris, pp168–177.

Squires CS and Shavlik JW. (1991) *Experimental analysis of aspects of the Cascade-Correlation learning architecture*, Machine Learning Research Group, Computer Sciences Department, University of Wisconsin — Madison, TR 91-1, neuroprose archive.

Stone JV and Lister R. (1994) On the relative time complexities of standard and conjugate gradient back propagation, in *The Fifth Australian Conference on Neural Networks*, Tsoi AC and Downs T, Editors, University of Queensland Electrical and Computer Engineering, pp242–245.

Thimm G and Fiesler E. (1995) *Evaluating pruning methods*, IDIAP, Switzerland, Preprint of paper accepted for publication by ISANN'95.

Thodberg HH. (1991) Improving generalization of neural networks through pruning, *International Journal of Neural Systems*, **1** (4): 317–326.

Thrun SB, Bala J, Bloedorn E, Bratko I, Cestnik B, Cheng J, De Jong K, Dzeroski S, Fisher D, Fahlman SE, Hamann R, Kaufman K, Keller S, Kononenko I, Kreuziger J, Michalski RS, Mitchell T, Pachowicz P, Reich Y, Vafaie H, Van de Welde W, Wenzel W, Wnek J and Zhang J. (1991) *The MONK's problems — a performance comparison of different learning algorithms*, Carnegie Mellon University, TR CMU-CS-91-197.

Tolstrup N. (1995) Pruning of a large network by Optimal Brain Damage and Surgeon: an example from biological sequence analysis, *International Journal of Neural Systems*, **6** (1): 31–42.

Tsaptsinos D, Mirzai AR and Leigh JR. (1992) Matching the topology of a neural net to a particular problem: preliminary results using correlation analysis as a pruning tool, in *Artificial Neural Networks* 2, Aleksander I and Taylor J, Editors, North-Holland, pp957–960.

Vamplew P and Adams A. (1991) *Real world problems in backpropagation: missing values and generalisability*, Artificial Neural Network Research Group, Department of Computer Science, University of Tasmania, TR R91-4.

Wang S-D and Hsu C-H. (1991) A self growing learning algorithm for determining the appropriate number of hidden nodes, in *International Joint Conference on Neural Networks*, IEEE, pp1098–1104.

Waugh S. (1994a) *Dynamic learning algorithms*, Artificial Neural Network Research Group, Department of Computer Science, University of Tasmania, TR R94-2.

Waugh S. (1994b) *Extensions to Cascade-Correlation training*, Artificial Neural Network Research Group, Department of Computer Science, University of Tasmania, TR R94-8.

Waugh S. (1995a) Extensions to Cascade-Correlation training, in *The Sixth Australian Conference on Neural Networks,* Charles M and Latimer C, Editors, University of Sydney, Electrical Engineering, pp21–24.

Waugh S. (1995b) Generating data sets for benchmarking, in *IEEE International Conference on Neural Networks,* IEEE and Causal Productions, pp2145–2148.

Waugh S. (1995c) *TasCas — a Cascade-Correlation simulator,* Artificial Neural Network Research Group, Department of Computer Science, University of Tasmania, TR R95-9, URL: ftp://ftp.cs.utas.edu.au/pub/ANNRG/Software/tascas4.0.tar.gz.

Waugh S and Adams A. (1993) *Comparison of inductive learning of classification tasks by neural networks,* Artificial Neural Network Research Group, Department of Computer Science, University of Tasmania, TR R93-5, extended abstract appears in The 6th Australian Joint Conference on Artificial Intelligence, World Scientific, (1993), p447.

Waugh S and Adams A. (1994) Connection strategies in Cascade-Correlation, in *The Fifth Australian Conference on Neural Networks,* Tsoi AC and Downs T, Editors, University of Queensland, Electrical and Computer Engineering, pp1–4.

Waugh S and Adams A. (1995) *Pruning within Cascade-Correlation,* in *IEEE International Conference on Neural Networks* IEEE and Causal Productions: Perth, WA. pp1206–1210.

Weigend AS, Rumelhart DE and Huberman BA. (1990) Generalization by weight-elimination with application to forcasting, in *Advances in Neural Information Processing Systems 3,* Lippmann RP, Moody JE and Touretzky DS, Editors, Morgan Kaufmann, pp875–882.

Weigend AS, Rumelhart DE and Huberman BA. (1991) Generalization by weight-elimination applied to currency exchange rate prediction, in *International Joint Conference on Neural Networks,* IEEE, pp837–841.

Weiss SM and Kulikowski CA. (1991) *Computer systems that learn: classification and prediction methods from statistics, neural nets, machine learning, and expert systems,* Morgan Kaufmann: San Mateo.

Wynne-Jones M. (1991a) Constructive algorithms and pruning: improving the multilayer perceptron, in *13th IMACS World Congress on Computation and Applied Mathematics,* Vichnevelsky R and Milier JJH, Editors, pp747–750.

Wynne-Jones M. (1991b) Node splitting: a constructive algorithm for feed-forward neural networks, in *Advances in Neural Information Processing Systems 4,* Moody JE, Hanson SJ and Lippmann RP, Editors, Morgan Kaufmann, pp1072–1079.

Yang J. (1991) *Experiments with the Cascade-Correlation Algorithm,* Department of Computer Science, Iowa State University, TR 91-16.

Yeung D-Y. (1991) Automatic determination of network size for supervised learning, in *Internation Joint Conference on Neural networks,* IEEE, pp158–164.

Zheng Z. (1993) A benchmark for classifier learning, in *The Sixth Australian Conference on Artificial Intelligence,* Rowles C, Liu H and Foo N, Editors, World Scientific, pp281–286.