

GUARANTEEING GENERALISATION IN NEURAL NETWORKS

John Gareth Polhill

**A Thesis Submitted for the Degree of PhD
at the
University of St Andrews**



1995

**Full metadata for this item is available in
St Andrews Research Repository
at:**

<http://research-repository.st-andrews.ac.uk/>

Please use this identifier to cite or link to this item:

<http://hdl.handle.net/10023/12878>

This item is protected by original copyright

Guaranteeing Generalisation in Neural Networks



A thesis submitted to the
UNIVERSITY OF ST. ANDREWS
for the degree of
DOCTOR OF PHILOSOPHY

By

John Gareth Polhill

Department of Mathematical and Computational Sciences
University of St. Andrews

October 1995



ProQuest Number: 10170673

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



ProQuest 10170673

Published by ProQuest LLC (2017). Copyright of the Dissertation is held by the Author.

All rights reserved.

This work is protected against unauthorized copying under Title 17, United States Code
Microform Edition © ProQuest LLC.

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

TH

B970

Abstract

Neural networks need to be able to guarantee their intrinsic generalisation abilities if they are to be used reliably.

Mitchell's concept and version spaces technique is able to guarantee generalisation in the symbolic concept-learning environment in which it is implemented. Generalisation, according to Mitchell, is guaranteed when there is no alternative concept that is consistent with all the examples presented so far, except the current concept, given the bias of the user. A form of bidirectional convergence is used by Mitchell to recognise when the no-alternative situation has been reached.

Mitchell's technique has problems of search and storage feasibility in its symbolic environment. This thesis aims to show that by evolving the technique further in a neural environment, these problems can be overcome.

Firstly, the biasing factors which affect the kind of concept that can be learned are explored in a neural network context. Secondly, approaches for abstracting the underlying features of the symbolic technique that enable recognition of the no-alternative situation are discussed. The discussion generates neural techniques for guaranteeing generalisation and culminates in a neural technique which is able to recognise when the best fit neural weight state has been found for a given set of data and topology.

I, John Gareth Polhill, hereby certify that this thesis, which is approximately 95 000 words in length, has been written by me, that it is the record of work carried out by me and that it has not been submitted in any previous application for a higher degree.

Date: 8.4.96 Signature of candidate: _____

I was admitted as a research student under Ordinance No. 12 in 1991 and re-registered as a candidate for the degree of Doctor of Philosophy in 1992; the higher degree study for which this is a record was carried out in the University of St. Andrews between 1991 and 1995.

Date: 8.4.96 Signature of candidate: _____

In submitting this thesis to the University of St. Andrews I understand that I am giving permission for it to be made available for use in accordance with the regulations of the University Library for the time being in force, subject to any copyright vested in the work not being affected thereby. I also understand that the title and abstract will be published, and that a copy of the work may be made and supplied to any bona fide library or research worker.

Date: 8.4.96 Signature of candidate: _____

I hereby certify that the candidate has fulfilled the conditions of the Resolution and Regulations appropriate for the degree of Doctor of Philosophy in the University of St. Andrews and that the candidate is qualified to submit the thesis in application for that degree.

Date: 8/4/96 Signature of supervisor: _____

ARS LONGA, VITA BREVIS EST

Acknowledgements

This thesis would not have been possible without the help of Mike Weir, my supervisor, and the financial support of the EPSRC. I am also extremely grateful to my parents, Chris and John Polhill for their financial and moral support. I would like to thank Inman Harvey at the School of Cognitive Sciences, Sussex University for some helpful advice on implementing genetic algorithms, and the Department of Mathematical and Computational Sciences at St. Andrews University for providing the computer facilities and putting up with all the demands I made on CPU time (with special thanks to Brian, Norman, Dave and Colin; and the secretaries, Helen and Margaret). I would like to thank Bath Information and Data Services for their extremely useful document search facilities. I would like to thank Katy Harrison for helping me through some difficult times, the Wild Bunch for helping me through some good times, Robert Bruce for being my guru, Gordon McPhate for putting up with me for a year and Tom Sambrook for the chrysanthemums. Last, and most, I would like to thank the Frimble Wizard, for the magic spell..

In Defence of No Defence

A great deal of neural network research is funded by military grants. I would like to make it clear that I do not approve of defence work, and that I would prefer it if this thesis was not used in any way as part of a defence project or a defence-funded project. My reasons for opposing defence work are simple. Firstly, whilst I accept that in certain situations, war is the only alternative short-sighted people can see, I do not think it is right to encourage it by providing more and more sophisticated means of defending ourselves or attacking others. Secondly, and more importantly, it is becoming increasingly clear that many clients of the British defence industry are not to be trusted with a firework, never mind tanks, fighter-planes, laser-guided missiles, etc. Finally, anyone who reads this is supposed to be intelligent, and violence is not an intelligent way to solve a problem.

Contents

1	Introduction.....	1
1.1	Introduction.....	1
1.2	Introduction to Symbolic AI.....	3
1.2.1	Search Techniques.....	4
1.2.1.1	Depth-First Search.....	4
1.2.1.2	Breadth-First Search.....	7
1.2.1.3	Best-First Search.....	8
1.2.1.4	Bidirectional Search.....	9
1.2.2	Introduction to Learning in Symbolic AI.....	12
1.2.3	Assessment of Symbolic AI.....	14
1.3	Introduction to Neural Networks.....	17
1.3.1	Specification of Neural Networks.....	18
1.3.2	Training Neural Networks.....	21
1.3.2.1	The Perceptron.....	23
1.3.2.2	Back-Propagation.....	26
1.3.2.3	Introduction to Genetic Algorithms for Training Neural Networks.....	31
1.3.3	Assessment of Neural Networks.....	34
1.4	Can Symbolic AI and Neural Networks Co-operate?.....	37
1.5	Summary of the Rest of the Thesis.....	39
2	Mitchell's Symbolic Technique.....	41
2.1	Introduction.....	41
2.2	Searching For No Alternative.....	45
2.2.1	The Search Space.....	45
2.2.2	Bidirectional Search in Mitchell's Technique.....	49
2.2.3	Candidate Elimination.....	53
2.2.4	The Version Space Algorithm and Examples.....	57
2.2.4.1	A Simple Example.....	58
2.2.4.2	A More Complex Example.....	59
2.2.5	Formalism of Mitchell's Technique.....	68
2.3	Problems With Mitchell's Technique.....	76
2.3.1	When No-Alternative Cannot be Found.....	77
2.3.2	When the Data are Inconsistent.....	79

	2.3.3	When the Concept is Disjunctive	82
	2.3.4	Controlling the Size of the Boundary Sets	83
	2.4	Conclusion	85
3		Generalisation In Neural Networks.....	87
	3.1	Introduction.....	87
	3.2	Issues in Generalisation	91
	3.2.1	Assumptions about the Underlying Function.....	91
	3.2.2	Assumptions about the Data	94
	3.3.3	The Over-Fit/Under-Fit Dilemma	94
	3.2.4	Over-Fit and Under-Fit in the Classification Paradigm.....	97
	3.3	Generalisation in Neural Networks in the Literature	98
	3.3.1	The Validation Technique	98
	3.3.2	Average Generalisation Error	104
	3.3.3	Vapnik Chervonenkis Theory.....	109
	3.3.4	Comparing VC and Average Generalisation Theories	113
	3.3.5	Bayesian Frameworks for Generalisation	115
	3.4	The Mitchellian View	120
	3.4.1	Generalisation	120
	3.4.2	Relating Mitchell's Technique to Neural Networks	121
	3.5	Conclusion	124
4		Issues in Topology Determination.....	127
	4.1	Introduction.....	127
	4.2	The First Hidden Layer.....	134
	4.2.1	Regions and the Requirement for a Second Hidden Layer	134
	4.2.2	Overcoming the Requirement for a Second Hidden Layer	137
	4.2.2.1	1D Input Space	137
	4.2.2.2	2D Input Space	139
	4.2.2.3	Higher Dimensions of Input Space	143
	4.2.3	One Hidden Layer or Two?	143
	4.3	The Second Hidden Layer	146
	4.3.1	Number of Units Required.....	148

	4.3.1.1	One Output Unit.....	148
	4.3.1.2	Many Output Units	151
	4.3.2	Further Hidden Layers	153
4.4		A Brief Look at Sigmoid Units.....	154
	4.4.1	The Increased Power of Sigmoid Units	154
	4.4.2	Approximation of the Chequer-Board With a Single Hidden Layer Using Sigmoid Units	158
	4.4.3	The Possibility For Using More Than Two Hidden Layers.....	160
4.5		Conclusion	164
		Appendix	166
4.A		Mapping the 4-Bit Parity Problem onto a Unit Square Using Two Units in the First Hidden Layer, Whilst Preserving Their Separability	166
5		Neural Implementation Based on Weight Space	170
	5.1	Theory of the Weight Space Technique.....	170
		5.1.1 Bidirectional Search Using Hypercones.....	173
		5.1.1.1 Monotonically decreasing angle between N and X	175
		5.1.1.2 Freedom of movement and hypercones.....	176
		5.1.1.3 Convergence.....	181
	5.1.2	Implementing Candidate Elimination Using Error Functions	181
		5.1.2.1 Minimising the Distance from the Current Hypercone	182
		5.1.2.2 An Error Function which is Zero for all Correctly Classified Patterns	184
		5.1.2.3 Combining Correct Classification with Minimum Distance from the Current Hypercone	186
	5.1.3	Relation to the Symbolic Technique	189
5.2		Algorithm and Experimental Results	192
	5.2.1	Algorithm	192
	5.2.2	Results for a Simple Experiment	195
	5.2.3	Problems with the Technique	198

5.3	Generalisation For Hidden Units	200
5.3.1	Symmetries in Hidden Units	201
5.3.2	Measuring Weight Space Angle with Hidden Units	203
5.4	Conclusion	206
6	Neural Implementation Based on IO Space.....	208
(i)	Development of Theory	208
6i.1	Partially Ordering the IO.....	208
6i.1.1	Representing the IO.....	208
6i.1.2	Partially Ordering the Grids	213
6i.2	Learning Using Grids.....	216
6i.2.1	Construction and Use of Grids	216
6i.2.2	Generalisation and the Relation to Mitchell	219
6i.2.3	Comparison with the Validation Technique	220
6i.2.4	Features of Learning Using Grids	222
6i.3	The H/H0 Paradigm	227
6i.3.1	Methodology of the H/H0 Paradigm.....	227
6i.3.2	Experiments on the H/H0 Paradigm	229
6i.3.2.1	Efficient Exhaustive Search	229
6i.3.2.2	Demonstration of Termination	232
6i.3.2.3	Maximum Number of Hypothetical Gridpoints at Locking	236
6i.3.3	Assessment of the H/H0 Paradigm.....	239
6i.4	The B/W Paradigm	240
6i.4.1	Using the Partial Ordering.....	240
6i.4.2	Inherently Better Fit	242
6i.4.3	False Locking Using the B/W Paradigm	246
6i.5	The I/I0 Paradigm.....	250
6i.6	Conclusion to Part (i)	254
(ii)	Experiments on the I/I0 Paradigm	257
6ii.1	Experiments and Results	257
6ii.1.1	Example Run	258
6ii.1.2	Comparison with Perfect Training	265
6ii.1.3	Comparison with H/H0	267
6ii.1.4	Comparison with Validation	272

6ii.1.5	Validation Set Size	275
6ii.1.6	Scalability of the Technique	277
6ii.1.7	A Symbolic Problem.....	279
6ii.2	Assessment of the I/I0 Paradigm.....	284
6ii.3	Conclusion to Part (ii)	287
	Appendices	290
6.A	Proof of Locking for the H/H0 Paradigm	290
6.B	Genetic Encoding of Neural Network and Selection Mechanism	292
6.B.1	Genetic Encoding of Neural Network Weight State	292
6.B.2	Selection Mechanism.....	293
6.C	Crossover and Mutation Probabilities.....	294
7	Conclusions and Further Work	296
7.1	Achievements	296
7.2	Further Work.....	298
7.2.1	Keeping I0 Strictly Behind in the Ordering	298
7.2.2	Using the Network Trained from the Previous Familiarisation	299
7.2.3	Relaxing the Training Requirements Further	300
7.2.4	Proposal of New Instances	300
7.3	Conclusion	301
A	Software Manual.....	304
A.1	General Points and File Formats	304
A.1.1	Format of Topology Files	305
A.1.2	Format of Weight Files.....	306
A.1.3	Format of Pattern Files.....	307
A.2	Grid Technique Simulators	307
A.2.1	Perfect Training Version Using Exhaustive Search	308
A.2.2	Imperfect Training Version Using a GA	322
A.3	IO Visualisor.....	331
	Bibliography	341

1 Introduction

1.1 Introduction

Mitchell's concept and version spaces technique¹ is a technique for learning symbolic descriptions of concepts from a set of instances. A series of instances and non-instances of the concept are presented to the learner until there is only a single concept that is consistent with the series. The learner has then learned the concept and will correctly determine whether future examples are instances or non-instances of the concept.

To some extent, when humans learn concepts, much the same kind of processes are happening at an abstract level. For example, when teaching our children to speak we point to objects and recite the word. The child is then expected to repeat the word back to the parent. There are two things to be learned here: firstly, the correct pronunciation of the word, and secondly the class of objects which the word refers to. At first, the child may have no idea of what is being pointed to — nor indeed what is to be inferred from the gesture of pointing. Yet after a series of instances of the word taught by the parent and non-instances corrected by the parent when the child fails to generalise correctly, the word is learned, and the child can distinguish "car" from "lorry" and "van", for example.

We can all remember concepts we may have had difficulty in learning, such as the conditional perfect tense in French, long division in arithmetic, or how to deal with a box junction when driving a car. Although the moment when we finally grasp the concept does not have us all jumping out of the bath shouting "Eureka!" to no-one in particular, it must be said that there is a certain joy at having understood something.

It is this eureka sense of grasping a concept that Mitchell's technique manages to achieve in a computer, in a limited sense, when the correct concept is converged upon. There is no alternative but the learned concept

¹Mitchell, 1982

at this stage and therefore the concept can be used with confidence in future classifications.

Mitchell's technique is based in the domain of symbolic Artificial Intelligence (AI) though its appeal is not restricted to that discipline. The eureka experience should be open to other paradigms and in this thesis the aim is to make it available to neural networks. The view is taken that the tenets of Mitchell's technique which enable the concept to be learned can be abstracted from the symbolic domain in such a way that they can be applied in neural networks.

The potential benefit of Mitchell's technique, as far as neural networks are concerned, is that it enables guaranteed generalisation within the limits of the assumptions made by the user. Authors have yet to harness the generalisation abilities which are a natural and primary ability of neural networks. If this could be done — even in a limited fashion — then neural networks could more readily justify themselves as an alternative approach to conventional classification and function approximation methods.

For Mitchell's technique, there is the benefit of showing a wider applicability than just symbolic AI. Symbolic AI, which is discussed in section 1.2, suffers from certain difficulties — particularly with regard to noise toleration — which limit its use. These limits are imposed by default on Mitchell's technique which, in the abstract, might not be restricted in this way.

Neural networks are introduced in section 1.3. There it is shown that neural networks have their own difficulties. In section 1.4, it is argued that the strengths and weaknesses of symbolic AI and neural networks seem to complement one another. Consequently it is argued that there is scope for making the more general point that neural networks and symbolic AI may be able to borrow ideas from one another — or even co-operate fully to provide more effective hybrid techniques. Existing examples of such co-operation are discussed.

This thesis is an attempt to evolve a technique based on borrowing ideas from Mitchell's technique to provide a benefit both for Mitchell's technique in the abstract, and for neural networks. Section 1.5 gives a

summary of the rest of the thesis which documents the research undertaken in attempting this goal.

1.2 Introduction to Symbolic AI

Symbolic AI is the domain of Artificial Intelligence which develops techniques for simulating intelligence on the basis of the physical symbol system hypothesis of Newell and Simon:

A physical symbol system has the necessary and sufficient means for general intelligent action.²

In essence, a physical symbol system is a system which operates on a set of symbols which have some kind of structure. Symbolic AI techniques are therefore techniques which are based around the manipulation of symbols.³ More specifically, the techniques are characterised by a mind-based approach. This means that they approach problems in a high-level fashion, using the symbols and rules that humans use to describe the problems, without insisting on any neurological or biological realism:

Artificial Intelligence research is not aimed at simulating neural networks, for it is based on another kind of faith: that probably there are significant features of intelligence which can be floated on top of entirely different sorts of substrates than those of organic brains.⁴

The key to the symbolic approach to problems is state representation. The capability to represent states within the problem domain enables the representation of the solution state and the current state. Together with a set of rules for proceeding from one state to another, symbolic AI approaches are able to find the desired state from a given current state. The basic mechanism by which this is done is termed *search*. Section 1.2.1 outlines some search techniques.

²Newell & Simon, 1976, p. 116

³Rich & Knight, 1991, p. 8

⁴Hofstadter, 1980, p. 572

There are many applications of symbolic AI in domains such as vision, natural language understanding, making plans, and game playing. This thesis, however, will draw from machine learning — another domain of AI. Section 1.2.2 indicates some approaches to machine learning in symbolic AI.

Section 1.2.3 gives some assessments of the symbolic approach to problem-solving.

1.2.1 Search Techniques

For the purposes of illustrating the search techniques, consider the famous problem by Lewis Carroll of getting from one English word to another of the same length by changing one letter at a time. All the intervening words must also be English. It is desired to achieve this in as few steps as possible. For example, it is possible to get from BLACK to WHITE in seven steps:

	BLACK
1	BLANK
2	BLINK
3	CLINK
4	CHINK
5	CHINE
6	WHINE
7	WHITE

The search can be thought of as a tree of words starting from the first word as the trunk, with branches extending for each subsequent word, which differ by one letter from their predecessors. Figure 1.2 shows some of the branches for the BLACK to WHITE search.

1.2.1.1 Depth-First Search

One approach to the problem is to start with BLACK and then find a word that differs from BLACK by one letter, such as SLACK. Then the search proceeds by finding a word that differs from SLACK by one letter, and so on. The search is stopped when the goal is reached or some prespecified maximum number of words have been visited, in which case the strategy

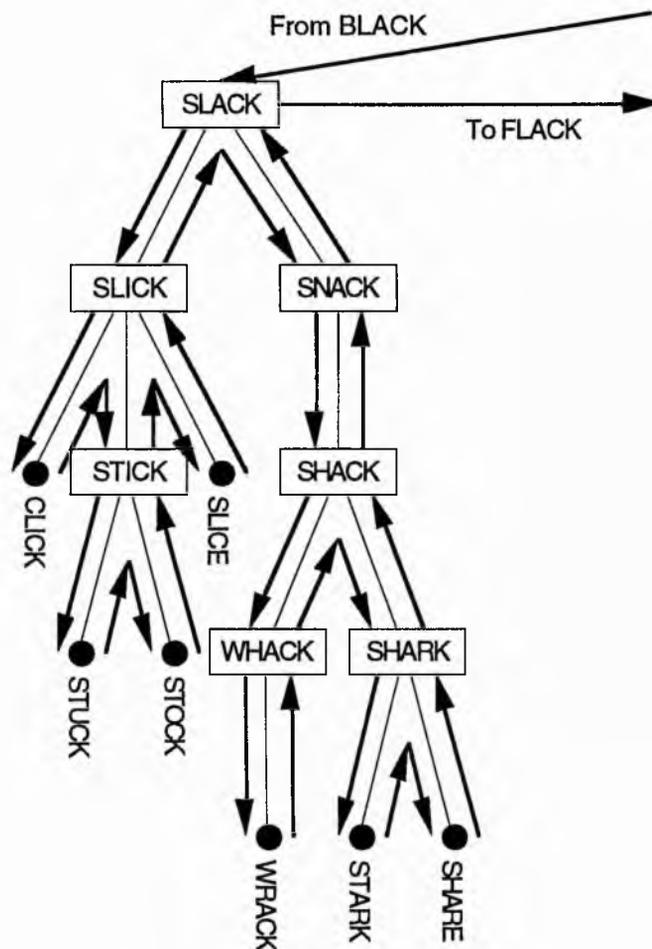


Figure 1.3 — *How the right hand side of the search tree in figure 1.2 is searched using the depth-first strategy.*

Even given such a measure, the depth first search does not guarantee the minimum number of steps. For example, it might find a longer route from BLACK to WHITE, such as BLACK, BLINK, BRINK, BRINE, BRIBE, TRIBE, TRITE, WRITE, WHITE, with eight steps rather than seven.

Depth-first search without path storage has the advantage of requiring little storage space, but the disadvantages of being costly in terms of time taken to find a solution. It is also not able to guarantee the solution in the minimum number of steps, should that be desired (as it is with this problem). Often, however, search spaces are so large, and problems so hard, that it is enough to be able to find a solution and depth-first search may be capable of doing this in various cases.

1.2.1.2 Breadth-First Search

An alternative strategy is to generate all the words that have one letter different from BLACK, i.e. SLACK, FLACK, BLOCK and BLANK, and then find the corresponding words for each of these, and so on, until WHITE is found. This is the breadth-first search strategy. Figure 1.4 shows how breadth-first search expands the first two layers of the search tree in figure 1.2 layer by layer. Ideally, breadth-first search should only have to remember the words in the current layer being expanded. This is so that the next layer can be produced. If there is the possibility of going back to a previous layer from the current layer, however, as is the case in this problem, the previous layers will have to be remembered as well to avoid repetitions.

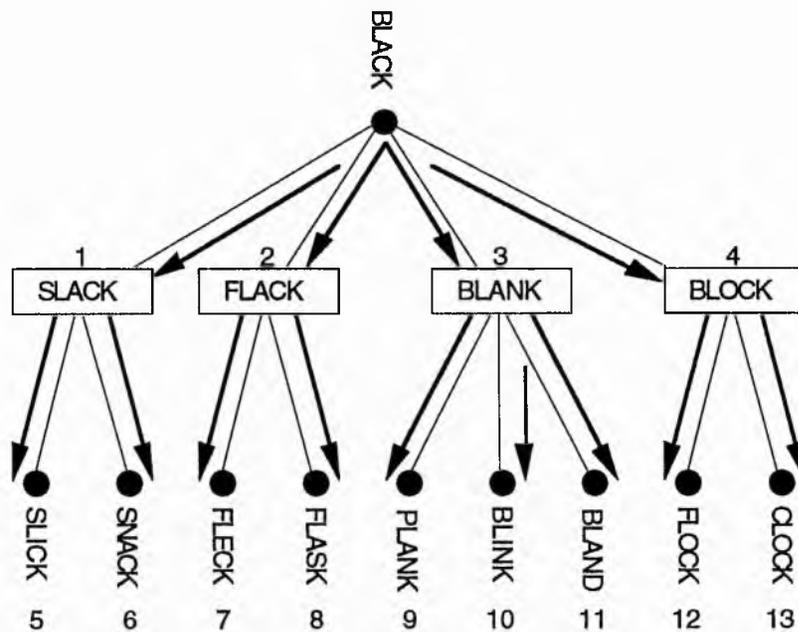


Figure 1.4 — Expansion of the first two layers of the search tree in figure 1.2 by breadth-first search. The order in which the nodes are expanded is indicated next to each node.

In contrast to depth-first search, breadth-first search can guarantee the minimum number of steps, should that be required. However, there is a cost. By the time the seventh layer of the search is reached (which contains WHITE), there are 134 words in the layer. A large number of problems, including this one, have an exponential growth in the number of nodes to

expand during the search. In a game of chess, for example, where (for the sake of ease) let us assume there are 16 possibilities for each move, there are 16^7 , or roughly 10^8 possible states of play after seven moves.

Breadth-first search therefore has the advantage of being able to guarantee the solution with the fewest steps, if that is required. The disadvantage is that for large search spaces there is an excessive demand on memory.

1.2.1.3 Best-First Search

Large search spaces cannot be feasibly searched using brute-force methods such as breadth-first and depth-first search. It is therefore necessary to find the means by which the search can be cut down. Such means are called heuristics. For example, a heuristic for depth first search might be to use a *depth-bound*. This means the search backtracks when a pre-defined maximum number of steps have been taken, to cut losses in the sense of not exploring a branch too far down. Such heuristics are blind heuristics, and may apply for any problem. There are also knowledge-based problem-specific heuristics, however, which can be used to explore the search space for the given problem more intelligently.

Best-first search uses a heuristic evaluation function to decide which nodes in the current search tree are to be explored first. The heuristic evaluation function for a given problem gives an indication of how good a candidate move is.

In the Lewis Carroll game, for example, the heuristic evaluation function might be the number of letters which are equal to the corresponding letters in the target word. Since the minimum number of moves is also required for this problem, a fraction is added to this heuristic evaluation function. Let μ_{max} be the maximum number of moves which will be considered by the search. Let μ be the number of moves so far. The fraction $(\mu_{max} - \mu) / \mu_{max}$ will give a number which is less than 1, and is larger for a fewer number of moves. Suppose there is a heuristic evaluation function, $h(w)$, which is to be maximised. There is the following equation for $h(w)$, where w is a word, and C_w is the number of letters in w that are equal to the target word:

$$h(w) = C_w + \frac{\mu_{max} - \mu}{\mu_{max}} \quad [1.1]$$

Let $\mu_{max} = 10$ for the purposes of this example. Best-first search begins by expanding the first node (which is BLACK in the example). The values of the heuristic evaluation function are calculated for each new node generated, and the nodes and their heuristic evaluation function values are put on a list of nodes to be explored. The search then proceeds by taking the first item with the highest value of the heuristic evaluation function from the list of nodes to be explored, and adding the nodes attained by expanding this item (which is always a leaf node on the current search tree) to the front of the list of nodes to be explored, along with their heuristic evaluation function values. Figure 1.5 shows how best-first search would approach the problem in the search tree of figure 1.2.

Best-first search does not guarantee the solution with the minimum number of steps, however, should that be required. For the restricted search space of figure 1.2, the search in figure 1.5 shows the search finding the CHINE branch, which leads to WHITE in the optimum number of moves, when the search is done properly. With all five-letter words available to the search, rather than just those in figure 1.2, a sub-optimum path may be found if SLACK is the first word expanded from BLACK. Figure 1.6 shows the nodes expanded in a search which takes 8 moves instead of 7. The 8 steps in the solution in figure 1.6 are: BLACK, SLACK, SLICK, SLICE, SPICE, SPINE, SHINE, WHINE and WHITE.

Best-first search is a more feasible method of searching the space than breadth-first search, and will often take less time than depth-first search.

1.2.1.4 Bidirectional Search

An alternative strategy is to search from two directions — one starting at the initial state, the other starting at the goal. The search then proceeds in two directions, and hopefully meets in the middle. Bidirectional breadth-first search will gain over straightforward breadth-first search in that fewer nodes in the search tree must be expanded. For example, in the BLACK to WHITE search, only 109 nodes would need to be examined in order to find the solution. With a unidirectional search the number of

nodes examined is 332 — just over three times the number needed with a bidirectional search.

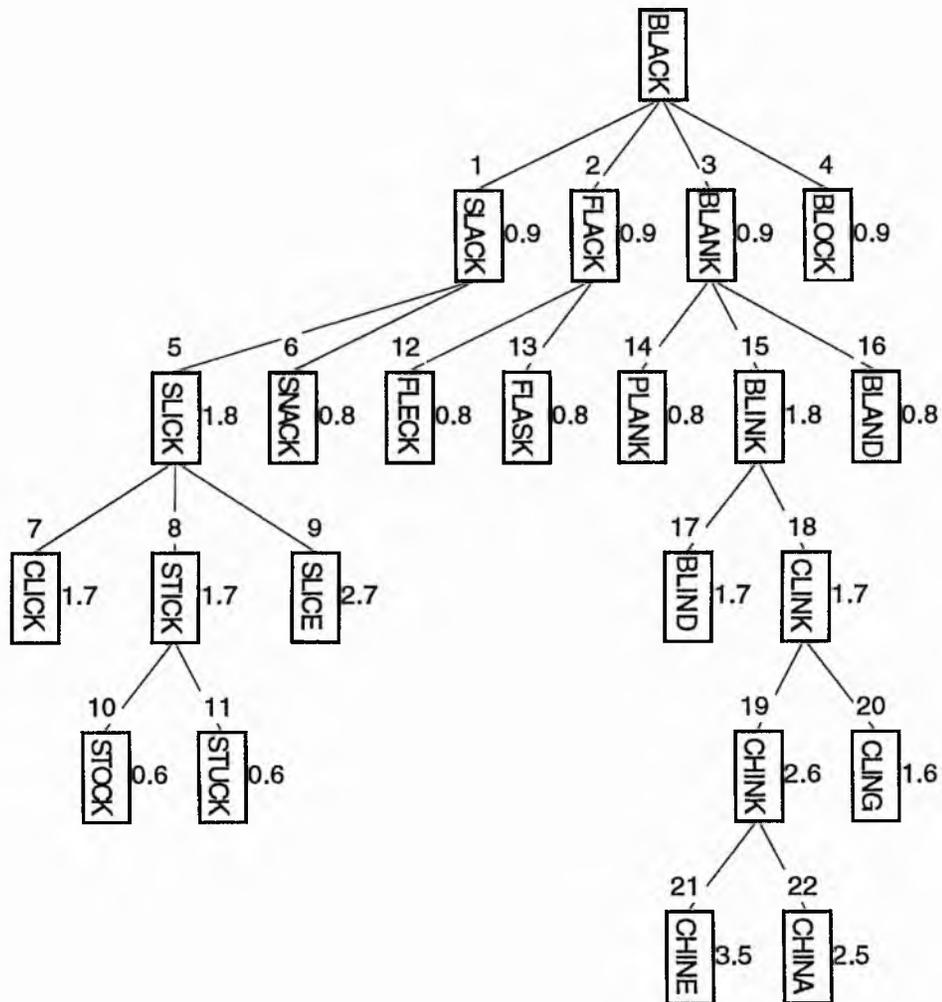


Figure 1.5 — Searching the tree in figure 1.2 using a best-first search. The numbers above each word indicate the order in which the words are explored. The value of the heuristic evaluation function is indicated to the right of each word.

Bidirectional best-first search, however, is not such a good idea. This is because it will not necessarily be the case that each direction will explore the same branch of the search. In general, any search employing a

heuristic has the possibility of suffering from this problem.⁵ Figure 1.7 shows the BLACK and WHITE directions passing each other in the search.

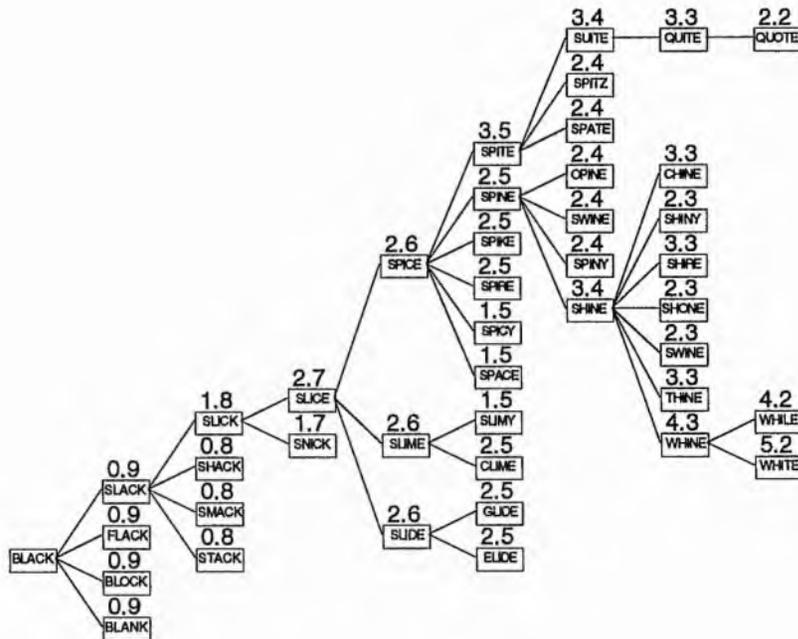


Figure 1.6 — The search tree for a non-optimum best-first search to get from BLACK to WHITE. The number above each word is the heuristic evaluation function value for that word.

Bidirectional depth-first search may also fail to make any savings, if the depth-bound is set inappropriately. Too high a depth-bound, and the two directions may pass each other. Too low a depth-bound, and the two directions cannot meet.

⁵Rich, 1983, p. 60

learning. Machine learning has many different paradigms,⁶ indicated below, all of which are aimed at finding a concept description that will be useful in future.

- *Rote Learning.* Here, the learner memorises the positive and negative training instances. No representation of the concept is needed. In order for future instances to be correctly classified, they must belong to the set of training instances.
- *Learning by Instruction.* In this case, the learner is given a concept description by a teacher. This must be translated into the learner's own internal concept description which can then be used in future.
- *Learning by Analogy.* The learner transfers an already acquired concept into another analogous domain.
- *Learning from Examples.* Under ideal circumstances, in which the data are noiseless, a concept description is found by the learner which matches all the positive instances and does not match any of the negative instances. When the data are noisy, heuristics must be employed which enable the correct concept to be learned.
- *Learning by Observation and Discovery.* This is also known as *unsupervised learning*. With no teacher, there is no given classification of the instances. It is up to the learner to make its own classifications. There is also the possibility that there is more than one concept to be learned from the observations made.

This thesis will be looking at learning from examples. Given a representation of the concepts, it is possible to use the instances as a basis for a search of concept space. Mitchell's technique is an example of this, and it has been described as a least-commitment exhaustive breadth-first search of concept space.⁷ (The least-commitment strategy is that of not trying to patch up incomplete solutions, but waiting for more information

⁶Carbonell et al, 1983, pp. 8-11

⁷Rich & Knight, 1991, p. 468

before making a more complete specification.⁸⁾ This description, however, does not do the technique justice. It ignores the savings made by the bidirectional search that Mitchell uses, and also ignores the extensive pruning carried out by the technique during the search, which is enabled by a partial ordering of concepts. Chapter 2 has more detail on this.

There are other techniques for searching concept space. Winston's arch learner,⁹ for example, employs a depth-first strategy.¹⁰ The approach relies on near-miss instances, which have a single difference from positive instances of the concept. Winston's program requires a very careful teacher if the concept is to be learned quickly, since it is sensitive to the order of presentation of the instances.¹¹

1.2.3 Assessment of Symbolic AI

Symbolic AI techniques are used in a wide variety of real-world domains. The main practical application to come from research in symbolic AI has been expert systems. Expert systems are the embodiment in a computer of the rules which experts apply when solving problems in their particular domain of expertise. For more detail, see Charniak and McDermott.¹²

Examples of expert systems are programs such as MYCIN¹³ and CADUCEUS,¹⁴ which give medical diagnoses, and PROSPECTOR,¹⁵ which predicts the location of deposits of various ores. These programs provide accurate information, where possible, and are also able to give explanations of how and why they reach the conclusions they make. These explanations inform the user of the rules invoked during decision making.

⁸Rich, 1983, p. 258

⁹Winston, 1975

¹⁰Rich & Knight, 1991, p. 469

¹¹Rich & Knight, 1991, p. 462

¹²Charniak & McDermott, 1985, Ch. 8

¹³Shortliffe, 1976

¹⁴Pople et al, 1975

¹⁵Duda et al, 1980

Expert systems exploit the narrowness of the domains in which they are applied to enable computers to make suggestions. Problems with large numbers of rules and too much computational information cannot be addressed by computers in a reasonable amount of time. By narrowing the domain in which they are applied, computers can be effectively used.

However, this narrowness leads to brittleness. This brittleness means that an expert system, once designed, can only cope with the problem domain it was designed to cope with. The rules and general principles cannot be applied in other domains. Brittleness is discussed by several authors,¹⁶ and in general, it refers not only to the narrowness of the domains AI systems can be effectively be used in, but also to the intolerance of these systems to noisy, novel and inconsistent data within a given domain.¹⁷

Duda and Shortliffe see this as a paradox in simulating intelligent behaviour:

Paradoxically, it has proved much easier to emulate the problem-solving methods of some kinds of specialists than to write programs that approach a child's ability to perceive, to understand language, or to make "commonsense" deductions. Many human experts are distinguished by their possession of extensive knowledge about a very narrow class of problems. It is this very limitation that makes it feasible to provide a computer with enough of the knowledge needed to perform those tasks effectively.¹⁸

Symbolic AI aims to provide a high level account which as a consequence tends to deal with clean rather than noisy data. In order to cope with noise, rules must be seen as guidelines rather than statements of absolute truth and symbolic representation schemes must allow for the fact that reality does not always divide itself up into neat, non-intersecting compartments. Noise toleration may be improved by a more flexible, adaptive approach.

¹⁶E.g. Aha, 1992, p. 267; Boden, 1987, p. 499; Hanson & Burr, 1990, p. 472; Holland, 1986

¹⁷Coombs et al, 1992, p. 247

¹⁸Duda & Shortliffe, 1983, p. 262

Other approaches, outwith mainstream symbolic AI, take these factors into consideration and are also sometimes incorporated into symbolic AI. Fuzzy logic, for example, is a formalism for approximate reasoning developed by Zadeh,¹⁹ and another example is the use of genetic algorithms,²⁰ which adopt an evolutionary approach to system design.

Noise has more implications in concept learning using symbolic AI than in other domains in which symbolic AI is applied. This is because it can give rise to incorrectly learned concepts, or even the loss of the ability to learn anything. Mitchell's technique requires unwieldy adaptations to cope with noise (see chapter 2). Other techniques for coping with noise in concept learning using symbolic AI involve using nearest-neighbour techniques.²¹

¹⁹Zadeh, 1973

²⁰Goldberg, 1989

²¹E.g. Aha & Kibler, 1989; Hirsh, 1990

1.3 Introduction to Neural Networks

Neural networks adopt a different approach to simulating intelligence from symbolic AI. They may be seen as a brain-based approach, rather than a mind-based approach. They offer the hope of being able to simulate the aspects of human intelligence that symbolic AI has difficulty with, such as vision and speech, though it is also believed that they might model higher psychological processes:

Though the appeal of PDP models is definitely enhanced by their physiological plausibility and neural inspiration, these are not the primary bases for their appeal to us. We are, after all, cognitive scientists, and PDP models appeal to us for psychological and computational reasons. They hold out the hope of offering computationally sufficient and psychologically accurate mechanistic accounts of the phenomena of human cognition which have eluded successful explication in conventional computational formalisms; and they have radically altered the way we think about the time-course of processing, the nature of representation, and the mechanisms of learning.²²

However, not all authors with an interest in neural networks are concerned with intelligence and biological reality:

Our subject matter is computation by artificial neural networks. The adjective "neural" is used because much of the inspiration for such networks comes from neuroscience, not because we are concerned with networks of real neurons. Brain modelling is a different field and ... our prime concern is with what the artificial networks can do, and why. ...

We emphasise the theoretical aspects of neural computation. Thus we provide little or no coverage of ... implications for cognitive science or artificial intelligence.²³

²²McClelland et al, 1986, p. 11

²³Hertz et al, 1991, p. xix

Therefore, there is an argument for asserting that neural networks are a discipline in their own right, distinct from AI. Some of the early enthusiasm for AI, and its subsequent failure to keep all the promises it made, has meant that those in neural networks are more cautious about stating their goals. Rather than attempting a complete simulation of human intelligence, neural networks are also to be seen as useful tools in the workshop of the computer scientist and the statistician.

Although it is some fifty years since the pioneering work of McCulloch and Pitts,²⁴ real enthusiasm for neural networks did not develop until the advent of training algorithms that could train more sophisticated neural networks than the Perceptron.²⁵ This did not happen until the last decade, and hence neural networks have yet to establish themselves as being as reliable as, and superior to standard techniques. This is necessary if neural networks are to survive as a discipline. Papers with titles such as "Neural Networks: A New Method for Solving Chemical Problems or Just a Passing Phase?"²⁶ serve to emphasise this point, and indicate that there is some urgency if neural networks are to establish themselves as valid replacements for standard techniques.

Section 1.3.1 describes the terminology of neural networks, and how they are constructed. Section 1.3.2 discusses three methods for training neural networks. Section 1.3.3 gives an assessment and critique of neural networks.

1.3.1 Specification of Neural Networks

A neural network consists of a set of interconnected neural processing elements, called *units*. These are sometimes called *neurons*, since they are based on a loose simulation of biological neurons. Each unit consists of a set of *fan-in* connections which provide the input to the unit, *fan-out* connections along which the output of the unit is sent, and functions for relating the input of the unit to the output of the unit. The connections are

²⁴McCulloch & Pitts, 1943

²⁵Rosenblatt, 1958

²⁶Zupan & Gasteiger, 1991

weighted, and the process of training the neural network to give the desired behaviour is that of deducing these weights.

The input to a unit is called the *excitation*. The excitation function is a function of the outputs of the units at the other end of the fan-in connections and the weights assigned to those connections. Typically, the excitation takes the form of equation [1.2] (where N is the number of fan-in connections), though sometimes a product, or combination of products and sums is used instead of the sum.

$$excitation_j = \sum_{i=1, N} output_i \times weight_{ji} \quad [1.2]$$

The excitation is used to compute the *activation* of the unit. The activation of the unit refers to the unit's state, which is usually either *on* or *off*, though sometimes the analogue value is of interest. In the binary case, an activation value of 1 represents the unit being on, and an activation value of 0 represents the unit being off. The activation function must be non-linear in multi-layer neural networks, since the addition of an extra layer when the neural network uses linear activation functions does not increase its capabilities.²⁷ Typical non-linearities are the Heaviside, or step-function [1.3], a semi-linear function [1.4], or a sigmoid function [1.5] (where β is a positive constant).



$$activation_j = \begin{cases} 1 & excitation_j > 0 \\ 0 & excitation_j \leq 0 \end{cases} \quad [1.3]$$



$$activation_j = \begin{cases} 1 & excitation_j \geq \beta \\ \frac{1}{2} \left(\frac{1}{\beta} excitation_j + 1 \right) & -\beta < excitation_j < \beta \\ 0 & excitation_j \leq -\beta \end{cases} \quad [1.4]$$



$$activation_j = \frac{1}{1 + e^{-\beta \times excitation_j}} \quad [1.5]$$

²⁷Hertz et al, 1991, p. 108

Usually, a *bias* is added to the excitation, which enables the activation to cross the 0.5 activation threshold for any value of the excitation, rather than just zero. Thus, the full excitation function is given in [1.6]:

$$excitation_j = \left(\sum_{i=1, N} output_i \times weight_{ji} \right) + bias_j \quad [1.6]$$

There is then an output function, which provides the final output of the unit given the activation. In this thesis, the output is set to be equal to the activation. The behaviour of a unit is summarised in figure 1.8.

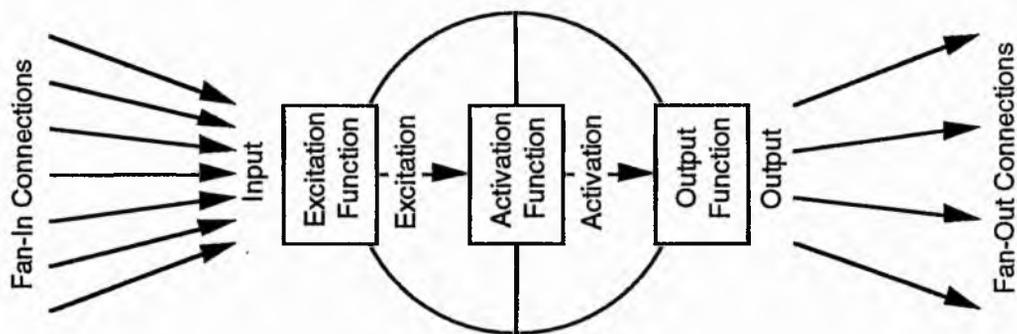


Figure 1.8 — Stages in calculating the output of a neuron from its input.

In some neural networks, there are neurons with no fan-in connections. These are called *input* units, and their outputs are usually given by a set of data. Similarly, there are neurons with no fan-out connections, which are called *output* units. The neural networks considered in this thesis all have input units and output units. Units which are neither input nor output units are called *hidden* units.

The excitation of each input unit is specified by the user, and may be seen as belonging to one axis of an input space. The input space contains the set of all possible inputs to the network. The Input to Output (IO) behaviour of the neural network is the output of each output unit for each point in input space.

The connections in a neural network may, in theory, be from any unit to any other unit. In practice, however, the units are often organised into layers, with the outputs of units in one layer being connected to the inputs of the units in the next layer closer to the layer of output units. These are called *layered feed-forward* networks, and are illustrated in figure 1.9.

Networks with cyclic connections, such as Hopfield networks²⁸ in which each unit sends its output along connections to every other unit, are called recurrent networks.

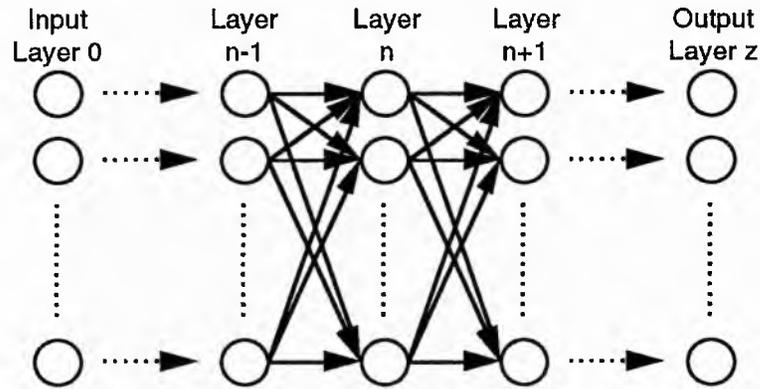


Figure 1.9 — A layered feed-forward network, where all fan-in connections to units in an arbitrarily chosen layer n , come from units in layer $n - 1$, and all fan-out connections go to units in layer $n + 1$.

The organisation of the units and connections in a neural network is called the *topology* of the network. In layered feed-forward networks, the topology is a specification of the number of layers, and the number of units in each layer. Full connection between each layer is usually assumed. The term *feed-forward* will be used to infer layered feed-forward networks henceforth in this thesis, which will only be considering such networks.

1.3.2 Training Neural Networks

The main thrust of much neural network research is the attempt to find a weight state that gives the desired IO behaviour. The process of finding a weight state is called *training*. In this thesis, as in many neural network training frameworks, the desired IO behaviour is trained using a set of p samples from the desired IO behaviour, termed the *training* or *pattern* set. The pattern set may have to be taken from real-world measurements, in which case the desired IO behaviour may not be known for all cases, and erroneous behaviour is only revealed during future trials of the trained network by taking further samples.

²⁸Hopfield, 1982

Rumelhart and Zipser identify four main paradigms for training neural networks:²⁹

- *Auto Associator.* Input patterns are stored in the network and then the network is used to retrieve the patterns as output, on the basis of input which may be corrupted or resembling versions of the patterns stored.
- *Pattern Associator.* Patterns consisting of IO pairs are presented and the output must be associated with the input for each pattern. Once the IO pairs have been trained, the correct output should be retrieved for each trained input. The pattern associator is more general than the auto-associator in that the desired output is not necessarily the same as the input used in training.
- *Classification Paradigm.* This is strictly a subset of the problems tackled by the pattern association paradigm. With pattern association, a given input may have a desired output which involves any number of the output units firing, or the outputs may have analogue targets. In the classification paradigm each output unit is trained to represent the input as belonging to a particular class, as distinct from the classes which are represented by other output units. Thus, with the classification paradigm, exactly one output unit should fire for each input.
- *Regularity detector.* Salient features in the input population are to be discovered by the network. There is no pre-specified output for the network. The system groups together inputs with common salient features, creating its own classification scheme accordingly.

As has been seen, learning by example occurs in symbolic AI, and the neural pattern association and classification paradigms. This thesis will focus on these paradigms of neural network training. It is these paradigms which correspond most closely with the kind of learning that takes place in Mitchell's technique.

²⁹Rumelhart & Zipser, 1986, p. 191

The following sections examine three methods for training neural networks in the classification and pattern association paradigms. In section 1.3.2.1 the Perceptron training algorithm is discussed since it provides an example of step-function architectures which are the subject of the discussion of topology considerations in chapter 4, and are used in the neural networks in chapter 6. This leads on to the discussion of back-propagation in section 1.3.2.2, which is the basis for training the neural networks used in chapter 5. A Genetic Algorithm (GA) may also be used to train neural networks, and a brief introduction to GAs is given in section 1.3.2.3, since it is the training mechanism used in chapter 6.

1.3.2.1 The Perceptron

The Perceptron was one of the earlier neural networks, developed in 1958. It was designed by Rosenblatt,³⁰ and used a feed-forward neural network with the structure outlined in figure 1.10.

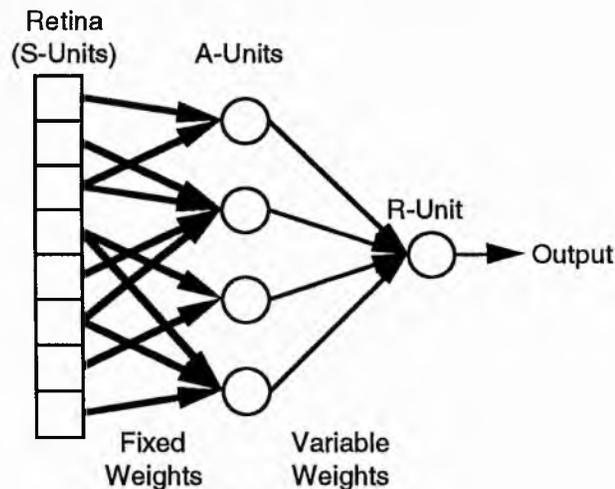


Figure 1.10 — An elementary perceptron with a single output unit.³¹ The S-units are stimulus units, and have fixed weights (represented by the thicker lines) to the A-units. The weights from the A-units to the R-unit are trained.

³⁰Rosenblatt, 1958

³¹Rosenblatt, 1962, p. 99

The S-units are stimulus units, which feed input to the A-units, which are a layer of intermediate units, called "association" units by Rosenblatt.³² The R-unit is a response unit, which gives the output of the network. In general, there may be one or more R-units in a perceptron architecture. The units all have step-function activation functions. The weights to the A-units from the S-units are fixed, and are not trained. From henceforth, therefore, the S-units will be ignored, and the A-units will be treated as the input units. The neural network trained is, in modern terms, effectively a feed-forward network with no hidden layers and step-function activation functions.

The essence of the perceptron training algorithm is as follows. Let the training set, P , contain n patterns, each pattern, p , with an input vector, \mathbf{y}_p , for the I inputs of the perceptron, taken from $\{0, 1\}^I$, and target vector, \mathbf{t}_p , for the J outputs of the perceptron, taken from $\{0, 1\}^J$. Let w_{ji} be the weight from input unit i to output unit j of the perceptron, where $i = 0$ implies the bias unit.

Weights are initialised by choosing small random values. Then the weights are changed for each pattern according to equation [1.7]:

$$\Delta_p w_{ji} = (t_{pj} - o_{pj}) y_{pi} \quad [1.7]$$

where $\Delta_p w_{ji}$ is the change to be made to w_{ji} for pattern p , and o_{pj} is the output of output unit j for pattern p , calculated using a threshold activation function as per equation [1.8]:

$$o_{pj} = \begin{cases} 1 & \sum_{i=1, n} w_{ji} y_{pi} + w_{j0} > 0 \\ 0 & \text{otherwise} \end{cases} \quad [1.8]$$

A single *cycle* of training refers to changing the weights for all n patterns. Rosenblatt's *perceptron convergence theorem* states that if it is possible to correctly classify all members of P , some finite number of cycles is sufficient to find a solution weight state.

³²Rosenblatt, 1958, p. 389

Consider a perceptron with a single output unit. Let \mathbf{R}^I be the underlying space of all inputs which are taken from $\{0, 1\}^I$. Consider the boundary between an input classified as 1 by an R-unit, u_j , and one classified by 0. This is given by equation [1.9]:

$$w_{j1}y_1 + w_{j2}y_2 + \dots + w_{jI}y_I + w_{j0} = 0 \quad [1.9]$$

This is the equation of a hyperplane in input space. All inputs classified as 1 will be on one side of the hyperplane, all those classified as 0 will be on the other side of the hyperplane. Those inputs which are on the hyperplane will be (arbitrarily) assigned a classification of 0. The perceptron will be able to correctly classify any training sets whose classes are linearly separable. It was thought that there would be very few problems that perceptrons would be unable to solve.³³ However, a book by Minsky and Papert³⁴ indicated an important problem that is not linearly separable, and hence could not be solved by a perceptron. This is the XOR problem, indicated in figure 1.11. It is not possible to separate the black and white points with a single line, and hence the XOR problem cannot be realised by a perceptron. Minsky and Papert's book showed that there are a large number of problems which require the solution of XOR, and hence this counter-example could not be ignored.

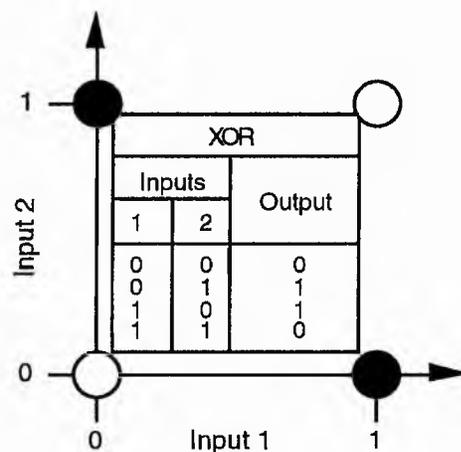


Figure 1.11 — *The XOR problem, and its graphical representation. White circles indicate an output of 0, and black circles indicate an output of 1.*

³³Rosenblatt, 1962, p. 97

³⁴Minsky & Papert, 1969

The XOR problem can, however, be solved by a network with a single hidden layer, but Rosenblatt's algorithm cannot train networks with hidden layers.

1.3.2.2 Back-Propagation

The answer came in the form of the back-propagation algorithm, which is usually accredited to Rumelhart et al,³⁵ even though others had thought of it earlier.³⁶ The back-propagation algorithm uses sigmoid activation functions, which are differentiable. The output of a unit u_k for pattern p , o_{pk} , may then be any real number between 0 and 1 non-inclusive. The difference between the output and the target, t_{pk} , (which may be set to a small distance, δ , from 0 or 1) is then an analogue number between $-(1 - \delta)$ and $+(1 - \delta)$. The sum over all patterns and output units of the square of this difference is termed the error, E , [1.10]. Since the activation function is continuous and differentiable, the partial derivative of the error in terms of the weights may be calculated, which provides the means by which the change to any weight can be found.

$$E = \frac{1}{2} \sum_p \sum_k (t_{pk} - o_{pk})^2 \quad [1.10]$$

Back-propagation performs gradient descent of the error/weight surface, in the hope of finding a weight state with minimum error. Gradient descent travels down the slope of the potential function until it reaches a local minimum. (See figure 1.12.)

For neural networks, the change in weights, $\Delta_p w_{ji}$ is proportional to the partial derivative of the error/weight surface with respect to the weights:

$$\Delta_p w_{ji} \propto -\frac{\partial E_p}{\partial w_{ji}} \quad [1.11]$$

³⁵Rumelhart et al, 1986

³⁶Bryson & Ho, 1969; Werbos, 1974; Parker, 1985

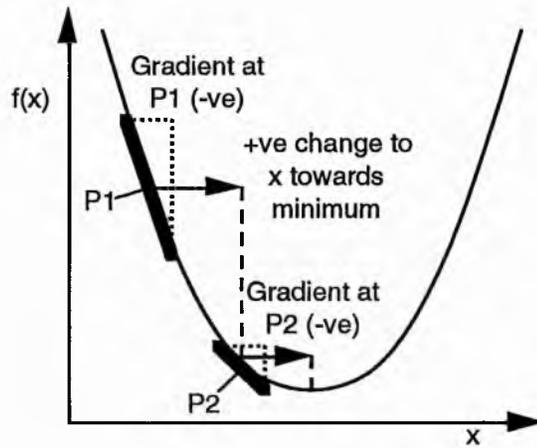


Figure 1.12 — Gradient descent reaches the minimum of a function $f(x)$ by making a series of steps in the opposite direction to the gradient.

Using the chain rule, Rumelhart et al expand the partial derivative of the error with respect to the weight to give the following equation for changing the weight w_{ji} between two units, u_i and u_j for pattern p .³⁷

$$\Delta_p w_{ji} = \eta \delta_{pj} o_{pi} \quad [1.12]$$

where η is a positive constant, o_{pi} is the output of u_i and δ_{pj} is the *error signal* of u_j . The error signal is the partial derivative of E_p with respect to the excitation of u_j , x_{pj} . For output units, u_k , this is given by:

$$\delta_{pk} = f'(x_{pk})(t_{pk} - o_{pk}) \quad [1.13]$$

For hidden units, u_j , there is no target, and the change in weights is based on the error signals of the units in the layer immediately after u_j . The error signal of a hidden unit u_j is thus given by [1.14], where k sums over the units in the higher layer:

$$\delta_{pj} = f'(x_{pj}) \sum_k \delta_{pk} w_{kj} \quad [1.14]$$

Equations [1.13] and [1.14] show the importance of having a differentiable activation function. For the sigmoid activation function in [1.5], the partial derivative of the activation with respect to the excitation is given by:

³⁷Rumelhart et al, 1986, pp. 322-327

$$\frac{\partial(1 + e^{-x_{pj}})^{-1}}{\partial x_{pj}} = o_{pj}(1 - o_{pj}) \quad [1.15]$$

Back-propagation consists of a forward pass, in which the output of the network for the input of pattern p is calculated, followed by a backward pass, in which the error is propagated backwards through the network, giving the change needed for each weight. The series of weight states adopted as the network moves through the error/weights surface may be termed the *weight trajectory*,³⁸ or equivalently the *weight path*. Weights are initialised to be small, non-zero, random values.

Unlike the perceptron, back-propagation does not guarantee convergence if it is possible to correctly classify all of the patterns using the given topology. This is because it is possible for the error/weight surface to contain local minima, from which the weight path cannot escape. For example, in the XOR problem, back-propagation finds a local minimum for roughly 1% of weight paths.³⁹ Another problem comes in the form of steep-sided, shallow ravines. These may take a long time to escape, and hence may be mistaken for local minima.⁴⁰ Ravines are a more common occurrence than local minima, and the failure to train XOR using back-propagation roughly 15% of the time reported by Weir⁴¹ is largely attributable to these ravines. Ochiai et al have more detail on the difficulties of ravines.⁴²

The general method used for trying to escape local minima and speed up training through ravines is to add a momentum term.⁴³ The momentum is represented by a fraction, α , of the last weight change, which is added to the current weight change, giving the following overall formula for the weight change at time t :

³⁸McClelland & Rumelhart, 1988

³⁹Beale & Jackson, 1990, p. 79

⁴⁰Hertz et al, 1991, p. 129

⁴¹Weir, 1991, p. 376

⁴²Ochiai et al, 1994

⁴³Rumelhart et al, 1986, p. 330

$$\Delta w_{ji}(t) = \alpha \Delta w_{ji}(t-1) + \eta \sum_p \delta_{pj}(t) o_{pi}(t) \quad [1.16]$$

However, even with a momentum term, back-propagation is a deterministic algorithm, and finding a local minimum rather than a global minimum is entirely dependent on the initial weight state.⁴⁴ Baba has suggested a non-deterministic method which uses small random perturbations to the current weight state in order to escape local minima.⁴⁵

Problems of separation, including those involving local minima and non-separating global minima are discussed by Brady et al, who discovered that there were certain linearly separable problems for which back-propagation would never find the solution weight state, whereas the perceptron would.⁴⁶ One such problem is shown in figure 1.13. Without pattern S , back-propagation finds a separation that lies along the Input 2 axis. In this situation all the $2n$ patterns have error 0, since the output of the output unit is exactly equal to their non-extreme targets which are a small, constant distance from 1 or 0.

Now consider a new problem, in which the training set in the previous problem is augmented by the pattern S . Although there are separating hyperplanes, such as W , it is no longer possible to exactly realise the same analogue target values for all the training patterns in the same class. Any separating solution has an error proportional to n . There also exist non-separating solutions with constant error, i.e. any hyperplane lying along the Input 2 axis. Therefore, for large enough n , such a non-separating solution will have lower error than any separating solution. Hence starting training with a hyperplane lying along the Input 2 axis and using the gradient descent algorithm of back-propagation will fail to move the hyperplane to a separating position since the latter has the higher error. In general, the point may be made that there are some problems for which the set of weight states with global minimum sum of squared error does

⁴⁴Kolen & Pollack, 1990; Guo & Gelfand, 1991

⁴⁵Baba, 1989

⁴⁶Brady et al, 1988

not intersect with the set of weight states with correct classification of all the patterns.⁴⁷

The solution, suggested by Sontag and Sussmann,⁴⁸ is to make the error function zero for outputs which are closer to 1 or 0 than their target. With this adaptation, all the $2n$ patterns (as well as S) have zero error with separation W . Thus, for separable problems (those which can be realised by the given topology) the error is zero when all the patterns are correctly classified. There is then always an intersection between the set of weight states with global minimum error of Sontag and Sussmann's error function, and the set of weight states that correctly classify all the patterns.

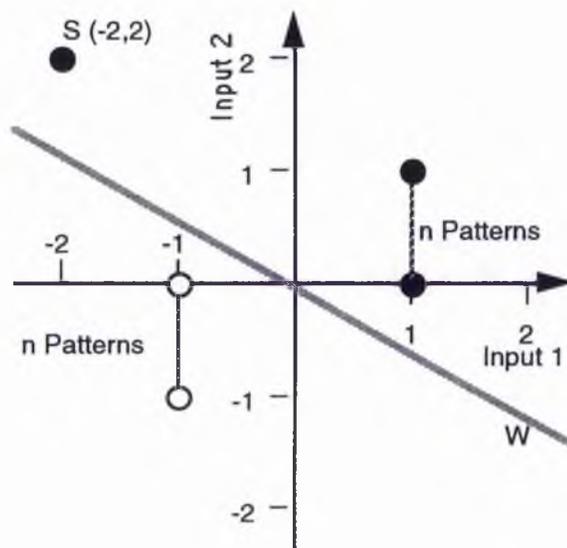


Figure 1.13 — W is a correctly classifying separation of the pattern set, with constant error. The pattern set consists of $2n + 1$ patterns. The dashed lines indicate $n - 2$ patterns which are evenly spaced between their delimiting patterns. Targets close to 1 are indicated by black circles, those with targets close to 0 are indicated by white circles. For sufficiently large n , a solution lying along the Input 2 axis has lower error than W .

Back-propagation using the sum of squared error function is also found to be sometimes suboptimal in the case of inseparable problems. This is

⁴⁷Brady et al, 1988, p. 650

⁴⁸Sontag & Sussmann, 1988

because the sum of squared error approximates the Bayesian *a posteriori* probabilities of each class.⁴⁹ This requires a more complex topology than that needed to realise the class boundaries with global minimum misclassification error, since it is a more complex problem.⁵⁰ The set of weight states that minimise the Bayesian *a posteriori* probability of misclassification for a given topology and problem may not intersect with the set of weight states with global minimum raw misclassification error.

1.3.2.3 Introduction to Genetic Algorithms for Training Neural Networks

GAs are a technique in their own right. They are used in a variety of optimisation problems for which conventional algorithms are too slow. For example, Fang et al use GAs to find solutions to the Job-Shop Scheduling Problem.⁵¹ In this problem, there are a number of jobs to be done which require a number of units of time on each of a number of machines. It is desirable to organise these tasks such that they can be done in the minimum time, making the most efficient use of the machines. Fang et al point out that conventional search algorithms are too slow because the problem is “one of the worst” NP-complete problems.⁵² They show that a GA is capable of producing comparable results to the conventional techniques in a fraction of the time.

Neural networks can be trained using GAs, particularly for the purpose of minimising discontinuous error functions such as misclassification error. This avoids the problem of approximating misclassification error using a continuous error function as in gradient descent.

GAs adopt an evolutionary approach to optimisation. Introductory texts on GAs include books by Holland⁵³ and Goldberg.⁵⁴ The logic of the GA approach is that evolution worked for optimising biological systems, and

⁴⁹Richard & Lippmann, 1991

⁵⁰Telfer & Szu, 1994, p. 809

⁵¹Fang et al, 1993

⁵²Fang et al, 1993, p. 375

⁵³Holland, 1975

⁵⁴Goldberg, 1989

therefore if it can be simulated in computers, it may also work for optimising computational systems.

Given a representation scheme for candidate solutions using binary strings, and a heuristic evaluation function for deciding how good a candidate solution is, a GA works as follows: Firstly, an initial random population of candidate solutions is generated. The heuristic evaluation function value for each member of the population is calculated. These are then used as a basis for generating the next generation of candidate solutions. Those members of the population with the best heuristic evaluation function values are given the best chance of producing offspring in the subsequent generation. Offspring may be generated through direct reproduction, in which each offspring is identical to its parent. There are also genetic operators which may be applied, however, which extend the search beyond the scope of the initial population. The procedure is repeated until a prespecified maximum number of generations is reached, or until an acceptable heuristic evaluation function value is found.

There are two genetic operators, *crossover*, and *mutate*. The latter is the simpler of the two. A random bit in the binary string is selected, and its value changed to the opposite bit value. Figure 1.14 shows an example of the mutate operator.

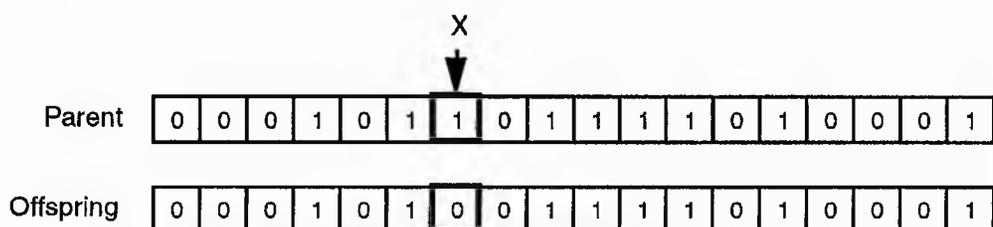


Figure 1.14 — *Mutation. A random bit, X, is selected in the parent binary string, and the offspring is identical to its parent with the exception of bit X.*

The crossover operator requires two parents. A random point in the binary string is selected, and the parents' binary strings are swapped thereafter to give the offspring. Good strings are more likely to be chosen as parents, consequently useful or important bit sequences in the binary string which might be responsible for the high heuristic evaluation

function values of the parents tend to be preserved in the offspring. Figure 1.15 shows the crossover operator.

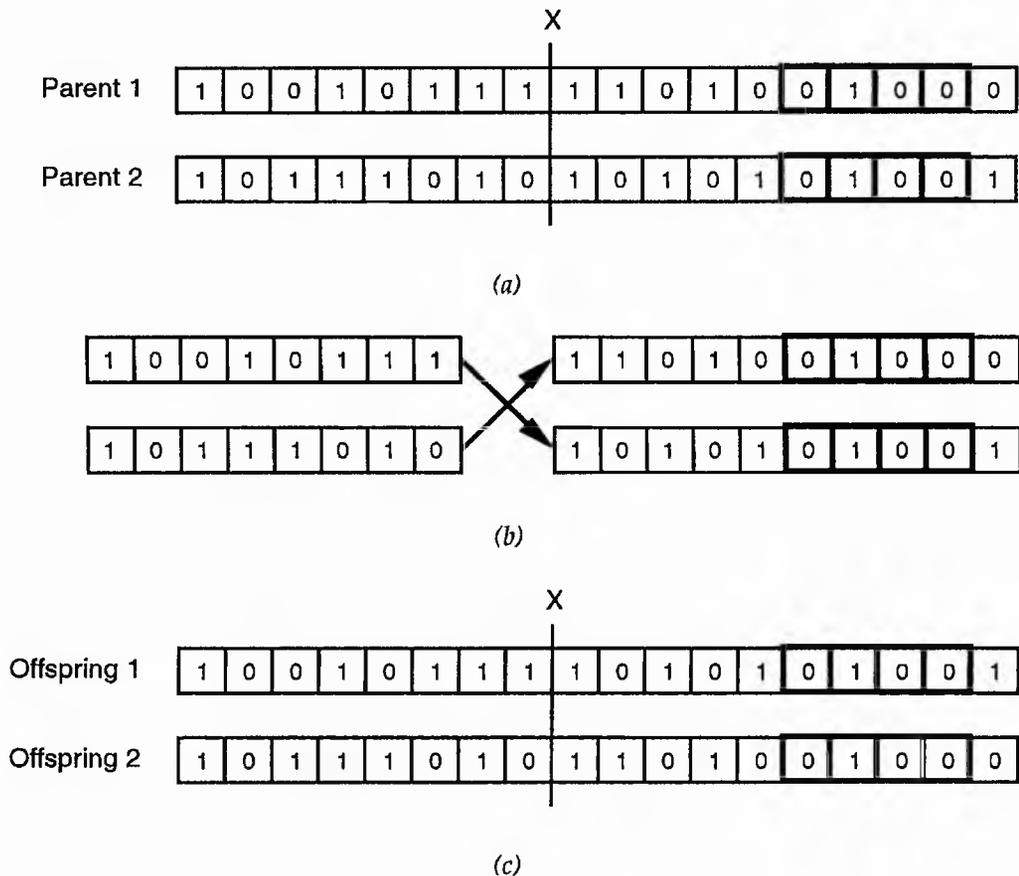


Figure 1.15 — The crossover operator. (a) A random point X in the parent strings is selected. (b) The strings after X in the parents are swapped to give (c), the offspring. Note that potentially worthy sequences such as the bold outlined bit sequence are preserved by this operation.

Usually, there is a probability for each operator being applied during offspring generation. These values must be chosen carefully. If the probabilities are too low, the search will be too slow. If they are too high, good solutions may not be retained from one generation to the next.

Encoding the neural network into a binary string can be quite simple for a fixed topology size. The values of the weights are readily encoded into binary values, and these weight values are then evolved until a satisfactory neural network is found. Since there is no need for a

differentiable activation function, the simpler step-function may be used, even though there may be hidden units in the network.

GAs are not necessarily a panacea for the difficulties with back-propagation and other training algorithms however. There is no guarantee that the optimum is always reached within a reasonable time.

Nevertheless, there are several authors who use GAs, at least in part, to train neural networks.⁵⁵ Kitano, for example, uses GAs to find a near optimum solution which is then used as the initial weight state for back-propagation.⁵⁶

1.3.3 Assessment of Neural Networks

The motivation for studying neural networks is discussed by Hertz et al:

The brain has ... features that would be desirable in artificial systems:

- It is robust and fault tolerant. Nerve cells in the brain die every day without affecting its performance significantly.
- It is flexible. It can easily adjust to a new environment by "learning" — it does not have to be programmed in Pascal, FORTRAN or C.
- It can deal with information that is fuzzy, probabilistic, noisy, or inconsistent.⁵⁷

The extent to which these features can be transferred into artificial neural systems is partly a measure of the success of the discipline.

In describing the appeal of the neural network approach, McClelland et al indicate three properties of neural networks which are to be observed.⁵⁸

⁵⁵E.g. Fogel et al, 1990; Sikora, 1992; Smalz & Conrad, 1994

⁵⁶Kitano, 1990

⁵⁷Hertz et al, 1991, p. 1

⁵⁸McClelland et al, 1986, pp. 29-30

- *Graceful degradation.* Removal of a unit from a neural network does not result in a dramatic loss of performance. Performance decreases gradually as more and more units are removed.
- *Default assignment.* When an input is presented with similar properties to other inputs, the neural network is likely to give the same (possibly erroneous) output.
- *Spontaneous generalisation.* Even in the case of inputs which are radically different from other inputs, neural networks are able to provide an output.

Therefore there is evidence that neural networks are indeed successful in modelling the desirable properties of the brain that are mentioned by Hertz et al. Default assignment may be seen as toleration of noisy information. If an input is presented which is corrupted by noise, and is not too far from the original pattern, the neural network may produce the correct output.

This thesis will be focusing on the spontaneous generalisation and noise toleration benefits of neural networks. Graceful degradation may be seen as being largely a hardware issue, and the neural networks implemented during the work of this thesis are all software-based. (See appendix A.)

These benefits of neural networks are also the sources of some of the difficulties involved with using them. Being able to generalise naturally is a considerable advantage, but is there any way to be sure that the neural networks are generalising correctly? A further issue is that of the topology. How many hidden units should there be, and in how many layers? These, and other issues, are indicated by Hertz et al, in the introduction to their book.⁵⁹

At the heart of all these problems is the issue of what it is neural networks actually do. Having trained the neural network on a set of patterns, it would be useful to be able to discover, at a high level of understanding,

⁵⁹Hertz et al, 1991, pp. 8-9

what it is the neural networks have learned. However, this is not an easy undertaking:

A major weakness of the neural network approach to artificial intelligence is that the knowledge learned by a network is difficult to interpret.⁶⁰

Shavlik also discusses the problem of rule extraction from neural networks, and reviews existing techniques in the literature.⁶¹ However, the real problem is the question of comprehensibility:

An extraction algorithm must produce reasonably comprehensible rules, but without a good measure it is hard to compare alternative approaches.⁶²

To conclude, neural networks have many potentially beneficial properties. However, these properties must be harnessed and understood if neural networks are to survive as rivals to conventional methods. Of particular benefit would be the ability to make guarantees about generalisation, which would enable the trained neural network to be used with confidence in real-world applications.

If there is no way to be able to make these assurances by extracting and then checking the rules the neural network has learned, then perhaps there might be alternative neural approaches to providing certainties about generalisation, which at least merit some research. If such methods proved effective, it would then be possible to side-step the whole issue of comprehensible rule extraction, whilst still giving assurances about neural network behaviour with confidence. This is the approach taken in this thesis.

⁶⁰Fu, 1994, p. 1114

⁶¹Shavlik, 1994, pp. 326-327

⁶²Shavlik, 1994, p. 327

1.4 Can Symbolic AI and Neural Networks Co-operate?

Neural networks have the desirable properties of being able to generalise and tolerate noise. By contrast, these features are weaknesses of the symbolic AI approach, which suffer from brittleness. Conversely, symbolic systems have the desirable properties of having comprehensible rules and of being able to explain their behaviour. This means that within the narrow domains they are applied, there is a degree of certainty about the behaviour of the symbolic systems. This corresponds to a weakness for neural networks which are currently unable to guarantee generalisation and cannot provide explanations for their behaviour.

The strengths and weaknesses of symbolic AI and neural networks seem to be opposites in one another, so there is the scope for making gains for both disciplines through hybrid implementations. Sun and Bookman summarise the three main approaches for co-operation between symbolic AI and neural networks:⁶³

- *Localist.* Symbols are attached to the nodes in the neural network. These are then used, when the network is trained, to describe its behaviour. However, one of the problems of this approach is that standard training algorithms do not take account of the symbol/node correspondence during and after training.⁶⁴
- *Distributed.* This is almost purely a neural network approach. The symbols are represented in a distributed fashion, using several nodes at a time. However, there is only a limited scope for the kinds of symbolic mechanisms (such as matching) that can be performed, due to limited representational capabilities of neural networks, whose presently fixed numbers of inputs cannot have the same depth of structure as symbolic representation schemes with variable-length input expressions.

⁶³Sun & Bookman, 1993, pp. 20-21

⁶⁴Shavlik, 1994, p. 327

- *Combined.* Here, a symbolic and neural system combine, with techniques using varying degrees of cohesion, from a loose coupling between two separate modules to a completely integrated system.

An example of a distributed system is Dyer and Lee's DYNASTY,⁶⁵ in which neural networks are trained using back-propagation on a set of propositions. These are used in the task of understanding goal and plan based stories.

Bahrami and Dagli's HIPS⁶⁶ illustrates a loosely coupled combined system, in which an expert system and a neural network co-operate in the solution of the packing problem.

Romaniuk and Hall have a fully integrated combined system in the form of SC-net.⁶⁷ They adopt a distributed representation scheme using a neural network. The rules are either directly encoded into the network, or trained, using an instance-based learning approach, which modifies the network topology and biases as instances are presented.

This thesis adopts a different approach, however, in the way that it aims to implement an existing symbolic technique in neural networks by abstracting the tenets of the technique that enable guaranteed generalisation. Mitchell's concept and version spaces technique is able to guarantee generalisation once the no-alternative situation is reached. However, the symbolic technique is criticised because the search space can be so large (even for simple concepts) that representing it using boundary representatives at the extremes of the partial ordering becomes infeasible.⁶⁸ It is also restrictive in the assumptions it makes with regard to the quality of the instances, which must be noiseless, and described using relevant attributes, which are assigned values for each instance.⁶⁹

⁶⁵Dyer & Lee, 1995

⁶⁶Bahrami & Dagli, 1994

⁶⁷Romaniuk & Hall, 1993

⁶⁸Booker et al, 1989, p. 268

⁶⁹Aha, 1992, p. 267

Nevertheless, the ability to guarantee generalisation is very useful and would be of great benefit to the neural network community. This thesis represents a body of work carried out on the assumption that the tenets of Mitchell's technique which guarantee generalisation can be transferred into neural networks, and that the technique's drawbacks can be addressed in so doing. Should this prove possible, there would be the ability to make some guarantees about the future performance of the neural network (without resorting to comprehensible rule extraction), and Mitchell's technique would be implemented in an environment which can tolerate noisy data.

1.5 Summary of the Rest of the Thesis

This chapter has introduced symbolic AI and neural networks and shown that there is some correspondence between their advantages and disadvantages. There has also been discussion of the possible benefits of a hybrid approach, with specific relevance to Mitchell's concept and version spaces technique.

Chapters 2 to 4 are review chapters.

Chapter 2 discusses Mitchell's technique implemented in a symbolic environment in some detail. It shows how Mitchell efficiently represents a shrinking number of candidate concepts under the weight of the instances through the use of a partial ordering and boundary representatives. The property of convergence is also shown, as a situation is reached whereby no alternative concept is possible. It is this that enables Mitchell's technique to make guarantees about generalisation.

Chapter 3 examines existing techniques for generalisation in neural networks, discussing their practical applicability, and their generalisation performance. These are contrasted with Mitchell's approach.

Chapter 4 discusses the behaviour of neural networks. One aspect is that of deciding the topology of a neural network. The chapter also analyses the various principles involved in the spontaneous generalisation of neural networks through providing an output for any input. Some novel insights are given into the number of units required in the second hidden

layer of threshold unit networks, and into the possible use of more than two hidden layers in sigmoid unit networks.

Chapters 5 and 6 describe two attempts to implement Mitchell's technique in a neural environment. Chapter 5 adopts a weight space approach, which indicates some important lessons to be learned about the symmetrical nature of the weight space of neural networks. The technique in chapter 5 only works for restricted examples, and is unsatisfactory. Chapter 6 adopts a grid-based approach based on input and output (IO) space, showing how the problems encountered in the technique of chapter 5 may be overcome.

Chapter 7 provides some concluding remarks, and possible directions for future research. Assessments are made about the degree to which the technique in chapter 6 has been successful in combining Mitchell's technique and neural networks to mutual benefit.

2 Mitchell's Symbolic Technique

2.1 Introduction

Mitchell recognised that generalisation is the essence of learning. No human being needs to be exhaustively trained using every possible instance of a concept. For computers to learn concepts, therefore, they must be able to generalise. Mitchell's Concept and Version Spaces technique, a symbolic AI technique, is able to learn a concept through its ability to recognise the *no-alternative* situation.

Fundamentally, the no-alternative situation is the situation whereby there have been enough instances to limit and contain the generalisation to only one possibility. In other words, there is no alternative generalisation that is consistent with all the instances of the concept the learner has been shown.

When the concept is known by the teacher in advance, and instances are carefully chosen, surprisingly few instances are required to generate the no-alternative situation, given certain restrictions, or *biases*, as Mitchell calls them.¹ (See the example in section 2.2.4.1.) This makes Mitchell's technique appealing to those who wish to be sure that the right concept has been learned.

The bias of a given concept learning technique is defined as any restrictions in the ability of the technique to represent or learn concepts or instances. For example, imagine the concept of "yellow or red flowers" is to be learned. If one cannot represent the colour of a given instance, then the concept cannot be learned. Neither can the concept be learned if one cannot represent the disjunction of yellow flowers and red flowers in the concept description.

Yet if one is maximally general in instance and concept representation, the learning process becomes unfeasible, since all objects in the universe must be shown, and the learner told whether they are red or yellow flowers or

¹Mitchell, 1980

not. There is no basis for generalising from one instance to another. Indeed, with maximum generality, one cannot even represent colour, never mind the property of flower-ness.

To categorise a certain range of wavelengths of light as red, for example, would deny the ability to learn concepts that required a more specific distinction of colour, such as "blood red" or "ruby red". Indeed, for maximum generality in representation, one cannot make any categorisations. The colour of the flower could not be referred to as "red" — one could only refer to a specific measure of colour, such as the average wavelength of light reflected by the petals of the individual flower in the instance under consideration.

An infinite number of instances of red wavelengths of light would be needed to cover the range of real number wavelengths that are associated with the colour "red". The difficulty of learning disjunctive concepts with Mitchell's technique, however, means that the various wavelengths of light as associated with "red" could not be grouped together by disjunction such that they could be distinguished from those wavelengths of light associated with "yellow". So, in order to generalise, one would have to categorise in any case.

This categorisation and determination of relevance of properties is a bias, and will inhibit the learning of concepts requiring a different categorisation. However, the representation of instances necessarily implies categorisation of properties, and knowing which attributes may be relevant when describing the instance.

Hence Mitchell argues that biases in a concept learning system, far from hindering the ability to generalise, actually enable the inductive leap from the already learned classification of the training instances to giving the correct classification of a new, untrained instance.

The generalisation language ... is biased in the sense that it does not allow describing every possible set of instances. ... Provided that this biased generalisation language allows describing the correct generalisation, the unambiguous classification of [an unseen training instance] is the correct classification. This ... provides an interesting

insight into the significance of initial biases for allowing inductive leaps during generalisation.²

If an unbiased generalisation system is one that only uses information from the training instances to make decisions, without any further assumptions, then:

...it is not surprising that an unbiased generalisation system cannot make classifications of instances other than the training instances ... [since] classifications of new instances do not logically follow from the classifications of the training instances.³

An unbiased generalisation system, according to Mitchell, is in essence, nothing more than a look-up table.⁴ Some of these issues are addressed in a paper by Denker et al⁵, and will be explored in more depth in Chapter 3.

In general, the bias in Mitchell's technique relates to the restrictions in the concepts that can be learned. It takes two forms. Firstly, in terms of representation, it relates to the way in which a specific instance or generalisation language favours the learning of certain concepts over other concepts, to the extent that some concepts are unlearnable with the given languages. Secondly, the learning mechanism itself may be biased. This can be due to the fact that the technique favours the learning of certain kinds of concept, and does not favour others (such as disjunctive concepts in this case), but is also due to underlying assumptions made about the nature of concepts. These points are summarised by Utgoff, who indicates the following sources of bias in a learning technique:

- 1 The language in which the [concepts] are described.
- 2 The space of [concepts] that the program can consider.
- 3 The procedures that define in what order [concepts] are to be considered.

²Mitchell, 1982, p. 217

³Mitchell, 1980, p. 4

⁴Mitchell, 1980, p. 3

⁵Denker et al, 1987, § 10-15, 17,18

- 4 The acceptance criteria that define whether a search procedure may stop with a given [concept] or should continue searching for a better choice.⁶

The generalisation problem for Mitchell is summarised as follows:⁷

Given:

- A language in which to describe instances,
- A language in which to describe generalisations [or concepts],
- A matching predicate that matches generalisations to instances,
- A set of positive and negative training instances of a target generalisation [or concept] to be learned.

Determine:

- Generalisations within the provided language that are consistent with the presented training instances.

When there is only one generalisation within the provided language that is consistent with the presented training instances, then the no-alternative situation has been achieved. All future instances presented will be correctly classified, within the constraints of the instance and generalisation languages and matching predicate.

Finding the no-alternative situation is a problem which Mitchell approaches using what he calls a "bidirectional search"⁸ strategy. This search strategy is explained and explored in the rest of this chapter. Mitchell's symbolic technique is used as a reference point during the development of the neural techniques (Chapters 5 and 6), and the bidirectional search strategy is the foundation of each.

⁶Utgoff, 1986, p. 107

⁷Mitchell, 1982, p. 204

⁸Mitchell, 1982, p. 213

2.2 Searching For No Alternative

2.2.1 The Search Space

Mitchell searches for the no-alternative situation in concept space. This space is set up using an instance language, a concept description (or generalisation) language, and a matching predicate that links the two. The matching predicate is used to determine whether or not an instance is consistent with a concept description.

Firstly, consider the space of all objects. A subset of this space is describable using a given instance language. Here, an object is represented by a p dimensional vector, where there are p object attributes being taken into consideration. Each object has a specific attribute value for each element of the vector. Each attribute value has its own unique symbol. The attributes are arbitrarily ordered in the vector.

An instance may be regarded as having an unordered set of objects as stimulus, with fixed cardinality, q , and a corresponding response. Training instances have the target response given. The task of the concept learning system during training is to reproduce the given target response of the stimulus for all training instances. Other instances, not used for training, do not have a target response. Once training is completed, the task of the concept learning system is to correctly predict the response of these unlearned instances, using the concept learned during training.

The target response depends upon the concept being learned. In Mitchell's technique, the target response is a binary decision, which is whether the stimulus is a positive or negative instance of the concept. For Mitchell, a concept is a matching filter, which enables the discrimination of the class of matching sets of objects in accordance with their attribute values. Other possibilities for the target response will be discussed in later chapters.

For example, if we were considering the observation of wild flowers, some relevant attributes might be the Latin name, and the month and place

seen.⁹ Irrelevant attributes could be such things as vehicle type, or year of publication. Thus, we might have the following instance (where $q = 1$):

{[Chrysanthemum vulgare, August, Derbyshire]}⁺

The + indicates that the instance is a positive instance for the current concept being learned, whatever that may be.

If we were considering poker hands, relevant attributes would be the suit and the number of the card. Here, an instance is a set of five objects, and hence we might have the following negative instance of a royal flush in Hearts:

{[2, Hearts] [7, Spades] [Ace, Diamonds] [3, Clubs] [10, Clubs]}⁻

A separate language is used to allow the representation of a concept. A concept, for Mitchell, is represented by an unordered set of q p -dimensional *general object description* vectors. The general object description vectors must have the same attributes for each dimension as in the instance language for the objects. Otherwise, the generalisation language has a bias that could prevent the desired concept from being learned.

Relative to a given language of instances, an unbiased generalisation language is one which allows describing every possible subset of these instances.¹⁰

Nevertheless, since Mitchell's technique is not strictly able to cope with disjunctive concepts (see later), all generalisation languages will have a degree of bias. Thus it will not be possible to represent concepts which allow the description of *every possible* subset of instances describable in the instance language. For example, it is not possible to represent a concept that allows the description of the following two instances only:

{[2, Hearts] [7, Spades] [Ace, Diamonds] [3, Clubs] [10, Clubs]}⁺
 {[3, Hearts] [7, Spades] [Ace, Diamonds] [3, Clubs] [10, Clubs]}⁺

⁹Taken from McClintock & Fitter, 1961

¹⁰Mitchell, 1980, p. 2

As mentioned in the introduction, bias in the generalisation language actually enables the concept to be learned. However, whilst all generalisation languages are biased, the extent of the bias should be kept under control. For example, in the poker scenario, if the generalisation language only has the attribute of suit, it would be possible to learn the concept of a flush in Hearts, but not a royal flush in Hearts, as would be possible if the number of the card was added to the language. However, note that the inability to learn disjunctive concepts means that one cannot learn the concept of a flush in any suit — royal or otherwise. As Mitchell puts it, it is a question of the *appropriateness* of the bias:

With the choice of a generalisation language the system designer builds in his biases concerning useful and irrelevant generalisations in the domain. This bias constitutes both a strength and a weakness for the system: If the bias is inappropriate, it can prevent the system from ever inferring correct generalisations; if the bias is appropriate, it can provide the basis for important inductive leaps beyond information directly available from the training instances.¹¹

Each element of each vector contains either an attribute value symbol, or a special symbol indicating a wild card. The wild card is a symbol that will match any attribute value in that dimension. An asterisk (*) will be used to represent the wild card in subsequent examples.

For example, the concept of all flowers observed in Derbyshire, whatever the time of year, or Latin name, would be represented as follows:

{[* , * , Derbyshire]}

Or, we might define the concept of a flush in Hearts, in which case the concept would be represented thus, where the asterisk is a wild card allowing any card number to be matched:

{[* , Hearts], [* , Hearts], [* , Hearts], [* , Hearts], [* , Hearts]}

The choice of attribute values is also important when creating the generalisation and instance languages. The case of colour is a useful

¹¹Mitchell, 1982, p. 223

example. What one person might call red, another might call scarlet, or blood red. In Mitchell's technique it is very important that one is precisely consistent about the attribute values chosen. It is fine to call something red if the concept is about distinguishing red from blue. This is not the case, however, if the concept is about distinguishing blood red from brick red.

The concept representation is not complete without the matching predicate. The matching predicate determines the output response, given the concept description and an instance. If an instance, which is a set of q object vectors (and a target response if it is a training instance) matches with a concept description, which is a set of q general object description vectors, then the output response is positive. Otherwise it is negative.

In general, the matching predicate is given by the user, since it depends on the instance and generalisation languages under consideration. An example of a matching predicate is given below, for use with the instance and generalisation languages given in the examples above.

Matching occurs at two main levels: the level of the general object description vector and the object vector; and the level of the instance and the concept. An object matches with a general object description vector if each element of the object vector matches with corresponding element in the general object description vector. Two elements match either if they are the same symbol, or if the element from the general object description vector is a wild card. The instance matches the concept if each object vector in the instance matches with a unique general object description vector in the concept.

For example,

{[Chrysanthemum vulgare, July, Cumbria]}

matches with

{[* , * , Cumbria]}

but

{[Iris spuria, June, Dorset]}

does not.

The matching predicate is used to define consistency. A concept is consistent with a positive instance if it matches the instance. Conversely, a concept is consistent with a negative instance if it does not match the instance. Hence, a concept is consistent with a set of positive and negative instances if and only if it matches all the positive instances, and does not match any of the negative instances.

Given enough instances for a given instance and concept language, there will be only one concept description that is consistent with all the instances shown. This is the no-alternative situation, and it is this that the search is being conducted to find. Therefore the search must be conducted in the space of all concept descriptions for the current language.

2.2.2 Bidirectional Search in Mitchell's Technique

Even when a search is conducted in the space of only those concepts describable by the current generalisation language, the search space is large. If \mathbf{a} is a vector of dimensionality p , where p is the number of attributes each object has, and each element, a_j , of \mathbf{a} contains the number of permissible attribute values (not including the wild card) for that attribute, then equation [2.1] shows the number, n , of possible different general object description vectors there are for the generalisation languages used here.

$$n = \prod_{j=1,p} (a_j + 1) \quad [2.1]$$

For example, in poker, there are two attributes: suit and card number. There are four suits, and thirteen cards in each suit. The vector \mathbf{a} is [4 13]. The number of possible general object description vectors includes the possibility for a wild card being placed in either attribute. This is equivalent to saying that there is an extra suit, and an extra card number. Hence the number of possible general object description vectors, n is 70.

Equation [2.2] shows the size of concept space for the generalisation languages used here, W , where there are q objects in an instance.

$$\#W = {}_{n+r-1}C_r \quad [2.2]$$

where ${}_n C_r$ is the notation for the number of combinations of n things taken r at a time, without repetitions.¹² Since a given general object description vector may be repeated in a single concept, repetitions are allowed. The number of combinations of n things taken r at a time with repetitions is the same as the number of combinations of $n + r - 1$ things taken r at a time, without repetitions.¹³ Hence $(n + r - 1)C_r$ is the number of combinations of n things taken r at a time, with repetitions.

In the poker concept space, there are five objects in an instance, since there are five cards in a hand of poker. The number of possible concepts is therefore ${}_{70+5-1}C_5$, or 16 108 764. Note that this allows repetitions of cards, and hence this is not the number of possible poker hands from a single deck of cards. This figure is ${}_{52}C_5$, or 2 598 960, for comparison.

Bidirectional search, or search that proceeds from two initial points in the search space, is part of the key to enabling the recognition of the no-alternative situation. Bidirectional search alone cannot achieve this. To have that knowledge requires more than just searching from two initial states. It is also necessary to have only a single point where the two directions of the bidirectional search meet. Bidirectional search *per se* allows for the possibility of more than one meeting point of the two directions. In conjunction with search heuristics, however, to be discussed in the next section, and a partial ordering of the search space, there is the possibility for a single meeting point of the bidirectional search, and hence the assurance of the no-alternative situation.

Mitchell partially orders concept space using the more-specific-than relation. A concept, c_1 is more-specific-than another concept, c_2 if and only if all the instances matched by c_1 form a proper subset of all the instances matched by c_2 .

For example,

¹²James & James, 1992, p. 66

¹³James & James, 1992, p. 67

$$c_1 = \{[Viola\ odorata,\ *,\ New\ Forest]\}$$

is more-specific-than

$$c_2 = \{[*,\ *,\ New\ Forest]\}$$

but not more-specific-than

$$c_3 = \{[*,\ April,\ *]\}.$$

This is because the set of instances matched by c_1 is not a proper subset of the instances matched by c_3 . For example, the instance:

$$\{[Viola\ odorata,\ March,\ New\ Forest]\}^+$$

is matched by c_1 (and c_2), but not by c_3 .

The partial ordering is used to guide the bidirectional search. Two sets of concept descriptions are used, S and G , the members of which form the maximal and minimal elements of all finite chains in the partial ordering.

S is the set of the most specific concept descriptions that are consistent with all of the instances shown at any stage during training. That is to say, there is no concept description more specific than any member of S that is consistent with all the instances shown so far.

G is the set of the most general concept descriptions that are consistent with all of the instances shown at any stage during training. So all concept descriptions that are consistent with the instances and are not members of G are more specific than members of G , and no member of G is more specific than any other member of G .

The most specific concept descriptions are those which are only as general as is necessary such that they match all the positive instances. Conversely, the most general concept descriptions are those which are only as specific as is necessary such that they do not match all the negative instances.

S and G , therefore mark the most extreme points (in terms of the partial ordering), or sets of points, that are consistent with the instances shown at a given time during training. The set of all concepts that lies between and including S and G in the partial ordering is called version space. Since

version space contains all concepts that are consistent with the instances shown so far, the search can be restricted to version space. Mitchell summarises the above as follows:

The advantage of the version space strategy lies in the fact that the set G summarises the information implicit in the negative instances that bounds the acceptable level of generality of hypotheses, while the set S summarises the information from the positive instances that limits the acceptable level of specialisation of hypothesis. Therefore, testing whether a given generalisation is consistent with all the observed instances is *logically equivalent* to testing whether it lies between the sets S and G in the partial ordering of generalisations.¹⁴

Version space is the essence of Mitchell's technique. At any time, during training, version space contains only those concepts that are consistent with the instances shown. This means that all the positive instances shown are instances of all concepts in version space, and all the negative instances shown are not instances of all concepts in version space. Therefore, all the possibilities for the final no-alternative concept are contained in version space at any given time, since all the possibilities for the final concept must match all the positive instances so far, and not match any of the negative instances so far.

To see how version space, V , cuts down the search for the instance and generalisation languages used here, it is worth showing the size of version space at the initial stage, after the presentation of the first positive instance. (Version space is undefined before this.) This is given in equation [2.3], where there are p elements in the general object description vector, and q general object description vectors in a concept:

$$\#V_0 = 2^{pq} \quad [2.3]$$

Intuitively, this can be understood by realising that all the possible concepts in version space have either a * or not a * in each individual element of the set of general object description vectors. This is because, after the presentation of the first positive instance, the possible attribute

¹⁴Mitchell, 1982, p. 215

values for each attribute are known. In order to match the positive instance, every concept in version space must have either a wild card, or the same attribute value as in the instance, in each element of each general object description vector.

In the poker world, where the number of possible concepts is roughly sixteen million, the size of version space after the first positive instance is presented is 2^{10} , or 1 024, which represents a scale down factor of just under sixteen thousand. This is certainly a remarkably large reduction in the volume of search space. In the proverbial hunt for needles, it is roughly equivalent to cutting down the search from a haystack to a mere armful of hay.

S and G, and the partial ordering are the means by which version space is represented in Mitchell's technique:

In general, the number of plausible versions can be very large (possibly infinite) when the language of patterns for rules [concepts] is complex. The key to an efficient representation of version spaces lies in observing that a general-to-specific ordering is defined on the rule pattern space by the pattern matching procedure used for applying rules. The version space may be represented in terms of its maximal and minimal elements according to this ordering.¹⁵

2.2.3 Candidate Elimination

In the initial state, before any instances have been presented, G is the concept that matches all instances, and S is the concept that matches no instances. Version space is initially all concept descriptions possible with the current generalisation language.

Candidate elimination is the process of eliminating candidate concepts from version space as instances are presented. This is in order to maintain consistency with the presented instances. Since version space is represented by S and G, the elimination of candidate concepts is represented by making changes to S and G.

¹⁵Mitchell, 1977, p. 306

There are two kinds of change to S and G that can be made: *updating* and *selection within*. The change made depends on whether the instance is positive or negative. The need for these two kinds of change can be observed through the nature of S and G, and the corresponding effect of a positive or a negative instance.

To be maximally specific, each member of S must be only as general as is necessary to match all the positive instances. When a positive instance is presented, any member of S that does not match the instance (i.e. classifies the instance as negative) must be made more general, by the *minimum amount* necessary such that the member of S does match the new instance. This is called *updating S*.

When a negative instance is presented, any member of S that matches the instance (i.e. classifies the instance as positive) is removed from S, since that member of S cannot be made more general to prevent the positive classification. This is called *selecting within S*.

To be maximally general, each member of G must be only as specific as is necessary to avoid matching any negative instance. When a negative instance is presented, any member of G that matches the instances must be made more specific, by the *minimum amount* necessary such that the member of G does not match the new instance. This is called *updating G*.

When a positive instance is presented, any member of G that does not match the instance (i.e. classifies the instance as negative) is removed from G. This is because that member of G cannot be made more specific such that the positive instance is correctly classified. This is called *selecting within G*.

The two processes of selection within and updating maintain the consistency of each member of the sets S and G with the instances presented so far. Furthermore, there is the assurance that S can only be made more general, and G can only be made more specific. Any change to S or G means the two sets are brought closer to convergence. The action appropriate to the kind of instance presented to S or G is summarised in table 2.1.

Updating and selection within maintain S and G on the boundary of version space. These processes are the representation of the shrinking of version space as instances are added, just as S and G are the representation of version space itself. Ultimately, given sufficient instances, version space shrinks to a singleton. S and G represent the same concept at this point, which is the no-alternative situation.

Instance Class	Classified as	By a member of	Action
Positive	Positive	S	None
Positive	Negative	S	Update S
Negative	Positive	S	Select within S
Negative	Negative	S	None
Positive	Positive	G	None
Positive	Negative	G	Select within G
Negative	Positive	G	Update G
Negative	Negative	G	None

Table 2.1 — Summary of the actions appropriate on presentation of an instance to S or G.

Section 2.2.5 contains a proof that S and G fully represent the boundaries of version space, and that updating and selection fully implement the change to version space when a new instance is added. It will also be shown that these changes can be made without the need to refer to previous instances, or to the rest of version space. If it were necessary to refer to the rest of version space, then S and G would not represent version space well enough, and there would be no point in using them. The ability to avoid referring to previous patterns is also an advantage in terms of processing time.

The process is called *candidate elimination* by Mitchell, because it eliminates candidate concepts from version space as they become inconsistent with the instances that are presented.

By modifying the version space in the above way, all rules (and only those rules) which conflict with the new training instances are eliminated from consideration.¹⁶

¹⁶Mitchell, 1977, p. 309

These eliminated candidate concepts from version space need not necessarily be members of S or G , because S and G are the boundary representatives of version space, and an instance may deny concepts that lie between S and G . The elimination of these concepts is implicit in the elimination of concepts from S and G during selection within and updating.

Any concept that lies on a chain in the partial ordering between a member of S , $S(x)$, and a member of G , $G(y)$, is removed from version space if either $S(x)$ or $G(y)$ is removed from S or G during selection within, unless the concept lies on a different partial order chain between other members of S and G . During updating, if $S(x)$ is updated to the set S_x , then all concepts are eliminated from version space that lie on a chain in the partial ordering between $S(x)$ and $G(y)$, but do not lie on a chain in the partial ordering between a member of S_x and $G(y)$, unless the concepts lie on a different partial order chain between other members of S and G . Therefore the search does not exhaustively examine all members of version space in order to determine the no-alternative situation.

This might appear controversial in the light of Rich and Knight, who claim that "the algorithm involves exhaustive, breadth-first search through the version space."¹⁷ However, the issue depends on what is meant by "exhaustive". Certainly, Mitchell exhaustively considers all alternatives that might lead to a solution at a given stage — as represented by the many members of S and G . It is not necessarily true, however, that Mitchell exhaustively considers all points in version space when working towards a solution. This can be seen in the version space example (2) in the next section.

An "exhaustive, breadth-first search through the version space" could be taken to imply that every possible concept in version space is considered, until the concept that is consistent with all the instances is found. To sum up the search in this way denies the methods Mitchell uses to prune the search, without compromising the possibility of reaching the no-alternative situation. Furthermore, no mention is made of the

¹⁷Rich & Knight, 1991, p. 468

bidirectionality of the search, which is the key to the termination condition: the convergence of S and G to the same single concept.

2.2.4 The Version Space Algorithm and Examples

Mitchell's version space learning algorithm is given below¹⁸:

- 1 Get the first positive instance, t_0 .
- 2 Initialise S so that t_0 is the only instance of S.
- 3 Initialise G so that all instances are instances of G.
- 4 For all subsequent instances t_n :
 - 4.1 If t_n is a positive instance:
 - 4.1.1 If any member of G classifies t_n as negative, remove that member from G. (Selection Within)
 - 4.1.2 If any member of S classifies t_n as negative, make that member of S more general in all possible ways by the minimum amount necessary to make the instance be classified as positive by S. (Updating)
 - 4.2 If t_n is a negative instance:
 - 4.2.1 If any member of S classifies t_n as positive, remove that member from S. (Selection Within)
 - 4.2.2 If any member of G classifies t_n as positive, make that member of G more specific in all possible ways by the minimum amount necessary to make the instance be classified as negative by G. (Updating)
 - 4.3 If $S = G$, display: Concept (S) found; stop.
- 5 Display: Version space lies between (S) and (G).
- 6 Stop.

For the instance and generalisation languages used in the examples in this chapter, the algorithms for updating S and G are given at the end of section 2.2.5.

¹⁸Mitchell, 1982, p. 213

The following two examples illustrate the algorithm in practice. The first is simple, the second is more complex and also shows how version space decreases in size with each instance. Thus it may be seen from the second example that members of version space are removed which are not members of S or G. Since they therefore will not be considered by S or G at any point in the future, Mitchell's technique is not an exhaustive search of version space in the sense that not every single concept in version space is investigated when developing a solution.

2.2.4.1 A Simple Example

Given the first positive instance (a honeysuckle):

$$\{[\text{Lonicera periclymenum, June, Essex}]\}^+$$

S is chosen such that it matches this instance only, and G such that it matches all instances:

$$S = \{[\text{Lonicera periclymenum, June, Essex}]\}$$

$$G = \{[* , * , *]\}$$

The second instance is a negative instance (a bluebell):

$$\{[\text{Hyacinthoides non-scripta, May, Yorkshire}]\}^-$$

This is not an instance of S, therefore no member of S is removed during selection within. G must be updated in every way that moves towards a member of S, being made specific by the minimum amount necessary such that G does not match the negative instance.

$$G = \{[\text{Lonicera periclymenum, * , *}] \{[* , June, *]\}$$

$$\{[* , * , Essex]\}$$

The third instance is another positive instance:

$$\{[\text{Lonicera periclymenum, July, Hertfordshire}]\}^+$$

G is selected within. All those concepts in G that do not match the positive instance are removed. $\{[* , June, *]\}$ does not match the positive instance, neither does $\{[* , * , Essex]\}$. Thus, G now contains only one concept:

$$G = \{ \{ [Lonicera periclymenum, *, *] \} \}$$

S is now updated by the minimum amount necessary such that it matches the new instance. This means generalising the month and place dimensions. S is now represented as follows:

$$S = \{ \{ [Lonicera periclymenum, *, *] \} \}$$

S and G are now equal singletons. The no-alternative situation has been achieved, and there can be no other concept consistent with the three instances presented. As stated in the introduction, this example shows that only a few instances are needed. This could be regarded as surprising in the light of the size of concept space. With roughly 1300 flowers in *The Pocket Guide To Wild Flowers*,¹⁹ 100 counties in the UK, and 12 months in the year, there are just over 1.7 million possible concepts, from equation [2.2]. It should be pointed out, however, that after the presentation of the first instance, version space contains only 8 candidate concepts, from [2.3].

2.2.4.2 A More Complex Example

In this example, we consider an instance language and a generalisation language designed to learn a concept which pertains to two objects, rather than one. This more complex example enables the possibility of showing a case whereby S has more than one member.

This situation cannot arise for instance languages used to describe only one object at a time. It is best to illustrate this with an example. Take the concept found from example 1:

$$c = \{ [Lonicera periclymenum, *, *] \}$$

The two wild cards are there because two positive instances occurred, with different attribute values for those attributes. For S to have more than one member at any given stage, there must be more than one specific attribute value for a given attribute that is consistent with all the positive instances seen so far. This cannot happen with one object only, since where attributes differ in positive instances, one must put in a wild card.

¹⁹McClintock & Fitter, 1961

There is no other possibility that ensures the consistency of S with all positive instances.

Thus, for a two-object concept, consider the concept of all flowers observed in the Chilterns at any time of the year, and all daisies (*Bellis perennis*) observed anywhere, at any time of the year. It just so happens, however, that all observations have taken place in April and May, and in the Chilterns and in Devon. Also, the only data available is for the daisy and the cowslip (*Primula veris*).

Given the first positive instance, of a cowslip observed in the Chilterns in April and a daisy observed in Devon in May, all members of version space can be constructed, as in figure 2.1. Membership of S or G is shown with a bold outline.

At this stage, S is simply the set containing the single concept that matches the first instance only:

$$S = \{ \{ [\textit{Primula veris}, \textit{April}, \textit{Chilterns}] \\ [\textit{Bellis perennis}, \textit{May}, \textit{Devon}] \} \}$$

G is the most general concept, i.e.:

$$G = \{ \{ [* , * , *] [* , * , *] \} \}$$

The next instance to be presented is another positive instance:

$$\{ \{ [\textit{Bellis perennis}, \textit{April}, \textit{Devon}] \\ [\textit{Primula veris}, \textit{May}, \textit{Chilterns}] \} \}^+$$

This causes the removal of many concepts from version space, as shown by the shaded concepts in figure 2.2, which includes the membership of S. There are two possibilities for the minimal updating of S. The first is whereby the concept is all flowers seen anywhere in April and May, the second is whereby the concept is cowslips observed any time in the Chilterns, and daisies sighted at any time in Devon. S is now the following set of concepts:

$$S = \{ \{ [\textit{Primula veris}, *, \textit{Chilterns}] [\textit{Bellis perennis}, *, \textit{Devon}] \\ \{ [* , \textit{April}, *] [* , \textit{May}, *] \} \} \}$$

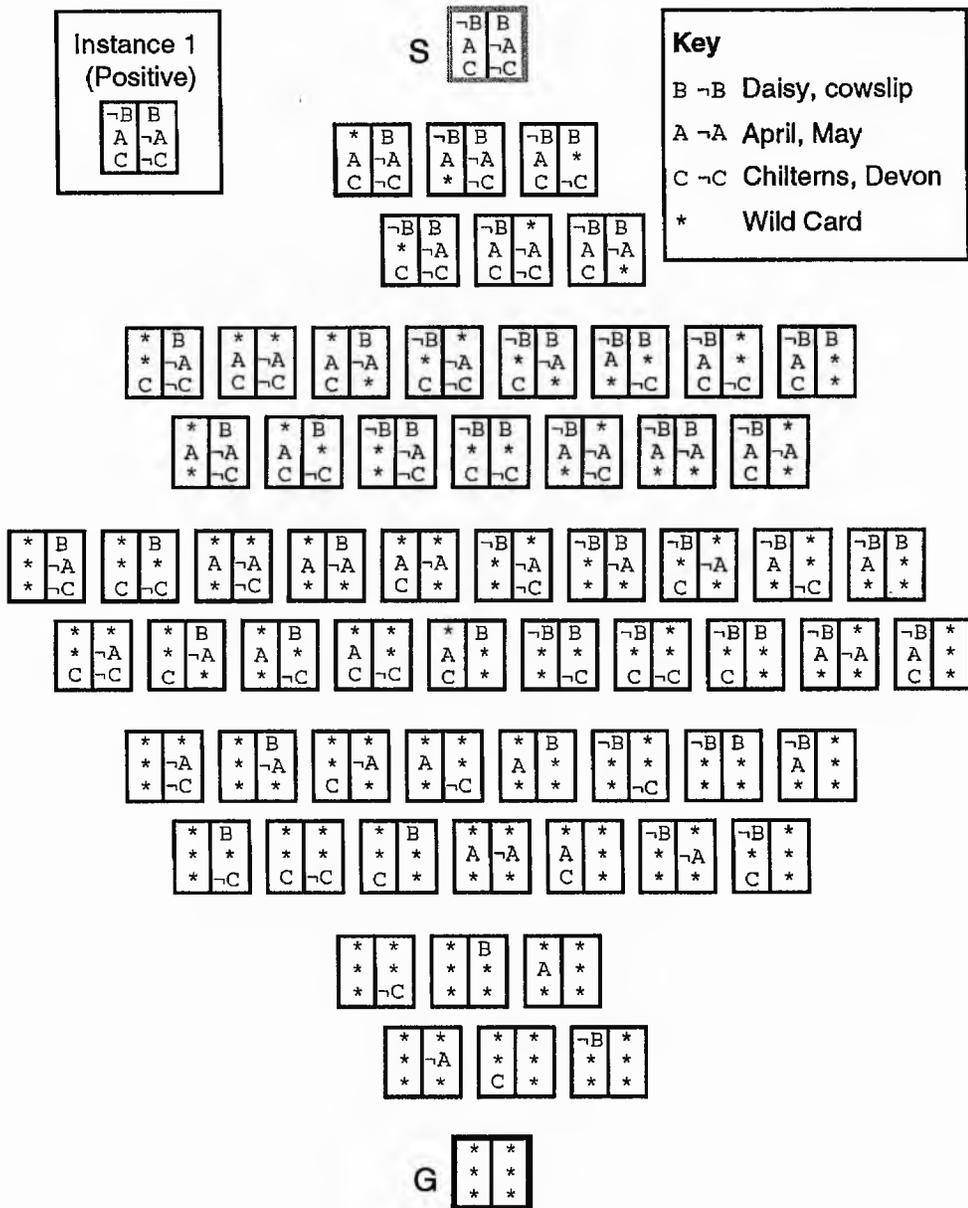


Figure 2.1 — Version space after the first positive instance. Membership of S or G is shown with a thick outline — S in grey, G in black. Since the final concept is known already in this example (see text), the various elements can be represented by the attribute value, or not-the-attribute value. Note, however, that the same non-desired attribute value is used throughout — i.e. cowslip instead of daisy, May instead of April, and Devon instead of the Chilterns. This keeps the number of possible concepts to a maximum at any stage during learning, since other non-desired attributes will lead to the placing of a wild-card for that attribute in the concept.

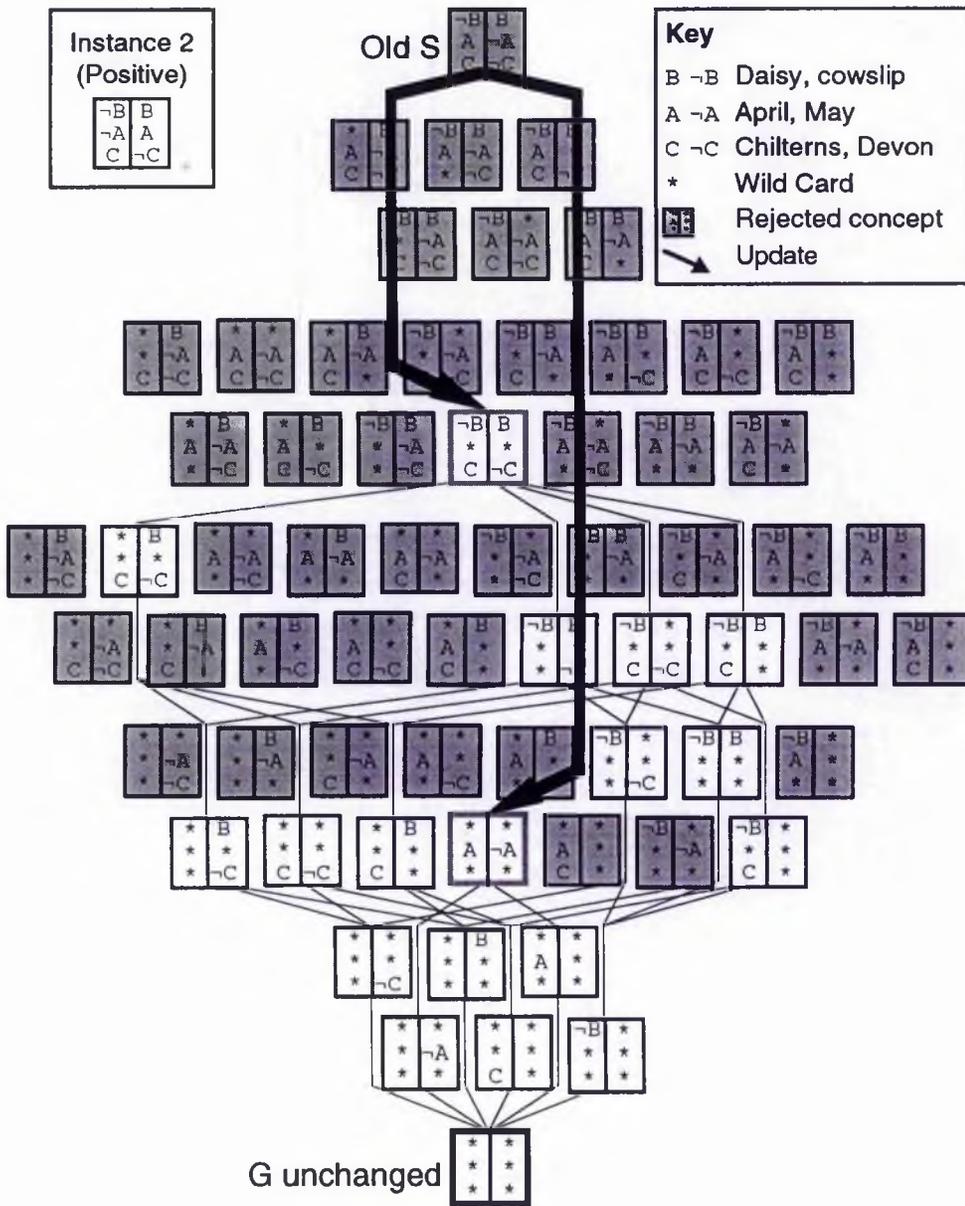
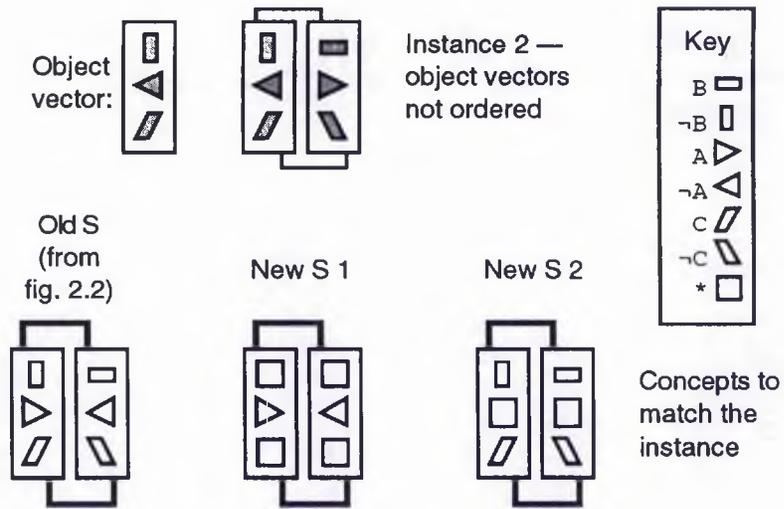
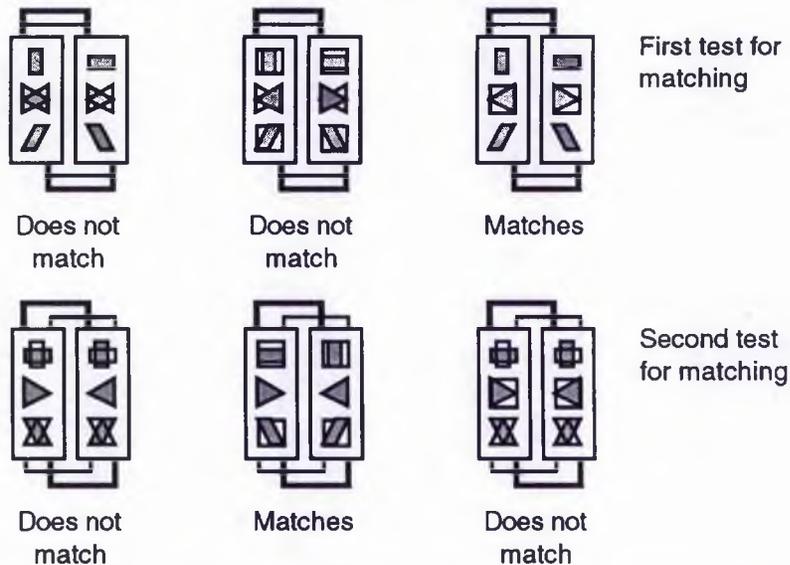


Figure 2.2 — Version space after the second instance is presented (unshaded concepts only). The shaded concepts indicate eliminated concepts. The arrows show how S is updated, and the bold outlines show membership of S or G. The matching predicate is illustrated in figure 2.3. The thin lines between the concepts indicate the chains in the partial ordering of version space.



Matching tested for all arrangements of the object vectors.
Two possibilities in this case.



If the concept matches the instance for one or more of the arrangements of the object vectors, then the concept matches the instance. Thus New S 1 and New S 2 match instance 2, but Old S does not match instance 2.

Figure 2.3 — The matching predicate used in these examples explained. The attribute values are represented by shapes, which are solid in the case of an instance, and are holes in the case of a concept. To match the instance, the holes in the concept must fit the shapes in the instance, rather like the child's toy. The wild card fits any shape.

In figure 2.2, it is possible to observe the double membership of S. Since there is no order to the objects in the instance, there are two ways in which one can be maximally specific to maintain consistency with the positive instances. Note that the fact that one member of S has fewer wild cards than another does not mean that it is more specific. Specificity is defined over the subset of all instances (taken from instance space, not just those seen at a given stage during learning) matched by a given concept (this will be defined formally in equation [2.6]), not by the number of wild cards in the general object description vector. Hence, the most specific concept so far could either be a cowslip in the Chilterns and a daisy in Devon, or any two flowers, one of which is observed in April, and another in May.

There is no change to G, since all positive instances are matched by the most general possible concept.

For G to change requires a negative instance, and it just so happens that the next instance is indeed negative:

$$\begin{aligned} & \{[Primula\ veris,\ April,\ Devon] \\ & [Primula\ veris,\ April,\ Chilterns]\}^- \end{aligned}$$

The effect on the remains of version space is shown in figure 2.4. There are two possibilities for minimally making G more specific in order to avoid matching the new instance, one moving towards each member of S:

$$G = \{ \{[*,\ *,\ *]\ [*,\ May,\ *]\} \{[*,\ *,\ *]\ [Bellis\ perennis,\ *,\ *]\} \}$$

S is unchanged by the third instance. The fourth instance, also negative, is given below:

$$\begin{aligned} & \{[Bellis\ perennis,\ April,\ Devon] \\ & [Primula\ veris,\ May,\ Devon]\}^- \end{aligned}$$

The effect on version space is given in figure 2.5. S is selected within first, which removes the $\{[*,\ April,\ *]\ [*,\ May,\ *]\}$ concept from S. This means the more general version of this concept in G: $\{[*,\ *,\ *]\ [*,\ May,\ *]\}$ cannot be updated, therefore it too is discarded.

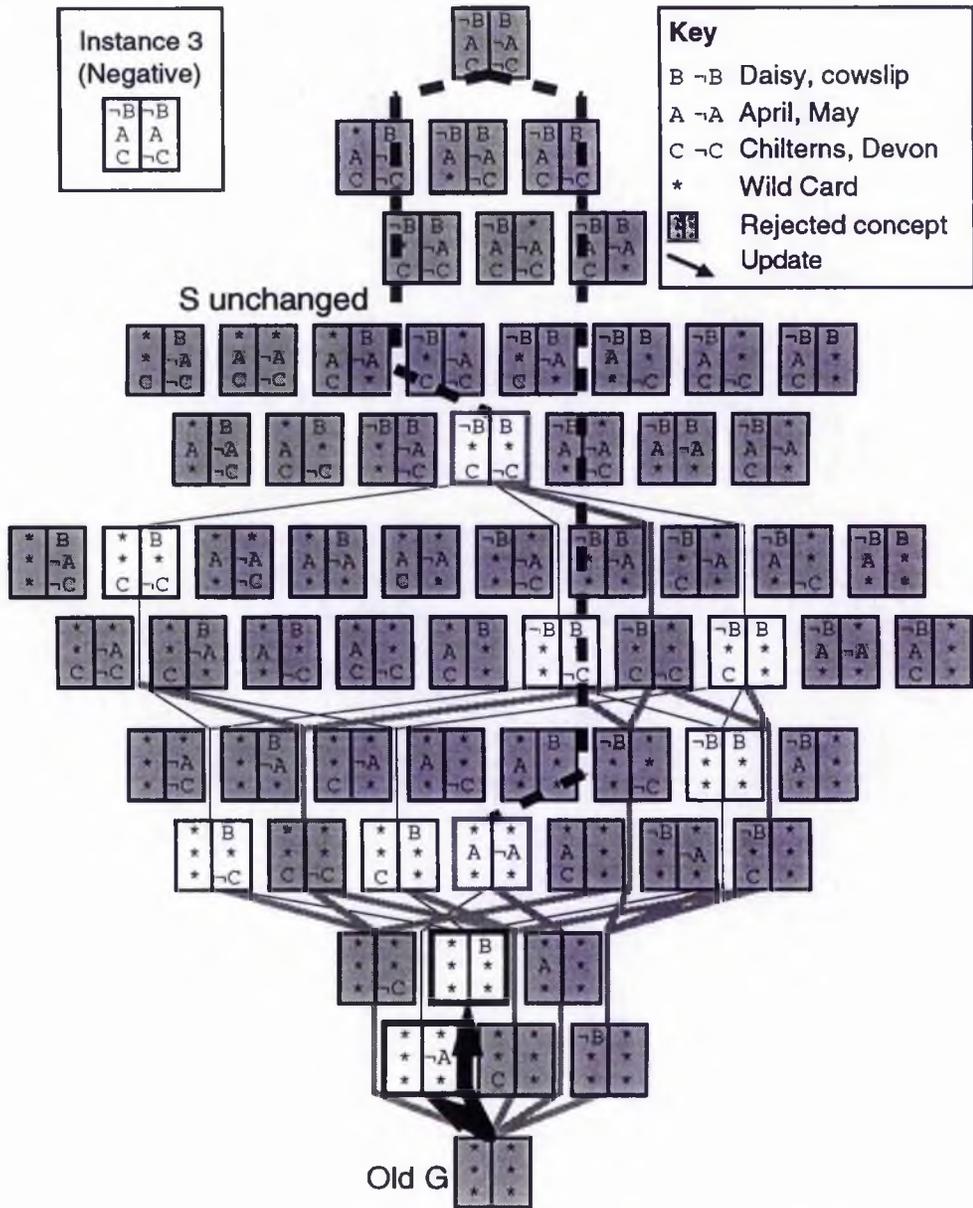


Figure 2.4 — Version space after the third instance is presented. The shaded lines indicate eliminated chains. The dashed lines indicate the updating history.

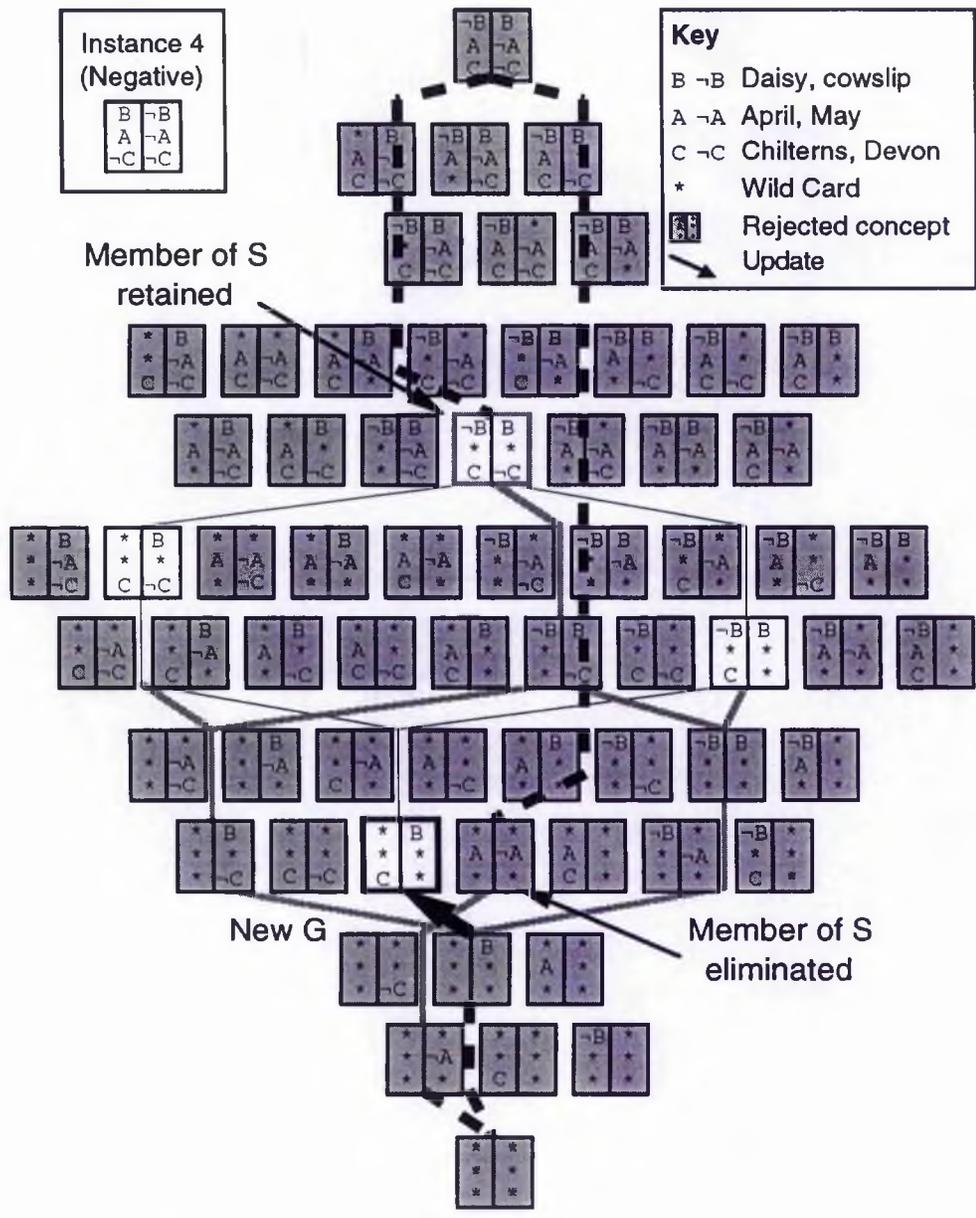


Figure 2.5 — Version space after instance 4 is presented.

The membership of S and G once the other member of G has been minimally updated is therefore:

$$S = \{ \{ [Primula\ veris, *, Chilterns] \} \{ [Bellis\ perennis, *, Devon] \} \}$$

$$G = \{ \{ [* , * , Chilterns] \} \{ [Bellis\ perennis, * , *] \} \}$$

Finally, instance 5, a positive instance, causes S and G to converge to a singleton (figure 2.6):

{[Bellis perennis, April, Chilterns]
[Bellis perennis, May, Chilterns]}+

Thus, the final concept is:

$S = G = \{ \{ [*, *, Chilterns] [Bellis perennis, *, *] \} \}$

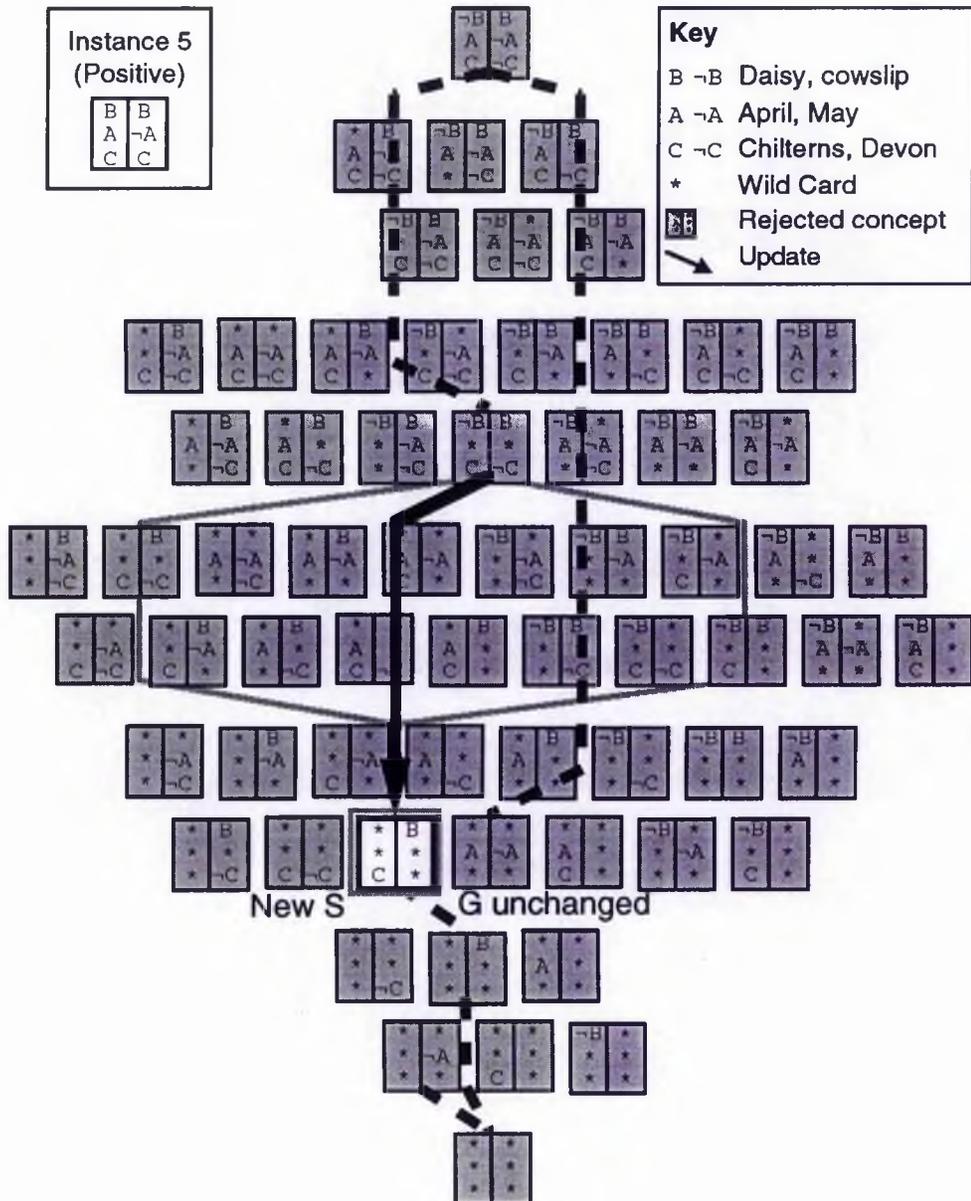


Figure 2.6 — S and G converge after instance 5 is presented.

2.2.5 Formalism of Mitchell's Technique

Mitchell gives a formalism for his technique in chapter 3 of his thesis.²⁰ In this section, part of that formalism is reviewed, with regard to proving the following points:

- That version space may be represented by S and G.
- That updating and selection within maintain S and G at the boundary of version space.

It will also be shown in this section that updating and selection within can be implemented without the need to refer to past instances or to version space, for the family of instance and generalisation languages to which those used in this chapter belong.

Let I be the set of all instances representable by the instance language. Let T be the subset of I whose classification is known, and hence can be used to learn the concept. T is the union of two non-intersecting sets of positive and negative instances, T_+ and T_- respectively.

A concept is learned through a sequence of training instances presented from T_+ and T_- . Let P_{+t} be the subset of T_+ that has been presented by time t , and P_{-t} be the subset of T_- that has been presented by time t . Let P_t be the union of P_{+t} and P_{-t} — the set of all instances that have been presented by time t .

Let W be the set of all concepts representable by the current generalisation language. Let $M(w, p)$ be a matching predicate, where $w \in W$ and $p \in P_t$. M is true if p is classified by w as positive, and false if p is classified by w as negative.

Mitchell then defines the consistency predicate, K as in [2.4] below:

$$K(w, P_{+t}, P_{-t}) \Leftrightarrow \left(\left(\forall p \in P_{+t} \right) M(w, p) \right) \wedge \left(\left(\forall p \in P_{-t} \right) \neg M(w, p) \right) \quad [2.4]$$

²⁰Mitchell, 1978, pp. 56-73

The goal of the concept learning problem is, given W , M and P_t , to find the subset of W which are consistent with P_t . This is called the version space, V_t of P_t . Mitchell gives V_t a formal definition as per [2.5]:

$$V_t = \{w \in W \mid K(w, P_{+t}, P_{-t})\} \quad [2.5]$$

Members of W are partially ordered by the *more specific than or equal to* relation (\geq), defined in [2.6]:

$$(\forall w_1, w_2 \in W)((w_1 \geq w_2) \Leftrightarrow (\{i \in I \mid M(w_1, i)\} \subseteq \{i \in I \mid M(w_2, i)\})) \quad [2.6]^{21}$$

This relation provably has the properties that it is reflexive, antisymmetric, and transitive.²² These, for Mitchell, are the necessary and sufficient conditions for a partial ordering. Borowski and Borwein agree with Mitchell as to the necessary and sufficient conditions for a partial ordering.²³ James and James give an equivalent definition, but from the point of view of the set rather than the relation.²⁴

As mentioned in sections 2.2.2, 2.2.4.2, and as defined in equation [2.6], specificity is measured by looking at the instances recognised as positive by one concept relative to another concept. This is not just the instances shown so far, but the space of all instances. This begs the question of how specificity may be assessed, since it is undesirable to generate every possible instance. For the generalisation languages used in the examples in this chapter, one concept c_1 is more specific than another concept c_2 if and only if for all general object description vectors in c_1 there is a unique general object description vector in c_2 for which each element of the general object description vector in c_2 is either equal to the corresponding element in c_1 , or is a wild card. This ensures that there are no instances matched by c_1 that are not matched by c_2 and hence the instances matched by c_1 form a proper subset of those matched by c_2 .

The strict relation *more specific than* ($>$) is also defined by Mitchell [2.7]:

²¹Mitchell, 1978, p. 59

²²Mitchell, 1978, p. 59

²³Borowski & Borwein, 1989, p. 437

²⁴James & James, 1992, p. 297

$$(\forall w_1, w_2 \in W)((w_1 > w_2) \Leftrightarrow ((w_1 \geq w_2) \wedge (w_1 \neq w_2))) \quad [2.7]^{25}$$

This is used to define the subset of maximally and minimally specific elements of a subset of W , A . The set of maximally specific elements of A is the set $\text{MAX}(A)$, defined in [2.8], and set of the minimally specific elements of A is the set $\text{MIN}(A)$, defined in [2.9]:

$$\text{MAX}(A) = \{a \in A \mid (\forall b \in A) \neg(b > a)\} \quad [2.8]^{26}$$

$$\text{MIN}(A) = \{a \in A \mid (\forall b \in A) \neg(a > b)\} \quad [2.9]^{27}$$

[2.8] and [2.9] can be used to give the definition of the sets S_t and G_t . S_t is the set of maximally specific elements of version space at time t , and G_t is the set of minimally specific elements of version space at time t :

$$S_t = \text{MAX}(V_t) \quad [2.10]^{28}$$

$$G_t = \text{MIN}(V_t) \quad [2.11]^{29}$$

At any time, Mitchell is content that the two sets S and G represent a version space V for a set of instances, P if and only if the condition in [2.12] is satisfied:

$$(\forall w \in W)((w \in V) \Leftrightarrow (\exists s \in S)(\exists g \in G)(s \geq w \geq g)) \quad [2.12]^{30}$$

The proof of [2.12] is given in Mitchell, 1978, pp. 62-63, and is reproduced below:

First prove that:

$$(\forall w \in W)((w \in V) \Leftarrow (\exists s \in S)(\exists g \in G)(s \geq w \geq g)) \quad [2.13]$$

²⁵Mitchell, 1978, p. 59

²⁶Mitchell, 1978, p. 59

²⁷Mitchell, 1978, p. 60

²⁸Mitchell, 1978, p. 60

²⁹Mitchell, 1978, p. 60

³⁰Mitchell, 1978, p. 60

Consider an arbitrarily chosen concept, $w \in W$ and $s \in S$ such that $s \geq w$. By $S \subseteq V$ from [2.10] we have:

$$(\forall p \in P_+)(M(s, p)) \quad [2.14]$$

and by $s \geq w$ (axiom) from [2.6]:

$$\{p \in P \mid M(s, p)\} \subseteq \{p \in P \mid M(w, p)\} \quad [2.15]$$

Thus, from [2.14] and [2.15]:

$$(\forall p \in P_+)(M(w, p)) \quad [2.16]$$

Similarly, for arbitrarily chosen $g \in G$ such that $w \geq g$:

$$(\forall p \in P_-)(\neg M(g, p)) \quad [2.17]$$

and by $w \geq g$ (axiom) from [2.6]:

$$\{p \in P \mid M(w, p)\} \subseteq \{p \in P \mid M(g, p)\} \quad [2.18]$$

Taking the complement of each set:

$$\{p \in P \mid \neg M(w, p)\} \supseteq \{p \in P \mid \neg M(g, p)\} \quad [2.19]$$

Thus, from [2.17] and [2.19]:

$$(\forall p \in P_-)(\neg M(w, p)) \quad [2.20]$$

By [2.16] and [2.20] $w \in V$.

Secondly, it is necessary to prove that:

$$(\forall w \in W)((w \in V) \Rightarrow (\exists s \in S)(\exists g \in G)(s \geq w \geq g)) \quad [2.21]$$

This follows, provided that there exists a chain that contains w , with a maximum and a minimum element. From [2.10] and [2.11], any chain, A of V that contains w will have a member of S as its maximum element, and a member of G as its minimum element. The only possibility is that the chain has no maximum or minimum element. This depends on the matching predicate and generalisation language. If there exists a chain

with no maximum or minimum element, then the generalisation language is said by Mitchell to be inadmissible.³¹

Hence, for the class of admissible languages, version space is represented by S and G . The generalisation languages used in the examples in this chapter are admissible.

To prove that updating and selection within maintain S and G as the boundary representatives of version space, Mitchell gives the proof for the presentation of a new negative instance³² — the proof for a positive instance following by analogy.³³

The proof for a negative instance is given in Mitchell, 1978, pp. 64-65, and is reproduced with a slight change in terminology below:

Consider the version space at time t , V_t , for the set of instances shown so far, and $P_t = P_{+t} \cup P_{-t}$, with the boundary representatives, S_t and G_t . Consider also the version space at time $t + 1$, V_{t+1} , after the presentation of a negative instance, i . $P_{-t+1} = P_{-t} \cup \{i\}$, and $P_{t+1} = P_{+t} \cup P_{-t+1}$. The boundary representatives of V_{t+1} are S_{t+1} and G_{t+1} .

Theorem: If the sets S_{t+1} and G_{t+1} obey the following equalities, then G has been updated, and S has been selected within such that S_{t+1} and G_{t+1} are the boundary representatives of V_{t+1} :

$$G_{t+1} = \text{MIN}(\{v \in V_t \mid \neg M(v, i)\}) \quad [2.22]$$

$$S_{t+1} = \{s \in S_t \mid \neg M(s, i)\} \quad [2.23]$$

[2.22] gives the formula for updating G . [2.23] gives the formula for selection within S . Both are based only on the state of affairs at time t and the new negative instance, i .

Proof: By definition of the consistency predicate [2.4], we have:

³¹Mitchell, 1978, p. 61

³²Mitchell, 1978, pp. 64-65

³³Mitchell, 1978, p. 65

$$K(v, P_{+(t+1)}, P_{-(t+1)}) \Leftrightarrow (K(v, P_{+t}, P_{-t}) \wedge \neg M(v, i)) \quad [2.24]$$

and by definition of V_t [2.5]:

$$V_{t+1} = \{w \in W \mid K(w, P_{+(t+1)}, P_{-(t+1)})\}$$

$$\text{By [2.24]:} \quad \Leftrightarrow V_{t+1} = \{w \in W \mid K(w, P_{+t}, P_{-t}) \wedge \neg M(w, i)\}$$

$$\text{By [2.5]:} \quad \Leftrightarrow V_{t+1} = \{w \in W \mid (w \in V_t) \wedge \neg M(w, i)\}$$

$$\text{Simplifying:} \quad V_{t+1} = \{v \in V_t \mid \neg M(v, i)\} \quad [2.25]$$

Therefore, from [2.25]:

$$G_{t+1} = \text{MIN}(V_{t+1}) = \text{MIN}(\{v \in V_t \mid \neg M(v, i)\}) \quad [2.26]$$

The first half of the theorem [2.22] is proven.

To prove [2.23], it is first necessary to prove that $S_{t+1} \subseteq S_t$.

Consider arbitrary s in S_{t+1} . We wish to prove that $s \in S_t$ for all s . We begin with:

$$(\forall v \in V_t)((v \in V_{t+1}) \vee (v \notin V_{t+1})) \quad [2.27]$$

By definition of S_{t+1} [2.10] we know that:

$$(\forall v \in V_{t+1})((\forall s \in S_{t+1})(\neg(v > s))) \quad [2.28]$$

$$\text{Therefore:} \quad (\forall v \in V_t)((\forall s \in S_{t+1})(\neg(v > s)) \vee (v \notin V_{t+1})) \quad [2.29]$$

$$\text{From [2.25]:} \quad (\forall v \in V_t)((v \notin V_{t+1}) \Rightarrow M(v, i) \Rightarrow ((\forall s \in S_{t+1})(\neg(v > s)))) \quad [2.30]$$

$$\text{Thus:} \quad (\forall v \in V_t)((\forall s \in S_{t+1})(\neg(v > s))) \quad [2.31]$$

$$\text{But:} \quad S_t = \text{MAX}(V_t) = \{v \in V_t \mid (\forall w \in V_t)(\neg(w > v))\} \quad [2.32]$$

Therefore $s \in S_t$ for all $s \in S_{t+1}$, and we have proven that $S_{t+1} \subseteq S_t$.

$$\text{Since:} \quad S_{t+1} = \text{MAX}(V_{t+1}) = \text{MAX}(\{v \in V_t \mid \neg M(v, i)\}) \quad [2.33]$$

$$\text{and:} \quad S_{i+1} \subseteq S_i \subseteq V_i \quad [2.34]$$

$$\text{we have:} \quad S_{i+1} = \{s \in S_i \mid \neg M(s, i)\} \quad [2.35]$$

and the theorem is proven.

Mitchell gives the formulae for updating S [2.36] and selection within G [2.37], on presentation of a new positive instance, j .³⁴ The proof of these formulae is analogous to the proof of [2.22] and [2.23], according to Mitchell.

$$S_{i+1} = \text{MAX}(\{v \in V_i \mid M(v, j)\}) \quad [2.36]$$

$$G_{i+1} = \{g \in G_i \mid M(g, j)\} \quad [2.37]$$

The formulae for selection within [2.23] and [2.37] require only the knowledge of the current instance, and the old S and G . However, the formulae for updating [2.22] and [2.36] refer to the old version space, which might imply an algorithm that inspects the whole of version space to find the new members of S or G when updating.

The procedures for updating S and G are specific to the generalisation language, as Mitchell states in his thesis:

The functions UPDATE-G [and] UPDATE-S ... depend upon the particular concept description language.³⁵

Mitchell does give a general strategy for updating S ,³⁶ which basically entails generalising each member of S as much as is necessary, and in every way possible until the positive instance is matched, until S contains only concepts that match the positive instance. Then members of S are removed that are more general than other members of S , or are not more specific than a member of G .

³⁴Mitchell, 1978, p. 65

³⁵Mitchell, 1978, p. 36

³⁶Mitchell, 1978, p. 41

For the generalisation languages used in the examples in this chapter, updating can be done without reference to previous patterns, and without exhaustively examining every member of version space. The algorithm for updating S for the examples in this chapter is given below:

- 1 For each member of S , s (where S is an ordered array of concepts):
 - 1.1 If s matches the new positive instance, retain s at the beginning of S .
 - 1.2 Otherwise generate new members of S :
 - 1.2.1 Generalise s in as many ways as possible by adding a $*$ in one field. Let a given generalised s be t .
 - 1.2.2 If t is not more specific than a member of G , discard t .
 - 1.2.3 If there is a member of S that matches the new positive instance which is more specific than or equal to t , then discard t .
 - 1.2.4 If t does not match the new positive instance, add t to the end of S .
 - 1.2.5 If t does match the new positive instance, add t to the beginning of S .
- 2 Repeat until the end of S is reached.
- 3 Remove from S any concept that is more general than another member of S .

The algorithm for updating G is similar, except that the instance is negative, and hence concepts are retained in G that do not match the instance. Members of G are made more specific by replacing a wild card with the corresponding attribute value from a member of S .

The algorithm retains those members of S that match the new instance (1.1). If these were maximally specific elements of the old version space, then they must be maximally specific elements of the new version space. Those members of S that do not match the new instance are made more general in every way that is possible by adding only a single wild card (1.2.1). This is the minimum amount by which a member of S may be updated. If an updated member of S is not more specific than a member of G , it must be discarded (1.2.2), since it cannot be a member of version

space. Since S is the set of maximally specific elements of version space, if a member of S exists that correctly classifies the new positive instance and is more specific than an updated member of S , the updated member of S must be discarded (1.2.3). If an updated member of S still does not correctly classify the new instance, it is retained to be updated further (1.2.4). If an updated member of S does correctly classify the new instance, having not been discarded in 1.2.2 or 1.2.3, then it must be the maximally specific update (1.2.5).

The algorithm therefore chooses the most specific elements from the old version space that match the new positive instance to determine the update of S , as per the formula for updating S given in [2.36].

2.3 Problems With Mitchell's Technique

This section examines some of the possible problems and criticisms that may be aimed at Mitchell's technique. Although the examples used above required relatively few instances, these instances were carefully constructed in order to illustrate the technique. Section 2.3.1 looks at how Mitchell's technique copes when there are not enough instances to allow the generation of the no-alternative situation. It will also be apparent from these examples that Mitchell's technique is very sensitive to the consistency of the data. Inconsistent data leads to an unlearnable concept in Mitchell's technique. Section 2.3.2 examines this problem, and briefly looks at a technique which aims to deal with it.³⁷ Mitchell's technique may also be criticised for an inability to learn disjunctive concepts.³⁸ Section 2.3.3 explores this issue. Also, from a practical standpoint, the representation of version space may entail unfeasibly large sets S and G . This problem is briefly examined in section 2.3.4.

³⁷Hirsh, 1990

³⁸Rich & Knight, 1991, p. 469

2.3.1 When No-Alternative Cannot be Found

The problem that a large number of instances may be required to achieve the no-alternative situation is one of which Mitchell is well aware.³⁹ For example, let's take the state of S and G at the end of the second instance in the first example in section 2.2.4:

$$S = \{ \{ [Lonicera\ periclymenum, June, Essex] \} \}$$

$$G = \{ \{ [Lonicera\ periclymenum, *, *] \} \{ [*, June, *] \} \{ [*, *, Essex] \} \}$$

There are over 1300 flowers listed in *The Pocket Guide to Wild Flowers*,⁴⁰ from which the instances used here are taken. There are eleven months of the year that are not June, and roughly 100 counties in the United Kingdom and Republic of Ireland. This means there are approximately 1.4 million negative instances that could be presented, each of which would have no effect on either S or G. By no means can it be said, even for this very simple example that the no-alternative situation may be guaranteed with a small number of instances.

However, this can be looked at in two ways. Firstly, the fact that there are 1.4 million instances that one can already definitely classify as negative is an impressive generalisation from only two instances learned.

Secondly, the unconverged version spaces may still be useful. When the no-alternative situation is not achieved, Mitchell distinguishes two types of classification of subsequent instances: *ambiguous* classification and *unambiguous* classification. Ambiguously classified instances are those for which members of S and G give different classifications — some positive, some negative. Unambiguously classified instances are those for which all members of S and G give the same classification.

For example, the instance below is an ambiguously classified instance:

$$\{ [Lonicera\ periclymenum, July, Lancashire] \}$$

³⁹Mitchell, 1982, pp. 216-218

⁴⁰McClintock & Fitter, 1961

One member of G: {[*Lonicera periclymenum*, *, *]} classifies it as positive, the others, along with S, classify it as negative.

The following instance is unambiguously classified as negative, however:

{[*Pinguicula lusitanica*, July, Lancashire]}

The unambiguous classification, according to Mitchell, must be the correct classification, given certain conditions:

It can be proven that any ... unambiguous classification is a correct classification, provided that (1) the observed training instances were correct, and (2) the generalisation language allows describing the target generalisation.⁴¹

The case of ambiguous classification is more difficult. In general, there is no heuristic that may be used which is guaranteed to give the correct classification, since any concept in version space could be the correct concept. There are, perhaps, possibilities for providing a reasonable guess at the correct classification, however:

By considering outside knowledge or by examining the proportion of generalisations in the version space which match the instance, one might still estimate the classification of such instances.⁴²

In general, all instances whose attribute values differ from the first positive instance in every dimension must be unambiguously classified as negative, given the above two conditions, and the further condition that G is not the maximally general concept. The only guaranteed unambiguously classified positive instance is the first positive training instance. There may be others, but this depends on the individual case under examination. S and G may contain various concept descriptions at any stage, and it is quite possible that only the first positive instance will be an instance of all of them.

⁴¹Mitchell, 1982, p. 217

⁴²Mitchell, 1982, p. 217

One can therefore say that a new instance chosen at random is far more likely to be given unambiguous negative classification than unambiguous positive classification. The smaller version space is, however, and hence the closer S and G are to convergence, the greater the number of instances that can be given unambiguous classification, positive or negative.

Version space can be used to determine the best instance to maximise removal of candidate concepts:

The instance which will provide on the average the most useful information is the instance which comes closest to matching one half of the generalisations in the version space. Regardless of its classification, ... [this] will allow rejecting one half of the currently plausible generalisations. Thus, by testing each instance to determine what proportion of the generalisations in the version space it matches, the most informative training instance can be selected.⁴³

Thus, one has the possibility of developing an algorithm that could ask the instance provider for certain useful instances, and thereby increasing the likelihood of achieving the no-alternative situation in reasonable time. This, of course, assumes that the classifications of the desired instances can be found.

2.3.2 When the Data are Inconsistent

The problem of inconsistent data is far greater for Mitchell than that associated with insufficient training instances. The correctness of the first positive instance is crucial in building version space. If this instance is wrongly classified, or an element of it contains the wrong symbol, it is quite possible that the correct concept will not be a member of version space.

Subsequent instances, whilst not as crucial as the first, may still prevent the correct concept being found if they are inconsistent. However, an observation of Hirsh relating to the insufficient data problem can be taken

⁴³Mitchell, 1982, p. 218

to infer that early instances are more important in the development of the concept than those that are shown later:

As each instance throws away candidate concept definitions, the version space gets smaller and smaller. As the version space decreases in size, the probability that a randomly chosen instance will make a difference — will be able to remove candidate concept definitions — becomes smaller and smaller.⁴⁴

To show how Mitchell's technique fails when confronted with inconsistent data, let us again consider the state of version space after the second instance in the example in section 2.2.4.1:

$$S = \{ \{ [Lonicera\ periclymenum, June, Essex] \} \}$$

$$G = \{ \{ [Lonicera\ periclymenum, *, *] \} \{ [*, June, *] \} \{ [*, *, Essex] \} \}$$

Suppose the third instance, which lead to the no-alternative situation in the example, had been wrongly classified as negative:

$$\{ [Lonicera\ periclymenum, July, Hertfordshire] \}^-$$

In that case, the first member of G, which this is an instance of, would be made more specific in two ways, to avoid the classification as positive of a negative instance by a member of G:

$$\{ [Lonicera\ periclymenum, *, *] \} \rightarrow$$

$$\{ [Lonicera\ periclymenum, June, *] \} \text{ and}$$

$$\{ [Lonicera\ periclymenum, *, Essex] \}$$

Each of these is more specific than another member of G, hence G at the end of this instance would be the following set:

$$G = \{ \{ [*, June, *] \} \{ [*, *, Essex] \} \}$$

The concept that was to have been the correct, final concept ($\{ [Lonicera\ periclymenum, *, *] \}$) has now been pruned from the search.

⁴⁴Hirsh, 1990, p. 34

The problem is addressed to a certain extent in a paper by Hirsh.⁴⁵ Provided that the data has *bounded* inconsistency, one can "blur"⁴⁶ instances, by considering the training instances in the neighbourhood of each instance, and the concepts kept are those that correctly classify at least one of the set of the instance and its neighbours. A definition of neighbourhood must be given by the user.⁴⁷

In an ordered space of instances, bounded inconsistency means that misclassified instances are those that occur near (where nearness is defined) the boundary between those instances which are matched by the concept, and those instances which are not. The definition of nearness is important in that it has a strong effect on the likelihood of a single concept being found:

Ideally, the result of this learning process would be a singleton version space containing the desired concept definition. However, if not given enough data the final version space will have more than one definition. This also happens if the definition of nearby is too generous, since it will allow too many concept definitions into the version space, and no set of instances will permit convergence to a single concept definition. The definition of nearby should be generous enough to guarantee that the desired concept definition is never thrown out by any instance, but not too generous to include too many things (or in the worst case, everything).⁴⁸

For each instance, a version space is constructed that is consistent with the instance and its neighbours. The intersection of this version space with the version space formed from previous instances becomes the new version space, since this will contain concepts that have the minimum bounded inconsistency with all the instances so far. Concepts lying outside the intersection will either be inconsistent (beyond the bounds) with the instances so far, or with the new instance.

⁴⁵Hirsh, 1990

⁴⁶Hirsh, 1990, p. 33

⁴⁷Hirsh, 1990, p. 34

⁴⁸Hirsh, 1990, p. 34

Mitchell's approach to the problem was to maintain parallel version spaces.⁴⁹ If a concept cannot be found that is consistent with all the instances, then version spaces are constructed on the basis of consistency with all but one instance. If no concept can be found that is consistent with all but one instance, then consistency with all but two instances is used, and so on, using increasingly smaller subsets of the instances. This obviously requires the maintenance of several boundary sets S and G , and is extremely costly both in time and memory. As one would expect, Hirsh reports improvements in terms of time and memory required over Mitchell's method.⁵⁰

2.3.3 When the Concept is Disjunctive

Mitchell's technique itself has a bias in that it cannot learn disjunctive concepts — e.g. the concept of "a flower observed in Kent or a flower observed in Sussex". No representational language and matching predicate can be provided that overcomes this problem, since the very nature of the technique, namely in the updating operation, assumes that when positive instances differ, it is necessary to generalise. When disjunctive concepts are allowed, however, differences between positive instances could mean that they belong to different disjuncts of the concept. Rich and Knight observe that disjunctive concepts are fundamentally unlearnable in this technique:

Generalisation can easily degenerate to the point where the S set contains simply one large disjunction of all positive instances.⁵¹

For example, if S is the following set:

$$S = \{ \{ [Myosotis arvensis, July, Fife] \} \}$$

Then, given the positive instance of a forget-me-not observed in June in Fife:

⁴⁹Mitchell, 1978, Ch. 5

⁵⁰Hirsh, 1990, p. 36

⁵¹Rich & Knight, 1991, p. 469

{[Myosotis arvensis, June, Fife]}+

Instead of generalising the month, so that the concept is a forget-me-not observed in Fife at any time of the year, there is the possibility with disjunctive concepts being represented that the concept could be a forget-me-not observed in July in Fife, *OR* a forget-me-not observed in June in Fife. This process could continue, as more and more positive instances are added. *S* is simply the disjunct of all the positive instances, and is never generalised such that a given member of *S* matches more than one instance.

Yet, as Mitchell points out,⁵² biases can enable learning as well as hindering it. It is the very process of candidate elimination, the foundation of the whole technique, that undermines the ability of the technique to learn disjunctive concepts, and yet enables the technique to learn the concepts it is able to learn.

Mitchell's technique can still be used to learn disjunctive concepts, but again with costly adaptations.⁵³ The approach uses several passes over the training data. In the first pass, a concept is found such that the maximum number of positive instances are instances of the concept, whilst not allowing any of the negative instances. Those positive instances that are matched by the concept found are removed from the training set, and a further concept is found in a similar way. When all the positive instances have been covered, the final concept is then simply the disjunct of all the concepts found during the learning process.

2.3.4 Controlling the Size of the Boundary Sets

For large search spaces, the boundary set representation of version space can itself become unwieldy. Mitchell copes with this problem by selecting instances that tend to reduce the size of *S* and *G*. There are two kinds of instance that are useful for this purpose:⁵⁴

⁵²Mitchell, 1980

⁵³Mitchell, 1978, cited in Rich & Knight, 1991, pp. 469-470

⁵⁴Mitchell, 1978, p. 91

- Instances that match half the concepts in each boundary set.
- Instances that match with all the attribute values in the concept but one.

The first strategy reduces the size of boundary sets that are already large, and applies to the process of selection within. The second strategy is aimed at reducing branching during updating. With only one differing attribute, the number of ways of specialising G or generalising S is reduced to one. For example, given:

$$G = \{ \{ [Lonicera\ periclymenum, *, *] \} \{ [*, June, *] \} \\ \{ [*, *, Essex] \} \}$$

and a negative instance:

$$\{ [Rhododendron\ ponticum, June, Essex] \}^-$$

the two members of G that match the instance are specialised, and G becomes:

$$G = \{ \{ [Lonicera\ periclymenum, *, *] \} \\ \{ [Lonicera\ periclymenum, June, *] \} \\ \{ [Lonicera\ periclymenum, *, Essex] \} \}$$

Note that there is only one possibility to minimally specialise each member of G that matches the negative instance, such that the instance is no longer matched. This is because the negative instance differed in only one attribute from the attribute values available.

Now these specialisations of G mean that they are now more specific than another member of G. Hence they are removed, and G is reduced to a single member:

$$G = \{ \{ [Lonicera\ periclymenum, *, *] \} \}$$

Suppose the negative instance had differed in more than one attribute:

$$\{ [Rhododendron\ ponticum, June, Avon] \}^-$$

There is now only one member of G that matches the instance, and it can be specialised in two ways:

$$\{[* , \text{June} , *]\} \rightarrow \{[\text{Lonicera periclymenum} , \text{June} , *]\} \text{ and} \\ \{[* , \text{June} , \text{Essex}]\}$$

Although these two are more specific than other members of G, and would be removed anyway, the point is made that there are more ways to specialise G with a negative instance that differs in many attributes, than for a negative instance that differs in only one attribute.

2.4 Conclusion

Mitchell's technique has been shown to find the no-alternative situation for a certain subset of concepts. The no-alternative situation, or knowing when to stop learning is a valuable facility. That which is of value to symbolic AI might also be of value to neural networks. Neural networks stand to gain considerably if they can make use of this, or any other technique in such a way that enables the knowledge of when the no-alternative situation has been reached.

In the symbolic domain, the representational languages used provide both a limit on the concepts that can be learned, and also the means by which those concepts are learned. The efficient representation of a vast space of alternative concepts improves feasibility of concept learning. The candidate elimination processes enable effective pruning of the search without impairing the certainty of finding the correct concept, given sufficient data. The results of the technique are still useful even when there is not enough data to give complete convergence, and there is the possibility of finding out the most useful instance in terms of eliminating candidate concepts from version space. Mitchell's technique is also able to detect when it has been given inconsistent data.

Thus Mitchell addresses himself to the three outstanding problems he outlines for concept learning in his thesis:⁵⁵

⁵⁵Mitchell, 1978, p. 8

- Knowledge of when the concept has been learned.
- Proposal of informative new instances.
- Detection of and recovery from inconsistent data.

Of these, the most important problem is the first one. Surely the most useful facility of Mitchell's symbolic technique is the fact that he is able to detect the no-alternative situation, and thus is able to know when to stop training.

The apparent weakness of the technique, bias, is paradoxically what enables learning. It is of philosophical interest that the categorisations made of the attributes and attribute values lead to the enabling of certain concepts being learned, whilst inhibiting the learning of other concepts.

The bias of the technique itself — a difficulty in learning disjunctive concepts — is due in part to the fact that the technique is couched in the symbolic AI domain. There is no capability to acquire a global sense of an embodiment of IO behaviour in a concept. This is because concepts must be described using a language, and hence disjunction explicitly represented, rather than taken as part of a whole. In neural networks there is no need for this. The concept is taken from the whole IO behaviour, and the disjunctive nature of the concept is not an issue.

That Mitchell made a significant contribution to symbolic AI and concept learning through this technique is undeniable. It is my belief that his method can also be employed in such a way as to be of benefit to learning with neural networks as well.

3 Generalisation In Neural Networks

Generalisation is the ability of the learner to provide output for untrained, rather than just trained, input. The degree to which this output is consistent with the concept or function being learned indicates the degree of understanding the learner has of the concept. The study of generalisation aims to maximise the degree of fit the learner has with the concept being learned, without exhaustively training on all possible examples. In this chapter, after an introduction to some issues in generalisation in neural networks, section 3.2 examines more general issues in generalisation, and relates them to the choice of topology in neural networks, and section 3.3 discusses various ways of looking at generalisation in neural networks in the literature. Section 3.4 details the Mitchellian view of generalisation, and how it relates to neural networks.

3.1 Introduction

In the symbolic technique discussed in the preceding chapter, Mitchell had to use a special language in order to enable the representation of generalisation. Neural networks, however, are able to generalise by their own nature. Any neural network has a certain set of legal inputs, I , such as $\{1, 0\}^n$, or \mathbf{R}^n , for example, and an output can be calculated for any of these inputs, given a topology and a set of weights. If a training set, T , does not contain all members of I , then it is possible to calculate an output for an input that is not the input of a member of T .

If the inputs are taken from \mathbf{R}^n , then there are an infinite number of possible architectures that correctly classify a finite training set, T , and an infinite number of possible generalisations of T . There are an infinite number of possible topologies, since any number of hidden units can be used beyond the minimum number needed to correctly classify T . There are an infinite number of possible generalisations of T , since there are an infinite number of points outside T , each of which may be assigned a different output.

The IO pictures given in figure 3.1 show the wide variety of possible generalisations that are to be found from the same set of patterns, for different architectures. All of these have zero classification error on the training set, but what is to be said about their generalisations to the rest of input space?

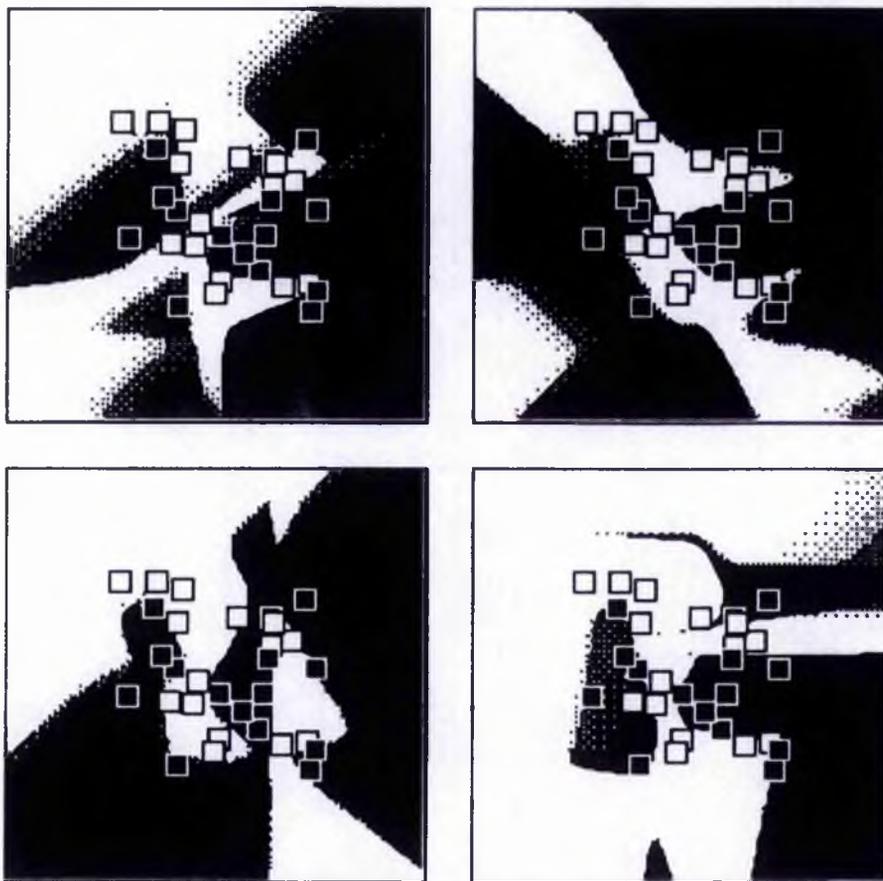


Figure 3.1 — *Various generalisations of a set of random patterns. Patterns with a target of 1 are in black, those with a target of 0 are in white. (Top left: cascade architecture¹ with 7 hidden units; Top right: standard feed-forward architecture with a 2*5*5*1 topology; Bottom left: standard feed-forward architecture with a 2*10*1 topology; Bottom right: cascade architecture with 9 hidden units.)*

Generalisation is clearly a natural and primary ability of neural networks. The ability to harness and understand this ability of neural networks

¹Fahlman & Lebiere, 1991

would be of benefit to the discipline. It would certainly have far reaching consequences in terms of the real-world applications to which neural networks could be reliably put to use.

Real-world training sets usually come from a distribution with an unknown underlying function. For example, one might consider using a neural network to predict the weather in a local area. The inputs could be readings of temperature, wind speed, and cloud cover, and the targets could be the same readings taken one hour later. The function that relates the readings between hourly intervals is unknown. If a group of people wanted to launch a hot-air balloon, and to know that the weather would be suitable over the one hour period it took them to do so, the prediction of the neural network would have crucial consequences. How then, is it possible to be sure that a neural network will be a reliable predictor of unseen instances? Without any knowledge of the underlying function, the accuracy of the prediction of the neural network can only be assessed on the basis of further sampling.

Much of the literature refers to "good", or "valid" generalisation.² By this, it is usually meant that the neural network is a reliable predictor of unseen examples. Yet what basis is there for making the claim that a given learning technique is more likely to produce a network with a good generalisation than any other learning technique? When comparing techniques, it is important to observe the assumptions made about the data, the nature of input and output space, and any restrictions to the architecture, since these assumptions and restrictions often form the basis of the authors' claims.

Some techniques assume noiseless data when constructing the neural network. This is also the assumption of Mitchell, in the symbolic technique, and of some other techniques, such as Wolpert's HERBIE.³ In such cases, an exact realisation of the training set is a necessary condition of good generalisation. However, often the data sets do contain noise, (by

²E.g. Baum & Haussler, 1989; Burton & Faris, 1991; Kanaya & Miyake, 1991; Mato & Parga, 1992; Sietsma & Dow, 1991

³Wolpert, 1989, 1990

which it is meant that a pattern in the training set might have a misleading target for its input), in which case, attempting an exact fit to the data will lead to poor generalisation.

Inputs may be taken from such sets as \mathbb{R}^n , $\{0, 1\}^n$, or $\{-1, +1\}^n$ as mentioned earlier. Outputs may be from $\{0, 1\}^m$, $\{-1, +1\}^m$, $[0, 1]^m$, $[-1, +1]^m$, or \mathbb{R}^m as is the case in function approximation. This is achieved in neural networks by not using an activation function (such as a sigmoid) on the output units, and using the excitation of the output unit as the output of the unit instead.⁴ Often, m is assumed to be 1, for the sake of simplicity.⁵

Techniques that aim to give good generalisation, must also give some definition of what is meant by "good". If by "good" it is meant that the network is a reasonably accurate predictor of unseen examples, then given a set of data, and no knowledge of the underlying function, there is no *a priori* way of deciding what constitutes a good generalisation beyond the data given, without making assumptions. Any generalisation of the data set is possible, and with no further information about the underlying function, it is impossible to give any rank to the alternatives, beyond these assumptions, which may be wrong. Therefore, techniques need to be realistic in their approach to generalisation.

⁴Hornik et al, 1989, p. 360; Kurkova, 1992, p. 502

⁵E.g. Hertz et al, 1991, pp. 148-149

3.2 Issues in Generalisation

In this section, the central issues in generalisation will be discussed. Although the thesis as a whole is concentrated on the classification paradigm, this section will be looking largely at function approximation. This is because the concepts are more easily described in this context. The concepts will be related back to the classification paradigm at the end of the section.

3.2.1 Assumptions about the Underlying Function

The most important issue in generalisation is probably that of the assumptions made about the underlying function of a particular data sample.⁶ A statistician, when trying to fit a set of data, will make some assumptions about the nature of the underlying function before trying to derive a function which fits the data as closely as possible. For example, take the three points in figure 3.2(a). A statistician might fit a linear (b), polynomial (c), or non-polynomial (d) function to these datapoints. Each would result in a very different generalisation.

In neural networks, the type of function fitted to the data is, perhaps, of less concern, especially when there are authors who state that neural networks can fit any function arbitrarily closely.⁷ The hope is that a neural network will somehow discover the underlying function without any *a priori* assumptions about what sort of function it is. This issue is addressed by Geman et al, in what they see as a trade-off between bias — the *a priori* assumptions made about the underlying function, and variance — the number of functions that a fitting mechanism can generate:

Much of the excitement about artificial neural networks revolves around the promise to avoid the ... process of articulating heuristics and rules for machines that are to perform nontrivial perceptual and cognitive tasks. ... We would naturally prefer to "teach" our machines

⁶Draghici, 1995, Ch. 3

⁷Funahashi, 1989; Hornik et al, 1989

by example, and would hope that a good learning algorithm would “discover” the various heuristics and rules that apply to the task at hand. ... Such a system may be said to be *unbiased*, as it is not a priori dedicated to a particular solution or class of solutions. But the price to pay for achieving low bias is high variance. A machine sufficiently versatile to reasonably approximate a broad range of input/output mappings is necessarily sensitive to the idiosyncracies of the particular data used for its training, and therefore requires a very large training set.⁸

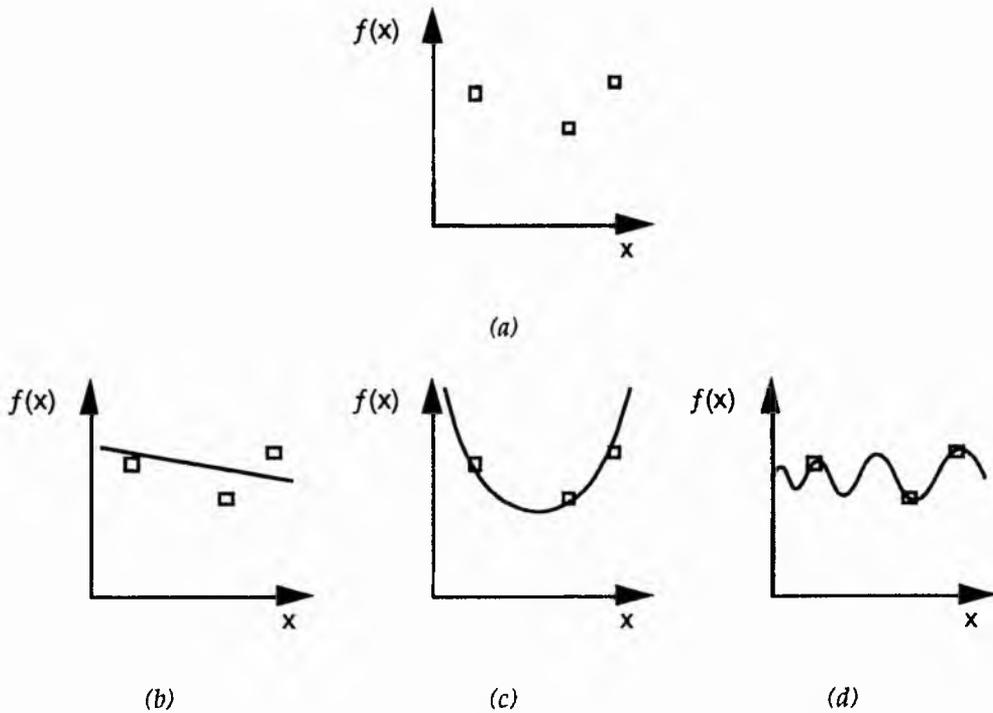


Figure 3.2 — Some data points (a), and possible fits to the data points by a linear (b), polynomial (c) and non-polynomial (d) function.

Of course, a specific neural network is not an unlimited function approximator. With a given number of units and weights, the kinds of functions that can be implemented are restricted. Therefore, the bias/variance trade-off is also related to the choice of topology:

⁸Geman et al, 1992, pp. 14-15

A small network, with say, one hidden unit, is likely to be biased, since the repertoire of available functions ... will in this case be quite limited. On the other hand, if we overparameterize, via a large number of hidden units ... then the bias will be reduced ... but there is then the danger of ... significant variance...⁹

Hence, whilst neural networks in general represent a universal, high-variance approach, capable (in theory) of approximating any function, a specific neural network topology has a bias in the number and kinds of functions it can implement.

Some authors attempt to circumvent this trade-off through the use of automated topology determination, in which the topology is adjusted during training.¹⁰ There are two main approaches: that of starting with a small topology and adding units, and that of pruning units from an initially overlarge topology. There are some differences in opinion as to the generalisation abilities of such techniques. Smieja claims that the constructive techniques do not generalise as well as back-propagation,¹¹ whereas Hertz et al, state that such algorithms have "encouraging results" in terms of generalisation.¹² Sietsma and Dow's pruning technique did not result in improved generalisation abilities:

The ... narrow, five-layer networks generalised far worse than their "parent" networks, indicating that fewer first-layer units, followed by more processing stages, does not lead to better generalisation.¹³

Chapter 4 discusses issues in topology determination, giving some insights into the capabilities of neural networks with various topologies in the classification paradigm.

⁹Geman et al, 1992, p. 12

¹⁰E.g. Baum & Lang, 1991; Brent, 1991; Fahlman & Lebiere, 1991; Fujita, 1992; Hirose et al, 1991; Kameyama & Kosugi, 1991; Lee & Chung, 1990; Mezard & Nadal, 1989; Radcliffe, 1993; Sietsma & Dow, 1991; Weigend et al, 1991a, 1991b; Wynne-Jones, 1993

¹¹Smieja, 1993, p. 369

¹²Hertz et al, 1991, p. 162

¹³Sietsma & Dow, 1991, p. 79

3.2.2 Assumptions about the Data

The next assumption that someone who is trying to fit a function to a sample from the underlying function relates to the quality of that sample. The data may be perfect samples of the underlying function, in which case, it is desirable for a fitting function to pass through all the datapoints. However, the data may also contain noise, in which case, a function which passes through all the datapoints will not generalise well.

Noise is rather a difficult issue. To a certain extent, noise can be regarded as inaccuracies in sampling the underlying function. However, there is also the possibility that the underlying function is more complex than expected, and the samples are all correct. Hence noise is unwanted variation which may come in various forms. Assumptions about the amount of noise in the data are therefore also biases, since they represent an *a priori* preference for one kind of fit rather than another.

3.3.3 The Over-Fit/Under-Fit Dilemma

Finally, there is the issue of over-fitting and under-fitting the data. Put simply, over-fitting is whereby there are too many peaks and troughs in the fitting function, relative to a simple interpolation of the data. Under-fitting means that there are too few peaks and troughs in the fitting function as far as the sample points are concerned. Both result in poor generalisation. For example, consider a set of points which lie on the underlying function in figure 3.3(a). Figure 3.3(b) shows an over-fitting function, and figure 3.3(c) shows an under-fitting function, which does not go through all the sample points.

The difference between the over-fit/under-fit dilemma and the bias/variance trade-off is that the former pertains to the fit to the data, whereas the latter pertains to the *a priori* assumptions which control the kind of fit that can be made.

If the data are noisy, then some datapoints must be assumed to be incorrect, and should not be fitted exactly by the fitting function. The amount of noise that is assumed in the data is a bias, since it will relate to the number of parameters chosen for the fitting function.

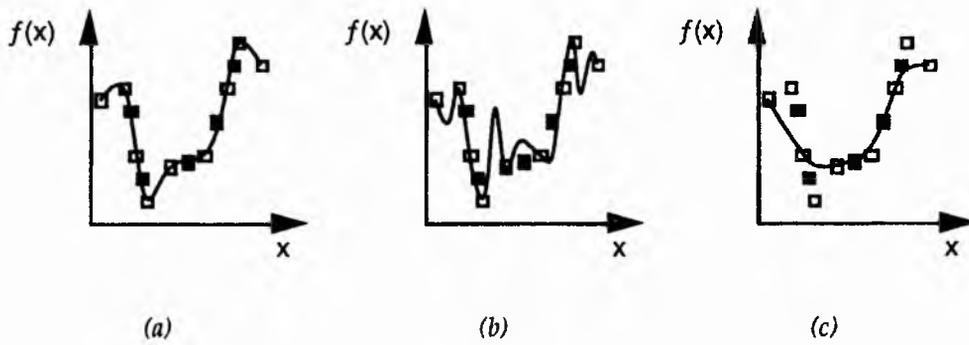


Figure 3.3 — (a) A sample of 14 points from the underlying function. (b) A function which over-fits the 9 white samples, but then does not fit all of the 5 black samples. (c) A function which underfits the 9 white samples.

Underestimating the degree of noise in the sample means the number of parameters used in the fitting function will be high relative to the underlying function. This may result in over-fit, since more datapoints have been assumed to be correct than actually are, and thus the fitting function may pass through incorrect datapoints. Conversely, overestimating the degree of noise in the sample means that too few parameters are used in the fitting function. This may result in under-fit.

Thus it may be seen that too much bias may result in under-fit, and too much variance may result in over-fit. Figure 3.4 shows a series of fits by neural networks with varying numbers of hidden units. The underlying function is sampled 100 times, and noise applied in order to encourage over-fit, where that is possible. Topologies with 2, 15 and 35 hidden units are trained on the sample, using a combination of GA training and back-propagation. The number of hidden units chosen is a bias. The 2 hidden unit topology under-fits the underlying function, and the 35 hidden unit topology over-fits it, since it is capable of a more complex fit. The 15 hidden unit topology gives a fit which is close to the underlying function.

The concepts of bias and variance must be separated from the concepts of over-fit and under-fit, however. There is no reason why the 15 or 35 hidden unit topologies, with higher variance than the 2 hidden unit topology, might not also under-fit the data, if they were so trained.

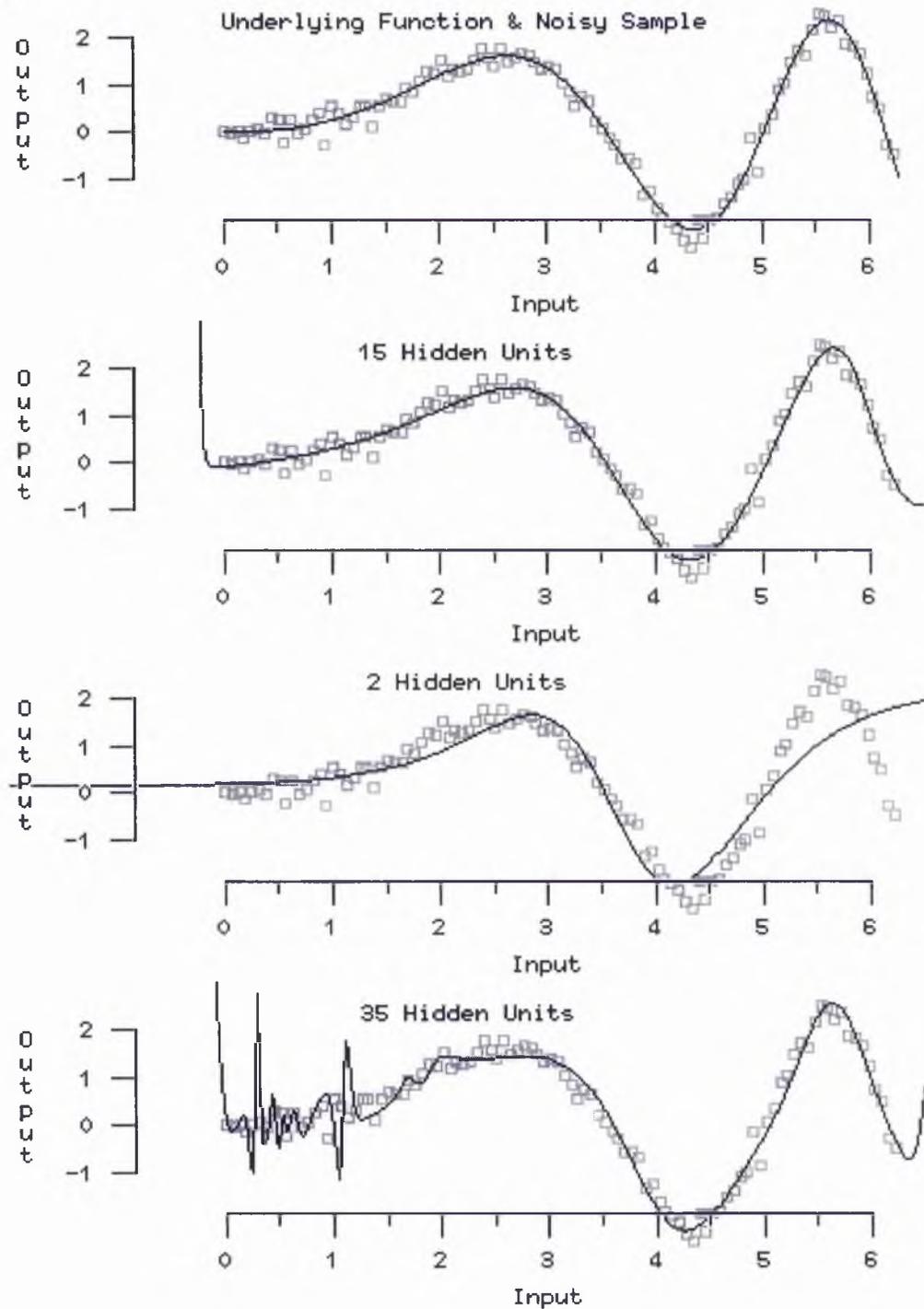


Figure 3.4 — From the top: The underlying function, and fits by 15, 2, and 35 hidden units to a sample which approximates the underlying function. The 2 hidden unit fit under-fits and the 35 hidden unit fit over-fits. The 15 hidden unit fit shows a good fit to the underlying function.

3.2.4 Over-Fit and Under-Fit in the Classification Paradigm

In the classification paradigm, there are only two possible values for an output unit. It is either active or inactive. The issue of over-fitting and under-fitting is slightly different from that of function approximation. Rather than relating to the number of peaks and troughs in the fitting function — which is equivalent to the value of the output of the neural network, the over-fit/under-fit dilemma may be seen as pertaining to the degree of complexity in the boundary found between inputs of different class. This would be measured relative to the simplest separation, given the data.

Consider a two-class scenario, for the sake of simplicity. There are two input axes, x and y . A decision boundary must be found which separates the black patterns from the white patterns, where black and white are the two classes. There are a set of patterns which belong to the training set, and a set of patterns which will be used to measure generalisation performance once training is finished.

Figure 3.5 shows some classification boundaries, with (a) representing a good boundary, which correctly classifies all the generalisation performance patterns. An over-fitting boundary is shown in (b) and an under-fitting boundary in (c). Neither of these correctly classify all the generalisation performance patterns.

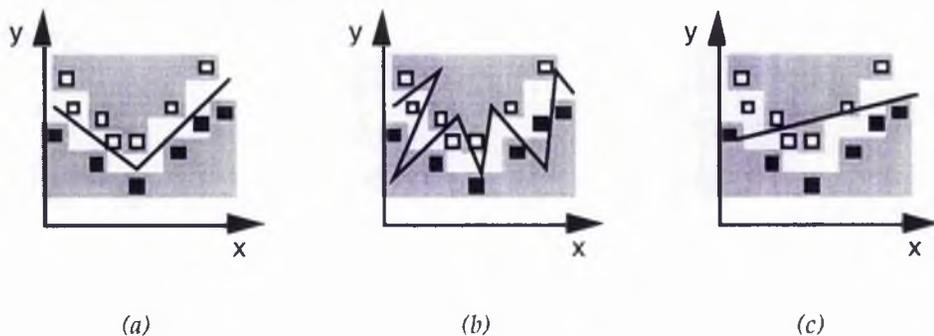


Figure 3.5 — A set of patterns are set aside for training (indicated by the patterns in the shaded areas). Another set of patterns will be used to measure the generalisation performance. (a) A good separation, which correctly classifies all the generalisation performance patterns (b) An over-fitting separation. (c) An under-fitting separation.

The same issues with noise apply in the classification paradigm as they do with function approximation. If the data are noisy, then there are patterns in the training set which are given the wrong class for the position of input space they occupy. If the degree of noise in the training set is underestimated and an appropriate topology chosen, the result may be an over-fitting class boundary. If the degree of noise is over-estimated, the topology chosen will yield an under-fitting class boundary. Both will result in poor generalisation ability. Chapter 4 has more detail on the complexity of decision region possible with a given topology.

3.3 Generalisation in Neural Networks in the Literature

The validation technique is a practical approach to training neural networks with a view to generalisation. This is discussed in section 3.3.1. A theory for calculating the average generalisation ability¹⁴ provides an alternative framework, which enables the prediction of the number of patterns that should be used to attain a given average generalisation performance. An overview of this is given in section 3.3.2.

As the relationship of neural networks to concepts in statistics and probability have been uncovered, deeper theoretical attitudes to generalisation have been forthcoming, and some techniques to go with them. A mathematical theory of Vapnik and Chervonenkis¹⁵ (abbreviated to VC hereafter) has been applied to provide the upper bounds on the number of patterns needed to be sure of good generalisation for a given neural architecture. This is discussed in section 3.3.3. The Bayesian view, outlined in section 3.3.4, is aimed at maximising the probability of a correct prediction for unseen points.

3.3.1 The Validation Technique

The validation technique is the most practical of the generalisation theories discussed in section 3.3. It is also the standard technique used in the neural community, and is based on the method of cross-validation in

¹⁴Schwarz et al, 1990

¹⁵Vapnik & Chervonenkis, 1971

statistics. There are many implementable variants along a basic theme. The data are divided into two sets, a training set, and a validation set. Only the training set is used to determine the values of the weights. The validation set is used, during training, to measure the generalisation ability of the network, and to decide when to stop training. The algorithm proceeds as follows:

[The validation technique] employs a network with an excessive number of free parameters and stops training before the network reaches overlearning on the training set. ... Training is stopped when the performance on the validation set ceases to improve.¹⁶

Weigend et al¹⁷ actually suggest dividing the data into three parts, and using a prediction set (not used for training or for assessing when to stop training) which gives a measure of the expected future performance.

Over-fit and under-fit relate to training time for back-propagation. The longer a network is trained, the more peaks and troughs in the function implemented by the network. This is because back-propagation uses low random initial weights, which leads to low excitation values for each unit. This yields a roughly uniform initial function. As training proceeds, the weights increase in magnitude, and the excitation values increase. This gives rise to functions with more peaks and troughs.

Using too many hidden units during training ensures that there can be no under-fit in the final solution, since there will definitely be more than enough hidden units to realise the underlying function. Initially, the validation error will be high, as the network under-fits the data. As training progresses, and the under-fit is reduced, the validation error goes down. As the network is trained beyond the point of under-fitting the data, the validation error increases, as there are more incorrect outputs given for the members of the validation set. The network is now beginning to over-fit the data. This is a good time to stop training, since there is likely to be a good compromise between over-fit and under-fit.

¹⁶Hasegawa et al, 1992, p. 2459

¹⁷Weigend et al, 1991a, p. 108

Validation need not be restricted to an equilibration of over-fit and under-fit with over-sized topologies, however. With topologies with too few units to exactly fit the data, a minimum of validation error can be used to indicate when there is a balance in the errors of the validation set and the training set. This balance at the minimum of validation error means that further training, although it leads to an improved training error, will result in poorer performance on the validation set.

Training too far with any topology, over- or under-sized, will lead to an excessive fit to the training set, and hence an imbalance between the fits to the training set and the validation set. This is undesirable, since it is likely to lead to poor generalisation. Lang et al give the following report of training too far with their neural network:

Peak generalisation occurred after ... 10 000 epochs, at which point the network got 95.4% of the training cases, and 91.4% of the testing cases correct. During an additional 10 000 epochs of training, the network's performance increased to 98.1% on the training set, but generalisation fell to 88.1%.¹⁸

Figure 3.6 shows the effect on the IO of overtraining, for a simple problem (a linearly separable set) whose targets have been corrupted by noise. An overlarge topology has been used for the simple problem, and if training continues sufficiently beyond the first minimum of the validation error, then the network begins to fit the noise.

¹⁸Lang et al, 1990, pp. 37-38

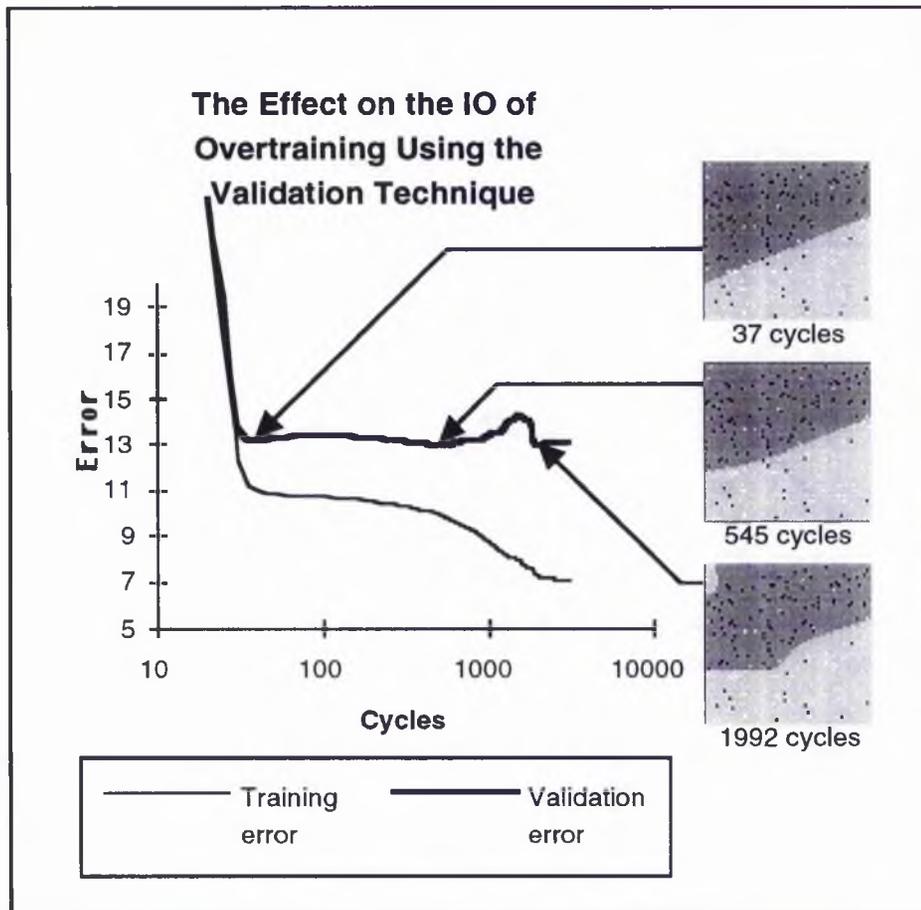


Figure 3.6 — *The change in the black and white picture produced by an overlarge topology for a noisy pattern set, over a period of 3 000 cycles. The topology had a single hidden layer of 5 units. Light grey indicates an output of 0 by the network, and dark grey indicates an output of 1. The pattern set is taken from a linearly separable pattern set of two hundred patterns, with 25% noise on the targets. The patterns are indicated by black and white dots on the IO picture. A black dot indicates a target of 1, and a white dot a target of 0. (The patterns are shown in more detail in figure 3.7.)*

The IO graph fits the noise more and more closely as training proceeds beyond the first minimum of validation error. At 37 cycles, the first minimum of validation error, the network realises a linear separation close to the underlying function. By 545 cycles, the separation is distorted slightly as it begins to fit the noise in the training set. By 1 992 cycles, the separation is severely distorted.

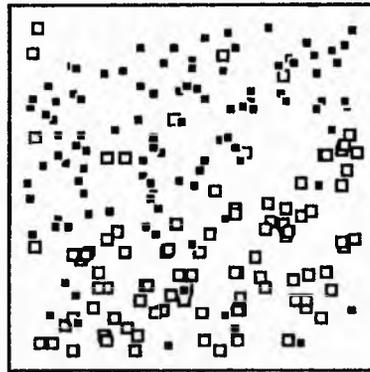


Figure 3.7 — Close up of the pattern set used in figure 3.6. Solid black squares indicate a target of 1, and the unfilled squares indicate a target of 0.

One problem with the technique is that the minimum of validation error stopped at need not be the best minimum in all cases. The original desired decision region in the above problem need not have been a linear partition, but the decision region shown at 1 992 cycles in figure 3.6. The same data set is used as for the simple problem, but now it is assumed that there is less noise. The same topology and initial weight state could also be used for this problem. However, the desired decision region, given these conditions, does not occur until the third minimum of validation error, which has lower training error relative to the first minimum of validation error. The decision to be content with a given minimum represents an assumption about the noise in the data.

Hence, the choice of which minimum to stop at is a bias. (See section 3.2.2, earlier.) This relates to the problem with validation discussed by Denker et al. Denker et al refer to validation as “rule extraction”¹⁹ (and do not assert any requirement for an over-sized topology). Given a training set, M , and a validation set, X , both of which are assumed to be representative samples of the rule to be extracted, they give a theoretical basis for expecting validation to work as follows:

The idea is to extract the rule from ... M , and extend it to ... X .²⁰

¹⁹Denker et al, 1987, pp.897-901

²⁰Denker et al, 1987, p. 897

Defining the extraction score as the accuracy with which the network predicts the validation set, Denker et al then go on to acknowledge a problem with the validation technique:

We emphasise that rule extraction is rather a slippery concept, since it is possible to change a network's extraction score (without changing the network) simply by changing one's mind about what rule was "supposed" to be extracted.²¹

This is a rather weak criticism of the technique, however. The assumption of representativeness must be broken if there is any major change of mind about the rule to be extracted. However, when the data are noisy, which must be taken to be the norm in real-world problems, the assumption of representativeness becomes an assumption of the *degree* of representativeness of the data. For the same data, differing assumptions of the amount of noise may yield different desired fits to the data, as is illustrated by the example in figure 3.6.

The validation technique makes no assumptions about the underlying function, other than that the data in the training set and the validation set are representative. It provides a means of deciding when to terminate training, at which point there is a balance between the memorisation of the training set, and the generalisation ability.

The validation technique provides no guarantees about the expected degree of fit beyond the available data. Even using a prediction set as per Weigend et al gives information about the available data only. The degree of expected fit to new data may only be based on the assumption that the data used for training and validation was representative. If this assumption is valid, then the technique should give satisfactory generalisation performance, but only in so far as the training regime produces the best fit to the training and validation data.

²¹Denker et al, 1987, p. 898

3.3.2 Average Generalisation Error

Schwarz et al²² provide a formalism for computing the distribution of generalisation abilities given the number of patterns, and a prior distribution of generalisation abilities which embodies some kind of prior knowledge about the problem. The formalism is also discussed in Hertz et al,²³ and the results are summarised in the following:

Let V_0 be the volume of weight space. This could be finite if weight values were only considered within a certain range. Hertz et al suggest a range of $[-10, 10]$.²⁴ Let $V_0(f)$ be the volume of weight space (in the same region of weight space) that implements a given function f . (There might be more than one weight state that implements a given function due to symmetries in weight space. Chapter 5 has more on these. There is also the possibility that the functions implemented by weight states in a given neighbourhood do not differ significantly.)

Let f_d be the target, or desired function. Let X be the set of all possible inputs, \mathbf{x} . A given training set, T , takes random members \mathbf{x}_i of X as input — the targets for each given by $f_d(\mathbf{x}_i)$. Let $E(f, \mathbf{x})$ be the following function:

$$E(f, \mathbf{x}) = \begin{cases} 1 & \text{if } f(\mathbf{x}) = f_d(\mathbf{x}) \\ 0 & \text{otherwise} \end{cases} \quad [3.1]$$

Let $g(f)$ be the probability that f agrees with f_d on any randomly chosen input — the generalisation ability of f . This is the mean value of $E(f, \mathbf{x})$ for all members of X :

$$g(f) = \langle E(f, \mathbf{x}) \rangle \quad [3.2]$$

Let $\rho_0(g)$ be the prior distribution of generalisation abilities, g . This is the distribution of the fraction of weight space under consideration that has generalisation ability g :

²²Schwarz et al, 1990

²³Hertz et al, 1991, pp. 148-153

²⁴Hertz et al, 1991, p. 148

$$\rho_0(g) = \frac{\sum_f V_0(f) \delta(g - g(f))}{V_0} \quad [3.3]$$

where $\delta(x)$ is the Kronecker delta:

$$\delta(x) = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{otherwise} \end{cases}$$

Consider p patterns from T . Let $V_p(f)$ be the volume of weight space consistent with a function f , and the p training examples, \mathbf{x}_i from T :

$$V_p(f) = V_0(f) \prod_{i=1, p} E(f, \mathbf{x}_i) \quad [3.4]$$

The product is 1 if the function is consistent with the training examples, and 0 otherwise. This represents the elimination of a function from the space of possible functions, if that function misclassifies a pattern. $V_p(f)$ may be estimated using $g(f)$ to give:

$$\hat{V}_p(f) = V_0(f) [g(f)]^p \quad [3.5]$$

The distribution of generalisation abilities after p patterns, $\rho_p(g)$ is then as per [3.6] below, where V_p is the volume of weight space consistent with p patterns (or the size of version space):

$$\rho_p(g) = \frac{\sum_f V_p(f) \delta(g - g(f))}{V_p} \quad [3.6]$$

Using the estimate of $V_p(f)$ in [3.5] above, $\rho_p(g)$ may also be estimated by substituting for $V_p(f)$ in [3.6]:

$$\begin{aligned} \rho_p(g) &\propto \sum_f V_p(f) \delta(g - g(f)) \\ &= \sum_f V_0(f) [g(f)]^p \delta(g - g(f)) \end{aligned} \quad [3.7]$$

$[g(f)]^p$ can be taken outside of the sum as g^p since if $g(f) \neq g$ the Kronecker delta evaluates to zero. Thus, for the sum over all f , $[g(f)]^p$ is effectively constant and equal to g^p . Therefore:

$$\sum_f V_0(f) [g(f)]^p \delta(g - g(f)) = g^p \sum_f V_0(f) \delta(g - g(f)) \propto g^p \rho_0(g) \quad [3.8]$$

Hence the estimate of $\rho_p(g)$ can be written in terms of the prior distribution, $\rho_0(g)$:

$$\hat{\rho}_p(g) = \frac{g^p \rho_0(g)}{\int_0^1 g'^p \rho_0(g') dg'} \quad [3.9]$$

where the integral is used to normalise the distribution.

The average generalisation ability after p patterns, $G(p)$, is the mean of the distribution of generalisation abilities:

$$G(p) = \int_0^1 g \rho_p(g) dg \quad [3.10]$$

This can be estimated as well, and hence expressed solely in terms of the prior distribution of generalisation abilities, by substituting for the estimate of $\rho_p(g)$:

$$\hat{G}(p) = \frac{\int_0^1 g^{p+1} \rho_0(g) dg}{\int_0^1 g'^p \rho_0(g') dg'} \quad [3.11]$$

This can be used to estimate the number of patterns needed to achieve a good average expected generalisation ability, given knowledge of the prior distribution of generalisation abilities. To calculate the prior distribution from [3.3] requires knowledge of the underlying function, however, which is not always possible. If the underlying function is not known, the prior distribution must be estimated:

The prior distribution of generalisation abilities $\rho_0(g)$ is computed by testing all networks ... on a randomly chosen set of ... examples, large enough to obtain the intrinsic generalisation ability of each network with a precision of at least 6%.²⁵

This kind of estimation requires sufficient data sample sizes to be sure of the precision of the estimation. If the amount of available data is limited, this may not be possible.

²⁵Schwarz et al, 1990, p. 380

Even if the underlying function is known, it is still necessary to exhaustively consider all the possible weight states. Severe restrictions on allowable weight values are required if the prior distribution is to be feasibly calculated. Schwarz et al use weight values of ± 1 .²⁶

For example, consider a simple topology with a single input unit and a single output unit. There are two weights: one weight, w , from the input unit to the output unit, and a bias weight, b , to the output unit. This topology separates the one dimensional input space into two halves, at the point $x = -b/w$. Let the weights, b and w be any non-zero integer between -10 and 10 inclusive. Let the input be any real number in the same range. The desired output is zero for all inputs between -10 and 3 inclusive, and 1 for all other inputs in the given range. Figure 3.8 shows the problem, and indicates how the generalisation ability is calculated. For any given weight state $f(w, b)$ with each weight taken from the specified set of values, the generalisation ability $g(f)$ of the weight state for this problem is given by:

$$g(f) = \begin{cases} 1 - \frac{|x-3|}{20} & w > 0 \\ \frac{|x-3|}{20} & w < 0 \end{cases} \quad \text{where } x = -\frac{b}{w} \quad [3.12]$$

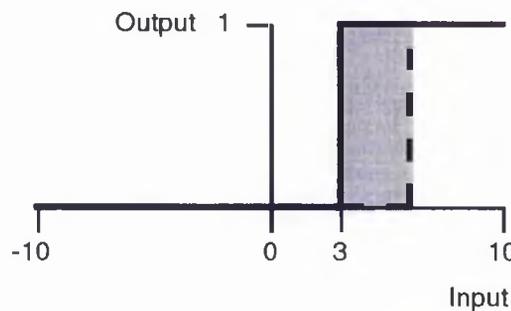


Figure 3.8 — The simple problem is illustrated by the solid line. The output is zero from -10 to 3 , and rises to 1 thereafter. A candidate solution, which is indicated by a dashed line, rises to 1 from 0 at a different point. The generalisation error is then represented by the shaded area. The generalisation ability of the candidate solution is then represented by the rest of input space between -10 and 10 .

²⁶Schwarz et al, 1990, p. 379

The prior distribution of generalisation abilities can then be calculated for this problem by considering each weight state, and assuming it represents a unit volume of weight space. The expected average generalisation ability after p patterns can then be calculated using [3.11]. Figure 3.9 shows how the average generalisation ability increases as the number of patterns is increased for this problem. Also shown is the decrease in the size of version space with increasing patterns.

There are strong links with Mitchell in this technique. It shows how, as functions are eliminated from the space of all possible functions under the weight of increasing numbers of patterns (see equations [3.4] and [3.5]), the average expected generalisation error increases (equations [3.10] and [3.11]). This is akin to the concept of version space shrinking, leaving fewer and fewer candidate concepts under consideration. The remaining concepts have a greater degree of agreement with the target concept, and hence the average generalisation ability of the remaining concepts is increased.

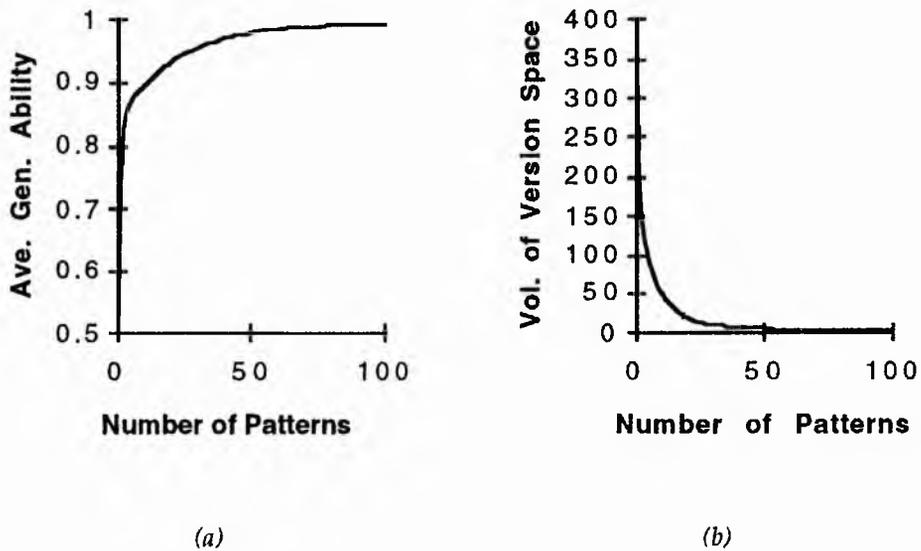


Figure 3.9 — (a) The effect on the average generalisation ability of increasing the number of patterns. (b) The corresponding effect on the size of version space.

3.3.3 Vapnik Chervonenkis Theory

The relationship of VC theory to neural networks is discussed in several papers.²⁷ The main concern of VC theory in neural networks is to provide a bound on the discrepancy between the estimated generalisation ability, $g_t(f)$ (equation [3.13]), based on a particular training sequence of p patterns, and the actual generalisation ability, $g(f)$ from equation [3.2].

$$g_t(f) = \frac{\sum_{i=1,p} E(f, \mathbf{x}_i)}{p} \quad [3.13]$$

where $E(f, \mathbf{x})$ is defined in equation [3.1].

The main result gives an upper bound on the probability that the function with the maximum discrepancy between $g_t(f)$ and $g(f)$ has discrepancy greater than ϵ , after p patterns have been trained:

$$P\left(\max_f |g_t(f) - g(f)| > \epsilon\right) \leq 4m(2p)e^{-\frac{\epsilon^2 p}{4}} \quad [3.14]^{28}$$

where $m(x)$ is a function which gives the maximum number of different target sets that are realisable by the neural topology being used in this case, over all sets of x input vectors taken from the set of allowable inputs, X , where each target may be chosen from two possibilities. The function is 2^x until x equals the VC dimension, d_{VC} of the topology. This is because when $x \leq d_{VC}$ all possible targets for x patterns are realisable using the topology. Thereafter, certain possibilities for the targets are unrealisable. Hence the VC dimension for a topology, A , may be defined as one less than the number of patterns needed before it is possible to have an unrealisable target set for A .

For example, consider a simple topology with two input units, one output unit, and a bias weight to the output unit. Inputs are taken from \mathbf{R}^2 . This topology allows weight states which are capable of an arbitrary linear

²⁷Introductory: Abu-Mostafa, 1989; Hertz et al, 1991, pp. 153-156; Watkin et al, 1993. More detailed work on VC theory is to be found in Baum & Haussler, 1989; Haussler et al, 1994; Parrondo & Van den Broeck, 1993.

²⁸Hertz et al, 1991, p. 154

separation of input space. Figures 3.10(a)-(c) show that it can separate any combination of unlike targets for 3 patterns in general position. (The case of all patterns with the same target is also separable.) Figure 3.10(d) shows an inseparable target set for 4 patterns. Hence the VC dimension of the topology, with inputs from \mathbb{R}^2 , is 3.

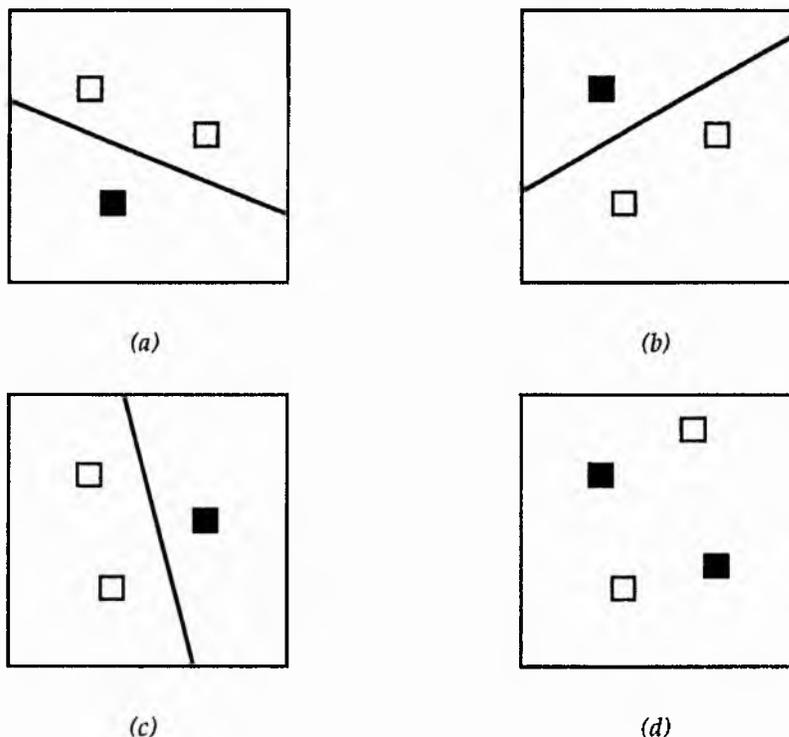


Figure 3.10 — (a)-(c) Show linear separability of any combination of unlike targets for three patterns in the same general position. (d) Shows inseparability of four patterns for a certain target set. Hence the VC dimension of a perceptron with 2 inputs and 1 output is 3.

In general, for any topology with n inputs, one output with a threshold activation function and a bias weight to the output, with inputs taken from \mathbb{R}^n , the VC dimension of the topology is equal to $(n + 1)$.²⁹

For topologies with hidden units, the VC dimension has not been so precisely formulated. Baum and Haussler give a lower bound for a feed-forward network with one hidden layer of h threshold units, taking input

²⁹Haussler et al, 1994, p. 99

from \mathbf{R}^n (from which n input units may be inferred), and with output $\in \{-1, +1\}$:

$$d_{VC}(n * h * 1) \geq 2 \left\lfloor \frac{h}{2} \right\rfloor n \quad [3.15]^{30}$$

where $\lfloor x \rfloor$ is the largest integer not greater than x , and $n * h * 1$ indicates a completely connected feed-forward topology with n input units, h hidden units and 1 output unit.

The value of $d_{VC}(n * h * 1)$ is approximately equal to the total number of weights in the network, W , for large n and h . A later paper by Baum shows experimental results which indicate that the VC dimension is approximately equal to W for networks with two hidden layers of threshold units.³¹

There is also the need to estimate the function $m(x)$. Where the VC dimension is finite, there is an upper bound on $m(x)$:

$$m(x) \leq x^{d_{VC}} + 1 \quad [3.16]^{32}$$

With a topology consisting of one input unit, and one output unit, it is possible to give a precise formula for $m(x)$. The topology is capable of a single separation of a one dimensional input space. For any number of patterns, p , there are therefore $2p$ different sets of outputs that can be assigned to the patterns. (For example, see figure 3.11.)

Using this very simple topology, for which $m(x) = 2x$, it is possible to give a precise indication of the number of patterns recommended for a given probability bound and discrepancy. This is shown in figure 3.12. It is clear from this graph that the discrepancy has a far greater effect on the number of patterns than does the probability bound. It is also clear that the number of patterns recommended is rather high. Roughly ten thousand patterns are required to ensure that the maximum possible discrepancy

³⁰Baum & Haussler, 1989, p. 157

³¹Baum, 1990, p. 2

³²Hertz et al, 1991, p. 154

between the estimated and actual generalisation error is greater than 0.1 with probability less than or equal to 0.1.

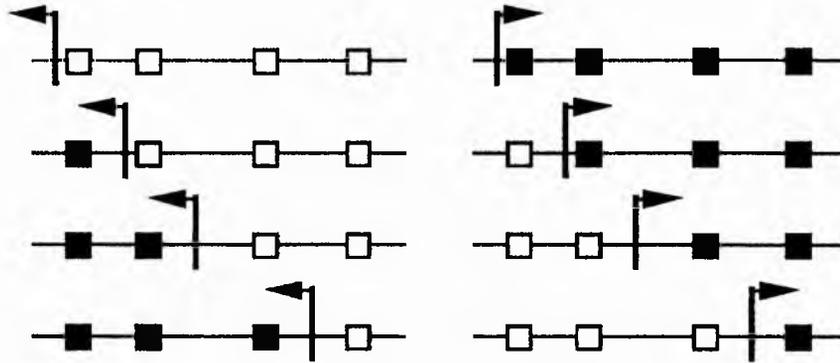


Figure 3.11 — *The eight possible sets of outputs for four patterns in a one dimensional input space being separated by a single hyperplane, which is indicated by the bold line. The arrow on the hyperplane indicates on which side of the hyperplane the output unit is 1. Black squares indicate an output of 1, white squares an output of 0.*

A further point is that the formula in [3.14] does not guarantee values in the range $[0, 1]$. For example, with a discrepancy of 0.1, 10 patterns, and $m(x) = 2x$, the probability bound in [3.14] is 158.01.

On a slightly different note, consider a situation whereby an exact realisation of a training set is required, and it is necessary to choose a topology which is certain to be capable of the realisation. If there were precise knowledge of the VC dimensions of different topologies, the VC dimension could be helpful in choosing the topology, by indicating the minimum topology needed to realise any combination of targets for that data set. This would provide certainty that the data set could be learned, and would remove the need for a trial and error approach in determining the topology.

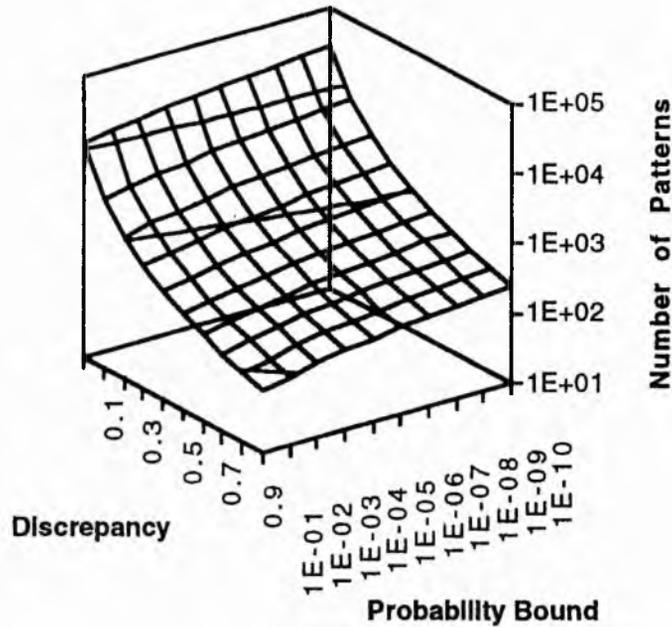


Figure 3.12 — Graph showing the number of patterns needed as the discrepancy, ϵ , and the probability bound are varied. The number of patterns is much more dependent on the discrepancy than the probability bound.

3.3.4 Comparing VC and Average Generalisation Theories

The average generalisation theory of Schwarz et al may be used to indicate the number of patterns needed to acquire a reasonable average generalisation ability. VC theory may be used, as indicated above, to provide the number of patterns needed to bound the discrepancy between the actual and estimated generalisation abilities to a given value, ϵ . However, it may also be used to give ϵ for a given number of patterns and probability bound. Using the number of patterns from the average generalisation ability, the discrepancy for a given probability bound may be calculated. Figure 3.13 shows this, for the problem described in section 3.3.2, which uses the same topology used in section 3.3.3 to give a precise formula for $m(x)$.

The VC formula [3.14], when rearranged in terms of ϵ , for $m(x) = 2x$, may produce values for the discrepancy which are greater than 1. Since the estimated and actual generalisation abilities ([3.13] and [3.2] respectively) are both in the range $[0, 1]$ inclusive, their difference must also be in this range. As section 3.3.3 indicated, the VC formula does not guarantee

values for the probability bound in the range $[0, 1]$ inclusive. Thus, there is no reason to expect the discrepancy to also be in an appropriate range.

The chart in figure 3.13 shows that the discrepancy is rather large, even for extremely good expected average generalisation abilities. (A discrepancy of 1 means either that the estimated generalisation ability is 1, and the actual generalisation ability is 0, or vice versa.) This could bring the usefulness of VC theory into question, particularly for real-world problems, where large numbers of patterns may not be available.

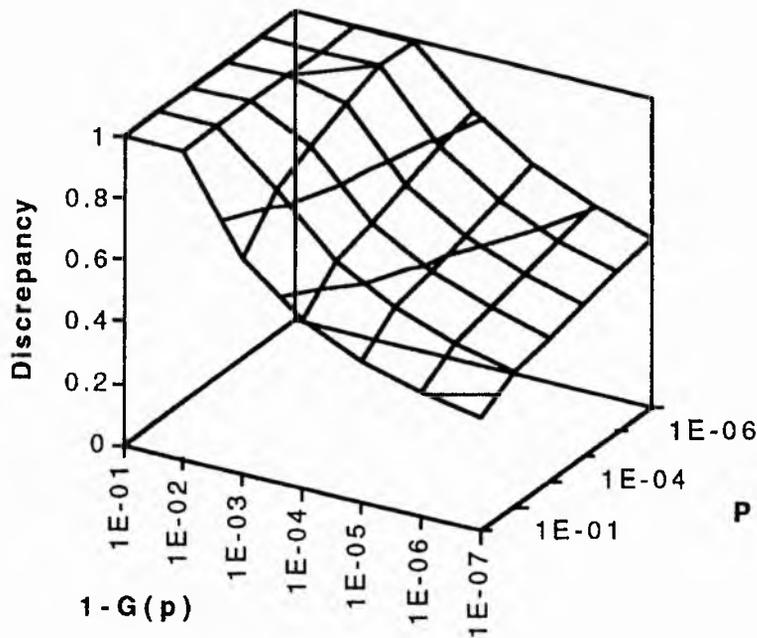


Figure 3.13 — Comparison of VC theory and average generalisation theory. The discrepancy between the estimated and actual generalisation ability is given for the average generalisation ability, $G(p)$, and probability bound, P .

3.3.5 Bayesian Frameworks for Generalisation

The relation of Bayesian probability theory to neural networks is discussed in several papers.³³ The Bayesian approach to generalisation is based on the Bayes optimal classification algorithm.³⁴ Given p inputs whose targets ($\in \{0, 1\}$) are known, this algorithm finds the most probable output of a new input, q . Let V_p be the volume of weight space that correctly classifies the p patterns seen so far. Let P_q^+ be the proportion of V_p that gives an output of 1 to q . Let P_q^- be the proportion of V_p that gives an output of 0 to q . If $P_q^+ \geq P_q^-$ then q is given output 1, otherwise q is given output 0. Evaluating P_q^\pm is equivalent to evaluating the Bayesian posterior probability that q has the given classification.³⁵ Eisenstein and Kanter refer to this as the Bayes₁ algorithm,³⁶ since one new input is tested at a time.

An important disadvantage with this technique, akin to the average generalisation error technique, is that it requires the knowledge of all the weight states that correctly classify the p patterns. Opper and Haussler³⁷ sample V_p by independently training N networks to minimum error from several different random initial weight states. This will result in different final solution trained weight states. A committee machine is then constructed which takes the outputs of the N networks as inputs, and gives output equal to the output given by the majority of the inputs. N must be odd so that there is always a clear majority one way or the other. For sufficiently large N , the output of the committee machine converges to the Bayes estimate. Compared with the validation technique, which only trains one network, Opper and Haussler's algorithm is very costly, particularly for large problems, and for large N .

³³E.g. Kanaya & Miyake, 1991; Levin et al, 1990; MacKay, 1992a-d; Opper & Haussler, 1991; Watkin et al, 1993

³⁴See Opper & Haussler, 1991, p. 2678, and references therein.

³⁵Watkin et al, 1993, p. 510

³⁶Eisenstein & Kanter, 1993. p. 3668

³⁷Opper & Haussler, 1991

In the case of a set, Q , of cardinality $n > 1$ patterns to be generalised to, there is a choice of two possibilities for deciding the outputs for the members of Q . Firstly, one can apply the Bayes₁ algorithm n times. This means selecting the largest proportion of V_p to be the target for the first member of Q , q_1 , then, of that sub-volume, selecting the largest proportion for q_2 , and so on up to q_n . The second possibility is to consider proportions of V_p for each possible set of targets for members of Q , and the largest proportion selected. This is termed the Bayes _{n} algorithm.³⁸ The difference between these algorithms is indicated in figure 3.14.

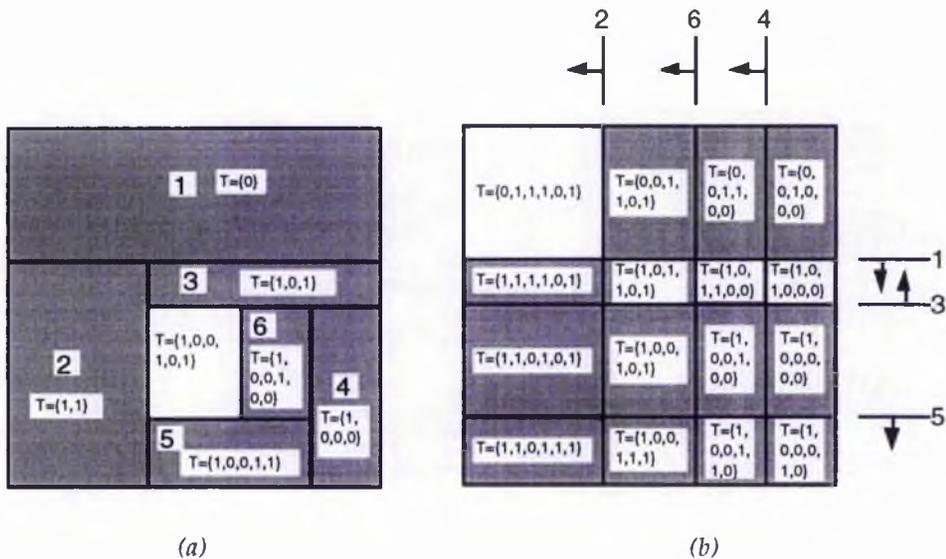


Figure 3.14 — The difference between Bayes₁ (a) and Bayes _{n} (b). In (a) each pattern is assessed sequentially, and the largest portion of the available weight states selected. In (b) all possible target sequences with the current architecture are assessed at once, and the largest area selected. Each pattern has a boundary hyperplane in version space which separates those weight states which classify the pattern as 1, and those which classify it as 0. The arrows indicate on which side of the boundary each pattern is classified as 1.

Eisenstein and Kanter distinguish between three kinds of generalisation in order to show that the Bayes₁ and Bayes _{n} algorithms are not necessarily optimal:

³⁸Eisenstein & Kanter, 1993, p. 3668

- (i) To maximise the probability of correct classification of all members of Q . In this case, the aim is to choose the maximum probability outputs for all members of Q simultaneously. For this, the Bayes _{n} algorithm is optimal.³⁹
- (ii) To maximise the average number of correct classifications of Q . Here, rather than considering all members of Q simultaneously, the aim is to be sure that on average, each member of Q will be correctly classified. For this, applying Bayes₁ n times is optimal.⁴⁰
- (iii) As (ii), but at least m patterns must be correctly classified. Here, neither Bayes _{n} nor Bayes₁ is optimal.⁴¹ This is because neither algorithm can guarantee a given number of correct classifications from Q , since there is no *a priori* basis for selecting among the possibilities according to the proportion of V_p that they occupy. Figure 3.15 shows this. It is hard to imagine that any algorithm could guarantee the correct classification of a given number of patterns, however. Yet Eisenstein and Kanter claim to have an algorithm which is optimal in this case, though their description of it is not straightforward:

The optimal strategy for [case (iii)] is a table which indicates for each type of partition of the VS to make a prediction following a particular part of the VS, among 2^n .⁴² [VS stands for version space, and is equivalent to V_p in the above discussion.]

MacKay has an alternative Bayesian approach which uses a single neural network.⁴³ The general approach is to find the simplest, most probable interpolant of the data.⁴⁴ This embodies Occam's razor, which is a principle stating that, given a choice, simple rules that explain a phenomenon are more likely to be true than complex rules.

³⁹Eisenstein & Kanter, 1993, p. 3669

⁴⁰Eisenstein & Kanter, 1993, p. 3669

⁴¹Eisenstein & Kanter, 1993, p. 3670

⁴²Eisenstein & Kanter, 1993, p. 3670

⁴³MacKay, 1992a-d

⁴⁴MacKay, 1992a, p. 426

In the classification paradigm, MacKay's approach to finding the maximum probability classification is to estimate the probability of each class for a given input, and choose the class with the highest probability.⁴⁵ Each output unit is dedicated to a particular class, and the analogue output of each unit is treated as an estimate of the probability that the current input has that class. This is in line with the view of Richard and Lippmann that neural network classifiers trained using back-propagation are good estimators of Bayesian *a posteriori* probabilities,⁴⁶ though MacKay suggests several refinements to back-propagation.⁴⁷

Bayes(n)	≥4	≥4	≥4
		≥4	≥4
	Bayes(1)		≥4

Figure 3.15 — Regions that have at least 4 correct patterns for a true target sequence of $T = \{0, 0, 1, 0, 0, 0\}$. There is no basis for finding these on the basis of the relative volumes of weight space they occupy, and hence neither the $Bayes_1$ nor the $Bayes_n$ algorithms find an acceptable solution.

MacKay's approach to classification is therefore more in the line of surface fitting than of raw classification, in which only the class boundaries are required. Telfer and Szu point out that more complex topologies are needed to estimate the Bayesian *a posteriori* probabilities than to perform raw classification.⁴⁸

This thesis is focused on estimating the class boundaries, rather than the *a posteriori* probabilities. This is because there is a better understanding of the topology needed to realise a given set of class boundaries than to fit a

⁴⁵MacKay, 1992d

⁴⁶Richard & Lippmann, 1991

⁴⁷MacKay, 1992b

⁴⁸Telfer & Szu, 1994, p. 809

given probability surface. MacKay deals with this by training using various topologies, and then choosing the best among these.⁴⁹

The technique in chapter 6 has the topology as a bias of the user, and it is therefore important that the user has a reasonable idea of the capabilities of a given topology before using the technique. These capabilities are discussed in detail in chapter 4.

Bayesian learning in general represents the embodiment of certain *a priori* ideas of what constitutes a good generalisation. The most probable generalisation, however, cannot be guaranteed to be the best generalisation, even if it is the most likely.

The Bayesian approach is akin to the Mitchellian approach for partially learned concepts. When the Mitchellian version space does not converge to a singleton, there will be some instances for which there is disagreement among the various members of S and G as to whether it does or does not match the target concept. For such an instance, one of Mitchell's strategies is to take the decision of the majority of the members of S and G. The main difference between the Bayesian approach and the Mitchellian approach is that Mitchell uses the boundaries of version space only, whereas the Bayesian approach samples the whole of version space.

⁴⁹MacKay, 1992b, pp. 450-451, 1992c, pp. 470-471

3.4 The Mitchellian View

3.4.1 Generalisation

In Mitchell's symbolic technique, no candidate generalisation is removed from version space until it is inconsistent with the data. Thus no preference is given by Mitchell's technique to one kind of generalisation over any other kind, within the given generalisation language. To Mitchell, generalisation can only be guaranteed when the cardinality of version space is 1.

Any generalisation of data involves making certain assumptions. Mitchell recognised this in the concept of *bias* in learning. To make generalisations requires bias, and without it, learning is simply the process of memorisation of the data in a look-up table form. This is because it is bias that enables the learner to look beyond the training examples to see the underlying pattern behind them, and thus to generalise. The nature of the bias determines the kind of underlying patterns the learner is able to see. Mitchell summarises this as follows:

The ability to make an appropriate "inductive leap" when generalising from a small set of training instances is only possible under *a priori* biases for choosing an appropriate generalisation out of the many possible.⁵⁰

Without the ability to choose one generalisation over the others, learning can only be a rote learning process, whereby training examples are memorised. This is because there is no basis for generalisation. This should not be confused with the memorisation/generalisation dilemma in neural networks. Neural networks generalise naturally. For Mitchell, generalisation requires a generalisation language. Without the generalisation language, there is no basis for generalisation, and hence rote learning is the only possibility. However, with a given generalisation language, certain concepts are representable, while other concepts are not,

⁵⁰Mitchell, 1980, abstract

and this is a bias. Since neural networks generalise naturally, a given neural network has a bias. This will be discussed in the next section.

Since the generalisation language is specified by the user, the bias is made explicit. The assumptions of the user about the nature of the underlying function to be learned are clear from the start, and are not built in to the technique. (Note, however, that the inability to learn disjunctive concepts does represent a built-in bias of the symbolic technique.)

Mitchell's technique presents a realistic attitude to generalisation through the acknowledgement and embodiment in the technique of the following crucial points:

- *Any* generalisation is a valid generalisation. To be sure of reaching the correct generalisation, one should not discard representable candidate generalisations until they become inconsistent with the data.
- The process of generalisation *requires* bias. It is often impossible to explore all the possible generalisations, since there could be an infinity of alternatives. Since all generalisations are legitimate, the bias is needed to indicate which kind of generalisations are preferred. (For example, in the symbolic technique, concepts without disjunction are preferred.) Concept learning then becomes feasible. However, it is important to be sure that the bias does not exclude the target generalisation.

In the design of the technique, Mitchell makes use of a partial ordering of version space to enable the efficient representation of version space using boundary sets. This avoids the need for an exhaustive search of version space.

3.4.2 Relating Mitchell's Technique to Neural Networks

For Mitchell, a concept is a link between instances, which groups certain instances under one category (those which match the concept) and other instances under another category (those which do not match the concept). If instances are seen as stimuli to a system, the only possible responses of the system are "Matches" or "Does not match".

In this thesis, a concept shall be taken to be a body of IO behaviour. This allows for a more active model, with the potential for a richer set of responses than the two possibilities with Mitchell's technique. The concept of danger, for example, may be seen as the linking of certain stimuli to appropriate responses. For instance, given the stimulus of "toadstool", the trained response might be "don't eat". In neural terms, this broadening of the idea of a concept allows for the possibility of many output units. One output unit would suffice to provide a strict link with the symbolic technique. An output of 1 could indicate, "the input matches the concept", and an output of 0, "the input does not match the concept".

Since the weights and topology are what give rise to the IO behaviour in neural networks, concept space can be seen as the weight space for a given topology. The set of weights that correctly classify the patterns presented so far is referred to as the version space for that set of patterns by some authors.⁵¹ Weight space, however, is an indirect way of looking at the IO behaviour. The view of version space in this thesis will be the space of all IO behaviours that are consistent with the data. This space may be limited to the space of all IO behaviours realisable by the given topology. This provides the link with weight space.

Version space in neural networks is much larger in comparison to the symbolic technique, if weight space or IO behaviour is taken to be the analogue. This is because the weights may have any real number as a value, as indicated earlier, whereas the symbolic case has a fixed, finite set of symbols for attribute values in version space. Reaching the no-alternative situation in neural networks, even within a certain degree of accuracy, could take a long time because there are so many possibilities. This is indicated by VC theory.

Instances are simply the training patterns. Training patterns combine input and target vectors for the input and output units of the neural topology. The neural analogue of the instance language is the specification of the sets from which the inputs and targets may be drawn. For example,

⁵¹E.g. Opper & Haussler, 1991, p. 2678; Watkin et al, 1993, p. 504

inputs might be from \mathbf{R}^n , and targets from the set $\{1, 0\}^m$, where n is the number of input units, and m is the number of output units.

The neural analogue of the generalisation language is the topology. This constrains the set of IO behaviours the neural network is capable of realising. The choice of topology represents the bias, in the Mitchellian sense, just as it does in the Geman et al sense, of the generalisation. In Mitchell's technique, the choice of generalisation language is made by the user. To maintain consistency with this, the neural implementations discussed in chapters 5 and 6 also either restrict the choice of bias (in terms of the topology), or leave it to the user. Therefore techniques for automatically determining the topology during training are not considered.

Mitchell's technique uses representatives of the boundaries of version space to mark the shrinking of version space as instances are introduced. In order to establish those boundaries, it is necessary to have an ordering of the concepts. Mitchellian neural partial orderings are hard to find because the general/specific relation is not there. There is no *a priori* way to order IO behaviour. For example, should a concept that has response "eat" to the stimulus "toadstool" come before or after one that has "don't eat", or "run away"? The work of this thesis centres around finding such orderings in order to implement Mitchell's technique in a neural environment.

A further requirement, once there is the possibility for representing the boundaries of version space, is the ability to make changes to those boundaries under pressure from the instances. These are the neural analogues of updating and selection within.

Thus, certain prime directives for a neural implementation of Mitchell's technique may be established. These are given below:

- The implementation must have an ordering of IO behaviour which enables boundary representation of version space, with a many-one correspondence between the IO behaviour and the ordering, and any two weight states deemed to have the same IO behaviour must also have the same value in the ordering. If this directive is not upheld, then two networks with the same IO behaviour might

have different values in the ordering. The no-alternative situation is detected by the networks having the same value in the ordering. Hence, without this directive, it is not possible to guarantee the detection of the no-alternative situation using the ordering.

- There must be mechanisms for updating and selection within.
- Updating must always be by the minimum amount necessary for correct classification, in one direction for the S analogue, and in the opposite direction for the G analogue.

3.5 Conclusion

Mitchell's technique has the advantage that it is not necessary to examine all the possible concepts, through the use of boundary representatives of version space during learning. Exhaustive search techniques, such as that of Schwarz et al, and the sampling technique of Oppen and Hausler, do not have this advantage, and suffer from relatively high computational costs.

The main advantage of the boundary representatives, however, is the ability to recognise the no-alternative situation. This is an important consideration for anyone who is trying to fit some data:

In some cases, we may be interested in global, rather than local questions. Not, "how good is this fit?", but rather, "how sure am I that there is not a *very much better* fit in some other corner of parameter space?"⁵²

The bidirectional convergence of the search enables this valuable property, since if there is no alternative but the current solution, then the fit must be the best possible. Having found and recognised the no-alternative situation also gives a terminating condition. There is no point in training further if it is known that there are no better alternatives. The validation technique also has a terminating criterion, but as indicated in section 3.3.1, there is ambiguity about which minimum of validation error should be

⁵²Press et al, 1988, pp. 517-518

used. Hence, the terminating criterion of the validation technique is not certain to recognise the optimum fit. MacKay's technique — which is also a unidirectional technique — also suffers from this disadvantage.

There is more potential for the Oppen and Haussler technique to indicate the no-alternative situation. If all the samples of version space give the same output for any randomly chosen input, then this might be taken to indicate the convergence of version space. This could, however, be due to poor or insufficient sampling of version space, rather than reaching the no-alternative situation.

The Bayesian literature relates to Mitchell's methods for partially learned version spaces. Selecting the most probable weight state from version space, or the most probable classification given the weight states in version space is a useful method for guessing the generalisation when it is clear that there are several alternatives for a given set of patterns.

VC theory and average generalisation theory provide measures which relate to the likely generalisation ability. The VC theory estimate has been shown in section 3.3.4 to place excessive demands on the number of patterns, through the high discrepancies between estimated and actual generalisation error it estimates even for extremely good average generalisation abilities.

Mitchell's technique is able to offer more than probabilities of generalisation, within a certain set of assumptions. If the no-alternative situation is reached, then Mitchell's technique can offer a guarantee that, given the assumptions, the generalisation is correct. This means that any unsatisfactory generalisation results arise from the assumptions of the user (and the designer of the technique), rather than being due to the probabilities not working in the favour of a good generalisation.

For example, if the average generalisation performance is estimated to be 90% on the basis of a set of patterns, the actual generalisation performance need not, in fact, have this value. The explanation of why the generalisation performance is different from the expected value does not rest on the assumptions of the user, so much as on the particular set of patterns chosen.

The VC and average generalisation theories show that in neural networks, large data sets are needed to constrain the number of alternative IO behaviours to a reasonable quantity — each with an acceptable generalisation performance. The Mitchellian guarantee, which rests on a single alternative, might seem rather a remote possibility. The techniques discussed in chapters 5 and 6 will both show neural implementations of Mitchell's technique which, within certain constraints, aim to achieve the no-alternative situation using a reasonable number of carefully chosen patterns.

4 Issues in Topology Determination

4.1 Introduction

Chapter 3 showed that the choice of topology had a strong effect on the generalisation — in terms of the bias a particular choice of topology has for the solution found. This chapter is included in order to provide guidelines for choosing a topology *a priori* when using the technique outlined in chapter 6, which assumes the topology is given as a Mitchellian bias of the learner.

This chapter will examine the work of various authors on the subject of *a priori* topology determination, looking at each layer in turn. Units with threshold activation functions [4.1] are considered throughout. If the output of a unit is 1, the unit may be said to be *active*, or *on*. Conversely, for an output of 0, the unit may be said to be *inactive*, or *off*.

$$output_k = activation_k = \begin{cases} 1 & \text{if } excitation_k > 0 \\ 0 & \text{otherwise} \end{cases} \quad [4.1]$$

The capabilities of networks with sigmoid units is briefly discussed in section 4.4. Sigmoid units have more complex capabilities than threshold units, so it is a good idea to look at the simpler threshold unit behaviour first. It is assumed that inputs are taken from \mathbf{R}^I , where I is the number of input units. One output unit is assumed for simplicity.

The neural networks that will be discussed here divide input space into continuous regions. Within any given region, the binary output is the same for all points in that region. Neighbouring regions may have the same output, or a different output. A given training point is therefore correctly classified if and only if it lies in a region with the same output as the target of the training point.

The main approach of this chapter is to look at the regions, rather than the patterns. This means the question addressed is more, "What is topology X capable of?" rather than, "What topology is necessary for problem Y ?" The problems encountered in this chapter are described in terms of regions,

rather than in terms of the patterns from a pattern set. However, it is not the norm to train regions with neural networks — training is usually done using a specific set of training patterns. To answer the latter of the two questions, using the work in this chapter, it would be necessary to have some idea of the number of regions that the pattern clusters indicate.

Neural network training algorithms attempt to divide input space up into regions that match with pattern clusters in the training set. How the network divides input space into regions is discussed in section 4.2, and the means by which the regions are assigned their outputs in section 4.3. Hartigan's book¹ contains several clustering algorithms, and finding the number of pattern clusters is also explored by some of the radial basis function literature.² It is not explored further here.

If one could train the regions directly, without using the patterns, then the problem of generalisation would be solved. The output for any given input would be precisely defined. However, this would either require an infinite sample size, or a very precise knowledge of the problem. This may be referred to as *infinite training*. The approach to training with neural networks uses a finite set of patterns to approximate the infinite training case. This may be referred to as *finite training*.

The problem of determining a topology comes down to understanding the capabilities of neural topologies. A pioneering work on this subject is a paper by Lippmann,³ which suggests an interesting theory. Although erroneous in its conclusions, this paper is frequently cited,⁴ and its theory of the capabilities of various topologies is even cited in textbooks.⁵ The theory is summarised simply in that networks with one hidden layer can only realise a single convex decision region. Convex is taken to mean that all points on a line between any two points in a region also belong to that region, and the decision region is the set of all inputs which give rise to an

¹Hartigan, 1975

²E.g. Mel & Omohundro, 1991, p. 761; Musavi et al, 1992, pp. 596-597; Roy et al, 1993

³Lippmann, 1987

⁴E.g. Barron, 1994, p. 33; Schmidt & Davis, 1994, p. 3389; Shonkwiler, 1993, p. 344

⁵Beale & Jackson, 1990, p. 87; Khanna, 1990, p. 71

output of 1. Networks with two hidden layers can realise any combination of convex regions.

Lippmann's reasoning in the case of one hidden layer only is based on the consideration of a very restricted set of weights from the first hidden layer to the output node.⁶ The weights from the hidden units to the output unit are all 1, and the bias weight to the unit is set so that the output unit is only activated when all of the hidden units are activated. This computes a logical AND. For N hidden units, the excitation of the output unit, u_k , is given in equation [4.2]:

$$excitation_k = \sum_{j=1,N} weight_{kj} output_j + bias_k \quad [4.2]$$

The weights considered by Lippmann to the output unit restrict [4.2] to the following equation only:

$$excitation_k = \sum_{j=1,N} output_j + c_k - N \quad [4.3]$$

where $0 < c_k < 1$. The excitation is greater than 0 (and hence the output equal to 1) if and only if $output_j = 1$ for all $j = 1, \dots, N$. The sum is then equal to N , and the lower bound for c_k ensures that the excitation is greater than 0. The upper bound for c_k ensures that the excitation is less than 0 when the sum is less than or equal to $N - 1$.

With these restrictions, Lippmann is correct to say that convex regions only may be realised with a single hidden layer. Take, for example, the three hyperplanes arranged as in figure 4.1. The regions are divided up into patterns of activity from the outputs of the three hidden units which give rise to the hyperplanes. With weights of 1.0 to the output units from the hidden units, and a bias weight of -2.5 , there is only one region which has an output of 1 — the region (if it exists) which contains all three hidden units firing simultaneously, as shown in table 4.1.

However, there is no need to impose these restrictions on the weights. Even keeping the weights to the output unit as 1.0, it is possible to generate a concave decision region with a bias weight of -1.5 . This is

⁶Lippmann, 1987, p. 16

shown in table 4.2. Other authors, including Lippmann himself, have realised this.⁷

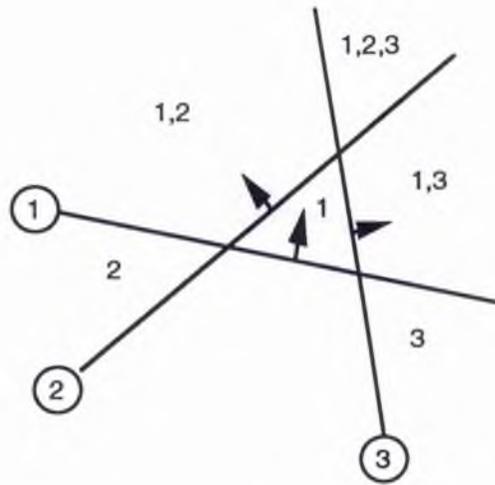


Figure 4.1 — Three hyperplanes. The arrows on each hyperplane show which side of the hyperplane the unit has an output of 1. The numbers in each region show which units are active (i.e. have an output of 1) in that region.

Hyperplane			Activation	Output	Decision Region
①	②	③	= ① + ② + ③ -2.5		
0	0	0	-2.5	0	
0	0	1	-1.5	0	
0	1	0	-1.5	0	
1	0	0	-1.5	0	
1	0	1	-0.5	0	
1	1	0	-0.5	0	
1	1	1	+0.5	1	

Table 4.1 — Calculation of decision region when weights are set in accordance with Lippmann. A convex region is formed.

Lippmann’s more useful result is to show that two hidden layers are sufficient to realise any desired decision region.⁸ This imposes a theoretical limit on the number of layers required in the topology for any

⁷Huang & Lippmann, 1988, p. 388; Li, 1991, p. 509; Lippmann, 1989, p. 50; Makhoul et al, 1989, p. 456; Wieland & Leighton, 1987, p. 387

⁸Lippmann, 1987, p. 16

problem. The first hidden layer, with restricted weights to the next hidden layer, forms the boundaries of the convex polygons which are realised by units in the second hidden layer via a logical AND network. Each output unit then takes a logical OR of the units in the second hidden layer whose combined polygons form the desired decision region of the output unit. These decision regions may be concave, disjoint, or convex. Hence all possible desired decision regions are realisable with two hidden layers. (See figure 4.2.) A logical OR network is constructed with weights to the output unit from the units in the second hidden layer all equal to 1, and the bias weight equal to any real number between 0 and -1 (non-inclusive).

Hyperplane			Activation = ① + ② + ③ -1.5	Output	Decision Region
①	②	③			
0	0	0	-1.5	0	
0	0	1	-0.5	0	
0	1	0	-0.5	0	
1	0	0	-0.5	0	
1	0	1	+0.5	1	
1	1	0	+0.5	1	
1	1	1	+1.5	1	

Table 4.2 — *Simply changing the bias weight from Lippmann's recommendations enables the formation of a concave decision region using one hidden layer only.*

Arguably, therefore, there is never any necessity for a feed-forward neural network with more than two hidden layers for classification, since two hidden layers are sufficient to realise an arbitrary decision region. Sections 4.3.2 and 4.4.3 have some considerations of the possibilities for using further hidden layers, however.

Since Lippmann's theory is incorrect, further analysis is needed. Section 4.2 looks in more detail at the kinds of decision regions that can be formed by networks with one hidden layer, and the real reasons for requiring a second hidden layer.

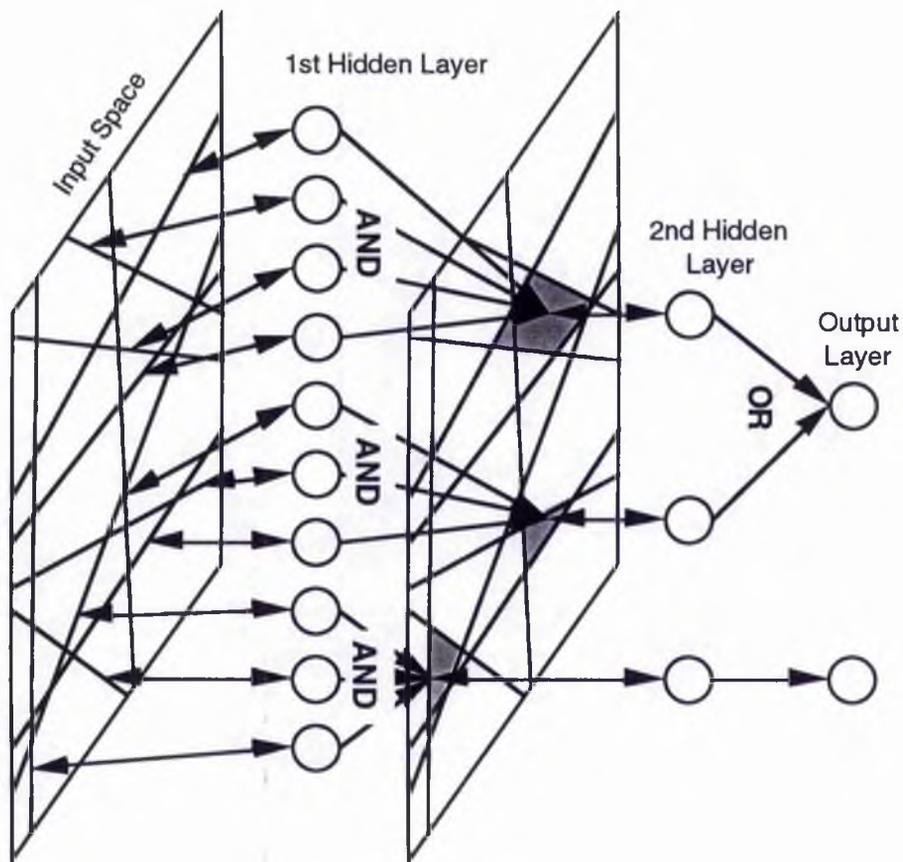


Figure 4.2 — How Lippmann constructs his topology. The hyperplanes of the first hidden layer bound the convex regions. Each unit in the second hidden layer combines the hyperplanes that border the convex region it is to fire for. An AND network is used so that the unit only fires within the region. Each output unit takes an OR of all the regions it is to fire in. In this way, each output unit can respond to any arbitrary combination of convex regions, forming disjoint and concave regions if required.

There is also the question of approximating the desired decision region, rather than exact realisation. Funahashi⁹ and Hornik et al¹⁰ both show that a neural network with one hidden layer can approximate any function (of which classification functions are a subset) to an arbitrary degree of accuracy. Cybenko has also shown this with specific relevance to

⁹Funahashi, 1989

¹⁰Hornik et al, 1989

classification.¹¹ This applies to a network using any activation function, provided that it is non-decreasing, and that the output is 1 at $+\infty$ and 0 at $-\infty$. The sigmoid and threshold activation functions are two such functions. Hornik et al use their result to conclude that failure to provide the desired degree of accuracy with a given neural network has three possible causes:¹²

- The training algorithm fails to find the optimum weight state.
- There are not enough hidden units.
- The relationship between input and output is non-deterministic.

Sections 4.2.2.1 and 4.2.2.2 support this work empirically for 1D and 2D inputs and classification problems. However, neural solutions which approximate decision regions using only one hidden layer may require excessive numbers of hidden units.¹³ Section 4.2.3 explores this issue in more detail.

These possible inhibitory factors in adhering to single hidden layer topologies necessitate a study of the second hidden layer. It will be shown that this layer enables an exact realisation without excessive numbers of units. This is covered in section 4.3.

Section 4.4 digresses from the discussion of threshold units only to explore the use of sigmoid units, and the increased capabilities thereof.

¹¹Cybenko, 1989

¹²Hornik et al, 1989, p. 363

¹³Cheng & Titterington, 1994, p. 19

4.2 The First Hidden Layer

The first hidden layer divides input space up into regions. Each region has a unique pattern of activity from the outputs of the units in the first hidden layer. 4.2.1 gives an explanation of how this happens, and indicates the circumstances under which a second hidden layer is necessary. Section 4.2.2 shows that in these cases, an arbitrarily accurate approximation can be achieved with a single hidden layer. Section 4.2.3 looks at the implications of this for deciding the topology.

4.2.1 Regions and the Requirement for a Second Hidden Layer

Each unit in the first hidden layer bisects input space into two halves. One half where the unit is on (i.e. has output 1), and the other where the unit is off (i.e. has output 0). The linear boundary in input space between the unit being on and the unit being off is termed the hyperplane of the unit. For all points on the hyperplane, the excitation of the unit is zero.

The position of the hyperplane, and on which side of the hyperplane the unit is active, is determined by the ratio and the sign of the weights. For example, in 2D input space (with axes x and y), a desired line of division of input space might be $y = -2x + 3$. If w_x is the weight of the unit to input x , w_y is the weight of the unit to input y , and w_b is the bias weight to the unit, the line of zero excitation is given by $w_x \cdot x + w_y \cdot y + w_b = 0$. There are a family of weights that implement the desired line of division, whereby $w_x = 2w_y$ and $w_b = -3w_y$. If the unit is to be active when $y > -2x + 3$, then w_y should have a positive value, and for $y < -2x + 3$, w_y should have a negative value.

When there are many units, each of which gives a different bisection of input space, there are many regions with linear edges. A region is bounded on its edges, by some or all of the hyperplanes. Each region is uniquely identified by the activity of the units in the first hidden layer. This is illustrated for a simple example in figure 4.1.

From figure 4.1, notice that there are seven regions formed. With three units, however, there are eight possible patterns of activity from the units

in the first hidden layer. Clearly not all of these eight possibilities are realisable. The unrealisable patterns of activity are termed *virtual cells* by Makhoul et al.¹⁴ Mirchandani and Cao,¹⁵ and Makhoul et al¹⁶ both give the formula for the maximum number of regions, R_{max} , that can be formed by N hyperplanes in an input space of d dimensions:

$$R_{max} = \begin{cases} 2^N & N \leq d \\ \sum_{j=0, d}^N C_j & N > d \end{cases} \quad [4.4]$$

where

$${}^n C_r = \frac{n!}{r!(n-r)!}$$

This means that given a certain desired number of regions, R , a lower bound for the number of hyperplanes required to implement R regions can be provided using [4.4] so that $R_{max} \geq R$. This is a lower bound — the given number of hyperplanes need not realise the maximum number of regions. Figure 4.3 gives an example of this.



Figure 4.3 — (a) Three hyperplanes arranged to realise the maximum 7 regions. (b) Three hyperplanes which realise only 6 regions.

The requirement for a second hidden layer for a given number of first hidden units is indicated by a linearly inseparable problem from the outputs of the first hidden layer to the targets for the regions they delineate. The simplest example of this, for 2D input is the “four-quadrant

¹⁴Makhoul et al, 1989, pp. 458-459

¹⁵Mirchandani & Cao, 1989, p. 661

¹⁶Makhoul et al, 1989, p. 456

dichotomy in $2D$ ¹⁷, or the “real-valued XOR”¹⁸, which will be called the *chequer-board problem*¹⁹ in this thesis, and is illustrated in figure 4.4. It is an infinite training problem, in which input space is divided into four portions. No two adjacent portions may have equal output. Four regions may be formed using two hidden units in a single hidden layer, whose targets for the chequer-board problem require the solution of XOR from the outputs of the hidden layer. The XOR problem is not linearly separable (by an output unit), and hence, a second hidden layer is required for an exact realisation.

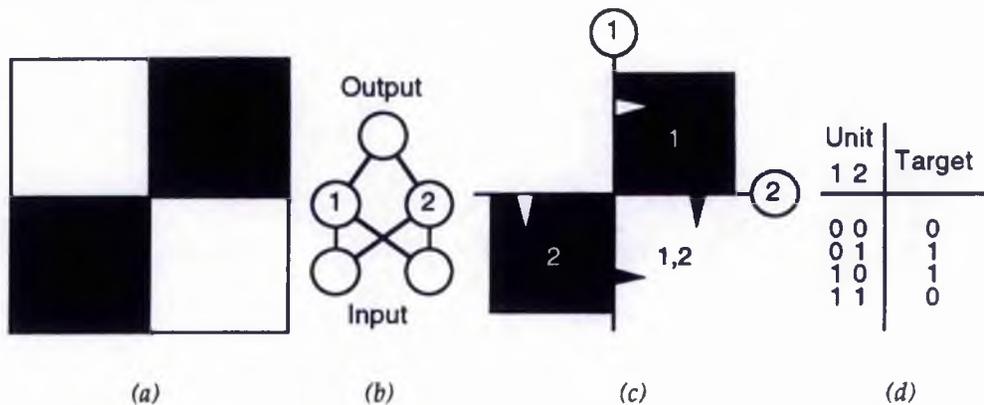


Figure 4.4 — (a) *The chequer-board problem. The problem is, in fact, unbounded, despite the square drawn round it.* (b) *A topology for attempting to solve the problem in (a).* (c) *Placing the hyperplanes on the boundaries of the regions.* (d) *The output unit cannot meet the requirements for the targets of the regions, since XOR must be solved, and hence this topology cannot realise the chequer-board problem.*

The case of the chequer-board might seem to be counter to the theory of authors such as Hornik et al, that only one hidden layer is necessary to realise a given problem. There is a deterministic relation between input and output, and one might think that there are enough hidden units to realise the targets of the regions.

¹⁷Cosnard et al, 1993, p. 2293

¹⁸Makhoul et al, 1989, p. 459

¹⁹Weir, 1993

This is an example of what may be termed *partitioning*, contrasted with *separation*. Here, partitioning is taken to mean that the hyperplanes have been placed along all borders between regions of opposite class. Separation implies that the output unit assigns the correct class to each region. This nomenclature applies henceforth. Two hidden units in a single hidden layer are sufficient to partition the chequer-board, but are not sufficient to separate it, because of the XOR problem at the first hidden layer. A second hidden layer is required for separation. In general, a single hidden layer may be sufficient for separation, though not always.

However, the chequer-board can be approximated, to an arbitrary degree of accuracy, without using a further hidden layer. This will be illustrated in section 4.2.2.2. It remains to be shown that no single hidden layer topology can exactly realise the chequer-board.

4.2.2 Overcoming the Requirement for a Second Hidden Layer

As indicated earlier, the work of Hornik et al claims that failure to realise the targets with a one hidden layer topology for a certain degree of accuracy during training is attributable to a lack of hidden units, given that the function is deterministic, and that the training algorithm returns a reasonable network. Topologies which partition the regions do not necessarily separate them, as shown by the example of the chequer-board attempted by a $2 \times 2 \times 1$ topology. One way to achieve a separation is to add a second hidden layer, which is discussed in section 4.3. In this section the scope for approximating the desired decision regions with a single hidden layer is discussed, for those problems in which a partitioning single hidden layer cannot also separate the desired regions.

The sections that follow go through the dimensions of input space that have been studied in the literature. An indication is given, where appropriate, of those problems which cannot be separated by a single hidden layer which only performs a partitioning.

4.2.2.1 1D Input Space

When input space is one dimensional, there is no requirement for a second hidden layer, since any set of 1D regions can be separated using a single

hidden layer. There is therefore no need to approximate the desired decision regions, since they can be exactly realised.

The argument for this uses the VC dimension. Consider a multi-layer perceptron with N hidden units in a single hidden layer, one input unit and one output unit. There are p patterns whose targets are to be realised. One hyperplane is placed between each pair of adjacent patterns with opposite target in input space. For any p patterns, $N = p - 1$ hidden units are needed to realise their targets in the worst case, in which no two adjacent patterns has the same target. The worst case will be assumed.

Now consider the VC dimension of a perceptron with N input units and one output unit. This is the top half of the multi-layer perceptron considered previously, and is represented by the shaded part of the topology shown in figure 4.5. The VC dimension of a perceptron with N inputs is $N + 1$. (See chapter 3.) This means that the perceptron can give the correct output for any targets of up to $N + 1$ patterns, inclusive. Thus, with a $1 \times N \times 1$ topology, in which each pattern generates a unique set of outputs from the N hidden units, the p patterns as 1D input to the $1 \times N \times 1$ topology may be seen as p patterns with N -D input to the perceptron. Since $N + 1 = p$ from above, the $1 \times N \times 1$ topology can realise the targets of the p patterns.

It follows that any desired set of regions may be separated, since a pattern may be seen as representing a region, with the boundary of any two adjacent regions lying at the midpoint between their respective representative patterns.

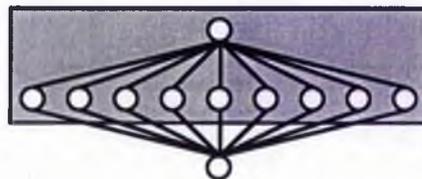


Figure 4.5 — A single hidden layer topology with one input dimension. The shaded box shows the part of the topology under consideration as a N input perceptron.

The realisability of any 1D classification problem using a single hidden layer is also proved using a different approach by Gibson and Cowan.²⁰

4.2.2.2 2D Input Space

Once the dimensionality of input space is more than 1, the maximum number of regions exceeds the VC dimension when more than 1 hyperplane is used. Thus there is always the possibility that a single hidden layer which partitions the regions will not separate them. Gibson²¹ has characterised the cases in which this possibility occurs for 2D input.

From Gibson,²² there is the distinction of various kinds of hyperplane:

- An *inconsistent* hyperplane. Consider the regions on either side of the hyperplane, from one end of the hyperplane to the other. There will be an equal number of regions, r , on either side of the hyperplane. Let $t_1(h_j)$ be the vector of targets for the regions on the active side of hyperplane h_j (i.e. the side of the hyperplane for which the unit has output 1). Let $t_0(h_j)$ be the vector of targets for the regions on the inactive side of the hyperplane. For any i , $1 \leq i \leq r$, $t_{1i}(h_j)$ is the target for the region exactly on the opposite side of the region for which $t_{0i}(h_j)$ is the target. Then the hyperplane is inconsistent if and only if there exist a and b such that $t_{0a}(h_j) = 0$, $t_{1a}(h_j) = 1$, $t_{0b}(h_j) = 1$, and $t_{1b}(h_j) = 0$. (See figure 4.6.)

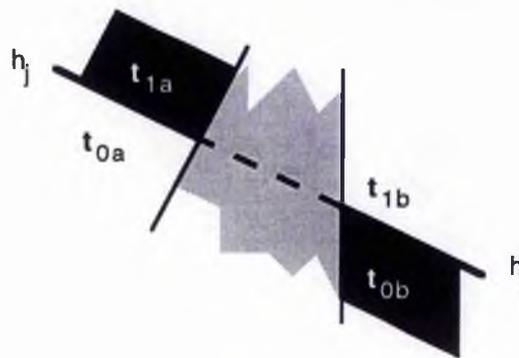


Figure 4.6 — An inconsistent hyperplane, h_j .

²⁰Gibson & Cowan, 1990, p. 1592

²¹Gibson & Cowan, 1990; Gibson, 1992, 1993, 1994

²²Gibson, 1994

- An *essential* hyperplane is one that forms part of the boundary of the desired decision region. Essential hyperplanes are necessary for a partitioning. Again, considering the vectors of targets for the regions on either side of a hyperplane, h_j , $t_0(h_j)$ and $t_1(h_j)$, h_j is essential if and only if there exists a such that $t_{0a}(h_j) \neq t_{1a}(h_j)$. (See figure 4.7.)

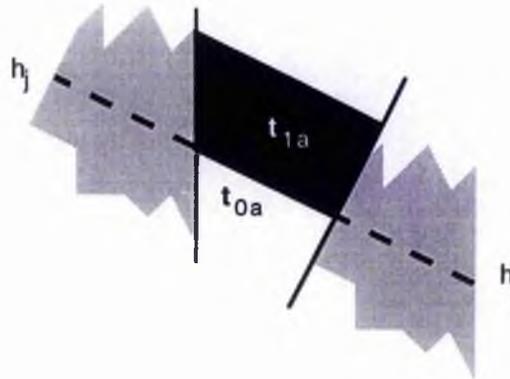


Figure 4.7 — An essential hyperplane, h_j .

Gibson then has the result that for any desired bounded decision region, S , for which no three essential hyperplanes intersect, S is realisable using one hidden layer if and only if no essential hyperplane is inconsistent.²³ This useful result means that it is possible to detect whether or not the essential hyperplanes can separate as well as partition the desired decision region, by inspection. However, where a separation is not possible by the essential hyperplanes, Gibson's technique does not tell you how many extra hidden units will be required, nor where to place their hyperplanes.²⁴

For 2D inputs, a desired decision region can always be approximated, even if it does have an inconsistent hyperplane, by placing a parallel hyperplane to each inconsistent hyperplane. (See figure 4.8.) The distance between the parallel hyperplanes indicates the degree of accuracy of the approximation. However, the greater the degree of accuracy, the larger the number of hidden units that are required.

²³Gibson, 1994, p. 914

²⁴Gibson, 1993, p. 991

Take, for example, the approximated chequer-board problem in figure 4.8. Using just the essential hyperplanes shown, the targets are unrealisable. Consider the excitation of the output unit:

$$out_A w_A + out_B w_B + out_C w_C + out_D w_D + bias = ex \tag{4.5}$$

where $out_j \in \{A, B, C, D\} \in \{0, 1\}$ is the output of a hidden unit, $w_j \in \{A, B, C, D\}$ is the weight from a hidden unit to the output unit, and $bias$ is the bias weight to the output unit. Considering regions AC, AD, BC and BD alone from figure 4.8, there are the following simultaneous inequalities from [4.5] which must be solved to realise the approximated chequer-board:

$$w_A + w_C + bias > 0 \tag{4.6}$$

$$w_A + w_D + bias < 0 \tag{4.7}$$

$$w_B + w_C + bias < 0 \tag{4.8}$$

$$w_B + w_D + bias > 0 \tag{4.9}$$

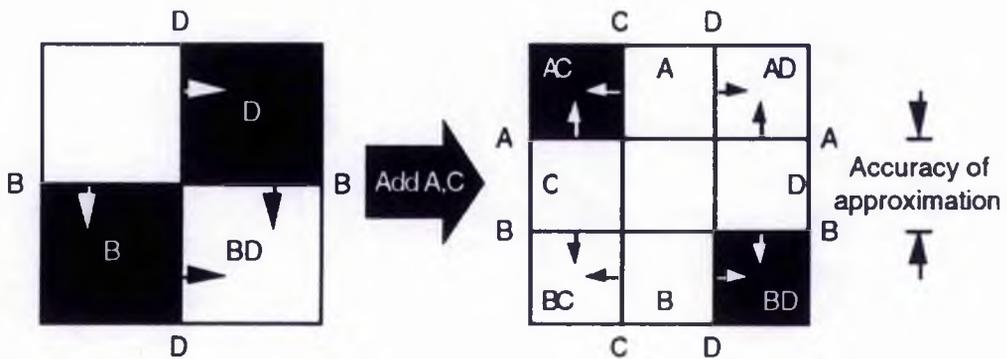


Figure 4.8 — The chequer-board approximated by placing parallel hyperplanes to the inconsistent hyperplanes, removing the inconsistency. The region must be bounded to comply with Gibson’s theorem.

Combining [4.6] and [4.7] gives $w_C > w_D$. Combining [4.8] and [4.9] gives $w_C < w_D$. Hence there is a contradiction, and weights cannot be found to achieve the desired decision region.

However, consider an IO diagram that is realisable using hyperplanes A to D in figure 4.8. For example, imagine trying to realise the same diagram as in figure 4.8, except for region AD which is black. This changes [4.7] to be greater than 0, and the contradiction is removed. (This could now be

realised with weights to the output unit, $w_A = 4$, $w_B = 2$, $w_C = 2$, $w_D = 4$, and $bias = -5$, for example.) An extra hyperplane can be added (E), as shown in figure 4.9, which, with a strong negative weight, $w_E < -w_A - w_D - bias$, to the output unit, would turn AD (now ADE) back to white. Hence, by adding in a non-essential hyperplane, the original desired decision region becomes realisable.

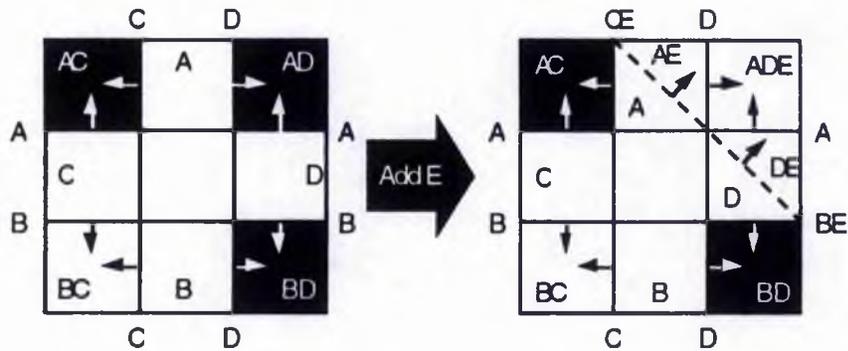


Figure 4.9 — A non-essential hyperplane (dashed line, hyperplane E) is placed, enabling the approximation to the checker-board in figure 4.8 to be realised without a second hidden layer.

However, as shown in figure 4.10, the closer the approximation to the checker-board, the larger the number of hidden units required.²⁵

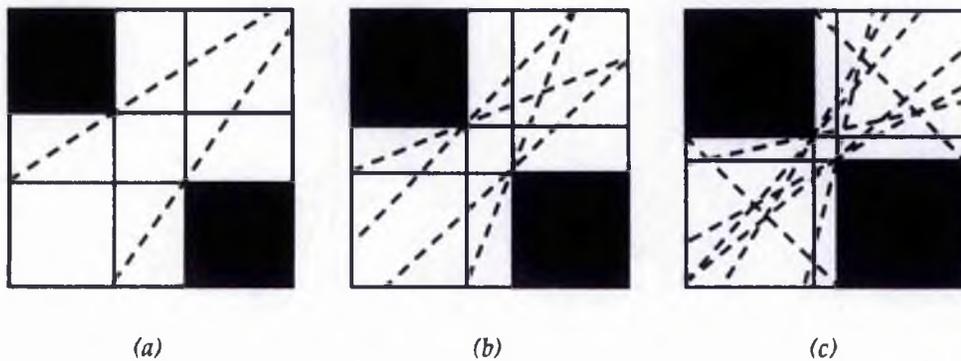


Figure 4.10 — The closer the approximation to the checker-board, the greater the number of non-essential hyperplanes necessary for realisation with one hidden layer. (a) has 2, (b) has 4 and (c) has 8 non-essential hyperplanes. (Non-essential hyperplanes are indicated by dashed lines.)

²⁵Gibson, 1992, p. 268

Therefore, for approximating functions with 1D or 2D input, an arbitrary degree of accuracy can always be achieved using a network with a single hidden layer — supporting the work of Hornik et al.

4.2.2.3 Higher Dimensions of Input Space

Gibson's theory does not generalise to higher dimensions, however.²⁶ Gibson has an example of a 3D decision region, with no inconsistent essential hyperplanes, which cannot be separated using just the essential hyperplanes.²⁷ (See figure 4.11.) However, there is an inconsistent line of intersection of two hyperplanes, as shown in figure 4.11, and this, perhaps, is the source of the problem.

Until further work is done in this area, little can be said about how approximation in n dimensions can be achieved.

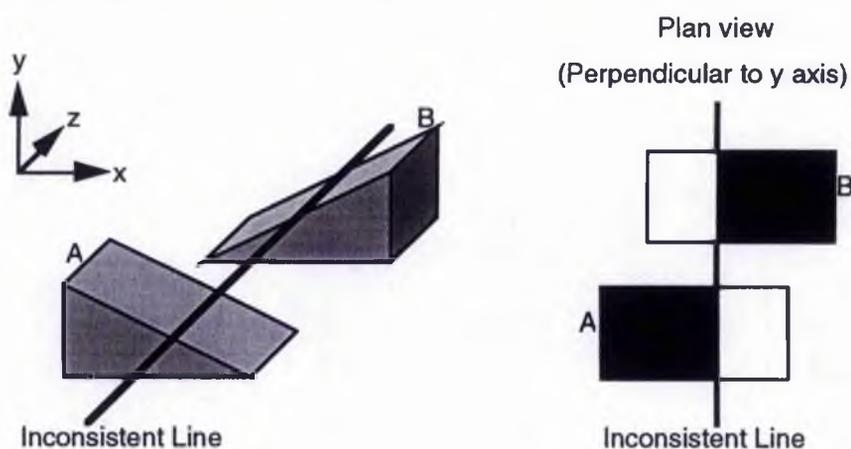


Figure 4.11 — An unrealisable 3D problem, and the inconsistent line of intersection of two of the hyperplanes, which is the possible cause of unrealisability.

4.2.3 One Hidden Layer or Two?

It is all very well to say that a single hidden layer can approximate any desired decision region, but this begs the question of whether it is worth

²⁶Gibson, 1994, p. 915

²⁷Gibson, 1994, Fig. 8, p. 917

it. The work of Gibson does not indicate where the non-essential hyperplanes can be placed, nor how many should be placed, and the increasing numbers of hyperplanes required as the approximation gets closer is a further disadvantage. Gibson acknowledges these criticisms.²⁸

In the example of the chequer-board, Gibson uses 10 hidden units to construct his approximation (with another 4 to bound it).²⁹ More accurate approximations would require more hidden units. Using two hidden layers rather than one, an unbounded, exact realisation of the chequer-board is possible using 4 hidden units — 2 in the first, and 2 in the second hidden layer — a difference of 6 units. For a strict comparison between the two methods, Gibson's approximated chequer-board can be realised using 4 units in the first hidden layer (and another 4 to make the boundary), and 2 units in the second hidden layer — a difference of 4 units. The closeness of the approximation to the chequer-board is changed simply by moving the hyperplanes in the first hidden layer. No further alteration is necessary.

A further difficulty occurs in training. Gibson attempted to train the single hidden layer approximation to the chequer-board using back-propagation, with disappointing results.³⁰ The fact that a realisation is possible with a given topology does not imply that it can be trained.

The theorem of Hornik et al is useful in that it indicates that one way to build a topology that will eventually be capable of generating (close to) the desired decision region is to add units to the first hidden layer until the desired degree of accuracy is achieved. The work of Gibson shows that there are certain practical difficulties with this strategy, which relate to the number and positioning of the non-essential hyperplanes, and to training. Chester also makes a similar point:

²⁸Gibson, 1992, p. 268; Gibson, 1993, p. 991

²⁹Gibson, 1994, p. 915

³⁰Gibson, 1992, pp. 268-270

The problem with a single hidden layer is that the neurons therein interact with each other globally, making it difficult to improve an approximation at one point without worsening it elsewhere.³¹

It would seem a better strategy, therefore, to use a network whose first hidden layer contained the essential hyperplanes only. Subsequent hidden layers can be used to overcome the difficulties of inconsistent hyperplanes. This strategy, however, is subject to the difficulty that it is necessary to have some idea of how many units will be required in the second hidden layer. This will be explored in the next section.

³¹Chester, 1990, p. 268

4.3 The Second Hidden Layer

The first hidden layer divides input space up into convex regions, bounded by the hyperplanes. Each region may be given a unique identifier, which is a combination of 1s and 0s from the outputs of the first hidden layer units. The outputs of the first hidden layer units all lie on a unit hypercube, of dimensionality N . The space in which this hypercube lies will be termed the *first hidden layer output space* (1HLO space), or equivalently, the *second hidden layer input space* (2HLI space). The purpose of the first hidden layer may then be said to be that of mapping input space onto the vertices of a hypercube.³² A second layer will be said to be needed from henceforth if a separation is not possible with the essential hyperplanes only.

Each region in input space is given a target, and since each region in input space is represented by a vertex of the 1HLO hypercube, targets can be assigned to each vertex. Some vertices will not be used (and hence have no target) if the number of hyperplanes exceeds the number of input units, as was indicated in the previous section. The need for a second hidden layer is indicated by the linear inseparability of the 1HLO hypercube.

The relationship between the 1HLO hypercube and input space is shown conceptually in figure 4.12, for a problem with three units in the first hidden layer, and two input units. The origin is not shown, because it is not used. If a given region is to be black, the target of the corresponding vertex becomes 1 — otherwise the target is 0. By looking at the targets to be realised on the hypercube, it is possible to gauge the number of units (if any) that will be required in the second hidden layer. For example, if regions 110, 010 and 011 were to be black, a single hyperplane will realise the targets of the vertices, and no second hidden layer is needed. If, however, regions 101, 010 and 011 were to be black, two hyperplanes are necessary to realise their targets, and a second hidden layer with two units is required.

³²Nilsson, 1965, p. 104

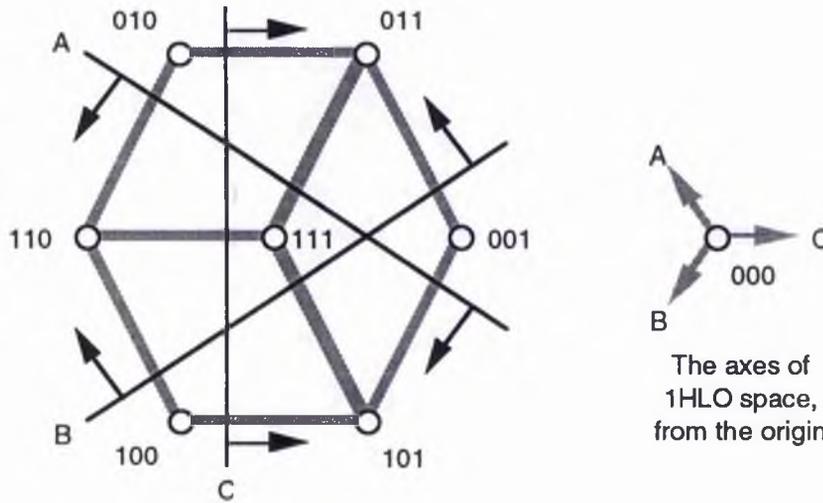


Figure 4.12 — *The relationship of input space to the 1HLO hypercube. On the left hand side, the hyperplanes in input space are indicated by thin, black lines. The arrows indicate which side of the hyperplane the unit is on. The numbers indicate the pattern of activity of the outputs of the units in the first hidden layer, in the order A, B, C. The thick, shaded lines indicate the edges of the 1HLO hypercube*

The effect of changing one of the hyperplanes, such that its unit is active on the opposite side is shown in figure 4.13. The 1HLO hypercube is rotated, but there is no effect on the number of hidden units required in the second hidden layer in order to separate a given desired decision region.

The number of units required in the second hidden layer is the number of hyperplanes necessary to realise the targets of the vertices of the 1HLO hypercube. In section 4.3.1, this number of units is examined in more detail.

In general, when considering the second hidden layer, it is useful to think of the first hidden layer as having performed a partitioning of the regions. The separating capabilities of the first hidden layer should be ignored. The purpose of the first hidden layer, as stated earlier, is to provide a unique vertex on the 1HLO hypercube for each region of input space — especially those which have a different target. It is up to the second hidden layer to place hyperplanes in the 1HLO/2HLI hypercube such that all the regions which are assigned different targets lie in a different region of 2HLI space.

This enables separation, and hence the correct output being assigned to each region.

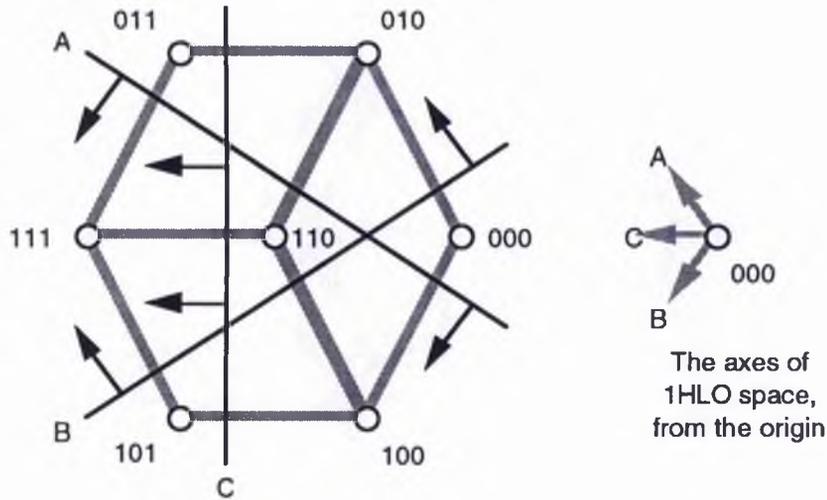


Figure 4.13 — The effect on the hypercube of changing the direction of one of the hyperplanes.

4.3.1 Number of Units Required

Section 4.3.1.1 considers the number of units required to perform a separation for one output unit only. Although only one output unit has been considered so far, it is necessary to consider the extra complexities of using many output units. This is considered briefly in section 4.3.1.2.

4.3.1.1 One Output Unit

This section illustrates an approach to realising the targets of the vertices of the 2HLO hypercube, which leads to a procedure for recommending the number of hidden units to be used in the second hidden layer.

The approach is based on the N -bit parity problem. To solve the parity problem, for each vertex of the hypercube, the output of the network must be 1 if there are an odd number of 1s in the co-ordinate of the vertex, and 0 otherwise. This problem is recognised by Rumelhart et al as being hard for neural networks, because “the most similar patterns (those which differ by

a single bit) require different answers.”³³ Hertz et al show that the N -bit parity problem is realisable with N hidden units with threshold activation functions in a single layer.³⁴ The positions of the hyperplanes for $N = 3$ are shown in figure 4.14. The first hyperplane to be placed goes through the midpoints of those edges of the hypercube which extend from the origin. Subsequent hyperplanes are placed parallel to the first until all vertices of the hypercube with different targets lie in different regions of input space. (Nilsson has the general result that any set of patterns can be realised by placing a number of parallel hyperplanes.)³⁵

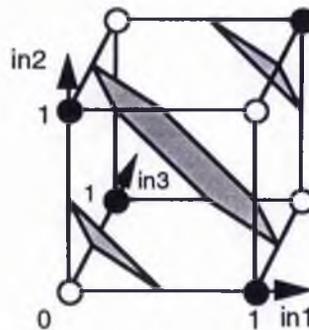


Figure 4.14 — Realisation of the 3 bit parity problem with 3 hyperplanes.

The parity problem is described as a “worst function” by Minsky and Papert, though they do not say that it is the worst case.³⁶ For the purposes of developing this procedure, however, the N -bit parity problem will be assumed to be the hardest problem for a hypercube of dimensionality N . “Hardest” here is taken to mean that the problem requires the most hyperplanes to solve, and that the solution of all other problems on the hypercube requires at least 1 fewer hyperplane. This is shown empirically for the 3D case in figure 4.15.

The logic behind all other problems requiring at least 1 fewer hyperplane is that if, for example, the vertex at the origin in figure 4.14 was to change its target, the hyperplane used to distinguish it from the three adjacent

³³Rumelhart et al, 1986, p. 334

³⁴Hertz et al, 1991, p. 131

³⁵Nilsson, 1965, p. 109

³⁶Minsky & Papert, 1988, p. 153

vertices is no longer required. Since the realisation of the 3 bit parity problem shown in figure 4.14 does not have to start at the origin, but can start from any vertex, 2 hyperplanes are sufficient for realisation of the targets if any vertex changes target.

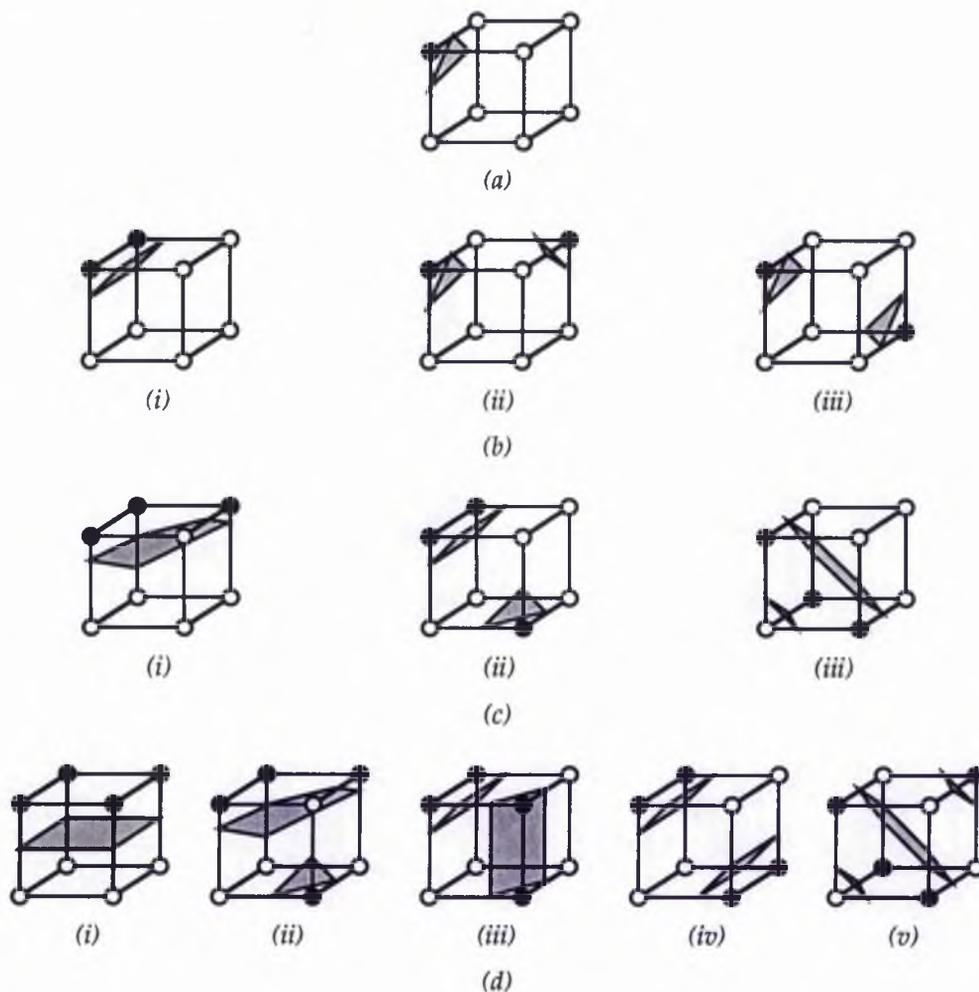


Figure 4.15 — All the possible different problems on the vertices of a cube. Equivalent problems can be achieved by rotating the hypercube or swapping the black and white targets. (a) 1 black, 7 white. (b) 2 black, 6 white. (c) 3 black, 5 white. (d) 4 black, 4 white. Note that only the 3-bit parity problem in (d)(v) requires 3 hyperplanes for realisation.

Let us assume that the N -bit parity problem is the hardest problem, with N hyperplanes required for solution, and at most $N - 1$ hyperplanes are required to solve any other problem. The procedure for calculating the number of hidden units in the second layer is then simply to use the

number of units required to realise the largest parity problem it is possible to get with the number of vertices that are used in the 1HLO hypercube. Thus, for the example in figure 4.12, seven vertices of the 1HLO hypercube are used, since there are seven regions in input space. The largest parity problem it is possible to generate on seven vertices is the 2-bit parity problem, and hence 2 hidden units in the second hidden layer should suffice for any desired final output. This gives the following formula for the number of units in the second hidden layer, M , in terms of the number of regions R the hyperplanes realise:

$$M = \lfloor \log_2 R \rfloor \quad [4.10]$$

where $\lfloor x \rfloor$ is the largest integer not greater than x . R may be reduced by only considering those regions which lie in a given bounded region of input space.

With a low ratio of input units to units in the first hidden layer, the number of units in the second hidden layer is likely to be significantly lower than the number of units in the first hidden layer. For example, with 3 input units, and 100 hidden units in the first hidden layer, the maximum possible number of regions from [4.4] is 166 751. From [4.10] the maximum number of hidden units required in the second hidden layer is 14.

4.3.1.2 Many Output Units

When there are many output units, each output unit may give its own classification to various regions of input space, which have been partitioned by the first hidden layer. This is an extra complication from the one output unit case. Here, each output unit will place its own, unique demands on the second hidden layer. Consider the case of a topology with two output units, two input units and three hidden units. An example of a possible partitioning of input space, and targets for the output units is shown in figure 4.16.

In order to cope with any possible set of targets for an output, it may be necessary for the second hidden layer to make each region available to each output unit, in order to enable the output unit to give the desired classification to those regions. The simplest way to do this is to have a unit in the second hidden layer dedicated to each region. Each vertex of the

1HLO hypercube that is used is distinguished from the rest of the hypercube by a unit in the second hidden layer. This is shown in figure 4.17, for the example in figure 4.16. Using this method, all of the possible outputs are realisable for any number of output units.

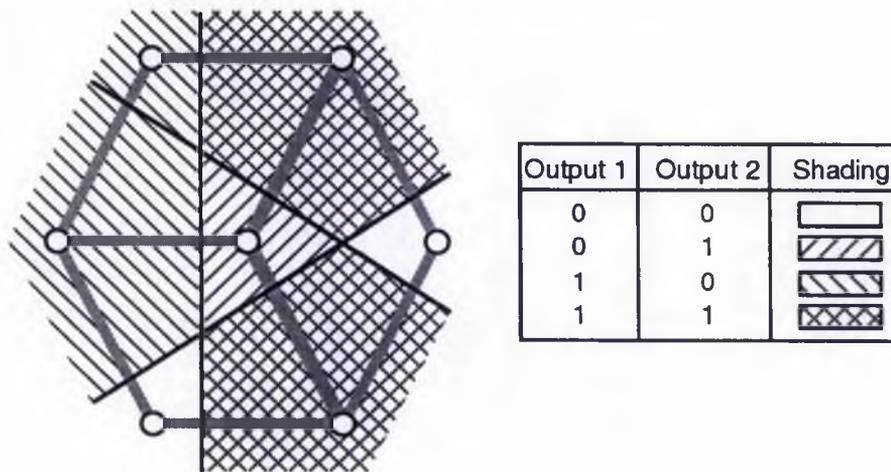


Figure 4.16 — A possible partitioning of input space, indicated by the thin, black lines, and the targets for the two output units, indicated by the shading of the regions. The 1HLO hypercube is represented by the thick, shaded lines, with the vertices indicated by the white circles.

The disadvantage with this method is that the number of regions may be prohibitively large for relatively small numbers of units in the input and first hidden layer. For example, with 4 input units, and 12 hidden units in the first hidden layer, there are 794 possible regions, using equation [4.4]. This is rather a large number of units to use in the second hidden layer, and it is unlikely that all of the 2^{794} possible outputs would be needed!

Another possibility is to apply the procedure in 4.3.1.1 once for each output unit. Thus, if there are O output units, the number of units in the second hidden layer should be $O.M$, where M is calculated as per [4.10]. Rather than 794 units in the second hidden layer indicated for the above example by the simple method, this method gives a very much smaller number: $\lfloor \log_2 794 \rfloor = 9$ units in the second layer for each output unit — a total of 36 units in the second hidden layer. Eighty-nine output units would be needed before this method exceeded the number recommended by the simple method.

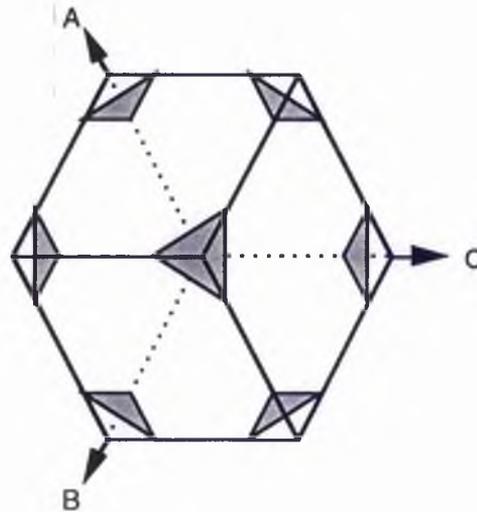


Figure 4.17 — Using 7 units in the second hidden layer to represent each region in input space, from figure 4.16.

4.3.2 Further Hidden Layers

The work of Lippmann outlined in the introduction made it clear that any desired decision region could be realised with a two hidden layer topology. However, this work was not based on a proper understanding of the capabilities of topologies with one and two hidden layers. It has been shown above that the purpose of the second hidden layer is to solve any parity problems that arise in the realisation of the targets of the 1HLO hypercube.

Section 4.3.1.2 showed that whatever the targets of the vertices of the 1HLO hypercube, a single hidden layer is sufficient for their realisation, albeit using a large number of units. This means that there is never any necessity for a third hidden layer, since a separation for any number of output units is always possible with two hidden layers. This result also indicates that if input space is the set $\{1, 0\}^N$, then there is never any need for more than one hidden layer.

The question of the use of more than two hidden layers would arise if it was possible to use fewer than the N units required in the second hidden layer to solve an N -bit parity in 1HLO space, and compensate this somehow with extra hidden layers. This is not possible with threshold

units, in a strictly layered feed-forward topology, since the patterns with different targets must be realised by the hyperplanes at each stage. If fewer units are used in the second hidden layer, therefore, there is no full discrimination of the patterns on the vertices of the 1HLO hypercube, which means that there will be a vertex on the 3HLI hypercube with a target of both 1 and 0 for two different patterns.

4.4 A Brief Look at Sigmoid Units

In this section, some of the increased computing powers of neural networks with sigmoid nonlinearities, reported by authors such as Dasgupta and Schnitger,³⁷ and Sontag³⁸ are examined. Section 4.4.1 explores these increased powers. It is demonstrated in section 4.4.2 that the chequer-board can be approximated without a second hidden layer, and without excessive extra units in the first hidden layer. Section 4.4.3 shows how sigmoid units are capable of using fewer units in the first hidden layer, in such a way that a third hidden layer is required.

4.4.1 The Increased Power of Sigmoid Units

Sigmoid units are capable of realising more sophisticated regions than threshold units. This is because the magnitude of the excitation must be taken into consideration, as well as the sign. Since the output of a unit can be any real number between 0 and 1, the outputs of the units in each layer lie inside a unit hypercube, and the weights of each layer transform the unit hypercube of the previous layer into the unit hypercube of the next layer. (The points on the surface of the hypercube are only reached when the weight to of one of the lower units is $\pm\infty$.) So long as no two points with different targets are mapped onto the same point in the hypercube, each layer will preserve the possibility of solving the problem. The requirement for a further hidden layer is indicated by the linear inseparability of the points of different targets in the hypercube.

³⁷Dasgupta & Schnitger, 1993

³⁸Sontag, 1989

The 3-bit parity problem, for example, can be solved using a 3*2*1 topology. This, it will be noted, means that the hyperplanes in the hidden layer no longer segregate all the different classes in different regions of input space. The way the network solves the problem is actually rather subtle. Two hyperplanes are used, parallel to one another, which are positioned in relation to the patterns as shown in figure 4.18.

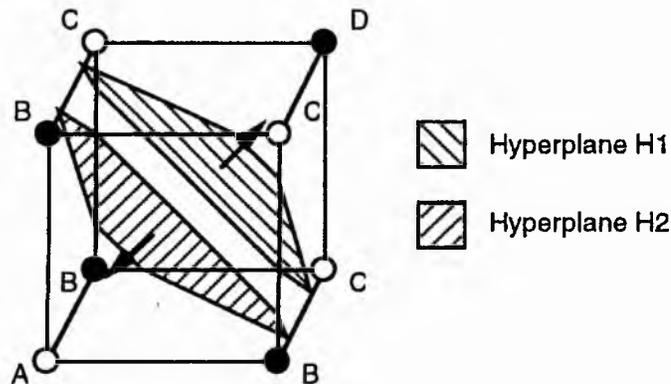


Figure 4.18 — Realisation of 3-bit parity using two sigmoid units in the first hidden layer. The arrows on each hyperplane show for which side of the hyperplane the output of the unit is more than 0.5. All patterns with the same letter are equidistant from each hyperplane and hence have the same output.

The magnitudes of the weights are set such that the patterns closer to H2 have relatively low outputs from the hidden units, in comparison to the two patterns further away. H1 has a high magnitude of weights. The output unit then has to realise the patterns in figure 4.19, which shows the outputs of the hidden units for each pattern. This illustrates how controlling the magnitude of the weights, as well as their ratio and sign enables a more efficient solution.

By drawing *double hyperplanes*,³⁹ which show the inputs for which the excitation is $-q$ and $+q$, where q (≈ 4.6) is the excitation required to generate an output of 0.99, it is possible to see the distance in input space between where the output of the unit is close to 0 and where it is close to 1. The double hyperplane drawn at $+q$ is called the *positive double*

³⁹Lansley & Clark, 1993; Storer, 1994

hyperplane and that at $-q$ the *negative double hyperplane*. Beyond $\pm q$, the behaviour of the sigmoid unit is the same as the threshold behaviour. It is in the regions between $+q$ and $-q$ that the extra powers of the sigmoid are observed. (This will be illustrated by the example in section 4.4.2.)

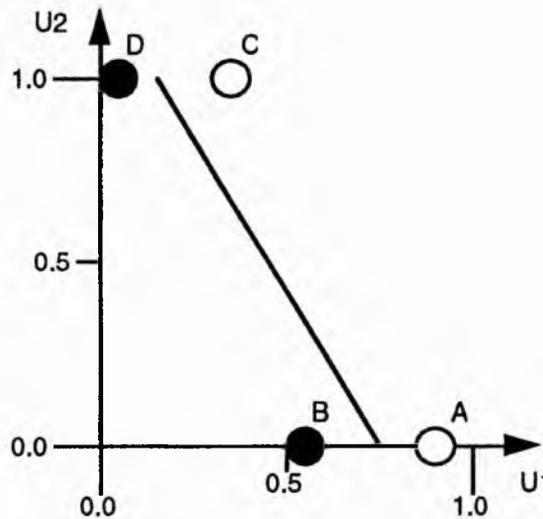


Figure 4.19 — Realising the targets of the patterns at the output layer. The U_2 axis corresponds to the outputs for the patterns for hyperplane H_2 in figure 4.18. The unit for this hyperplane has a large magnitude of weights, so the outputs for each pattern are close to 1 and 0. The U_1 axis corresponds to hyperplane H_1 in figure 4.18. This has a low magnitude of weights so that different output values are given for A, B, C and D. This enables a solution with a single hyperplane.

The closer the double hyperplanes are together, the smaller the regions between $+q$ and $-q$, and hence the closer the approximation to the threshold behaviour. Figure 4.20 shows this for a simple example of the interaction of two hidden units.

Figure 4.20 also shows that the closeness of the double hyperplanes is related to the magnitude of the fan-in weight vector to the unit. The larger the magnitude of the weight vector, the closer the double hyperplanes. The formula for the positive double hyperplane of a unit u_j with weights $w_{j1...jn}$ to inputs $y_{1...n}$, and bias b_j is given in [4.11]; that for the negative double hyperplane in [4.12].

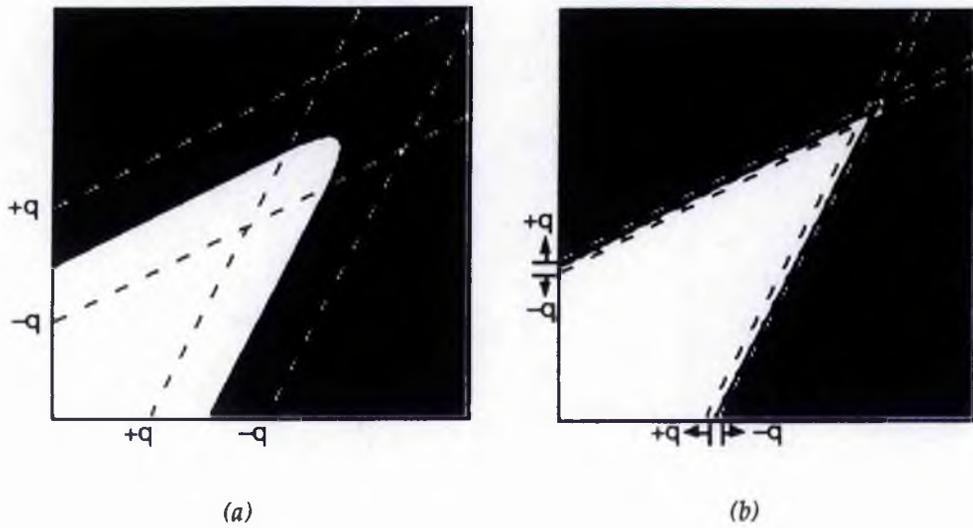


Figure 4.20 — The approximation to threshold units gets closer as double hyperplanes for each unit (parallel dashed lines) get closer. (a) Weight vector length to each hidden unit: $4\sqrt{2}$. (b) Weight vector length: 8.

$$w_{j1}y_1 + w_{j2}y_2 + \dots + w_{jn}y_n + b_j - q = 0 \tag{4.11}$$

$$w_{j1}y_1 + w_{j2}y_2 + \dots + w_{jn}y_n + b_j + q = 0 \tag{4.12}$$

Since these hyperplanes are parallel, the distance between them, d_j , is equal to the difference between their perpendicular distances to the origin:

$$d_j = \left| \frac{b_j + q}{\|w_j\|} - \frac{b_j - q}{\|w_j\|} \right| \tag{4.13}$$

where w_j is the fan in weight vector $[w_{j1}, \dots, w_{jn}]$ to unit u_j . Hence, the distance is given by the following proportionality:

$$d_j \propto \frac{1}{\|w_j\|} \tag{4.14}$$

Thus, as the magnitude of w_j approaches infinity, the distance between the double hyperplanes approaches zero, and the unit more and more closely approximates the behaviour of a threshold unit. Hence sigmoid units can realise any decision region that threshold units can, using the same topology.

Although no more than two hidden layers are ever necessary, given sufficient units in each layer, it is possible that further hidden layers can be used to compensate for smaller numbers of units in earlier hidden layers. This is illustrated in section 4.4.3.

4.4.2 Approximation of the Chequer-Board With a Single Hidden Layer Using Sigmoid Units

Figure 4.21 shows the approximation of the chequer-board, and how it is achieved. The double hyperplanes are marked as well. This approximation — requiring 4 hidden units, can be made for an arbitrarily large, but finite region of input space, since the distance between the double hyperplanes (which partially controls the size of the chequer-board) can be extended by decreasing the magnitude of the weights to each hidden unit, enlarging the frame, X . An unbounded approximation would require infinitely small weights, and is therefore not possible. The distance between the hyperplanes themselves (which also has an effect on the size of the chequer-board) can be controlled by repositioning the hyperplane, which is controlled by the direction of the weight vector to each hidden unit. Hence the enlargeable approximation to the chequer-board, within a given region of input space, can be achieved with a constant number of 4 hidden units.

To explain the behaviour of the network which produces the output in figure 4.21, consider hyperplanes A and C . The double hyperplanes mark where each unit has an output of 0.99 and 0.01. Region AC is the region for which both units have an output more than 0.99. Region aC is the region for which unit A has an output between 0.01 and 0.99, and unit C has an output of more than 0.99; and vice versa for region Ac . Region ac is the region for which units A and C both have outputs between 0.01 and 0.99, and it is this region that is of interest for generating the chequer-board.

The weights to the output unit are set, roughly speaking, such that whenever the output of A is sufficiently greater than the output of C , the output unit has output interpreted as 1. This is achieved by C having a negative weight to the output unit which is slightly larger in magnitude than the positive weight of A . Hence, in region AC , the output of A is never sufficiently greater than C for an output of 1, and the output is zero

in this region. However, in region Ac , where c is between 0.01 and 0.99, and A is always greater than 0.99, the output of A is sufficiently greater than the output of C for an output of 1. The slightly greater weight of C to the output unit means that C can be slightly less than 0.99, even though A is more than 0.99, and the output will be 0. This explains why the output is 0 just inside the positive double hyperplane of C in region Ac .

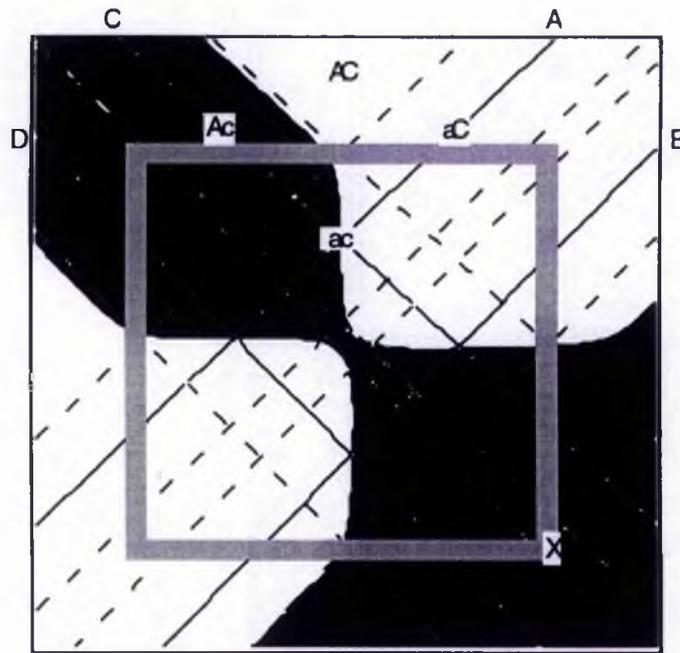


Figure 4.21 — An approximation to the checker-board using four hidden units, A , B , C and D . The checker-board is approximated only within a specific bounded region (inside the frame, X), which is where the outputs of the hidden units are between 0.01 and 0.99. The hyperplanes of the hidden units are marked with solid lines; the double hyperplanes with dashed lines. It is the double hyperplanes that are of interest when looking at the behaviour of networks with sigmoid units.

In region ac , which is bounded by the double hyperplanes of A and C , the black half shows where the output of A is sufficiently greater than that of C for an output of 1. This roughly divides the region into two halves. The curve in the bottom corner is where the output of C is low enough that the small, positive bias weight to the output unit compensates for the greater magnitude of weight of C than A to the output unit.

The interactions between the other combinations of hyperplanes is similar, and the combined effect gives rise to the chequer-board.

4.4.3 The Possibility For Using More Than Two Hidden Layers

Using an extra hidden layer to compensate for a lack of units in an earlier hidden layer can be illustrated in the following example. The classical feed-forward topology for the XOR problem is $2*2*1$. This uses two hidden units in a single hidden layer. However, using only one unit in the first layer, placing the hyperplane, as in figure 4.22(a), the XOR problem can be mapped onto the 1D space indicated in figure 4.22(b). The patterns then require two further hyperplanes to realise their targets. This means that an alternative $2*1*2*1$ topology can be found, which uses a second hidden layer to compensate for one unit in the first layer rather than two.

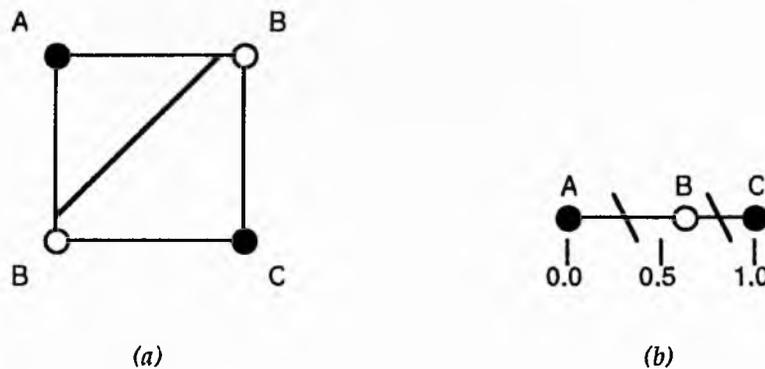


Figure 4.22 —Solution of the XOR problem using a $2*1*2*1$ topology. (a) First hidden layer. A single hyperplane is placed as shown. Targets with the same letter have equal output from the unit in the first hidden layer. The patterns are then mapped onto the 1D space shown in (b). Two hyperplanes (diagonal lines) in a second hidden layer are required to realise their targets.

There is no real advantage to the $2*1*2*1$ topology, since it requires more units, and more parameters⁴⁰ than the $2*2*1$ topology. It is also much more difficult to train. In tests, using standard back-propagation with a learning rate of 0.7 and a momentum of 0.9, the $2*2*1$ topology had an 83% success

⁴⁰N.B. The parameters of the network are the weights and biases.

rate with a limit of 1 000 training cycles,⁴¹ and the $2*1*2*1$ topology a success rate of 9% for the same limit. However, for higher-bit parity problems, savings can be made in terms of parameters using a reduced topology with an extra hidden layer.

The 4-bit parity problem, for example, which requires a $4*4*1$ topology with threshold units (using 9 units and 25 parameters) can be solved using a $4*3*1$ topology (with 8 units and 19 parameters), and a $4*2*2*1$ topology (with 9 units and 19 parameters). The $4*3*1$ solution is analogous to the $3*2*1$ solution of the 3-bit parity problem given above. The $4*2*2*1$ solution is more interesting. The 4-bit parity problem is mapped onto the 1HLO unit square depicted in figure 4.23, using two units in the first hidden layer. (Appendix 4.A explains how this is achieved.) This requires two hyperplanes to realise the targets.

Note, from figure 4.23 that by analogy with the $2*1*2*1$ solution of the XOR problem shown above, a $4*2*1*2*1$ topology could also be used for the 4-bit parity problem. This is a topology with three hidden layers, 10 units, and 20 parameters.

Experiments were done comparing the trainability of $4*2*1*2*1$, $4*2*2*1$, $4*3*1$ and $4*4*1$ topologies on the 4-bit parity problem. The results for 200 runs of maximum 50 000 cycles using standard back-propagation with a learning rate of 0.3 and momentum 0.9 are shown in table 4.3. This also shows the percentage of runs which had converged by 10 000 cycles. The $4*4*1$ topology successfully trains the patterns more quickly than do the other topologies, training becoming harder as the number of units in each hidden layer are reduced, or the number of layers is increased. (Lang and Witbrock have also observed "an order of magnitude" slow-down for each hidden layer added in their experiments.⁴² They linked this to the weakening of the error signal as it passes through each layer. The problem was countered by directly connecting each layer to *all* the layers above it.)

⁴¹One cycle is the calculation and implementation of the weight change for all the patterns.

⁴²Lang & Witbrock, 1988, p. 52

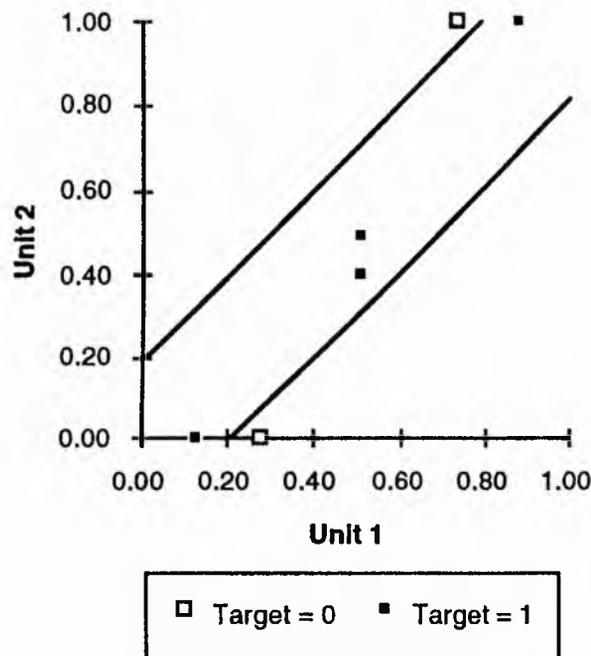


Figure 4.23 — Second layer of solution of the 4-bit parity problem using a 4*2*2*1 topology. The first hidden layer projects the problem onto a unit square. The targets are now separable using two hidden units in a second hidden layer, as indicated by the diagonal lines.

Topology	Convergence (cycles)	
	50000	10000
4*4*1	61%	41%
4*3*1	71%	14%
4*2*2*1	28%	17%
4*2*1*2*1	46%	11%

Table 4.3 — Comparison of convergence rates for various topologies for the solution of the 4-bit parity problem.

The results show that the 4*3*1 topology is slower to train than the 4*4*1 topology, but more likely to train in the long run. Similarly, the 4*2*1*2*1 topology is slower, but more likely to train in the long run than the 4*2*2*1 topology. Interestingly, this does not reflect the results for the comparison of the 2*2*1 and 2*1*2*1 topologies in training the XOR problem.

The 4*3*1 topology trains significantly better than the 4*2*2*1 and 4*2*1*2*1 topologies. This is because the latter topologies have roughly the same number of parameters, but a smaller volume of weight space that contains the solution state than the 4*3*1 topology. This is due, in part, to a

larger number of regions in weight space that contain the solution state for the $4 \times 3 \times 1$ topology, which relates to the symmetry of the ordering of hidden units in the hidden layer. Since it does not matter which hidden unit does what, there are $3!$ equivalent, and equally sized, regions in weight space that contain the same solution for the $4 \times 3 \times 1$ topology.⁴³ (This is called *unit ordering symmetry*, and is also explained in chapter 5.) The $4 \times 2 \times 2 \times 1$ and $4 \times 2 \times 1 \times 2 \times 1$ topologies both have $2 \times 2! = 4$, and therefore fewer equivalent regions that contain the solution state.

Unit ordering symmetry does not explain why the $4 \times 3 \times 1$ topology trains more successfully than the $4 \times 4 \times 1$ topology. The $4 \times 4 \times 1$ topology, however, has roughly 20% more parameters than the $4 \times 3 \times 1$ topology, and it may be that the reduced success rate is due to the higher dimensionality of weight space.⁴⁴ This is termed the "curse of dimensionality" in the literature.⁴⁵ It may also explain the improved results of those techniques such as that of Weigend et al⁴⁶ which minimise the number of links as well as the pattern error during training.

It may be concluded that sigmoid units have additional capabilities to threshold units, since the magnitude of each unit's weight vector must now be taken into consideration. These additional powers result in three observations. Firstly, networks with sigmoid units are able to realise more complex IO pictures than the same networks with threshold units. Secondly, there may be a reduced requirement for units in a given hidden layer, but without the requirement for a further hidden layer (such as in the $3 \times 2 \times 1$ solution of the 3-bit parity problem above). Finally, a further reduction in hidden units in a given layer may be compensated for by a further hidden layer. Hence, topologies with more than two hidden layers may be considered for sigmoid units. However, it should be pointed out that the fact that a topology is capable of solving a given problem does not imply that a given training algorithm will be capable of finding the solution weight state in a reasonable amount of time.

⁴³Denker et al, 1987, p. 886-887

⁴⁴Hrycej, 1990, p. 557

⁴⁵Hertz et al, 1991, p. 198

⁴⁶Weigend et al, 1991b

4.5 Conclusion

This chapter has shown some of the capabilities of various topologies in the classification paradigm. It also indicates heuristics for choosing the number of units in the first and second layers. The theorems of the various authors who prove that only one layer is necessary to approximate arbitrary mappings are useful theoretically, but are not practical. They are disadvantaged because they must restrict themselves to an approximation of the desired decision region rather than an exact realisation, and the number of units they use may be prohibitive.

It is possible to describe a method for constructing a two hidden layer topology for exact realisation, or for inexact realisation to a specified degree of accuracy. Adding a unit to the first hidden layer has the consequence of creating extra regions. This can be related to the number of units required in the second hidden layer to be sure of assigning the correct output to each region. To help summarise the points in the chapter, the following algorithm provides a constructive approach to a given problem using the principles discussed:

- 1 Let H_1 be the number of units in the first hidden layer.
- 2 Let H_2 be the number of units in the second hidden layer.
- 3 $H_1 = 1$.
- 4 $H_2 = 0$.
- 5 While the problem is untrainable in a reasonable time, or is not trainable to a desired degree of accuracy in reasonable time using the current topology:
 - 5.1 Increment H_1 .
 - 5.2 Calculate the maximum number of regions, R , that H_1 hyperplanes can form in input space, as per equation [4.4].
 - 5.3 Calculate H_2 on the basis of R (assuming the worst case), as per equation [4.10].
- 6 Stop.

If the number of desired regions is known in advance, then H_1 can be initialised to be the minimum number of hyperplanes whose maximum

number of regions is greater than the desired number of regions, using equation [4.4]. H_2 in this case is calculated on the basis of the desired number of regions, and remains constant throughout.

A further point is that the topology considerations do not account for the clustering of the data. It assumes that the number of regions has been decided already from the data, and this is not true in practice. Deciding the number of regions is itself a difficult task. The architectural considerations for ensuring good generalisation relate to constructing a topology that is capable of realising the desired regions, once the number of regions has been decided. The goodness of the generalisation is dependent upon the number of hidden units in the first layer, which corresponds to the number of regions formed by the network. More regions may mean a better fit to the data, but it may also mean over-fitting noisy data.

However, the positive side of the analysis in this chapter is that it shows what a network is capable of in its general response. This promotes a view towards learning which aims at training and generalisation simultaneously.

The Mitchellian approach does not deal with this side of the generalisation problem, except to say that whatever topology is chosen, this represents a bias of the learner. (See chapter 3.) This chapter has indicated some of the limits to concepts that can be learned using a given topology, and hence the bias the user might employ in the choice of topology for the neural implementation of Mitchell's technique discussed in chapter 6. (Chapter 5 discusses a restricted neural implementation for topologies without hidden units.)

Appendix

4.A Mapping the 4-Bit Parity Problem onto a Unit Square Using Two Units in the First Hidden Layer, Whilst Preserving Their Separability

A sample solution weight state to the 4-bit parity problem is given in table 4.A.1, and the outputs of each unit in the network for each of the 16 patterns in the 4-bit parity problem are reproduced in table 4.A.2.

Since the first and second hidden layers each have two units, it is possible to visualise the behaviour of the network in these layers, with relative ease in comparison with observing the placement of the hyperplanes of the units in the first hidden layer in the 4D input space of the problem. This is shown in figure 4.A.1. This shows that once the first hidden layer has placed its hyperplanes, it is not a problem for the rest of the network to assign the correct class to each pattern.

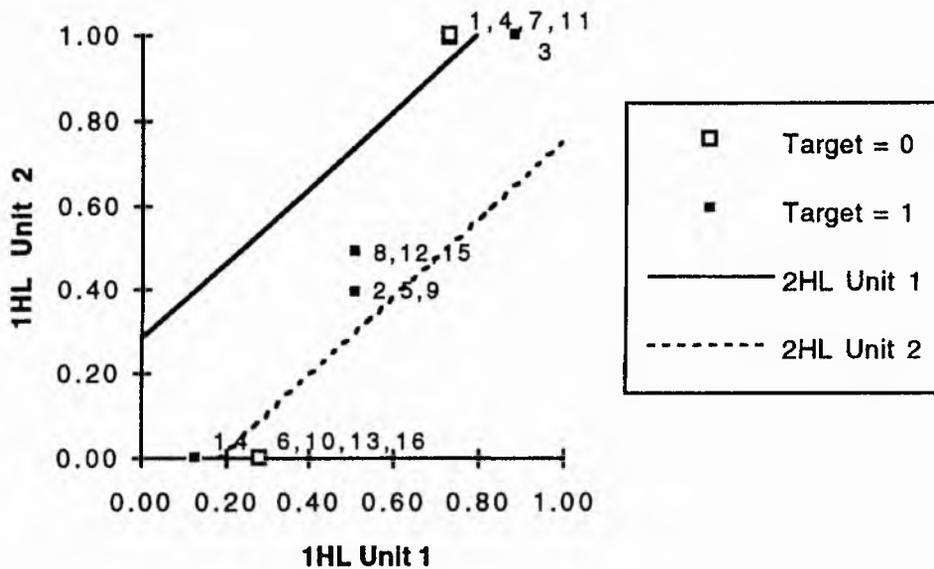


Figure 4.A.1 — First hidden layer outputs for each pattern in the 4-bit parity problem, and hyperplanes of the second hidden layer. The positions of the patterns in 1HLO space are marked as shown, and the numbers beside each correspond to the pattern numbers in table 4.A.2.

Using figure 4.A.1, it is clear that the hyperplane of the first unit in the first hidden layer passes through patterns 2, 5, 8, 9, 12 and 15, since the output of the unit is 0.5 for those points. Similarly, the hyperplane of the second unit in the first hidden layer passes through patterns 8, 12 and 15, and close to patterns 2, 5 and 9. Since this hyperplane is rather steeper, with a higher magnitude of weights (see table 4.A.1(a)), it is reasonable to assume that the hyperplanes occupy roughly the same space. Using two cubes to represent the 4D input space, figure 4.A.2 shows the positions of the hyperplanes for each unit in the first hidden layer.

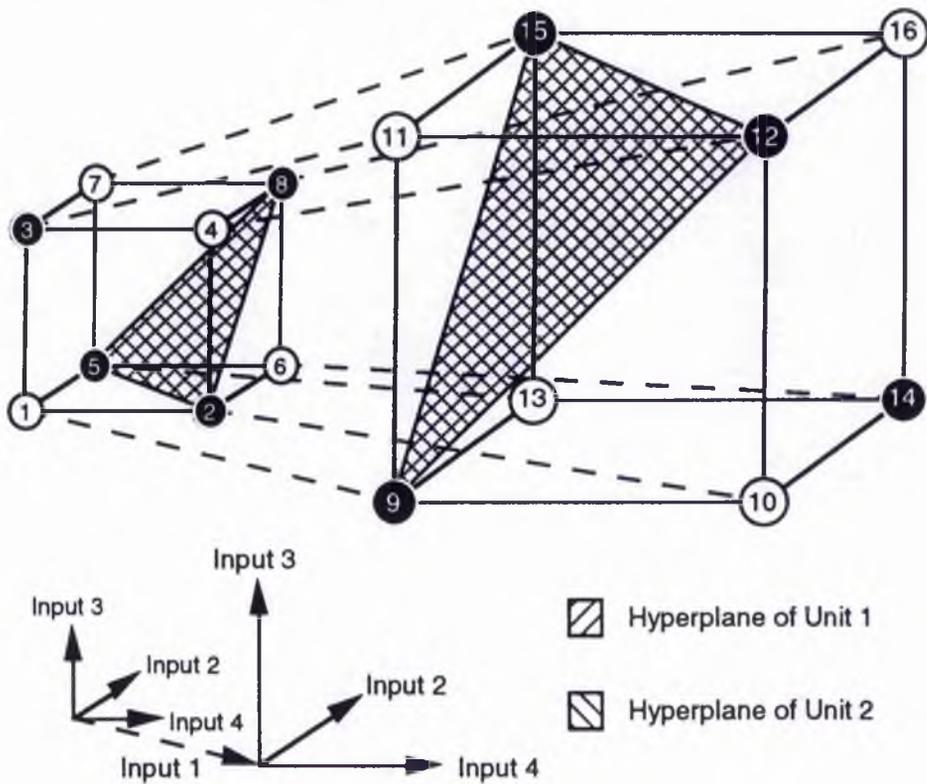


Figure 4.A.2 — Representation of the 4-bit parity problem as two 3D cubes. The dotted lines indicate a change in input 1, which replaces the extra dimension. The vertices of each cube are numbered according to the pattern they represent in table 4.A.2. Both hyperplanes have an output close to 1.0 for patterns 3 and 11 (see table 4.A.2), which means that each unit is active on the same side of its hyperplane as the other unit. Also, as discussed in the text, both hyperplanes are roughly in the same place.

There is a certain degree of similarity between this, and the hidden layer of the 3*2*1 topology solution of the 3-bit parity problem, shown in figure

4.18. However, an extra hidden layer is necessary here, because of the additional complexity of the problem, which results in an inseparable 1HLO hypercube.

Stork and Allen claim to be able to solve the N -bit parity problem with two hidden units, and no second hidden layer. However, they use a novel activation function.⁴⁷ Another attempt by Minor, using sigmoidal activation functions, with $0.5N$ hidden units does not use a strictly layered topology.⁴⁸ Such solutions are not possible with sigmoid units only in a strictly layered topology, with no connections that skip a layer.

Weights	Hidden Layer 1	
Input Layer	Unit 1	Unit 2
Bias	1.01103	6.53674
Unit 1	-0.977516	-6.94597
Unit 2	-0.977516	-6.94597
Unit 3	0.979558	7.31745
Unit 4	-0.977516	-6.94597

(a)

Weights	Hidden Layer 2	
Hidden Layer 1	Unit 1	Unit 2
Bias	-3.17681	2.00747
Unit 1	-9.9537	-10.7642
Unit 2	11.0873	11.5556

(b)

Weights	Output Layer
Hidden Layer 2	Unit 1
Bias	-5.98023
Unit 1	-14.5636
Unit 2	12.7787

(c)

Table 4.A.1 — *The weight state of the 4*2*2*1 topology used to solve the 4-bit parity problem. (a) Weight matrix between the input layer and the first hidden layer. (b) Weight matrix between the first and second hidden layers. (c) Weight matrix between the second hidden layer and the output layer.*

⁴⁷Stork & Allen, 1992, p. 924

⁴⁸Minor, 1993, p. 707

Pattern	Input				Hidden Layer 1		Hidden Layer 2		Output	Target
	1	2	3	4	Unit 1	Unit 2	Unit 1	Unit 2		
1	0	0	0	0	0.73	1.00	0.64	1.00	0.07	0
2	0	0	0	1	0.51	0.40	0.02	0.76	0.97	1
3	0	0	1	0	0.88	1.00	0.30	0.98	0.90	1
4	0	0	1	1	0.73	1.00	0.65	1.00	0.07	0
5	0	1	0	0	0.51	0.40	0.02	0.76	0.97	1
6	0	1	0	1	0.28	0.00	0.00	0.27	0.07	0
7	0	1	1	0	0.73	1.00	0.65	1.00	0.07	0
8	0	1	1	1	0.51	0.49	0.06	0.90	0.99	1
9	1	0	0	0	0.51	0.40	0.02	0.76	0.97	1
10	1	0	0	1	0.28	0.00	0.00	0.27	0.07	0
11	1	0	1	0	0.73	1.00	0.65	1.00	0.07	0
12	1	0	1	1	0.51	0.49	0.06	0.90	0.99	1
13	1	1	0	0	0.28	0.00	0.00	0.27	0.07	0
14	1	1	0	1	0.13	0.00	0.01	0.65	0.90	1
15	1	1	1	0	0.51	0.49	0.06	0.90	0.99	1
16	1	1	1	1	0.28	0.00	0.00	0.27	0.07	0

Table 4.A.2 — *The outputs of all the units in the network for each of the 16 patterns in the 4-bit parity problem.*

5 Neural Implementation Based on Weight Space

This chapter documents the implementation of Mitchell's technique in neural networks, which uses weight space as its basis. The implemented technique works only for restricted examples, but provided some valuable insights into the nature of neural networks, which led to the development of the technique in chapter 6. Section 5.1 explores the theory of the neural technique for networks without hidden layers, and pattern sets assumed to be linearly separable. Section 5.2 gives the algorithm and some experimental results. Section 5.3 elucidates some of the problems associated with generalising the technique to cope with hidden layers.

The technique discussed in this chapter is an introductory implementation of Mitchell's technique in a neural environment — given the restrictions of no hidden layers in the architecture, and of linearly separable problems. This chapter reveals the circumstances under which a series of important principles can be observed about the nature of Mitchell's technique in neural networks.

5.1 Theory of the Weight Space Technique

As indicated in the prime directives given in chapter 3, when developing a neural implementation of Mitchell's technique, it is crucial that a solution state has a single representation in whatever ordering is used to measure version space. If there are many representations of a solution state in the ordering, then it is not possible to detect the no-alternative situation. This is because the no-alternative situation is detected by the boundary representatives of version space being measured at the same point in the ordering. If the goal state has many possible representatives in the ordering, then the boundary representatives may have reached the goal state without being at the same point in the ordering. If this happens, the no-alternative situation is not detected.

This technique aims to address a problem from an earlier implementation which used output tolerance as a basis for the ordering.¹ The earlier implementation broke the directive that equivalent weight states must have the same value in the ordering, since it could not cope with weight scaling symmetry.²

To explain weight scaling symmetry in more detail, consider an output unit with a sigmoid activation function, as per equation [5.1]:

$$output = activation = \frac{1}{1 + e^{-(excitation)}} \quad [5.1]$$

The range of this function, the output interval, will be any real number from 0 to 1. The position of the separating hyperplane is given when the excitation is 0. This corresponds to an output of 0.5. When the output is binary interpreted, outputs greater than 0.5 are interpreted as 1, and outputs less than 0.5 are interpreted as 0. Hence the output is 1 for all points on one side of the hyperplane, and 0 for all points on the other side of the hyperplane.

$$output = \frac{1}{1 + e^{-(b \times excitation)}} \quad [5.2]$$

If the excitation in equation [5.1] is multiplied by a positive constant, b , as in equation [5.2], the output is still in the same half of the output interval, making no difference to the binary interpretation. Denker et al³ call this b -symmetry, and it is referred to by Watkin et al⁴ as the *gauge freedom* of the perceptron.

A technique was needed that used an ordering such that all weight states producing the same binary interpreted output would have the same measure in the ordering. If the weights to a unit are treated as a vector, then all linearly dependent weight vectors have an output in the same half of the output interval. This is captured in the concept of the spherical

¹Weir & Polhill, 1993

²Weir & Polhill, 1993, p. 9

³Denker et al, 1987, p. 886

⁴Watkin et al, 1993, p. 503

perceptron.⁵ Since the direction of the weight vector to an output unit is all that is important, the only weight states that need to be considered for the perceptron are those on the surface of a unit hypersphere. The technique in this chapter does not apply this restriction, however.

Therefore, an ordering is used in this implementation which measures the direction of the weight vector by measuring its angle relative to an arbitrary, constant, reference weight vector. All the weight states at the same position in the ordering therefore have a constant angle from the reference vector, and hence lie on the surface of a hypercone whose principal axis is the reference vector.

Learning is incremental, as in the symbolic technique, though the neural technique trains all the patterns seen so far and the new pattern, rather than just the new pattern. The weight states are encouraged to stay on the surface of the same hypercone during training, unless it is necessary to move away from the hypercone in order to correctly classify a new pattern. This is to be attempted using a cost function which measures the distance of the current weight state from the surface of the current hypercone, and is to be minimised using gradient descent. The cost function will be incorporated into a potential function which serves much the same purpose as updating and selection within in the symbolic technique. Weight space angle is changed by the minimum amount necessary to allow the correct classification of each new pattern as it is added to the training set.

Since the weight state only moves away from the hypercone in order to make a correct classification, it must be assumed that the training set is noiseless and realisable by the neural architecture used. If the network moves away from the hypercone in order to correctly classify an incorrect training pattern, then there is the possibility of premature convergence of the hypercones. Similarly, premature convergence is possible if the network moves away from the hypercone in order to correctly classify a pattern which cannot be realised by the topology. In fact, since in practical

⁵Watkin et al, 1993, pp. 520-521

circumstances the network is trained to zero misclassification error, an unrealisable pattern will result in an inability to train.

This technique is not designed to deal with networks with hidden units, and hence a perceptron learning algorithm could be used for training. However, gradient descent is needed in order to combine training for correct classification with a function for keeping the weight state near the surface of a hypercone. Hence back-propagation is used.

Section 5.1.1 looks at the use of hypercones to implement the ordering. Section 5.1.2 examines the implementation of the processes of updating and selection within using error functions. Section 5.1.3 relates the neural implementation back to the symbolic technique.

5.1.1 Bidirectional Search Using Hypercones

Figure 5.1 shows the structure of the hypercones for a three dimensional weight space. The reference vector is N_0 . Since the search is bidirectional, there are two hypercones, based on two neural networks, N and X , which are used to represent the boundaries of version space in terms of angle. Version space is the region of weight space lying between the two hypercones.

The *internal hypercone* marks the boundary of version space containing the weight state, N , at the minimum angle from N_0 that correctly classifies all the patterns presented so far. The *external hypercone* marks the boundary of version space containing the weight state, X , at the maximum angle from N_0 that correctly classifies all the patterns presented so far.

Separate hypercones are needed for each output unit of the neural network. For convenience, only networks with one output unit shall be considered henceforth. The procedure for many output units is the same as that for training each output unit individually.

The search proceeds from an angle, α_N , of 0 for N and an angle, α_X , of π for X from the reference vector N_0 . Patterns are added to the training set incrementally, as they are in Mitchell's technique. As each pattern is added, weight states for N and X are found that correctly classify all the patterns so far with the minimum change in their angles relative to N_0 .

The no-alternative situation is detected when N and X have the same angle relative to N_0 . It will be shown (in section 5.1.1.3) that when N and X reach the no-alternative situation their weight vectors will be collinear. This means the separating hyperplanes of N and X are in the same position in input space.

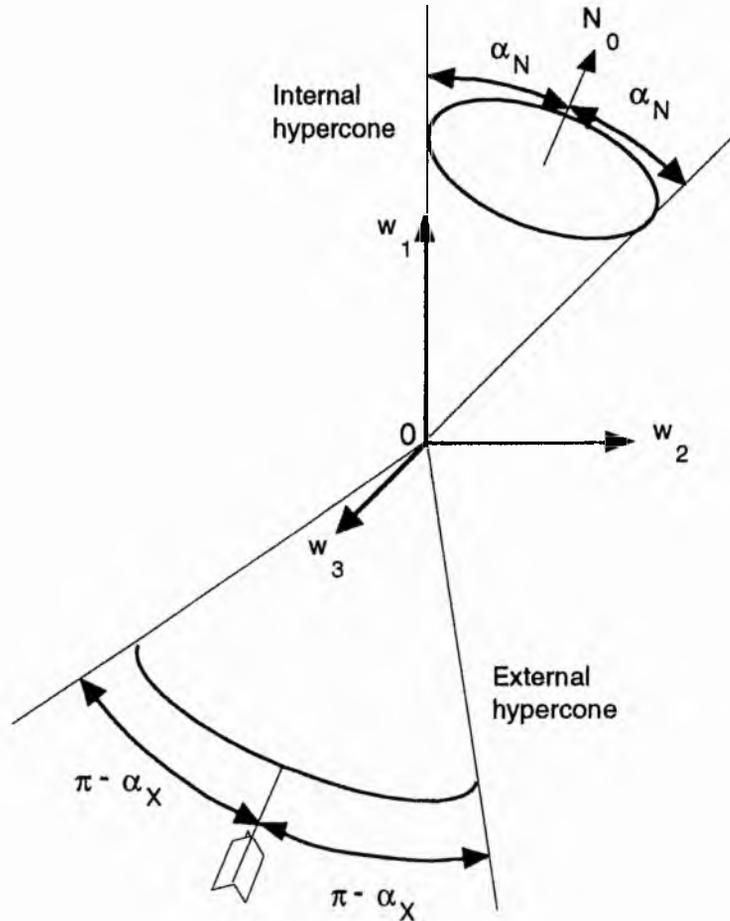


Figure 5.1 — The internal and external hypercones in 3D weight space. The internal angles of each hypercone are as shown.

In order to comply with the prime directives outlined in chapter 3, it is necessary to show that N and X proceed in a single, convergent direction only during the search through version space. This means that the angle of N must always be increasing, and the angle of X must always be decreasing. This will be argued in section 5.1.1.1. If it is not the case that N and X necessarily always move closer together, then there can be no guarantee of convergence.

It is also necessary to look at the terminating condition. Section 5.1.1.2 shows that whilst there is freedom of movement, there are always correctly classifying weight states at different angles relative to N_0 . Section 5.1.1.3 shows that N and X are therefore collinear at termination.

5.1.1.1 Monotonically decreasing angle between N and X

If there is no weight state with a smaller angle than α_N that is consistent with all the patterns presented so far, then there can be no weight state with a smaller angle than α_N that is consistent with the patterns presented so far and any number of additional patterns. Any change in weight state in order to permit the correct classification of a new pattern must lead to a greater value for the angle, α_N .

Similarly, if there is no weight state with a greater angle than α_X that is consistent with all the patterns presented so far, then there can be no weight state with a larger angle than α_X that is consistent with the patterns presented so far and any number of additional patterns. Any change in weight state in order to permit the correct classification of a new pattern must lead to a smaller value for the angle, α_X .

Therefore, as more and more patterns are added, N and X each only make movements towards one another, provided that at each stage, close to the minimum change is made to weight space angle (see figure 5.2) when finding a weight state for N and X that correctly classifies all the patterns presented. This means that when N and X meet (or pass slightly) in terms of angle, there is no other angle (within a certain accuracy) with correctly classifying weight states.

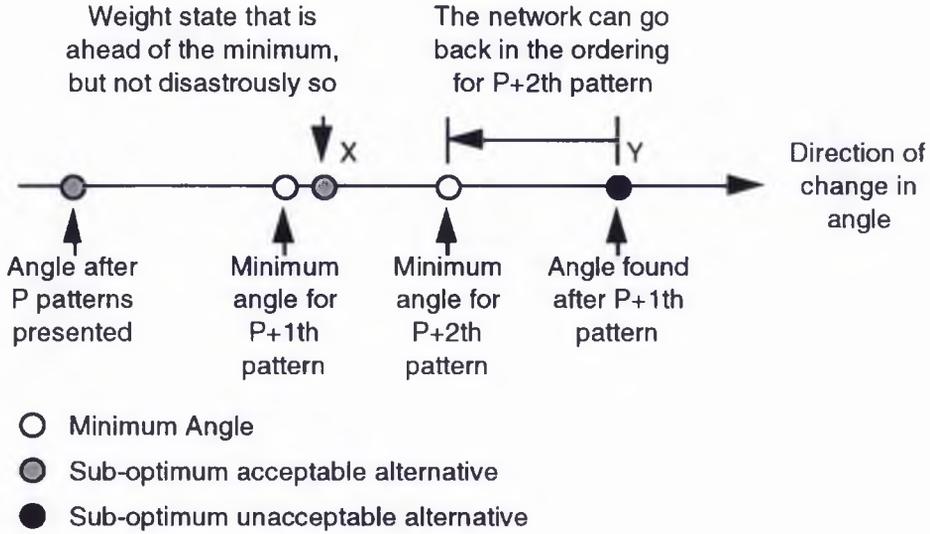


Figure 5.2 — There is the possibility for the network to go back in the ordering if the weight state strays too far from the hypercone of minimum change in angle. On presentation of the P+1th pattern, if the weight state moves to Y, there is the possibility that the network can go back in the ordering when classifying the P+2th pattern. X is preferred to Y in this sense.

5.1.1.2 Freedom of movement and hypercones

In IO space terms, the no-alternative situation is when the hyperplane has no *freedom of movement*. This is the situation whereby the patterns are so aligned that the hyperplane has no possibility to move whilst preserving correct classification, within a certain degree of accuracy. This is illustrated in figure 5.3.

With freedom of movement, there are always correctly classifying weight states on different hypercone surfaces.

To prove this, it is necessary to relate IO space to weight space. This allows the possibility of understanding how the angle of the hyperplane changes in IO space as the angle of the weight vector changes in weight space.

Consider the case of 2D input, and a network with 2D weight space (i.e. no bias weight). The hyperplanes of these weight states all go through the

origin of input space. The freedom of movement will be between two hyperplanes, as shown in figure 5.4(a).

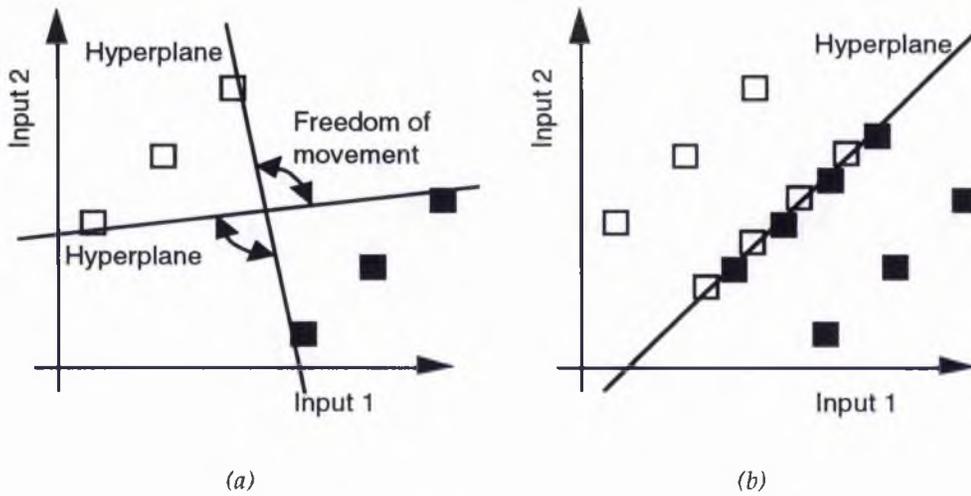


Figure 5.3 — Patterns for which the hyperplane (a) has freedom of movement, and (b) has no freedom of movement to within an acceptable degree of accuracy.

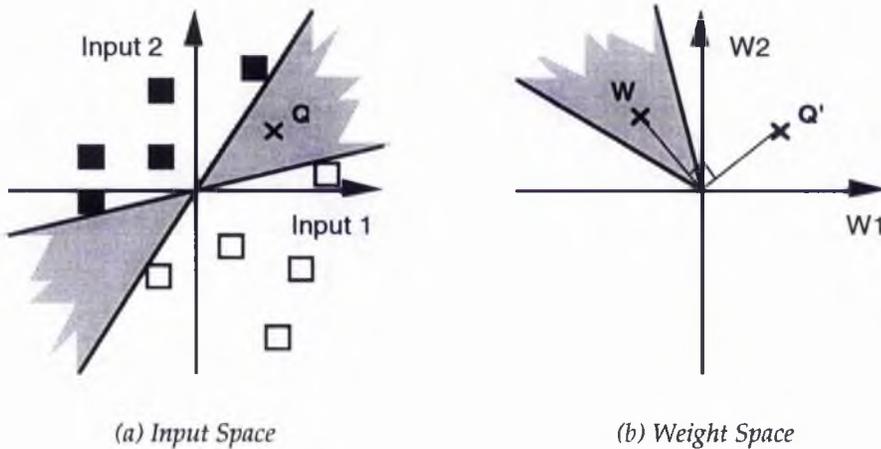


Figure 5.4 — (a) Shaded area shows freedom of movement of hyperplane, when there is no bias weight. Black indicates a target of 1, white indicates a target of 0. (b) The shaded region shows the region of weight space that gives the hyperplanes in the shaded region of input space.

For any weight state, W , with parameters w_1 and w_2 , the formula of the hyperplane is given by:

$$w_1 y_1 + w_2 y_2 = 0 \tag{5.3}$$

where y_1 and y_2 are the axes of input space. Consider an arbitrary point, $Q = (q_1, q_2)$ in input space, which determines a hyperplane. From [5.3], we know that:

$$w_1 q_1 + w_2 q_2 = 0 \quad [5.4]$$

Hence, a weight state that produces correct output with a hyperplane that goes through Q may be given by $w_1 = -q_2$ and $w_2 = q_1$, or by $w_1 = q_2$ and $w_2 = -q_1$, depending on which side of the hyperplane the output is to be 1 or 0. (In figure 5.4 $w_1 = -q_2$ and $w_2 = q_1$ gives correct output.) Therefore, for a point $Q' = (q_1', q_2')$ in weight space, where $q_1' = q_1$ and $q_2' = q_2$, a vector from the origin to Q' is perpendicular to the weight state whose hyperplane goes through Q in input space. From this, and information from the patterns as to which side of the hyperplane should be 1 or 0, it is possible to determine the region of weight space that contains weight states whose hyperplanes lie within the region of freedom of movement in input space. This is shown in figure 5.4(b).

Note that the hypercone is represented through two lines (K_1 and K_2 in figure 5.5), whereas the correctly classifying weight states in figure 5.4(b) occupy an area. It is clear that the surface of the hypercone always has a lower order of dimensionality than the portion of weight space that correctly classifies the patterns.

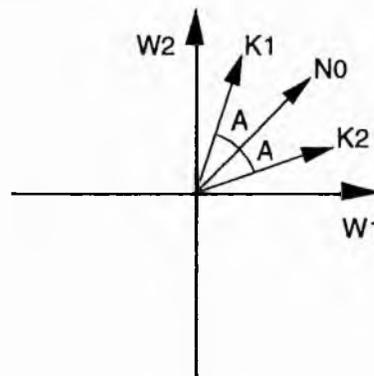


Figure 5.5 — K_1 and K_2 have the same angle A relative to a reference vector N_0 .

For the general n -D case, a small quantity, δ , (larger than the accuracy used to determine convergence) can be added to any of the weights (including the bias) of a separating hyperplane that does not lie too close

to any of the patterns, and still correctly classify the patterns, if there is freedom of movement. Therefore there is freedom of choice of the weights in all dimensions of weight space. Hence, for n -D weight space, correctly classifying weight states occupy a hypervolume, of dimensionality n , of weight space. Those on the surface of a hypercone occupy a hypersurface, of dimensionality $(n - 1)$.

There is a pathological potential counter-example to this, which is where two patterns of opposite class are in fact arranged close together, such that a change in any individual weight on its own results in misclassification. Changing the bias weight moves the hyperplane up or down, resulting in misclassification. Changing any other weight changes the gradient of the hyperplane about the intercept, resulting in misclassification. All separating hyperplanes pass through the point P . The situation is shown, for a 2D input space example, in figure 5.6.

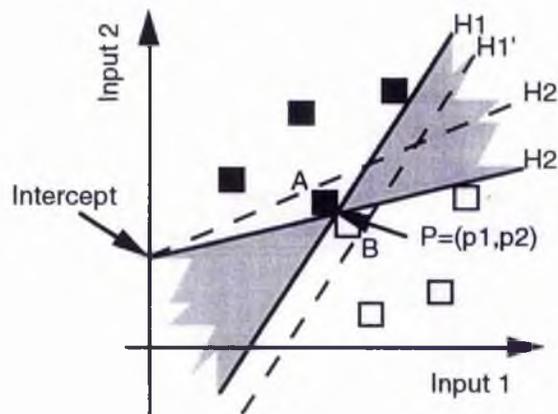


Figure 5.6 — Two patterns, A and B , restrict the freedom of movement of the hyperplanes, $H1$ and $H2$ to intersect at the point P . $H1'$ illustrates the bias weight restriction. Changing the bias weight only changes the intercept, whilst keeping the gradient the same, which results in the misclassification of B . $H2'$ illustrates the restriction on the other weights. Here, the weight from input 1 to the output unit is changed, which changes the gradient of $H2$, but retains the same intercept on the input 2 axis, resulting in the misclassification of A .

The set of weight states, A , whose separating hyperplanes pass through P , where $P = (p_1, p_2, \dots, p_n)$ in the n dimensional input space case must satisfy the following equality:

$$A = \{w_1, w_2, \dots, w_n, w_B \mid w_1 p_1 + w_2 p_2 + \dots + w_n p_n + w_B = 0\} \quad [5.5]$$

where w_B is the bias weight. This set corresponds to a hyperplane in weight space which passes through the origin, and has a normal in the vector $\mathbf{v} = (p_1, p_2, \dots, p_n, 1)$. A subset, W , of A correctly classify the patterns. This is indicated for the example in figure 5.6 by the shaded region in figure 5.7.

The set of weight states, C , lying on the surface of a hypercone with angle α from N_0 is given by:

$$C = \{\mathbf{w} \mid \mathbf{w} \cdot N_0 = \|\mathbf{w}\| \|N_0\| \cos \alpha\} \quad [5.6]$$

where $\mathbf{w} = (w_1, w_2, \dots, w_n, w_B)$ is a vector in weight space. There is no hypercone, the surface of which contains all members of W , except when $\alpha = \pi/2$ and $N_0 = (p_1, p_2, \dots, p_n, 1)$, which makes $C = A$ (remembering that $W \subseteq A$). This is a pathological counter-example within a pathological counter-example. Ignoring this second order of pathology, the essence of the argument is that with the restriction imposed by the two patterns close to P , the correctly classifying weight states W all lie on a hyperplane in weight space, which is flat. The hypercone surface, C , is curved, and is of the same dimensionality as the hyperplane A , on which all members of W lie. Hence W cannot lie entirely on the surface of the hypercone.

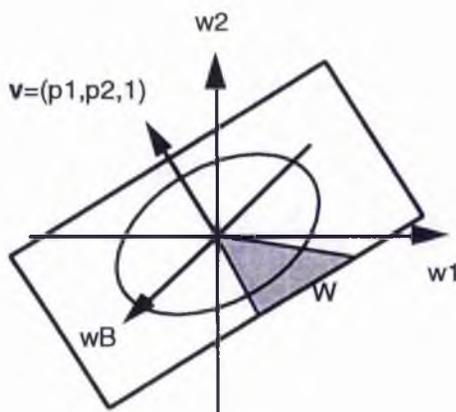


Figure 5.7 — The set of weight states that intersect with P in figure 5.6 lie on a hyperplane with normal vector \mathbf{v} , as shown. The shaded region (W) indicates those weight states which also correctly classify the patterns. The circle represents the hypercone with internal angle $\pi/2$, and $N_0 = \mathbf{v}$.

The weight states lying on the surface of a hypercone therefore either have a lower order of dimensionality than the set of weight states that correctly classify the patterns, or are on a curved, rather than a flat surface. Hence, there can be no single hypercone whose surface intersects with every single weight state in the correctly classifying hypervolume of weight space. When there is only one correctly classifying input space hyperplane, the hypervolume of correctly classifying weight states becomes a single line, which can lie on the surface of a hypercone in weight space.

The above shows then, that there are always weight states lying on the surfaces of different hypercones whilst there is freedom of movement.

5.1.1.3 Convergence

When there is no freedom of movement, all correctly classifying weight states are collinear. If N and X are collinear, they will lie on the surface of the same hypercone. If there is freedom of movement, N and X will lie on different hypercones, from 5.1.1.2, assuming they have been kept apart as much as possible. Therefore, N and X lying on the surface of the same hypercone is a necessary and sufficient condition for detecting no freedom of movement. This condition is termed *locking*.

5.1.2 Implementing Candidate Elimination Using Error Functions

A popular method for training neural networks is to use gradient descent, such as in the back-propagation technique.⁶ (For a discussion of gradient descent and back-propagation, see chapter 1.) Here, gradient descent is used to combine training with the directive of minimal updating in the ordering. This is achieved by using a potential function that adds a function which increases as the weight state deviates from the surface of the current hypercone to the sum of squared errors from standard back-propagation.

⁶Rumelhart et al, 1986

5.1.2.1 Minimising the Distance from the Current Hypercone

The idea of using additional cost functions in the potential function is not new.⁷ For example, Joerding and Meador⁸ add a function to the sum of squared error to encourage a weight state that satisfies certain conditions, enabling the explicit representation of prior knowledge about the function to be approximated (such as its monotonicity) during the training of the neural network.⁹

Here, a function is added to the sum of squared error which increases as the weight state deviates from the surface of the current hypercone. Let E_{D_j} be this function, w_j be the current weight vector, and v_j be a vector on the surface of the current hypercone. E_{D_j} should be proportional to the squared difference in angle of w_j from v_j . This would be $(\alpha - \beta)^2$ in figure 5.8. The angles of w_j and v_j are measured relative to N_0 .

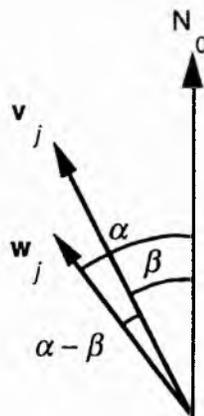


Figure 5.8 — Angles used to calculate the deviation from the current hypercone.

Instead of using the angles directly to measure the distance of the current weight state from the current hypercone, the cosines of the angles are used. Hence the scalar products of the weight vectors can be used as a

⁷Chauvin, 1989*, 1990*; Drucker & LeCun, 1991*; Hanson & Pratt, 1989*; Hinton, 1989*; Ishikawa, 1989*; Jean & Wang, 1994; MacKay, 1992b; Mozer & Smolensky, 1989*; Weigend et al, 1990, 1991. (Starred references are taken from Xu et al, 1992.)

⁸Joerding & Meador, 1991

⁹Joerding & Meador, 1991, p. 852

measure of the distance, which is easier for implementation purposes. The cosine of the angle has a one-one relationship with angles from 0 to π , which is the range of angles needed for consideration here.

Hence, E_{D_j} can be given by the following equation:

$$E_{D_j} = (\cos \alpha - \cos \beta)^2 \tag{5.7}$$

Therefore:

$$E_{D_j} = \left(\frac{\mathbf{w}_j \cdot \mathbf{N}_{0j}}{\|\mathbf{w}_j\|} - \frac{\mathbf{v}_j \cdot \mathbf{N}_{0j}}{\|\mathbf{v}_j\|} \right)^2 \tag{5.8}$$

To implement gradient descent, we set:

$$\Delta_D w_{ji} \propto -\frac{\partial E_{D_j}}{\partial w_{ji}} \tag{5.9}$$

where $\Delta_D w_{ji}$ is the change in weights due to the deviation of the weight state from the current hypercone.

Thus:

$$\Delta_D w_{ji} = -\frac{h}{\|\mathbf{w}_j\|} \left(\frac{\mathbf{w}_j \cdot \mathbf{N}_{0j}}{\|\mathbf{w}_j\|} - \frac{\mathbf{v}_j \cdot \mathbf{N}_{0j}}{\|\mathbf{v}_j\|} \right) \left(n_{0ji} - \frac{w_{ji} \mathbf{w}_j \cdot \mathbf{N}_{0j}}{\|\mathbf{w}_j\|^2} \right) \tag{5.10}^{10}$$

where h is a positive constant.

The angle cost function is designed to keep the weight path near the surface of the current hypercone as much as possible, whilst allowing the drive for correct classification of the patterns to guide the search as well. Of course, this method does not exhaustively search the surface of each hypercone for the best weight state before updating. This is not practical. The purpose of measuring the distance in terms of angle of the current weight state from the current hypercone is simply to prevent false locking

¹⁰The following results are useful when deriving equation [5.18], where m is the dimensionality of weight space:

$$\frac{\partial}{\partial w_{ji}} \mathbf{w}_j \cdot \mathbf{N}_{0j} = \frac{\partial}{\partial w_{ji}} (w_{j1}n_{0j1} + w_{j2}n_{0j2} + \dots + w_{ji}n_{0ji} + \dots + w_{jm}n_{0jm}) = n_{0ji}$$

$$\frac{\partial}{\partial w_{ji}} \|\mathbf{w}_j\| = \frac{\partial}{\partial w_{ji}} \sqrt{\sum_{k=1,m} w_{jk}^2} = \frac{1}{2\sqrt{\sum_k w_{jk}^2}} \frac{\partial}{\partial w_{ji}} (w_{j1}^2 + w_{j2}^2 + \dots + w_{ji}^2 + \dots + w_{jm}^2) = \frac{w_{ji}}{\|\mathbf{w}_j\|}$$

by keeping N and X apart as much as possible, so that at convergence, N and X indicate the correct separating hyperplane.

5.1.2.2 An Error Function which is Zero for all Correctly Classified Patterns

The weight change due to the deviation of the weight state from the surface of the current hypercone is added to the weight change due to misclassification of patterns. The latter is based on a modification of the standard sum of squared error from back-propagation.¹¹ The modified error function is zero for patterns on the correct side of the hyperplane.

Figure 5.9 compares various classification error functions. In standard back-propagation (a), the error function rises as the target is exceeded. This leads to the problem that the global minimum error point has misclassified patterns for certain classes of linearly separable pattern sets.¹² The proposal of Sontag and Sussmann¹³ (b) is to use an error function that cuts off this rise by giving zero error to patterns whose outputs exceed the target. The counter examples of Brady et al no longer apply. (Chapter 1 has more detail on this.)

Extending the cut-off point to 0.5 so that all patterns on the correct side of the hyperplane have an error of zero, yields the error function in (c). Using this error function, however, effectively trains all patterns to give an output tending to 0.5, which means zero error is achieved when the weights are all zero. This is undesirable, since the increasingly low gradients mean that the patterns may remain unseparated.

The problem is dealt with by using the error function in (d), which has relatively high gradients at an output of 0.5. The calculation of the gradient at this point forces patterns into the correct half of the output interval. Although the error function is discontinuous, it is possible to calculate a gradient at all points along the curve. The gradient at 0.5, and when the pattern is misclassified, is defined to be the same as the gradient of (a) at that point, and the gradient when the point is correctly classified

¹¹Rumelhart et al, 1986, p. 327

¹²Brady et al, 1988

¹³Sontag & Sussmann, 1988, 1989

(i.e. immediately beyond 0.5), is defined to be zero. For each point in weight space, each misclassified pattern contributes a direction to the suggested weight change. Steepest gradient descent on this error function always moves in the direction indicated by the misclassified patterns.

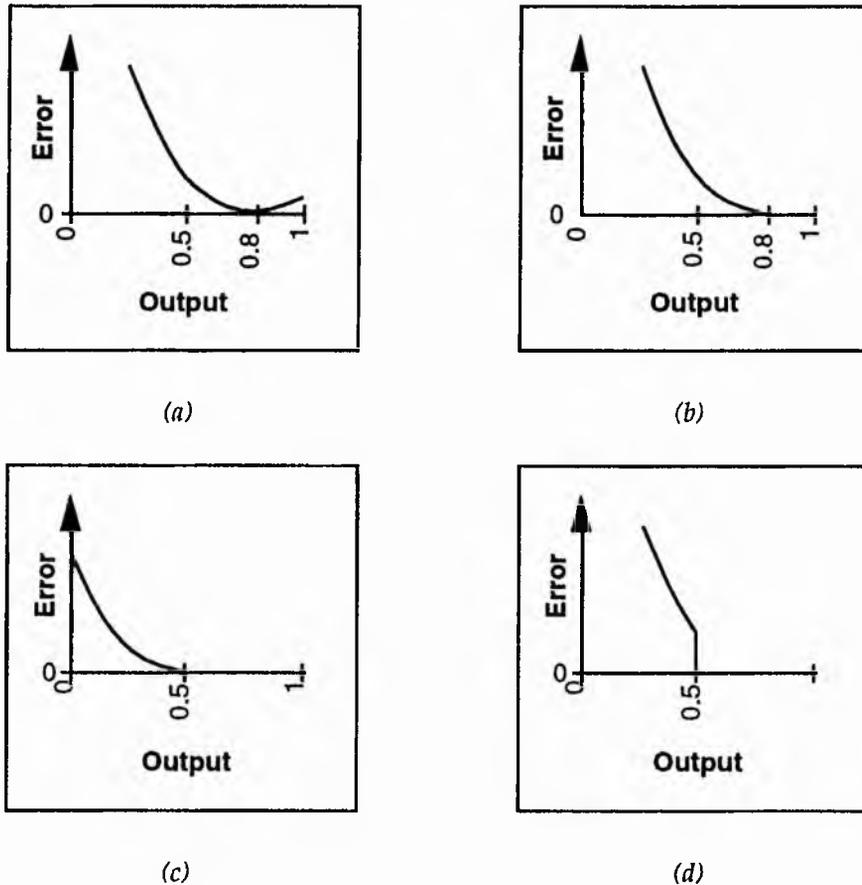


Figure 5.9 — Comparison of error functions, for a target of 0.8, and an output interval of $[0, 1]$. (a) Standard back-propagation error function. (b) Error function proposed by Sontag & Sussmann. (c) Extension of (b) so that there is zero error when the pattern is on the correct side of the hyperplane. (d) Error function used in this implementation.

This leads to an error surface which has sudden increases and decreases resulting in a jump in the error for small changes in the weights. Sudden decreases in error are not a problem, since it is an indication that a pattern has become correctly classified.

Sudden increases in error are more difficult to justify, since it indicates that a pattern, p , has become misclassified. However, the gradient after the

jump always leads to weight changes with a decrease in error — even though it may not point back to the correct classification of p if there is a greater gradient due to other patterns being misclassified. Sooner or later, once these patterns are correctly classified, p will be one of the only patterns left misclassified. The gradient to correctly classify p will become significant, and the weight path moves in that direction.

Since it is assumed that the problem to be solved is linearly separable, there is a region of zero error. The goal region, of correctly classifying weight states, is correctly represented by the global minima of the error function, since it is zero for all weight states in the goal region, and greater than zero for all other weight states.

Equation [5.11] gives the formula for the error function for pattern classification, E_{Cp_j} :

$$E_{c_p} = \begin{cases} \frac{1}{2}(t_{pj} - o_{pj})^2 & f(t_{pj}) \neq f(o_{pj}) \\ 0 & \text{otherwise} \end{cases} \quad [5.11]$$

where t_{pj} is the target of pattern p for unit j , o_{pj} is the output of unit j on presentation of the input of pattern p , and $f(x)$ is the following function:

$$f(x) = \begin{cases} 1 & x > 0.5 \\ -1 & x \leq 0.5 \end{cases}$$

5.1.2.3 Combining Correct Classification with Minimum Distance from the Current Hypercone

The total error is then the sum of the error due to misclassification of the patterns, and the distance in terms of angle of the current weight state from the surface of the current hypercone. This is given in equation [5.12], where p iterates over the patterns, and E_{D_j} is the measure of the distance of the current weight state for unit j from the current hypercone discussed in section 5.1.2.1, and defined in [5.8].

$$E_j = \sum_p E_{c_p} + E_{D_j} \quad [5.12]$$

In the symbolic technique, updating operates over a discrete search space. However, since weight changes, and hence changes to the concept, are made in a continuous search space, updating is a more gradual process.

With the overall error, E_j above, the network enters a local minimum when the drive to correctly classify patterns is matched by the drive to maintain the weight state on the current hypercone. At this point, the new pattern is not yet correctly classified, and this is the prime imperative. The hypercone must be updated, so that the ability of the network to reach a point whereby the new pattern is correctly classified is not impaired.

Therefore, when a local minimum is suspected, the current weight state becomes v_j , the reference vector for the hypercone. This means that the deviation from the hypercone becomes zero, since the hypercone has been moved to the position of the current weight state. The path to correctly classify the new pattern can then proceed. This is illustrated in figure 5.10.

There are two alternative heuristics for local minimum detection. One is to wait until the total error change becomes very small. Another is to wait until a weight change results in an increase in error. The problem with the former test is that it is possible for the weight path to be oscillating along the bottom of a gently sloping ravine,¹⁴ in which case, the error changes are small, but the network is not stuck in a local minimum. Also, the sudden jump in error with the error functions used here means that local minima cannot always be detected by small changes in error. Hence the latter method is used.

This, however, is subject to the problem that the jump up in error need not necessarily be due to the weight path being in a local minimum, and could be simply in the natural course of events as the weight path proceeds towards zero error. However, this technique uses incremental learning. This means that when a new pattern is introduced, the error will either be zero (if the pattern happens to be correctly classified by the current weight state), or there will be a direction of change due to the misclassification of the new pattern. This direction of change could be supposed to be unlikely

¹⁴Ochiai et al, 1994; Hertz et al, 1991, p. 129

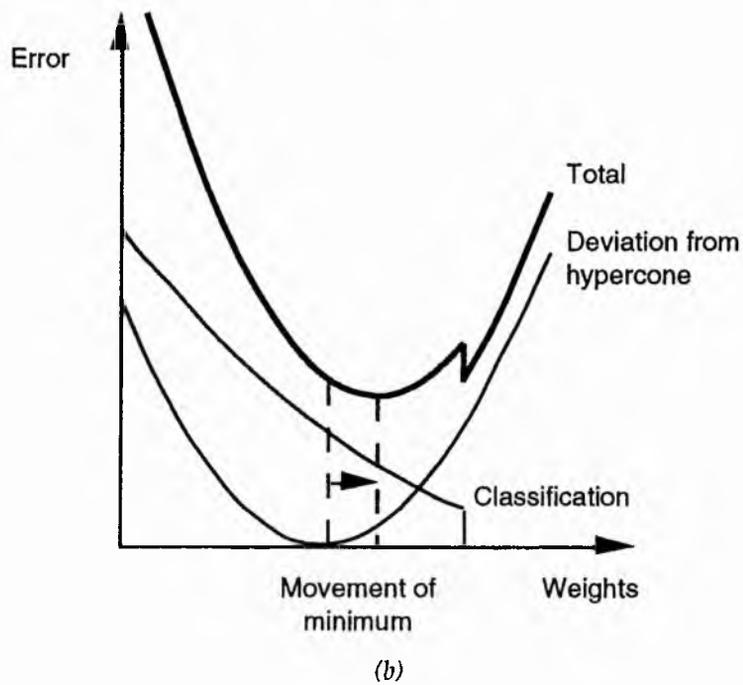
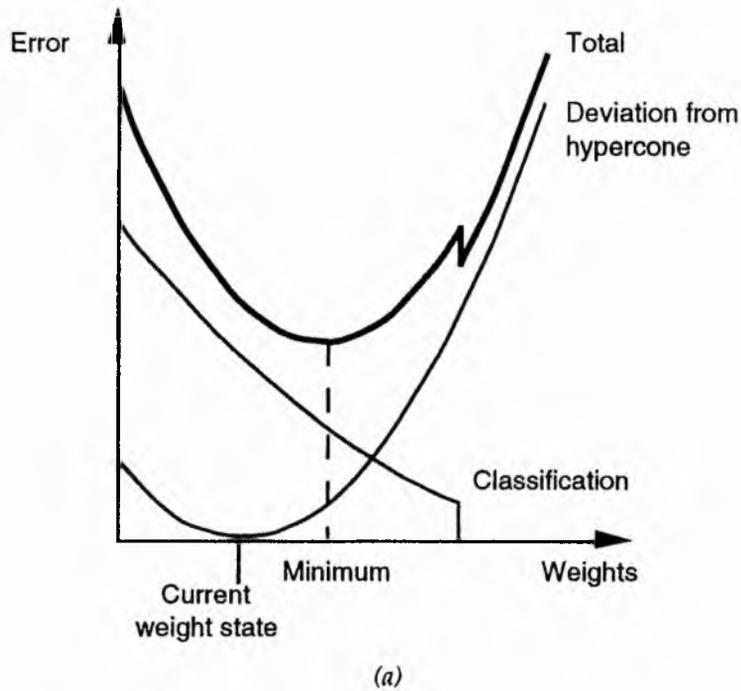


Figure 5.10 — Continuous updating. (a) At the minimum of classification and deviation from hypercone error, the pattern is not correctly classified. (b) The current hypercone is reset, and the minimum is moved towards correct classification. After sufficient hypercone resettings, the pattern is correctly classified, but with the minimum change in angle.

to point back towards the misclassification of the old patterns, and hence the likelihood of a sudden jump up that is not due to a local minimum is reduced. Since updates are small, any false updates relating to this problem will not cause difficulties during learning. This will be discussed later in the light of experimental evidence (section 5.2.3).

The training procedure makes use of functions which are designed with the intention of ensuring that a weight state is found using the shortest path (in terms of angle) from the weight state after the last update to a correctly classifying weight state. The effectiveness of the functions used here in achieving this is discussed in section 5.2.2.

5.1.3 Relation to the Symbolic Technique

This technique deviates from the original symbolic technique in four main ways, summarised as follows: Firstly, in the nature of the search space, which is a discrete graph of states in the symbolic technique, and a continuous multi-dimensional space in this neural implementation. This means that the style of search, and the means of representing version space boundaries, updating and selection within in the neural technique will differ significantly from the symbolic case. Thirdly, there is a difference in the ordering of the concepts. Finally, the neural technique cannot make use of partially learned concepts, as is possible with the symbolic technique.

The similarities lie in the incremental presentation of patterns, the convergence of version space boundary representatives, and in the ability to recognise the no-alternative situation.

The ordering in the neural technique is a total ordering of the angle of weight vectors measured relative to an arbitrary constant reference vector in n dimensional space, \mathbf{n}_0 . A relation is a total ordering if it is reflexive, antisymmetric, transitive and connected.¹⁵ Let $\mathbf{a} R \mathbf{b}$ represent the relation that the angle of \mathbf{a} relative to \mathbf{n}_0 is greater than or equal to the angle of \mathbf{b} relative to \mathbf{n}_0 , as per equation [5.13], where \mathbf{a} and \mathbf{b} are weight vectors:

¹⁵Borowski & Borwein, 1989, p. 422

$$\mathbf{aRb} \Leftrightarrow \cos^{-1}\left(\frac{\mathbf{a} \cdot \mathbf{n}_0}{\|\mathbf{a}\|\|\mathbf{n}_0\|}\right) \geq \cos^{-1}\left(\frac{\mathbf{b} \cdot \mathbf{n}_0}{\|\mathbf{b}\|\|\mathbf{n}_0\|}\right) \quad [5.13]$$

This relation is clearly a total ordering of angle, since greater than or equal to forms a total ordering of real numbers. However, it is not a total ordering of weight vectors. Consider the case of antisymmetry:

$$(\forall \mathbf{a}, \mathbf{b} \in \mathbf{R}^n)((\mathbf{aRb}) \wedge (\mathbf{bRa})) \Rightarrow \cos^{-1}\left(\frac{\mathbf{a} \cdot \mathbf{n}_0}{\|\mathbf{a}\|\|\mathbf{n}_0\|}\right) = \cos^{-1}\left(\frac{\mathbf{b} \cdot \mathbf{n}_0}{\|\mathbf{b}\|\|\mathbf{n}_0\|}\right) \quad [5.14]$$

From [5.14], it cannot be inferred that $\mathbf{a} = \mathbf{b}$, since \mathbf{a} and \mathbf{b} may be at different points on the hypercone. Since antisymmetry is also a necessary condition for a partial ordering, R does not form a partial ordering of weights either. Therefore, the use of weight vector angle does not give any ordering, either total or partial, to the weight vectors themselves.

As discussed in chapter 3, the neural equivalent of a concept is the IO picture. The basis of this technique is to give an ordering to the IO pictures through giving an ordering to the weight space angle. Although it is the weight space angle that is ordered, and not the weights or the IO pictures, it is nevertheless the case that this technique provides a mechanism for constraining the search through weight space such that the no-alternative situation can be recognised.

The continuous nature of the search space in the neural technique means that the style of the search is very different. Mitchell always exhaustively searches the boundary sets, and explicitly represents each boundary member of version space. This is not done in the neural case. The boundaries of version space in the neural case are the surfaces of the hypercones of N and X . It is infeasible and unnecessary to exhaustively search these. This is because the weight states of N and X are free to move in weight space, and are not restricted to particular chains in the partial ordering, as the boundary representatives are in the symbolic case. This means that one neural network representative is sufficient for each boundary. The neural network will move through weight space towards zero misclassification error at any stage during training. It is constrained using the angle measure, but by updating this, the ability to find zero misclassification error is not impaired.

The symbolic mechanisms of updating and selection within ensure that the minimum changes to the boundary sets S and G are made when a new instance is presented. The purpose of the angle function is to provide the means of ensuring that a new, correctly classifying weight state is found with minimum local change in weight space angle.

Whereas the symbolic technique is able to make use of partially learned concepts through the agreement of S and G on the classification of unfamiliar instances (see chapter 2), there is not the same usefulness in the neural case. The concept must lie between N and X , but this does not mean that if N and X agree on the classification of an untrained pattern that this pattern must therefore have that classification, because the hyperplanes of N and X do not represent the extremes of freedom of movement — they represent the extremes of weight space angle measured relative to an arbitrary weight vector. The two do not necessarily equate. Figure 5.11(a) shows the freedom of movement of the hyperplane for a problem in a 1D input space. Figure 5.11(b) shows the corresponding correctly classifying region of weight space. If the reference vector, N_0 , happens to correctly classify, as chosen in the diagram, then N stays at a minimum angle of 0 from N_0 , but does not represent an extreme of freedom of movement. X does represent one of the extremes of freedom of movement — that at maximum angle in weight space from N_0 .

There are similarities between the symbolic technique and the angle technique. A key similarity is that patterns are presented incrementally. A pattern is added, and if misclassified, the network is changed until the pattern is correctly classified, through a series of updates. There is, however, a difference in that previous patterns must also be retrained, in order to ensure that correctly classifying the new pattern does not lead to misclassification of old ones. This is because the correctness of the classification does not depend on the hypercone. Thus updating the hypercone for the new pattern alone does not of itself preserve correct classification of previous patterns, as is the case in the symbolic technique.

A further similarity is that the boundary representatives always move closer together in the ordering. This has been proved in section 5.1.1.1.

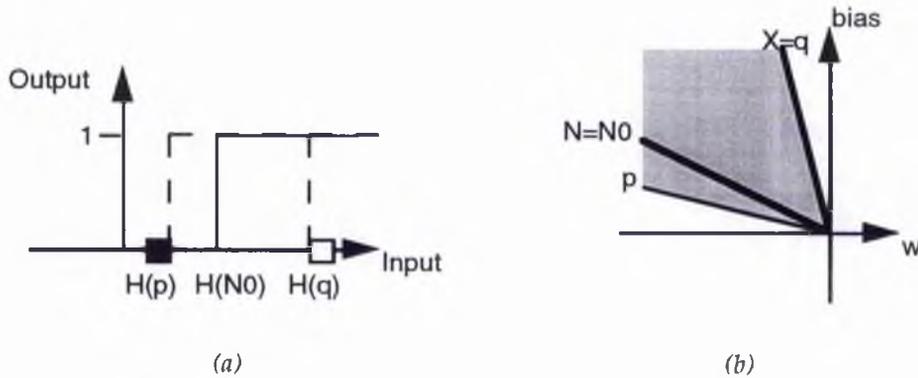


Figure 5.11 — The extremes of angle that correctly classify relative to N_0 need not be the extremes of freedom of movement. (a) A 1D problem in IO space. The hyperplane of N_0 , $H(N_0)$ happens to correctly classify the patterns (black/white squares). The extremes of freedom of movement are indicated by the hyperplanes $H(p)$ and $H(q)$. (b) Weight space. The perceptrons to perform the separation have a weight, w , to the input unit, and a bias weight. The weight vectors which correspond to separations $H(p)$ and $H(q)$ in (a) are indicated by p and q . The shaded region indicates all weight states that lie between p and q , and therefore correctly classify. N and X are indicated by bold lines.

5.2 Algorithm and Experimental Results

5.2.1 Algorithm

The algorithm for the implementation of the weight space technique is as follows. Note that in step 3 N_0 is set to N 's first trained weight state on the first pattern from step 2. N must be trained close to place the first pattern close to the boundary, as explained later, and illustrated in figure 5.13. Steps 2 and 3 ensure that no potential weight states are excluded from being searched.

1 Variables:

- 1.1 Let T = the training set.
- 1.2 Let P = the number of patterns.
- 1.3 Let C = the misclassification error.
- 1.4 Let D = the angle error, based on the angle between the hypercone reference vector and N_0 .

```

1.5   Let E = an error function to train the first
      pattern to be correctly classified and close to the
      current hyperplane of the network.
1.6   Let N = the internal hypercone network.
1.7   Let N0 = the reference vector.
1.8   Let Nh = the hypercone reference vector of N.
1.9   Let X = the external hypercone network.
1.10  Let Xh = the hypercone reference vector of X.
2     Train(T(0), T(0), N, E, _).
3     Let N0 = N.
4     Initialise X by setting all the weights of X to be -1
      multiplied by the corresponding weight in N0.
5     If T(0) is misclassified by X,
      Train(T(0), T(0), X, C + D, Xh).
6     For each subsequent pattern, T(n):
6.1   If T(n) is correctly classified by N, go on to 6.2
      else Train(T(0), T(n), N, C + D, Nh).
6.2   If T(n) is correctly classified by X, go on to 6.3
      else Train(T(0), T(n), X, C + D, Xh).
6.3   If N and X are collinear (within an acceptable
      tolerance) locking has been achieved. Terminate.
      Otherwise n = n + 1, and repeat from 6.1 until
      n > P.
7     Locking not achieved for patterns given. Stop.
8     Procedure:
      Train(From_pattern, To_pattern, Network, Error_function,
            Hypercone_network)
8.1   Train Network on all patterns from From_pattern to
      To_pattern using Error_function, until a minimum is
      detected.
8.2   If Error_function == E return.
8.3   Update hypercone. Hypercone_network = Network.
8.4   Repeat from 8.1 until C = 0.

```

The error functions C and D refer to the misclassification error, E_C , and the angle cost function, E_D , discussed in section 5.1.2.

The error function E in the algorithm above is simply the standard back-propagation error function, but with a target of 0.5 for the input of pattern $T(0)$. This trains the network to put $T(0)$ precisely on its hyperplane. This is necessary in order to enable N and X to have the maximum initial angle (π) from each other, whilst still correctly classifying the first pattern. If the first pattern is some distance from the hyperplane of N_0 , then it is not possible to have the initial weight vector of X close to an angle of π from the weight vector of N_0 . Figure 5.12 illustrates this.

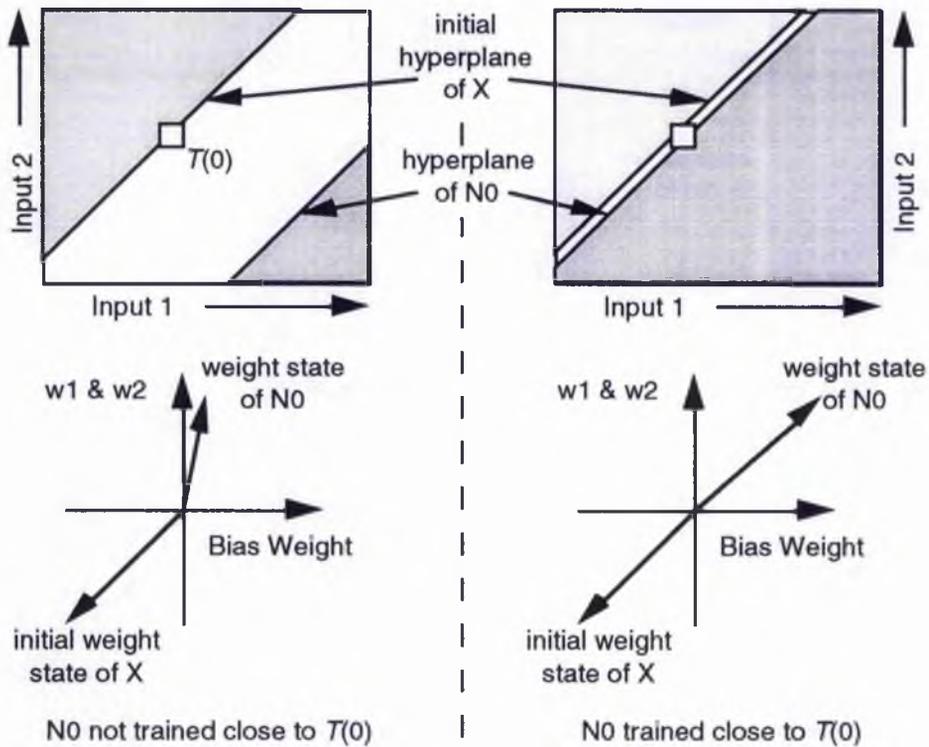


Figure 5.12 — The difference in initial weight space angle between X and N_0 when N_0 is not trained to be close to the first pattern, $T(0)$, as shown on the left, and when N_0 is trained to be close to $T(0)$.

If the initial weight vector of X is not close to an angle of π from the weight vector of N_0 , then it is possible for weight states at angles greater than the actual limit after $T(0)$ is introduced to be left untested. Alternatively, it is possible for the angle between N and X to increase. This is illustrated in figure 5.13, which shows that if $T(1)$ is presented as shown in figure 5.13(a), then N moves from N_0 to correctly classify, whilst X is unchanged, since it already correctly classifies, thereby increasing the angle between N

and X , as shown in figure 5.13(b). This is contrary to the requirement of the paradigm for a monotonic decrease in angle.

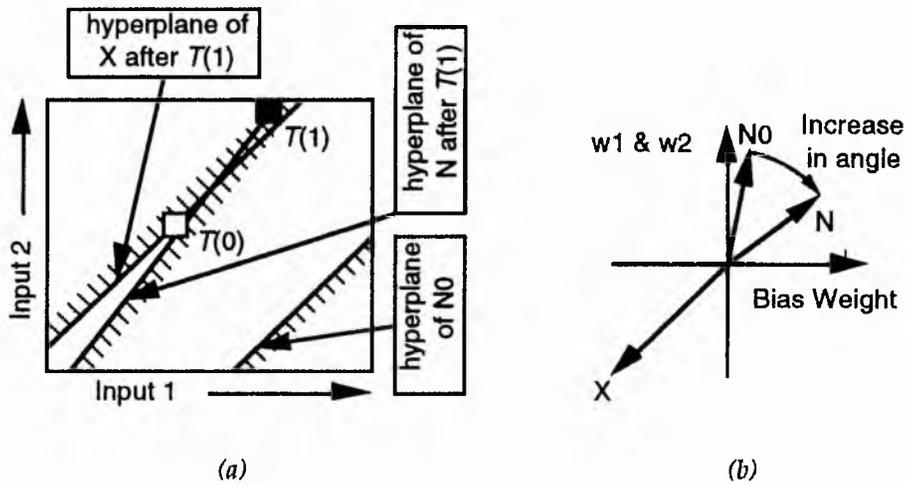


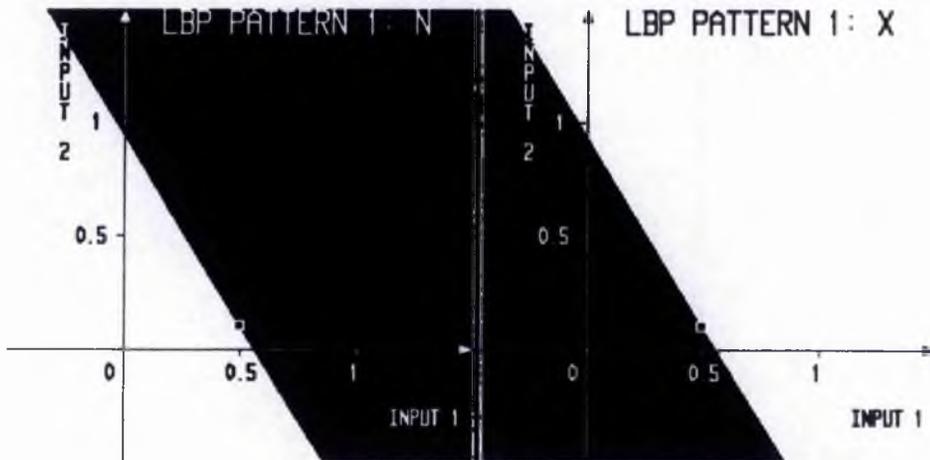
Figure 5.13 — (a) N moves from N_0 to correctly classify $T(1)$. The shading represents the side of the hyperplane with classification 1. (b) This move results in an increase in angle between N and X .

5.2.2 Results for a Simple Experiment

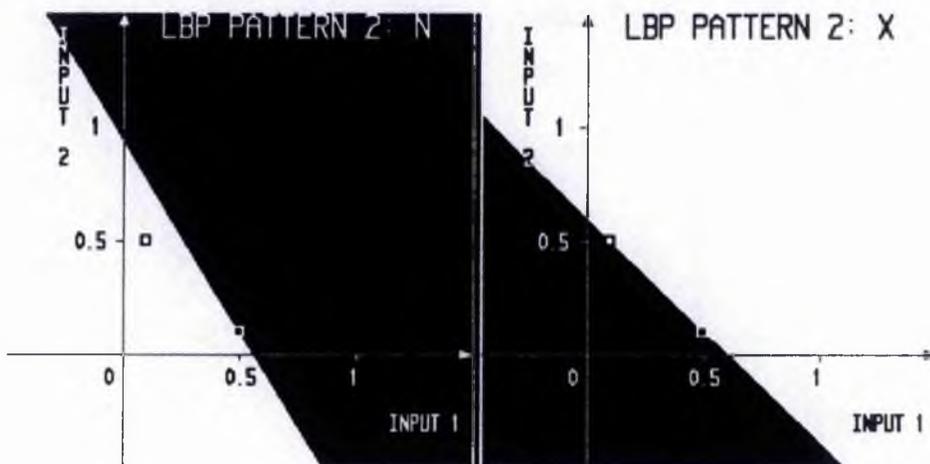
In this experiment, the technique is demonstrated for a simple 2D linear binary partition, which is learned over a sequence of five patterns. This shows the ability of the technique to achieve locking in low dimensional input space using relatively few patterns. All that is required for the hyperplane is enough patterns to pin the hyperplane in place. In 2D, only three patterns, two of one class, and one of the other class, all close to the hyperplane boundary, are required. Two extra patterns are used at a distance from the hyperplane boundary. These are not strictly necessary, but act as an aid to guiding the hyperplane in place.

Figures 5.14(a) to (e) show the IO sequence of N and X as they converge to the same hyperplane. The problem is clearly a very simple one, yet it serves well as a demonstrator of the way in which the technique works.

Figure 5.14 — Sequence of IO graphs for N and X leading to convergence. The output of the network, with two input units and one output unit is plotted for each point in input space covered by the region of the graph. Where the output is one a black point is plotted, and where it is zero, a white point is plotted.

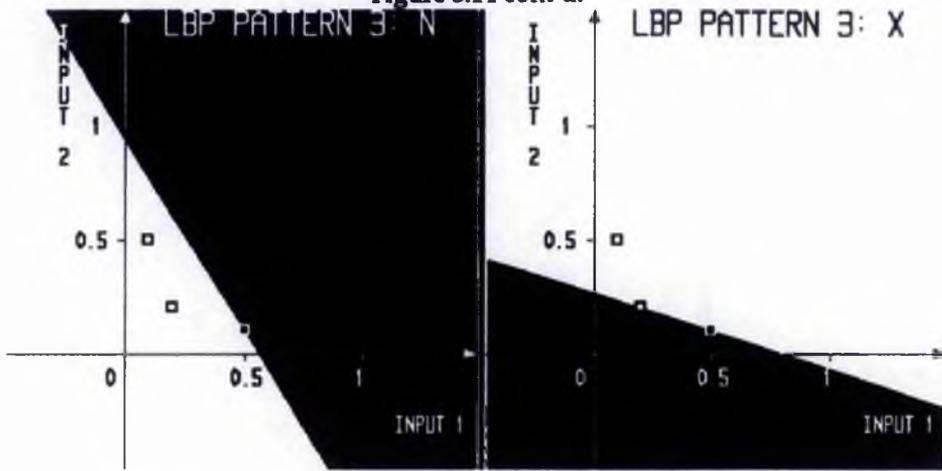


(a) The initial condition. The weights of N and X are virtually opposite, both just correctly classifying the first pattern, close to the border, X having been minimally updated to correctly classify the pattern after its weights were all set to -1 multiplied by the corresponding weight of N . The pattern is marked by a black square, with a white border, indicating a target of 1.

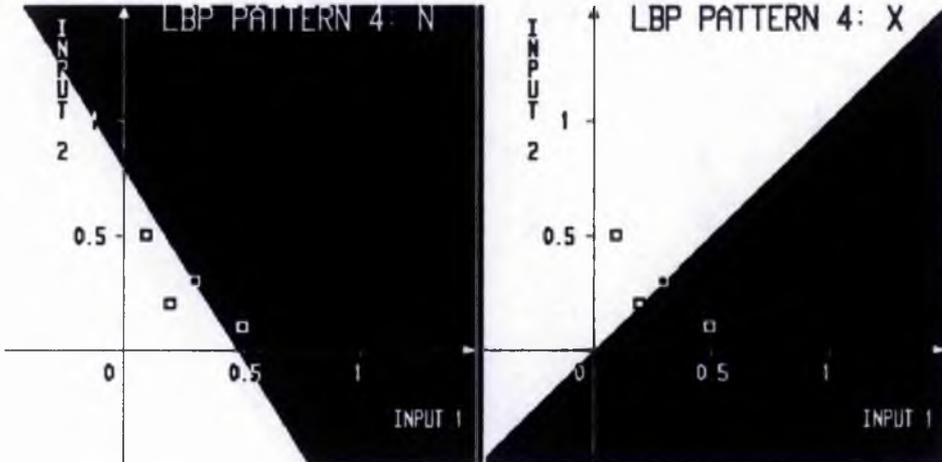


(b) The second pattern is introduced. N correctly classifies it already, hence there is no change. X misclassifies the pattern, and must move to correctly classify it. This is done with the minimum change in weight space angle. The new pattern is marked by a white square with a black border, indicating a target of 0.

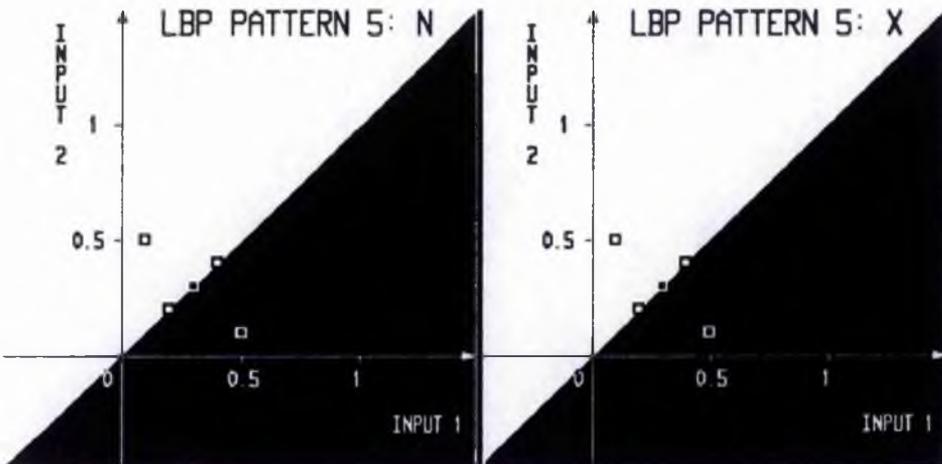
Figure 5.14 cont'd.



(c) The third pattern is introduced. Again, N correctly classifies this already, and does not change. X must change again, however.



(d) The fourth pattern is introduced. N and X move to correctly classify it.



(e) N and X converge after N moves to correctly classify the fifth pattern.

The nature of the classification error function means that misclassified patterns are always trained to be close to the boundary of the hyperplanes. However, since there is no change if the pattern is correctly classified, N and X do not show the extent of the freedom of movement of the hyperplanes for the patterns under consideration. This is clear from figure 5.14(b), where the IO behaviour of N would need to be nearly the opposite to that of X in order to indicate the freedom of movement.

5.2.3 Problems with the Technique

One of the main problems with the technique is that the detection of local minima is far more problematic than originally suspected. The naive views expressed in section 5.1.2 do not match with experimental results. Updating occurred as often as once out of every three training cycles in some runs. The reason for this is precisely due to the fact that weight states are always found that correctly classify the new pattern with the minimum change in angle.

Figure 5.15 shows a two dimensional weight space. Each line represents the boundary in weight space between the correct classification and the misclassification of a pattern. This boundary may be termed the solution manifold.¹⁶ The arrows show the direction of change that will be indicated by the error function. If a weight state just correctly classifies $P1$, the first pattern, it is close to the solution manifold for that pattern. When the second pattern, $P2$, is introduced, the direction of change will be in accordance only with attaining the correct classification of this pattern, since $P1$ is correctly classified. This leads to the misclassification of $P1$, and the error makes a sudden jump up. The new direction points back to the correct side of the solution manifold of $P1$, and the oscillation continues.

This kind of oscillation means that the method chosen in section 5.1.2 (which used an increase in error as an indication) cannot be used to detect a local minimum. For each oscillation, the error increases when the weight path moves to correctly classify $P2$ only, and misclassifies $P1$ as well as a result. This leads to a false update.

¹⁶Fernandez, 1994

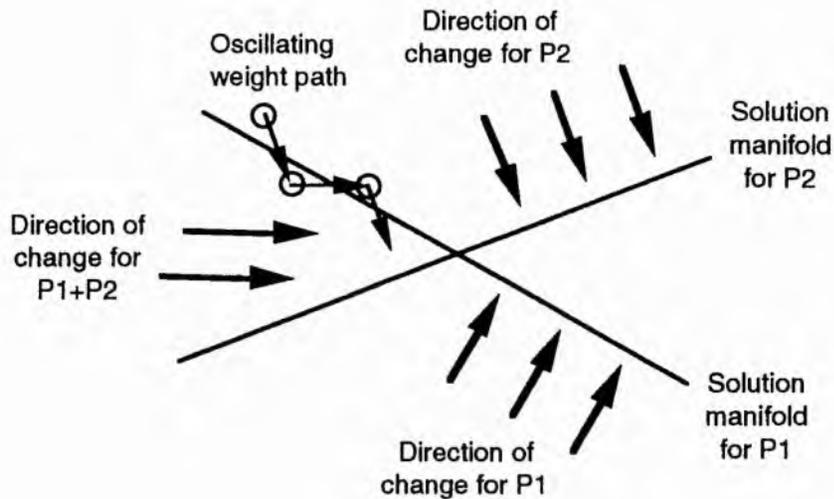


Figure 5.15 — *How the weight path can oscillate along the solution manifold of the last correctly classified pattern (P1) whilst trying to correctly classify the new pattern (P2). The circles indicate a series of weight states, and the arrows between the circles indicate the weight change.*

Since the technique used an increase in error to indicate the need to update, and updating occurred frequently, the deviation from the hypercone rarely had a significant effect on the weight change. However, the results in section 5.2.2 show the technique converging. How can this be? The answer lies in the fact that the weight states have started from an initial angle of π . This, coupled with the fact that the misclassification error alone ensures the minimum change is made that allows for correct classification of the patterns (provided that the learning constant is small), means that the technique can work fine without using angles at all. Subsequent experiments without using the deviation from the hypercone to constrain the weights have confirmed this.

There is therefore some doubt about whether the angle cost function has a useful effect on the weight path in practice. The simple nature of the IO problems this technique can cope with means that there is no need to further constrain the search using the deviation from the hypercone. Starting from opposite directions, and using an error function for classification which drops to zero once a pattern is correctly classified, is sufficient to prevent either network from going beyond the goal angle, and therefore from allowing a false locking situation to arise.

A further problem is the dependence of the technique on the accuracy of the patterns. Such assumptions may be acceptable in symbolic AI, but neural networks tend to be used for problems where the data cannot be guaranteed 100% noise free. In noisy situations, or situations whereby the neural architecture cannot provide an exact fit to the pattern set, it could be hoped that the technique might provide a "best fit" to the underlying data. However, as illustrated in figure 5.16, it is possible to have patterns from an unrealisable training set which have fixed the hyperplane such that there is no freedom of movement, and yet the hyperplane is not in the position of best fit to the underlying function.

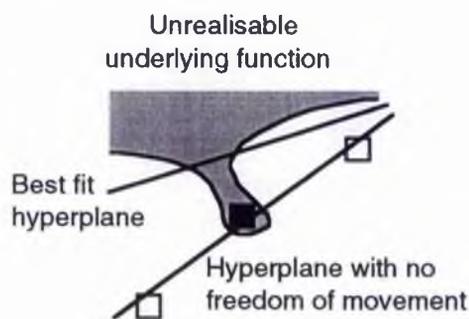


Figure 5.16 — *A hyperplane with no freedom of movement with a non-optimum position with respect to the underlying function.*

It was especially this point, combined with the difficulties in generalising the technique for hidden units outlined in the following section, that led to the development of the technique outlined in chapter 6.

5.3 Generalisation For Hidden Units

A great problem for the technique is the limitation to the networks it can be used to train. As mentioned in chapter 1, neural networks without a hidden layer cannot solve linearly inseparable problems. This means that there is a severe constraint on the problems that can be solved with this technique. This section documents some of the problems experienced with generalising the technique to apply for hidden units, which led, in combination with the problems outlined in section 5.2.3, to the development of a different neural implementation of Mitchell's technique, to be discussed in chapter 6.

5.3.1 Symmetries in Hidden Units

Symmetry of hidden units is an important issue when considering a neural implementation of Mitchell's technique based in weight space, where a concept is linked to IO behaviour. Given the directive that all equivalent solutions in terms of IO must have the same measured value in the partial ordering chosen, it is necessary to be sure that the partial ordering used takes into consideration all equivalent weight states.

The symmetries of weight states for neural architectures containing hidden units are indicated by Denker et al,¹⁷ and other authors.¹⁸ Whenever an ordering is given to the hidden units (such as in software simulations) there is a symmetry, since any re-ordering of the units in a given hidden layer leaves the output behaviour unchanged. This will be termed *unit ordering symmetry*.

There is also symmetry due to unit polarity. If the activation function has rotational symmetry about the origin, then changing the signs of the fan-in and fan-out weights will yield the same output behaviour.

If the symmetry of the activation function is about the intercept then the same kind of equivalence exists, but an adjustment to the bias weights of the units in the higher level is necessary¹⁹. This will be termed *unit polarity symmetry*. For example, consider the sigmoid activation function [5.15], which has rotational symmetry about the point $(net_j, out_j) = (0.0, 0.5)$:

$$f(net_j) = \frac{1}{1 + e^{-net_j}} \quad [5.15]$$

where net_j and out_j are the excitation and output of a unit, u_j .

If the signs of the fan-in weights w_{ji} to a unit u_j are all changed, then the effect is to change the sign of the excitation of u_j , net_j . Changing the sign of the fan-out weights, w_{kj} from u_j does not compensate entirely for this, since:

¹⁷Denker et al, 1987, pp. 886-887

¹⁸E.g. MacKay, 1992b, p. 455; Smieja & Mühlenbein, 1990, p. 261; Sussmann, 1992, p. 590

¹⁹Denker et al, 1987, p. 886

$$\frac{-w_{kj}}{1 + e^{net_j}} \neq \frac{w_{kj}}{1 + e^{-net_j}}$$

A correction, c_k , is needed to the bias weight of each fan-out unit, u_k :

$$c_k - \frac{w_{kj}}{1 + e^{net_j}} = \frac{w_{kj}}{1 + e^{-net_j}}$$

Rearranging:
$$c_k = \frac{w_{kj}}{1 + e^{-net_j}} + \frac{w_{kj}}{1 + e^{net_j}} \quad [5.16]$$

Multiplying the top and bottom of the right hand term on the right hand side of [5.16] by $\exp(-net_j)$:

$$\begin{aligned} c_k &= \frac{w_{kj}}{1 + e^{-net_j}} + \frac{w_{kj}}{(1 + e^{net_j}) e^{-net_j}} \\ \Leftrightarrow c_k &= \frac{w_{kj}}{1 + e^{-net_j}} + \frac{w_{kj} e^{-net_j}}{e^{-net_j} + 1} \\ \Leftrightarrow c_k &= \frac{w_{kj} (1 + e^{-net_j})}{1 + e^{-net_j}} \\ \Leftrightarrow c_k &= w_{kj} \end{aligned}$$

This result is summarised in equation [5.17] for the sigmoid activation function in [5.15].²⁰

$$w_{kj} f(net_j) = -w_{kj} f(-net_j) + w_{kj} \quad [5.17]$$

A further symmetry, not covered by Denker et al, is due to the nature of a class of problem that neural networks with hidden units are capable of solving. This exemplified by the approximation of a circle, for 2D input and one output unit, to an arbitrary degree of accuracy (see figure 5.17). Whether sigmoidal or threshold activation functions are used, a given degree of accuracy can be achieved using n hidden units in a single hidden layer, whose hyperplanes are arranged such that they combine to form an n -sided regular polygon around the circle.

²⁰Balchin, 1993

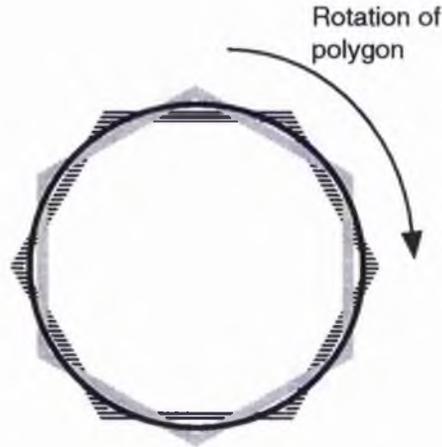


Figure 5.17 — *Rotational symmetry of a problem. A circle is being approximated by the hyperplanes of six hidden units forming a hexagon. Any angle of rotation of the hexagon yields the same approximation.*

By rotational symmetry, there are an infinite number of orientations of a given polygon that will approximate the circle to the same degree of accuracy. (See figure 5.17.) This will be termed *exact IO behaviour symmetry*. This presents a rather awkward challenge for a Mitchellian technique based on weight space. There is not an obvious method for mapping weight states that are equivalent in this way onto the same point in an ordering.

5.3.2 Measuring Weight Space Angle with Hidden Units

When hidden units are used, the symmetries outlined in the previous section must be overcome whilst preserving the directive that weight states with equivalent IO behaviour must have the same position in the partial ordering. This means a decision must be made as to how the angle is to be measured. The most obvious method is to form a hypercone over the entire weight space for all units. This could be termed a network-based approach. Other approaches could use a sum of angles over hypercones for all the units in a layer, a sum of angles over hypercones for each unit, or even an approach based on links alone.

Each of these alternatives must be considered to see if they break any of the symmetries discussed in the previous section.

- The network and layer based approaches both involve putting hypercones over more than one unit. However, it can be demonstrated (figure 5.18) that hypercones over more than one unit cannot cope with unit ordering symmetry.
- The unit based approach would use the sum of the angles to give the position in the ordering. This cannot cope with unit ordering symmetry either, as shown in figure 5.19. If the fan-in and fan-out weights of units 1 and 2 are swapped, the angle of unit 3 will have a different value. Therefore although the sum of angles for units 1 and 2 remains constant, the sum of angles for units 1, 2 and 3 does not.
- The link based approach cannot use hypercones, and would instead try and keep the difference in the sum of the distances between the current weight value and a reference weight value minimal, as per equation [5.18], where E_D^{link} is the sum of distances, v_j is the weight value after the last update, and m is the number of weights in the network:

$$\sum_{j=1,m} (|n_{0j} - v_j| - |n_{0j} - w_j|)^2 = E_D^{link} \quad [5.18]$$

Multiplying w_j by a positive constant yields a different value for E_D^{link} and hence the link based approach cannot cope with weight scaling symmetry — the original motivation for creating hypercones in the first place.

Therefore, hypercones cannot be used with hidden units without encountering difficulties with one of the many neural symmetries that occur once hidden units are allowed.

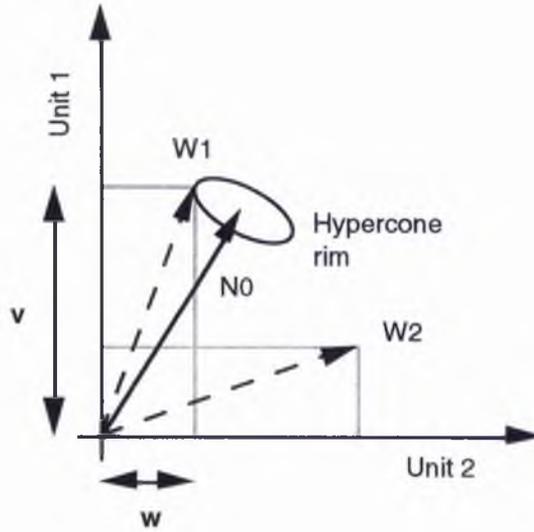


Figure 5.18 — Two weight states, $W1$ and $W2$, equivalent by unit ordering symmetry do not lie on the surface of the same hypercone, where that hypercone has been drawn over the weights of both units combined. $W1$ is a weight state with weight vector v to unit 1 and weight vector w to unit 2. Provided that the fan-out weights of units 1 and 2 are swapped, $W2$ is an equivalent weight state with weight vector w to unit 1 and v to unit 2. $W2$ does not lie on the surface of the same hypercone over units 1 and 2 as $W1$, and hence network, or layer based approaches, which have hypercones over more than one unit, do not have all equivalent weight states that lie on the surface of the same hypercone.

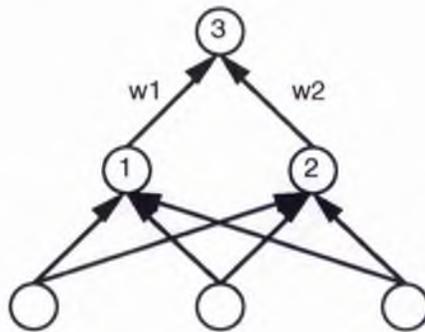


Figure 5.19 — A subsection of a neural architecture. $w1$ is the fan-out weight of unit 1 to unit 3, and $w2$ is the fan-out weight of unit 2 to unit 3.

5.4 Conclusion

The original purpose of developing this technique was to overcome weight scaling symmetry, which had been a problem for an earlier attempt to implement a Mitchellian style learning algorithm in neural networks. Hypercones are effective in this, and hence the technique achieves what it sets out to achieve. Although the angle ordering does not give an ordering of weight states, or of IO pictures, it does enable the detection of the no-alternative situation for a restricted class of problems.

With hindsight, there are ways in which the implementation could have been improved:

- The error function of Sontag and Sussmann (figure 5.9(b)) could have been tried. There was no essential need for the discontinuous error function. Although Sontag and Sussmann's error function gives an error to patterns which are on the correct side of the hyperplane, the weights can be scaled up, so that patterns close to the hyperplane have outputs close enough to 1 or 0 to be recognised as correctly classified. This would not have jeopardised the use of hypercones, which are designed to cope with weight scaling.
- Local minimum detection with the discontinuous error function need not be indicated using the error, and thus avoid the problems posed by discontinuity. A small change in the weights could be used instead, since it also indicates a shallow gradient. Also, with Sontag and Sussmann's error function, there are no problems with discontinuity, and hence a local minimum can be detected more reliably by a small change in error.

It seemed best to give an ordering to the weight space angle, since it is the weights that give rise to the IO behaviour (for a given neural architecture). Although the problems with the earlier implementation had been dealt with, the weight space angle technique brings to light several other problems. These indicate important principles for implementing Mitchell's technique in a neural environment:

- A weight space implementation must take into consideration the symmetries in weight space of equivalent solutions. These symmetries were brought to light through working on this technique. The inability of the present technique to use hidden units because of the symmetries is an unacceptable restriction.
- Neural networks are rarely given noiseless data, for which an exact fit guaranteeing good generalisation can be attempted. Ideally, a neural implementation of Mitchell's technique should be able to cope with noisy data by providing a good, inexact fit. Figure 5.16 shows that the weight space angle technique based on freedom of movement of the hyperplane cannot cope with inexact fit.

It is clear that a weight space implementation is unlikely to yield satisfactory results for a large enough subset of problems. It is necessary to look at ways of using IO space as the space of search for a Mitchellian implementation. If a method could be found of recognising when two neural networks have equivalent IO, without the need to compare their weight states, then the various symmetries uncovered during the process of this work can be ignored. It is also necessary to provide the basis for recognition of equivalent inexact fits to the data. The following chapter details the technique that arose from these thoughts.

6 Neural Implementation Based on IO Space

This chapter explores the development of a neural Mitchellian technique which uses IO space as the basis for the partial ordering of concepts rather than weight space. The result is a technique that is able, under certain conditions, to indicate when the global minimum misclassification error solution has been found to the given problem. This means that it is possible to guarantee the best fit possible to the data with the given fixed topology and a standard neural training algorithm. Generalisation is thereby guaranteed according to the assumptions of the user. These assumptions will be made clear in later sections, and pertain to the topology chosen, and the degree of accuracy in the input patterns.

The chapter is divided into two parts. The first part discusses the development of the theory, in the context of three paradigms which are based on IO space, each one building on the previous paradigms. The second part details experiments using the final paradigm, illustrating its ability to find when the best fit to the data has been found using a Genetic Algorithm (GA) to train the network. The technique is also compared with validation.

(i) Development of Theory

6i.1 Partially Ordering the IO

6i.1.1 Representing the IO

The weight space implementation described in chapter 5 suffered from various weight space symmetries. The techniques outlined in this chapter aimed to overcome these symmetries by using IO space as the basis for the Mitchellian search rather than weight space. The difficulty with using IO space is that there is no obvious way to give a partial ordering to the IO. Consider the four IO pictures for various neural topologies drawn in

chapter 3. (Figure 3.1.) How can one say that one of these pictures is partially ordered in some way relative to another?

The final solution to the problem arose out of an attempt to relate the generalisation error to the patterns. The generalisation error is the volume of input space that has a different classification from the underlying classification of input space. One of the problems with generalisation in neural networks is that a continuous picture is required from a set of discrete patterns. This is why the VC estimate of the number of patterns needed to bound the probability of worst-case generalisation is so high. (See chapter 3.) Effectively, it implicitly suggests that the only way to be sure of generalisation is to saturate input space with patterns so heavily that no other continuous IO picture is likely.

The number of patterns in the data set is finite and limited. The patterns nevertheless determine, through the trained weight state, the entire realised IO space. In a sense, a pattern stands for its local region of input space. The IO space can then be partitioned into regions, with one data point in each, using a Voronoi diagram,¹ for example. With a Voronoi diagram, the regions are set such that any point in a given region is closest to the pattern which was used to construct the region. Then, rather than calculating the error of a pattern on a *target - output* basis, as in back-propagation, calculate its error on the basis of the region of input space it represents. (See figure 6i.1.) By adding the regions of misclassification to provide a total, an approximation to the generalisation error is obtained. The approximation is valid up to the resolution provided by the data set.

Practically speaking the calculation of these Voronoi regions is somewhat infeasible, especially for higher dimensionalities of input space, and large numbers of patterns.² One way to get round this problem is to use a regular grid of patterns, which would enable each pattern to represent the same area of input space. Then, assuming unit error for each misclassified pattern, the generalisation error can be estimated. (See figure 6i.2.)

¹Voronoi, 1908

²Preparata & Shamos, 1985

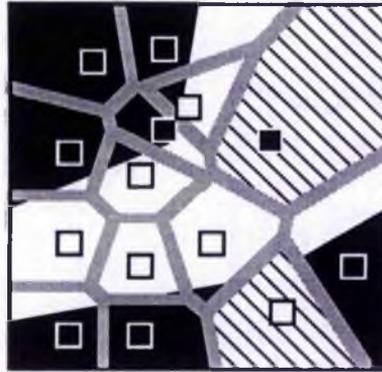


Figure 6i.1 — *Estimating the generalisation error by estimating the area of input space each misclassified pattern represents as being misclassified. The black and white areas indicate the underlying classification of the network, the striped areas indicate the generalisation error estimate.*

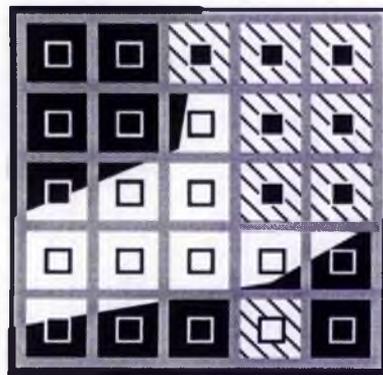


Figure 6i.2 — *Estimating the generalisation error using a grid. Since each gridpoint stands for the same area of input space, each misclassified gridpoint contributes the same error to the generalisation.*

Having considered the possibility of using a regular grid, further advantages of such grids become clear. Firstly, since a gridpoint is made to stand for a given area of input space, there is no need to saturate input space with patterns — leading to infeasible training times for a reasonable approximation to the underlying classification. Secondly, and more importantly for a Mitchellian approach, by sampling the input, a grid can be used to represent the IO relation of a neural network, in a manner which is independent of the symmetries in weight space. Given a partial ordering of the grids, it is no longer necessary to give an ordering to the weights, as was done in chapter 5, and hence the weight state by which

the IO is achieved is no longer a direct concern. This gives a method of representing the IO in a way which overcomes the difficulties encountered in chapter 5.

In chapter 5, the assumption was made that the data are noiseless, and an *exact fit* to the data was attempted. This assumption is not practical for real-world problems, in which noisy samples are to be expected. When noisy data are considered, there is the possibility of a further symmetry of solutions, which will be termed *inexact IO behaviour symmetry*. This occurs when a topology is incapable of realising a pattern set, but there is more than one way to achieve the global minimum misclassification error of the pattern set. (See figure 6i.3.)

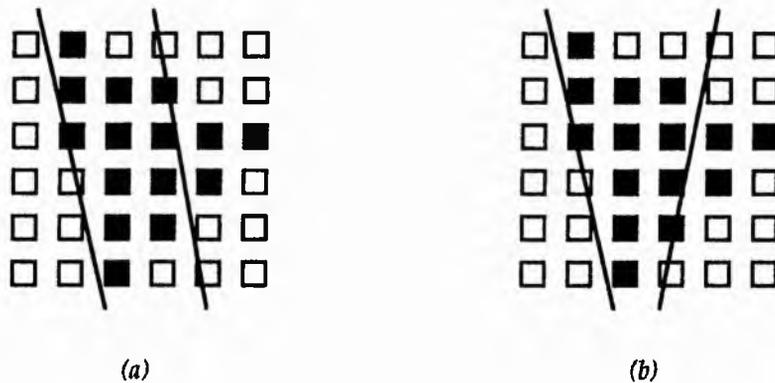


Figure 6i.3 — Two different ways to achieve the same global minimum misclassification error of five, using two hyperplanes.

Equivalence in the terms of inexact IO behaviour symmetry is represented as the same misclassification error of the grid. This means that grids are also able to cope with *inexact fit* solutions.

With noisy data, grids can have a beneficial blurring effect on the data, which, if the noise is not too high, may enable the elimination of the noise. The spacing between the gridpoints also reflects the accuracy to which the input may be measured. (See figure 6i.4.)

Grids have the further advantage that they represent an increased closeness to Mitchell's symbolic technique, since the grid as a whole can represent the entire instance language. This will be discussed in section 6i.2.1.

It is important to distinguish between three types of grid. Firstly, there is the target grid. The target grid is based on the targets of the patterns. The method used to decide the targets of the gridpoints is discussed in section 6i.2.1. Secondly, there is the output grid, which is based on the output of the neural network for the inputs corresponding to the gridpoints. The output and target grids have the same input points, which may be referred to as the *input grid*.

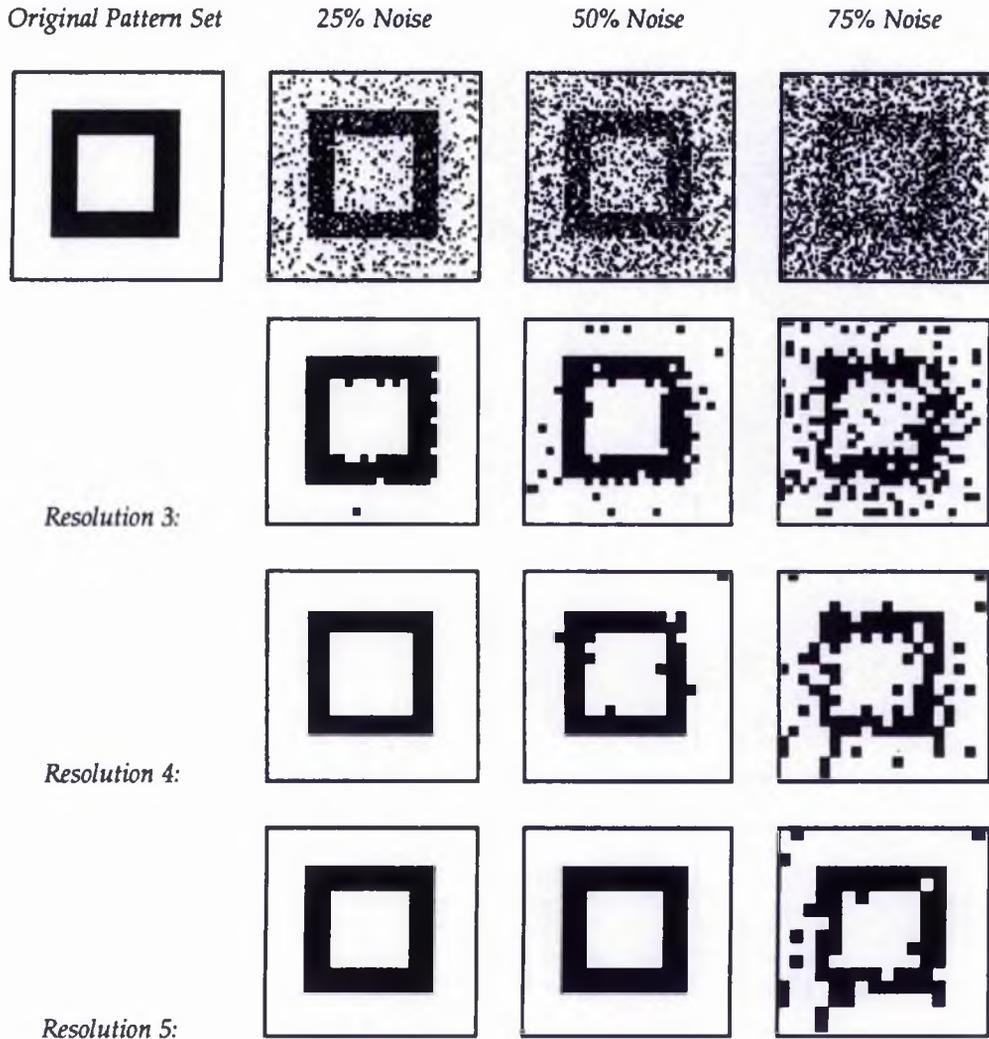


Figure 6i.4 — The partial restoration of the original pattern set from noise corrupted pattern sets using grids of increasing resolutions. Each gridpoint in the three resolutions given below the original pattern set (with varying degrees of noise) is assigned the target of the majority of the points in the area it occupies. More detail is given in section 6i.2.1.

The choice of topology for the neural network affects which of the target grids can be trained to give an IO with zero misclassification error. This represents a Mitchellian bias since, inevitably, certain IO relations are preferred to others, in that they can be realised exactly rather than inexactly. To keep this bias under user control, the topology is set by the user *a priori*, and is unchanged during learning.

6i.1.2 Partially Ordering the Grids

Before defining a relation which gives a partial ordering to grids, consider the following method for representing any grid, no matter what the dimensionality of input space, or the number of gridpoints in the grid. Given that the position of each gridpoint in input space is already known, what is of interest about a grid, be it a target or output grid, is the class assigned to each gridpoint, which is either 1 or 0. Thus, by ordering the gridpoints, the class assignments to a grid can be written as a binary string, regardless of the number of dimensions of input space. Thus, for example, a $4 \times 4 \times 4$ grid in a 3D input space, and an 8×8 grid in a 2D input space could both be represented by a 64 bit binary string.

Let the intersection of two grids, A and B , where the grids are of the same type and represented using binary strings, be defined by the AND binary string operator, as per equation [6i.1]. The intersection of two grids is a grid whose gridpoints are black where both A and B are black, and white otherwise, where "black" represents a target or output of 1 assigned to the gridpoint, and "white" represents a target or output of 0 assigned to the gridpoint.

$$A \cap B = \text{AND}(A, B) \quad [6i.1]$$

The reflexive, commutative and associative properties of the logical AND operator are carried across into the AND binary string operator.

Let the relation *has at least the same black gridpoints as* be denoted by \geq , defined in [6i.2]:

$$A \geq B \Leftrightarrow A \cap B = B \quad [6i.2]$$

The necessary and sufficient conditions for a partial ordering are that the relation that forms the ordering is reflexive, transitive and antisymmetric.³ The relation \geq is reflexive, since $A \cap A = A$, and hence $A \geq A$. If \geq is to be transitive, then it is necessary to prove [6i.3]:

$$(A \geq B) \wedge (B \geq C) \Rightarrow A \geq C \quad [6i.3]$$

Expanding the terms on the left hand side, using [6i.2], gives:

$$(A \cap B = B) \wedge (B \cap C = C) \quad [6i.4]$$

Using the left hand conjunct, we may substitute $A \cap B$ for B in the right hand conjunct to give:

$$\Rightarrow (A \cap B) \cap C = C \quad [6i.5]$$

Then, since \cap is associative, [6i.5] may be rearranged to give:

$$\Rightarrow A \cap (B \cap C) = C \quad [6i.6]$$

Substituting for $(B \cap C)$ using the right hand conjunct in [6i.4] gives:

$$\Rightarrow A \cap C = C \quad [6i.7]$$

Therefore, from [6i.2], $A \geq C$, and [6i.3] is proved.

If \geq is to be antisymmetric, it is necessary to prove [6i.8]:

$$(A \geq B) \wedge (B \geq A) \Rightarrow A = B \quad [6i.8]$$

Expanding the left hand side of [6i.8] using [6i.2] gives the following conjunction:

$$(A \cap B = B) \wedge (B \cap A = A) \quad [6i.9]$$

Since \cap is commutative, we may write:

$$A \cap B = B \cap A \quad [6i.10]$$

³Borowski & Borwein, 1989, p. 437

Using [6i.10] to substitute for $A \cap B$ in the left hand conjunct in [6i.9] gives:

$$(B \cap A = B) \wedge (B \cap A = A) \quad [6i.11]$$

Using the right hand conjunct, we may substitute A for $B \cap A$ in the left hand conjunct. Therefore $A = B$ and [6i.8] is proved.

Thus the relation \geq forms a partial ordering of grids since it is reflexive, transitive and antisymmetric.

The lattice of partially ordered 3×3 grids realisable by a 2^*1 topology is shown in figure 6i.5.

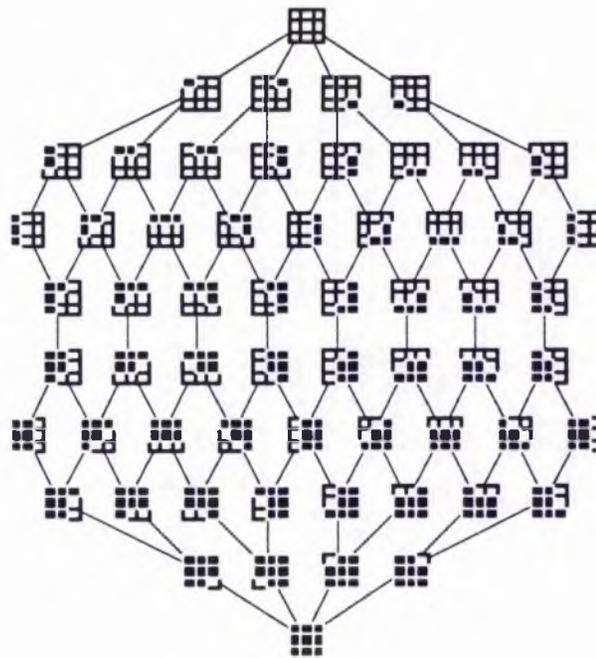


Figure 6i.5 — The lattice of partially ordered 3×3 grids realisable by a 2^*1 topology.

Therefore, it may be seen that by approximating the IO and the generalisation error through the use of a grid, it is possible to give a partial ordering to the IO. This opens up the possibility of implementing a Mitchellian technique using neural networks without the need to look at weight space.

6i.2 Learning Using Grids

6i.2.1 Construction and Use of Grids

In this section, methods for constructing a grid from a given set of patterns are examined. This enables comparisons with Mitchell's instance language to be made. The use of grids in the learning process itself is also discussed.

Once the position and spacing of the gridpoints has been decided, the targets of the gridpoints may be calculated. For each pattern, the nearest gridpoint to that pattern is found. The pattern then has a vote for the target of its nearest gridpoint. Once all the patterns have been assigned to a gridpoint, the gridpoint is given the target of the majority of the patterns assigned to it. Where the number of patterns of each class assigned to a gridpoint are equal, the gridpoint is given the target of the nearest pattern.

This method of representing the input allows for an extra closeness to Mitchell. For Mitchell, the inputs along each dimension (attribute) take on a discrete set of values, as they do here, in the grid. However, the brittleness of symbolic representation means that the symbol `navy_blue` would be treated differently from the symbol `royal_blue`. This is all very well when the concept to be learned requires such fine distinctions of attribute values, but what if it is only required to know whether or not the colour is blue as opposed to red? The symbol `navy_blue` is not equal to the symbol `blue`, and hence a positive instance risks not being recognised as such. Using a grid, however, the colours can be enumerated — using the wavelength of light, for example. Gridpoints can be placed for the wavelengths of significant colours, and instances which may contain the various hues of these colours will be mapped onto the appropriate gridpoint. Learning is then achievable with the desired degree of accuracy, and without the brittleness of symbolic representation.

When the output space is multi-dimensional each point on the input grid may be given different classifications by the output units. Therefore a separate output grid must be used for each output unit in order that each output unit's classification for each point on the input grid is represented.

Since each output unit has its own grid, it is easier to use a separate network with a given number of hidden units to train each output unit and grid than to try and reach the no-alternative situation for many grids at once. This gives rise to a modular topology. Figure 6i.6 shows an example of the topology used when there are three output units, with two input units, and three hidden units allocated to learn the grid for each output unit.

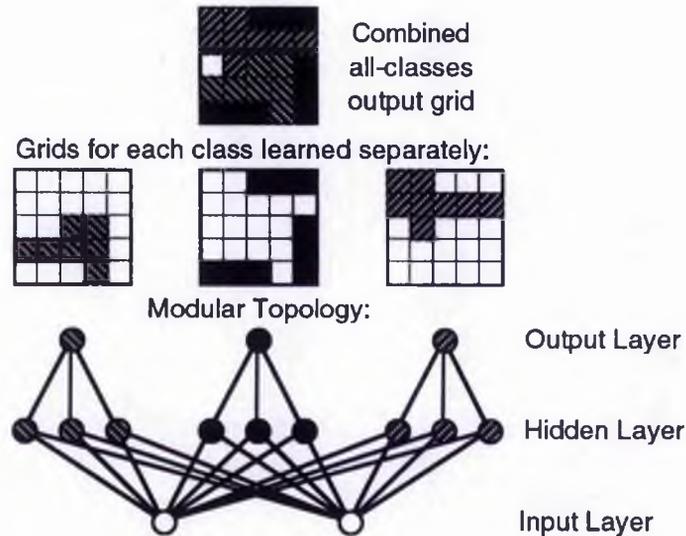


Figure 6i.6 — A modular topology used for learning a problem with more than one output unit.

There is also the distinction between two types of gridpoint, which introduces the incremental nature of the learning process. A *real* gridpoint is a gridpoint which is trained using targets assigned from the underlying patterns. A *hypothetical* gridpoint is a gridpoint whose target has not yet been assigned a value from the underlying patterns. The process of assigning gridpoints their true targets from the underlying patterns — turning hypothetical gridpoints into real gridpoints — will be termed *familiarisation*. During learning, the gridpoints are familiarised incrementally. The hypothetical gridpoints may be trained, even though they do not have targets assigned to them from the patterns. Hypothetical targets may be assigned to them instead. At all stages of learning, the *underlying* target grid is the grid with targets assigned on the basis of the targets of the underlying patterns.

At each stage of familiarisation, there will be a number of possible underlying target grids. These possibilities may be found by considering combinations of each of the possibilities for the hypothetical gridpoints. Each of these target grids has one or more best fit output grids. The set of all output grids that are best fit grids to one or more of the possible underlying target grids at a given stage of learning forms the version space. As familiarisation continues, the number of possible underlying target grids is reduced, and with it the set of possible best fit grids and hence the size of version space.

The boundary representatives may therefore have different target grids at a given stage during learning, since they may have different targets for the hypothetical gridpoints. When all the gridpoints are familiarised, the resulting target grids for the boundary representatives are identical to the underlying target grid. At the final stage of learning, therefore, there can be no difference in the target grids for the boundary representatives. All the gridpoints are real gridpoints, and do not have their targets assigned hypothetically.

There is a single optimum degree of fit to the underlying target grid. In some cases there are various alternative output grids with this optimum fit to the underlying target grid. These must be viewed as equivalent by inexact IO behaviour symmetry (see section 6i.1.1).

Although the no-alternative situation may be reached at an earlier familiarisation, it is at least possible to be sure that it is always reached by the time all the gridpoints are familiarised. After full familiarisation, the target grids for each direction are equal to the underlying target grid. Once the learning process has reached this stage there is never any alternative consistent with all the examples presented except the optimum fit.

The attitude to generalisation is made closer to Mitchell than in chapter 5. The user is responsible for the biases (the grid and topology) which affect the kind of generalisation that will be made from the given underlying data. It is then the responsibility of the training algorithm (such as back-propagation or a GA) to find the best weight state to fit the presented data, with the user's given bias.

Here, this means finding the global minimum error neural network. There are some authors who claim to have algorithms which are able to find the globally optimal neural network, such as Baba et al, for example.⁴ Baba et al combine back-propagation with a random walk to escape local minima. However, the proof that the global minimum can be found relies on having an infinite amount of time available.⁵ This is impractical, and the technique has no ability to recognise when the global minimum error has been found. To my knowledge there are no techniques in the literature that feasibly and reliably guarantee an ability to find the global best fit.

Since it is not, in practice, possible to always guarantee finding the globally optimum fit, then it would be useful to be able to recognise the occasions when it has been found. This is what Mitchell's technique has to offer, since it can recognise the no-alternative fit situation.

The development of the three paradigms described in later sections has this goal in mind: to recognise when the optimum fit has been found to the underlying target grid using standard training algorithms. This goal is to be achieved through maintaining parallels with Mitchell's symbolic technique in the neural implementations as much as possible. Generalisation is then guaranteed, as with the symbolic technique, on the basis of the assumptions of the user — which are expressed in terms of the topology and the grid chosen.

6i.2.2 Generalisation and the Relation to Mitchell

When learning using grids, there are two main ways in which generalisation can be viewed. Firstly, generalisation can be considered using the gridpoints only as examples to be generalised to — this may be seen as *discrete generalisation*. With this view of generalisation, it is important to be able to reach the no-alternative situation with hypothetical gridpoints remaining — otherwise it can be argued that there has been no discrete generalisation, since all the possible instances have been learned. Therefore, in discrete generalisation, the success of a trial may be

⁴Baba et al, 1994

⁵Baba et al, 1994, p. 1257

measured in terms of the number of hypothetical gridpoints left when the no-alternative situation is reached. The more the number of hypothetical gridpoints remaining when the no-alternative situation is reached, the more the amount of discrete generalisation.

However, the grid technique allows that there may be full familiarisation of gridpoints before the no-alternative situation is reached. There are then no hypothetical gridpoints to generalise to. The discrete view of generalisation, confined to the gridpoints, is consequently not broad enough.

A broader, more common view of generalisation in neural networks is that of providing output for any input, including points that are not gridpoints. This will be termed *continuous generalisation*. The grid has been used to approximate a continuous picture. In this case, it does not matter if all the gridpoints have been familiarised, provided that the no-alternative situation has been reached. Generalisation is guaranteed to the rest of input space, within the limits of the boundaries found with the given grid resolution, the bounds of input space under consideration, and the topology chosen. This maintains the maximum consistency with the Mitchellian view, which is that generalisation follows from finding the no-alternative situation, given the biases of the user.

One deviation from Mitchell, however, is that there is no need to maintain multiple boundary representatives. One boundary representative is sufficient for each direction. This is because neural search is not constrained to a given chain in the partial ordering for each boundary representative. Any grid can, in theory, be generated at any time. This is an advantage of the neural technique, since it can then tackle problems that might be infeasible with the symbolic technique due to there being too many boundary representatives.

6i.2.3 Comparison with the Validation Technique

Validation aims to find the weight state that will best predict the validation set, given the training set. It trains until it appears that no better prediction can be found, since a minimum of validation error has been reached. This however, does not preclude the possibility of there being a

better predictive weight state at a later minimum of validation, or using different initial conditions for the search.

The grid technique, by contrast, aims to find the best degree of fit to the grid that is possible with the given topology. The technique continues learning until a best fit has been found and recognised. At each stage preceding the no-alternative situation, it is known that a better degree of fit is possible.

Adherents of the validation technique will be aware of Denker et al's kibitzer criticism, which pertains to the idea that transferring patterns from the validation set to the training set means improving generalisation, since the patterns to be generalised to are being trained.⁶ Using patterns set aside for validation to train the neural network undermines the validation technique, since fewer patterns in the validation set means that the predictive powers of the neural network are not so heavily tested. Therefore, anyone who suggests putting patterns from the validation set into the training set is a kibitzer. (A kibitzer is an observer who offers unwanted advice.)

Since, when learning using grids, the gridpoints are familiarised in a sequence, there is the possibility that this kind of learning may be seen as being open to Denker et al's "kibitzer" criticism. This is because there is a rough correspondence between extended familiarisation, and taking patterns from the validation set and putting them in the training set.

However, it is important to note that the grid technique does not aim at good prediction of the hypothetical gridpoints. The object of the grid technique is to show that it does not matter whether the true targets of the hypothetical gridpoints are black or white. Consequently, the kibitzer criticism does not apply to the grid technique.

⁶Denker et al, 1987, p. 898

6l.2.4 Features of Learning Using Grids

- **Distinctions between Training, Generalisation and Learning**

In standard neural techniques, training is seen as separate from generalisation.

[Training] is typically done incrementally making small adjustments in response to each training pair, so that the [weights] converge ... to a solution in which the training set is "known" with high fidelity. It is then interesting to try patterns not in the training set, to see whether the network can successfully generalise what it has learned.⁷

In this thesis, training and generalisation are to be viewed together, under the heading of learning. The difference between generalisation and learning is that the former is an assessment of performance, and the latter is an incremental process of updating the concept specification until there is no alternative concept that is consistent with the available data. The difference between training and learning is that learning is aimed at providing a guarantee of generalisation when it terminates. Training makes no such guarantee. When taken separately training and generalisation do not offer any certainties about the correctness of the final concept found. Learning is a more comprehensive process which is able to offer such certainty once completed.

Training is then the process of finding an output grid for a given target grid. Learning is a higher level process, which provides an interpretation of the grids found by the training algorithm, and is based on a series of target grids to be trained. Essentially, training provides an output grid, and learning consists of a training sequence which culminates in the ability to know that the no-alternative situation has been reached.

- **Bidirectional Search**

The grid technique uses a bidirectional search, with one grid (and corresponding neural weight state) used to represent each direction of the search. Bidirectional search is used to enable the terminating condition to

⁷Hertz et al, 1991, p. 89

provide the guarantee that the no-alternative situation has been found. It also gives a measurement of the degree of learning achieved so far.

Conventional training techniques, such as back-propagation, are unidirectional. A single weight path is found from the initial state to a solution state. With only a single direction, it is not possible to recognise when the best fit weight state has been found for inexact fit. It may be argued, however, that bidirectional techniques take at least twice the amount of time to train than unidirectional techniques, though this does not take into consideration any heuristics for speeding up training that might apply in a specific implementation. See "Training Sequences" below.

Under ideal circumstances the weight path of a neural network during a search contains increasingly better weight states, in terms of their IO. So earlier IO behaviours along a weight path are rejected, and should never be revisited on the path to finding a solution. For any specific weight path it is only possible to look back along that path and say that all the IO behaviours along that path are worse. Whatever the circumstances, there is no possibility to look forward and say that all possible future IO behaviours arising from the continuation of the current weight path are also worse. This is why a unidirectional technique cannot make any guarantees about termination. There is always the possibility of finding a better weight state given further training.

Here, the IO behaviours are measured using a grid. A unidirectional technique can only reject the grids it has already passed through. Using a partial ordering would enable a unidirectional technique to eliminate other grids that are behind the visited grids without infeasible exhaustive search. However, such a technique then has a terminating problem. The single direction cannot look ahead and know that there is no better output grid than the output grid it is currently stopped at.

Using two directions which meet means that each direction may know what is ahead, since it has already been explored and rejected by the other direction. It is this that enables the guarantee to be made that it is time to stop, since neither direction can go ahead without impairing performance. This is the benefit that bidirectionality has to offer.

- **Use of Grids**

Grids are used to represent the IO behaviour of the neural network. The input positions and targets of the gridpoints are determined by the data sample. The networks are trained using the grids and not the data sample. Grids are a finite representation of IO space, and each gridpoint is taken to stand for its local area of input space. The error on the grid is a means for approximating the generalisation error.

This technique has advantages over methods that use the raw data only. The latter take relatively more time to train since the raw data may be expected to have a higher density of coverage of input space than the grid. This means that the time cost of using a bidirectional technique and a training sequence is not as high as it might seem at first glance. Also, if the raw data are irregularly distributed, the hyperplanes associated with the non-input units of the neural network are attracted to separating the regions more densely packed with training points. This occurs regardless of whether the resulting separations accurately reflect the underlying map or not, thus leading to poor generalisation.

- **Terminating Condition**

The grid technique has a terminating condition, which, if satisfied, guarantees that the best fit has been found to the grid, provided that the training conditions are met. The terminating condition is necessary to specify when the concept has been learned. This is when the two directions in the bidirectional search have reached equivalent grids in the partial ordering. This condition will be called *locking*, and this definition of locking supersedes the earlier definition in chapter 5.

After sufficient gridpoints have been familiarised, certain output grids in the version space may be globally optimum fits to all of the possible underlying target grids. These output grids are equivalent in the sense that they all have a globally optimum fit no matter what the true targets of the hypothetical gridpoints are. The terminating condition is designed to detect when these grids have been found by both of the directions in the search.

The terminating conditions of standard training techniques are based on local minima or preset maximum training times, and as such do not guarantee that the best fit grid has been found. Without a bidirectional search, standard techniques are unable to provide a terminating condition that makes this guarantee, since there is always the possibility of a better fit after continued training.

- **Training Sequences**

The grid technique is a learning technique which consists of a training sequence for each direction of the search. Each member of the sequence arises from the familiarisation of one or possibly more gridpoints. For each familiarisation, the target grids for each direction are trained using a new random initial weight state. To maintain maximum consistency with Mitchell, gridpoints are familiarised one at a time here. This is also part of a wider view, in which concept learning is seen as a process in which the instances are not all presented at once, but one by one.

The main practical benefit of this strategy is that the training algorithms are given more of a chance to find the optimum solution. The small, gradual changes to the target grids as the hypothetical gridpoints are familiarised one by one, allows minimal updating. Thus, if a training algorithm returns a sub-optimum grid at a certain stage, it is given more chances to catch up with the optimum classification in the ordering, and the distance to catch up is always kept as small as possible.

Standard iterative training techniques need only train once to find a solution — though in practice, several trials may be used to try and find better solutions using different parameters and initial weight states.⁸ This means that there may be an additional cost to training many times in this learning technique. This may be countered in that using grids, instead of large amounts of raw data, results in a saving on computer time.

⁸Weir, 1991, pp. 375-376

- **Fixed Topology**

The grid technique uses a fixed topology, which is interpreted as being a bias of the user. The aim of the technique is to find the best fit grid given the data sample, and the user's assumptions. By finding the best fit grid, it is able to provide the desired class boundaries.

Some techniques use a variable topology. This is because it may be difficult to know which topology to use in advance. Some guidelines for this have been given in chapter 4, however. This pioneering version of Mitchell's technique uses a fixed topology in order to investigate inexact fit, which is a problem for standard techniques. Variable topology systems risk over-fitting inexact fit problems. The simplest way to investigate inexact fit is to use a fixed topology.

- **Keeping the Two Directions Apart**

There will be two means described in this chapter of preventing the possibility of *false locking*, which is where the two directions in the search converge on a sub-optimum grid. The first is to use target reversal. Here, the output grids of the two directions are encouraged to be as different as possible by setting the hypothetical targets of one direction to be the opposite of the corresponding outputs of the other direction. The grids converge as the number of hypothetical targets diminishes and such targets have less effect on the trained grids.

The second is to use the partial ordering of grids. The partial ordering of grids is used in part to prevent false locking due to the training algorithm not returning the best fit grid at each stage of learning. This is achieved by ensuring that any sub-optimum grids are as far back in the ordering as possible.

The main use of the partial ordering, however, is to enable the elimination of output grids that were not on the weight paths of either of the two directions. Grids are eliminated from consideration by being behind the current grid for a direction in the ordering, as well as by being grids that have been actually visited in earlier stages of learning. This enables a space-wide elimination, and hence the certainty that, at locking, there is no output grid with a better fit to the underlying target grid.

If a potential function is used as a measure of fit, standard unidirectional techniques implicitly eliminate weight states and output grids of lower potential as the weight paths move to improved potential function values. Without the bidirectional search and partial ordering, though, these techniques are unable to know in general the totality of the output grids that have been eliminated. The partial ordering and bidirectional search combine to enable a guarantee of best fit to be made at termination. Standard techniques are unable to make such a guarantee.

6i.3 The H/H0 Paradigm

The H/H0 paradigm⁹ uses target reversal only to keep the two directions of the bidirectional search apart. Target reversal will be described in detail later. The partial ordering is not used. This means it does not eliminate any grids except those previously visited by one of the two directions, unless the locking condition is satisfied. Once the locking condition is satisfied, however, all other grids are guaranteed to be either as good a fit or worse than the grid found.

The purpose of this paradigm is to demonstrate the locking condition.

6i.3.1 Methodology of the H/H0 Paradigm

In the bidirectional search, one direction is represented by H, which is an actual output grid with an optimum fit to the real gridpoints only. H is not trained on the hypothetical gridpoints, and represents a hypothesis for the generalisation to the hypothetical gridpoints.

The other direction is represented by H0, which is the output grid that best fits the hypothesis that the generalisation of H is entirely incorrect. H0_T is the target grid for H0, with targets assigned in the following way. The real gridpoints have the targets assigned as per the underlying patterns. By contrast with H, H0 is trained on the hypothetical gridpoints. The hypothetical gridpoints are assigned the opposite classification to the actual output of H for those points. This is the method of target reversal. H

⁹Weir & Polhill, 1994b, 1995

and H_0 will be assumed to have global minimum classification error on the targets they are given. The training algorithm is therefore assumed to be *perfect*, in that it always provides the optimum fit. Figure 6i.7 has an example of target reversal.

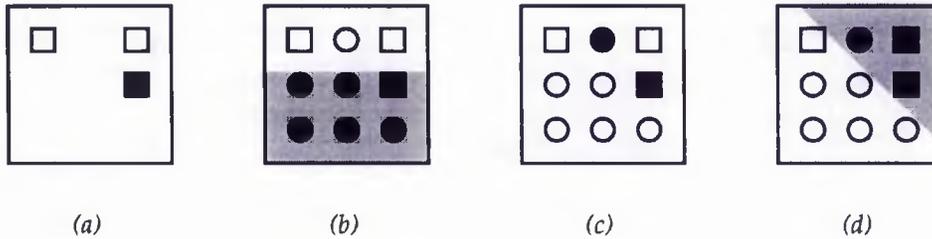


Figure 6i.7 — The target and output grids for H and H_0 . Squares indicate real gridpoints, and circles indicate hypothetical gridpoints. (a) Target grid for H , containing only the real gridpoints. Black squares indicate a target of 1, and white squares indicate a target of 0. (b) Output grid for H . The shaded region shows where H gives an output of 1. The outputs of the gridpoints are then indicated in black if the output is 1, and white if the output is 0. (c) H_{0T} , the target grid for H_0 . The real gridpoints retain their targets, whilst the hypothetical gridpoints have the opposite target to the classification of H . (d) Output grid for H_0 , with global minimum error.

During learning, the gridpoints are familiarised one by one until the no-alternative situation is reached. The no-alternative situation is recognised by the satisfaction of the following terminating condition: termination occurs when (i) H and H_0 have the same misclassification error on the real gridpoints, and (ii) H_0 has given the same classification as H on the hypothetical gridpoints, despite being trained on the opposite class. This means that H_0 must have misclassified all hypothetical gridpoints according to H_{0T} . When the above two conditions are satisfied, H and H_0 will be said to have *locked*. An example is given in figure 6i.8.

H_{0T} 's hypothetical targets are reversed from H 's classification. At locking, H_0 misclassifies all of H_{0T} 's hypothetical targets with globally optimal fit on H_{0T} as a whole. If any of the hypothetical gridpoints in H_{0T} were to be familiarised, there are two possibilities. If any of their real targets are different from their hypothetical targets in H_{0T} , then these gridpoints are correctly classified. If any of their real targets are the same as their hypothetical targets in H_{0T} , then they have already been trained with that

target value, and the globally optimal fit misclassifies it. If any of these targets could have been correctly classified with better overall fit, then this would have occurred. There is therefore no point in further familiarisation, since H and H0 will remain unchanged. The real targets will either be correctly classified or misclassified. In the latter case, if they could have been correctly classified, then they would have been during the training of H0. Appendix 6.A contains a proof that, at locking, H0 has global minimum error whatever the true targets of the hypothetical gridpoints.

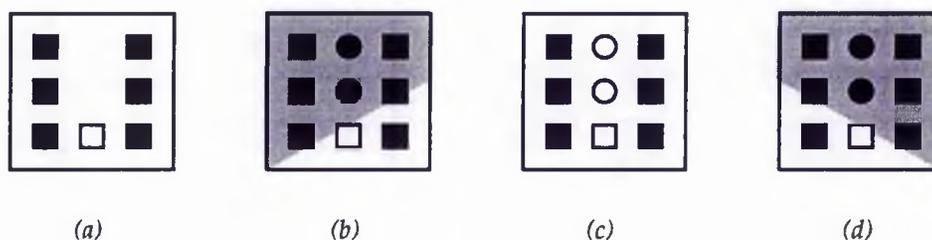


Figure 6i.8 — Target and output grids for H and H0 for an example of locking. (a) Target grid for H — the real gridpoints. (b) H's classification. (c) $H0_T$, with the hypothetical targets set as the reverse of H's classification. (d) H0's classification. This has the same error, 1, as H on the real gridpoints, and the same classification as H on the hypothetical gridpoints.

It can be seen from figure 6i.8(c) and (d) that, at locking, any change to the hypothetical targets in the grid in (c) agrees with the classification of H0 in (d). It follows that if H0 has global minimum error on (c), then it must have global minimum error when any of the targets are reversed. Termination occurs in general when target reversal has no effect.

6i.3.2 Experiments on the H/H0 Paradigm

6i.3.2.1 Efficient Exhaustive Search

When a technique requires global minimum error, it is necessary to be able to provide some means of acquiring it. This necessitates an exhaustive search in the case of inexact fit, since there is no other means of providing an absolute certainty of having found the global minimum error, if its value is uncertain *a priori*. An exhaustive search of weight space may seem like a daunting prospect, and indeed it is, though there are ways in which

the search can be approximated and cut down. The goal of the search is to find all the possible output grids, given an input grid, and a topology. If these are found, the global minimum error for a given target grid can always be found by iterating through the output grids, saving the best grid found.

The first measure for cutting down the search to be considered is to constrain the weight values. This is the approach adopted by Schwarz et al in their exhaustive search.¹⁰ The extent to which the weights are constrained represents a Mitchellian bias in learning. There is therefore no need to regard such constraint in a negative light. Since the ratio and the sign of the weights is all that is of significance in determining the output of a threshold unit (see chapter 4), the range of weight values considered is of less concern than the sample interval, provided that the range is centred at the origin.

The range and sample interval of weight values considered for each weight will affect the output grids the topology can generate. For the experiments in this section, all the output grids that could be found were used. The heuristic for finding as many output grids as possible was to keep the range of weight values constant, and decrease the sample interval by an order of magnitude until two consecutive samples gave the same number of possible output grids.

For a topology with n weights, the number of samples, $\#s$, required is given in equation [6i.12], where w_{max} and w_{min} specify the range, and i_w is the sample interval.

$$\#s = \left(\frac{w_{max} - w_{min}}{i_w} \right)^n \quad [6i.12]$$

For a simple 2*1 topology, with three weights, and a 3 x 3 grid, which requires a sample interval of 0.1 in a range of [-0.5, 0.5] for the weight values to obtain all the 58 possible output grids, the number of samples is 1 000. This is not an unreasonable number of samples, and indeed, 1 000 weight samples is perfectly acceptable in conventional training

¹⁰Schwarz et al, 1990, p. 379

algorithms, such as back-propagation. However, using a $2 \times 2 \times 1$ topology, which has nine weights, means the sample size is increased to 10^9 . This is an unreasonable number of samples, and hence a further measure is necessary in order to cut down the search.

The search can be dramatically cut down further by considering the purpose of each unit. For the purposes of this discussion, first layer units will be defined as all units with only and all the input units (and the bias, if used) as source. Similarly, second layer units will be defined as all units with only and all first layer units as source.

First layer units divide up input space, as discussed in chapter 4. All first layer units perform this same purpose. Finding all the possible output grids for a particular first hidden layer unit gives all the possible output grids for any first hidden layer unit. Thus, if all the possible output grids have been found for a 2×1 topology, this result can be re-used for all first layer units.

Using threshold units means that the only possible outputs from a unit are 1 or 0. This means that the outputs from any layer of units lie on the vertices of a unit hypercube. This may be seen as a grid. The output combinations of the first layer units therefore form an input grid to the second layer units. The total number of second layer output grids is given by the number of linear separations of the second layer input grid for the various input combinations to the second layer units.

For example, consider a $2 \times 2 \times 1$ topology, and a 3×3 grid. Using 1 000 samples of weight space for a first hidden layer unit gives the 58 possible output grids from such a unit. 1 000 further samples of weight space for the output unit gives the 14 possible linear separations of the unit square. To find all the possible output grids of the network, it is necessary to consider all combinations of output grids from the first hidden layer, for each of the 14 separations found for the output unit. This means examining 14×58^2 , or 47 096 output grids. Many of these are the same, and the eventual number of different possible output grids found is 320. The exhaustive search is complete, with 2 000 samples of weight space, and roughly 50 000 grid combinations — a significant reduction from examining 10^9 weight states.

Naturally, the search does not scale up well. Using a $2 \times 3 \times 1$ topology on a 3×3 grid entails 1 000 samples in the first layer, and 10 000 samples in the second, to get the 97 possible separations of the first layer units. The number of combinations of grids to be examined is then $97 \times 58^3 \approx 2 \times 10^7$, to find the 508 possible output grids — three orders of magnitude increase in the number of combinations for one extra hidden unit. A 4×4 grid requires a sample interval of 0.05 to get all the 174 first layer possibilities, using the same range as above. This means 8 000 samples of weight space. For a $2 \times 2 \times 1$ topology on a 4×4 grid, there are $14 \times 174^2 \approx 4 \times 10^5$ combinations of output grids to be examined, to find the 3 066 possible output grids — an order of magnitude increase in combinations for seven extra gridpoints relative to a 3×3 grid.

Despite these measures for making the search more efficient, therefore, the time taken to find all the possible output grids can still be extremely lengthy. It is therefore not surprising that heuristic search is used in neural networks when finding weight states. Problems are therefore restricted to very simple topologies, but this is nevertheless sufficient to show the basis of the technique at work.

6i.3.2.2 Demonstration of Termination

The fundamental questions of interest are whether or not locking is true or false, if locking is achieved before all the gridpoints have been familiarised, and if so, the maximum number of hypothetical gridpoints at locking for a given grid. False locking occurs when the output grids of H and/or H0 are not one of the best fit grids of the underlying target grid. The number of hypothetical gridpoints at locking is significant for discrete generalisation (as defined in section 6i.2.2).

Once all the output grids have been found using exhaustive search, the software then runs through all the possible target grids. The set of target grids will typically be a larger set of grids than the set of output grids, unless the topology is capable of exactly realising any set of targets on the grid. A prespecified number of random familiarisation orders is carried out on each target grid, with the gridpoints being familiarised one by one each time. Each sequence of familiarisation until locking occurs, or all the gridpoints have been familiarised, counts as a single trial. The software

counts the trials which terminate with various numbers of hypothetical gridpoints, along with the false locking cases.

The software has facilities for excluding equivalent target grids. Two target grids are equivalent if one is the binary inverse of another, or if there is rotational, or reflective symmetries between the two grids and/or their inverses. Figure 6i.9 shows a set of equivalent grids.

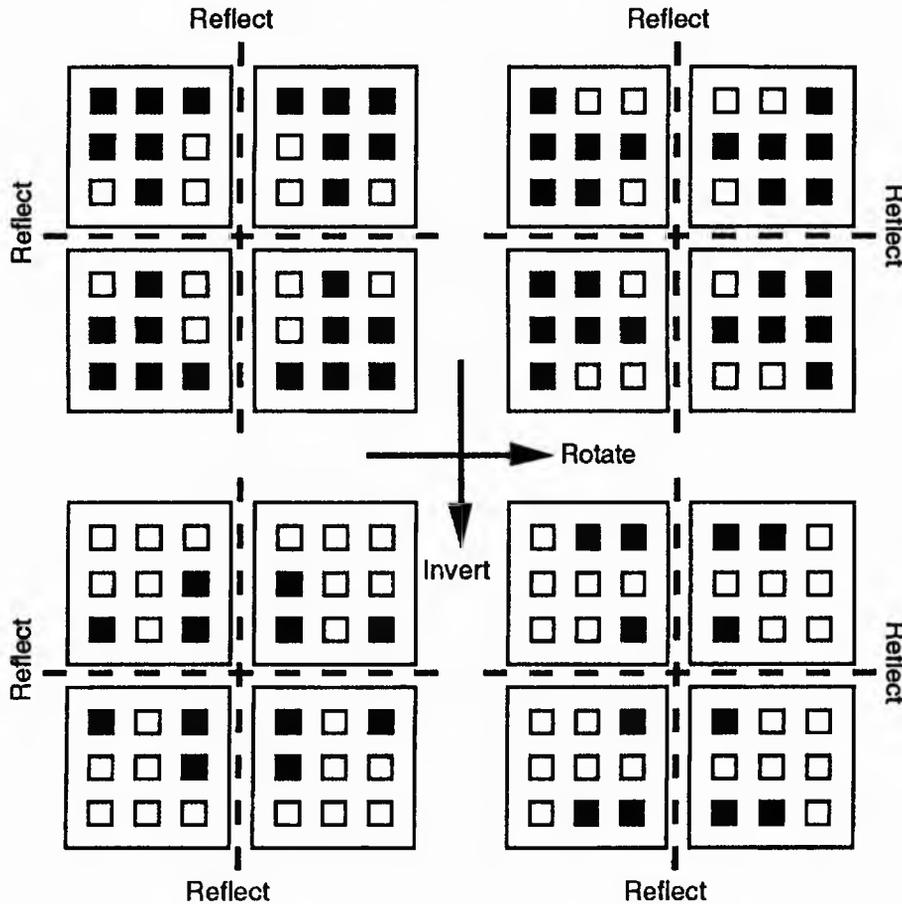


Figure 6i.9 — Sixteen grids equivalent by inversion, rotation or reflection.

Because the investigation is of inexact fit, those target grids which can be trained exactly by the current topology are ignored, and not used as underlying target grids.

The results for a 3×3 grid with the lower left-hand gridpoint having the co-ordinate (0, 0) and a spacing of 1.0 between the gridpoints, using a 2×1 topology, are as in table 6i.1. The choice of spacing and co-ordinates of the gridpoints is by and large arbitrary, and has no effect on the results. Any

regular 3×3 grid in input space has the same possible separations whatever its location and gridpoint spacing. Here each non-equivalent, inexact fit underlying target grid has 100 trials with a different random familiarisation order each time.

The results show that there is never any false termination, and that on average, there is roughly a 30% chance of terminating before all the gridpoints have been familiarised.

An example with 3 hypothetical gridpoints is given in figure 6i.10(a). Figure 6i.9(b) shows that the final grid found at locking by H and H0 is a global minimum error grid for all the target grids that can be generated by reversing the targets of the hypothetical gridpoints at locking.

Number of Hypothetical Gridpoints At Locking	Number of Terminations	Percentage of Total	Number of False Locking Runs
0	3134	70	0
1	1083	23	0
2	252	6	0
3	31	1	0

Table 6i.1 — Results for 100 trials of all non-equivalent inexact fit target grids.

```

Target grid (index 12):
oox  1  2  3
xoo  4  5  6
ooo  7  8  9

Global minimum error grids:
oox
ooo
ooo

Number of familiarised
gridpoints: 6
Familiarisation order:
3 8 6 7 2 1 4 5 9

H's targets:
..x ..x ..x ..x .ox oox
... .. .o .o .o .o
... .o. .o. oo. oo. oo.

H's outputs:
oox oox oox oox oox oox
ooo ooo ooo ooo ooo ooo
ooo ooo ooo ooo ooo ooo

H's errors:
0 0 0 0 0 0

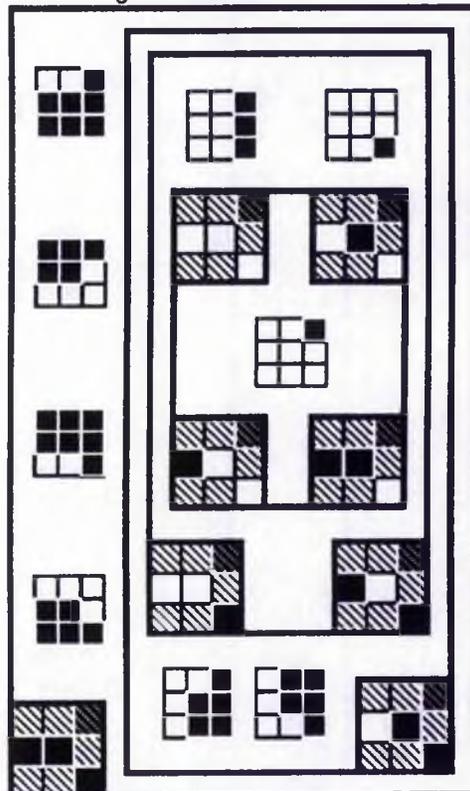
H0's targets:
xxx xxx xxx xxx xox oox
xxx xxx xxo xxo xxo xxo
xxx xox xox oox oox oox

H0's outputs:
xxx xxx xxx xxx xxx oox
xxx xxx xxo xxx xxx ooo
xxx xxx xoo oox oox ooo

H0's errors:
0 1 1 1 2 3
    
```

(a)

The grid found by H and H0 at locking in (a) is a best fit grid for all possible true targets of the 3 remaining hypothetical gridpoints. The possible underlying target grids in (b) are indicated by the larger grids, which have the real gridpoints shaded. There are 8 target grids, one for each possibility for the true targets of the hypothetical gridpoints at locking in (a). The target grids are shown contained within rectangles, each of which indicates a set boundary. The output grids are the smaller grids. All output grids inside a target grid's boundary are globally optimum fit grids for that target grid and any other target grids with surrounding boundaries.



(b)

Figure 6i.10 — (a) An example run, showing the output generated by the software for that run. 'x' indicates a target or output of 1, 'o' indicates a target or output of 0, and '.' in H's targets indicates a hypothetical gridpoint which is given no target. The underlying target grid is given at the top, and underneath it, the global minimum error grids for that target grid. There is only one in this case, and H and H0 converge on it after 6 familiarisations. (b) The grid found by H and H0 at locking is the optimum whatever the true targets of the hypothetical gridpoints.

6i.3.2.3 Maximum Number of Hypothetical Gridpoints at Locking

The maximum number of hypothetical gridpoints at locking for a given input grid is at most the maximum global minimum error, $E_{max(t)}$, out of all the target grids, given all the output grids. Since H0 must misclassify all the hypothetical gridpoints in H0_T before locking can occur, the earliest point at which locking can happen is when H0_T is a target grid whose maximum global minimum error is $E_{max(t)}$. H0 would then correctly classify all the real gridpoints, and the number of hypothetical gridpoints (all of which are misclassified) would be $E_{max(t)}$. For a 3×3 grid, and a 2×1 topology, $E_{max(t)} = 3$. Table 6i.2 shows all the values of $E_{max(t)}$ for various grids up to size 5×5 for a 2×1 topology.

		Maximum Global Minimum Error				
Grid Dimension Y	1	0				
	2	0	1			
	3	1	2	3		
	4	1	3	4	6	
	5	2	4	6	8	10
		1	2	3	4	5
		Grid Dimension X				

Table 6i.2 — Maximum global minimum error for any target grid for grids of dimension X by Y . (Y by X is equivalent.)

The results in table 6i.2 were found by generating all the possible output grids, and then running through all the target grids, to find a target grid with the maximum global minimum error. On reflection, however, these results can be explained, by considering a worst case grid for classification using a 2×1 topology, which has a single hyperplane. This is a grid which has alternating targets for the gridpoints. (See figure 6i.11.)

For grids with equal dimension, the global minimum error classification is indicated by the shaded decision region in figure 6i.11. The misclassified patterns will then always be a triangle number — specifically, the $(x - 1)$ th triangle number, where x is the dimension of the grid.

For other grids, where the grid dimensions, x and y are not equal, the grid may be divided into the largest square grid possible, with the remaining gridpoints as a rectangle. If x is the smaller dimension, an $x \times y$ grid would

be divided into an $x \times x$ grid joined onto a $(y - x) \times x$ grid. The minimum error classification is then the same as that for the $x \times x$ grid, with half of the remaining points misclassified. Figure 6i.12 has an example.

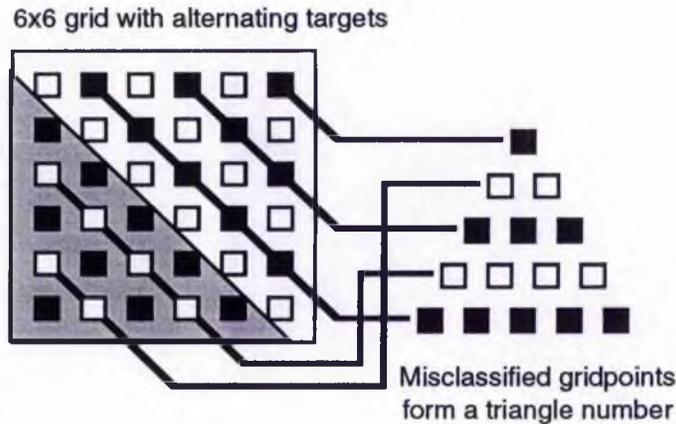


Figure 6i.11 — A grid with alternating targets, and a global minimum error classification, indicated by the shaded decision region. The misclassified patterns sum to a triangle number.

The general result, therefore, can be summarised in equation [6i.13], where $E_{max(t)}$ is the maximum global minimum error over all the possible target grids, $x \leq y$ are the grid dimensions, and $\lfloor a \rfloor$ is the largest integer not greater than a .

$$E_{max(t)} = \frac{x(x-1)}{2} + \left\lfloor \frac{x(y-x)}{2} \right\rfloor = \left\lfloor \frac{x(y-1)}{2} \right\rfloor \quad [6i.13]$$

As the grids approach infinite size, the maximum global minimum error approaches half the number of gridpoints. This shows that for larger grids, there is the possibility of terminating with close to half of the gridpoints remaining hypothetical. This result applies only to 2×1 topologies, however, though there may be similar results for other $n \times 1$ topologies.

Intuitively, it is unreasonable to expect locking to take place before half the gridpoints are familiarised. This is because the number of hypothetical gridpoints outnumbers the number of real gridpoints. The hypothetical gridpoints, with reversed targets from H's classification must all be misclassified before H0 can lock with H. Before the number of real gridpoints outweighs the number of hypothetical gridpoints, H0 will be

able to get lower error by correctly classifying the hypothetical gridpoints, rather than misclassifying them.

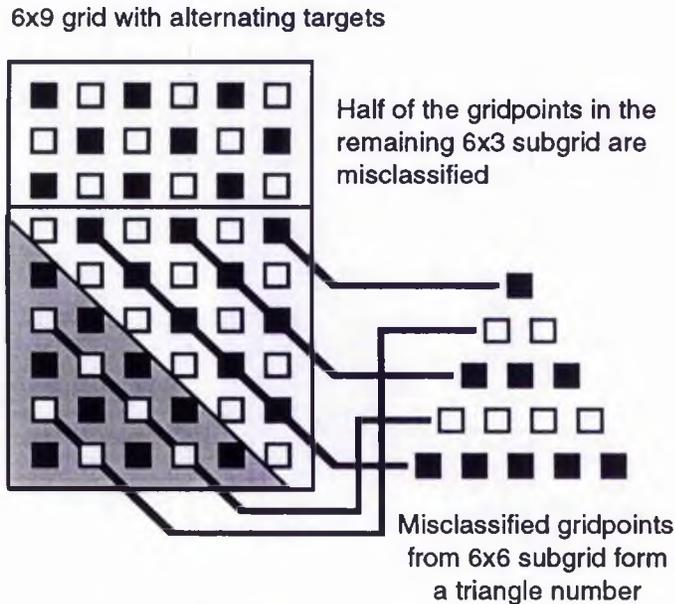


Figure 6i.12 — *A grid with unequal dimensions. The global minimum error is equal to that for the largest equal-dimensioned subgrid, plus half of the remaining gridpoints. As the hyperplane is moved upwards, the six white gridpoints which currently border the hyperplane become misclassified. Then the next six black gridpoints are correctly classified, and the overall net change in error is zero.*

The effect of adding hidden units is also worth considering. Since the number of inexact fit target grids is reduced when hidden units are added, it was thought that a 3×3 grid was too small. A 4×4 grid was used instead, and the results are summarised in table 6i.3. Ten runs were done on each inexact fit, non-equivalent target grid. There were a total of 43 090 runs for the 2^*1 topology, and 41 060 runs for the 2^*2^*1 topology. The fewer runs for the 2^*2^*1 topology are due to the extra grids that can be fit exactly, using a 2^*2^*1 topology rather than a 2^*1 topology. Of the runs for the 2^*1 topology, 72 terminated with 5 hypothetical gridpoints, and 5 terminated with 6 hypothetical gridpoints. The 2^*2^*1 topology had 98 terminations with 3 hypothetical gridpoints, and 1 termination with 4 hypothetical gridpoints. Both results show how rare it is to terminate with the maximum number of hypothetical gridpoints.

The rarity of termination with the maximum number of hypothetical gridpoints is to be expected. If an underlying target grid has global minimum error E , then in order to terminate with the maximum number of hypothetical gridpoints, the E gridpoints which are misclassified must be the last to be familiarised. This is an example of the more general point that the number of hypothetical gridpoints at locking is related in part to the order in which the gridpoints are familiarised.

There are also results in table 6i.3 that show clearly that using just one extra hyperplane to separate the grids means an increased likelihood of terminating only after all gridpoints have been familiarised for the same grid.

Number of Hypothetical Gridpoints At Locking	Topology		2*1 & 2*2*1 Number of False Locking Runs
	2*1	2*2*1	
	Percentage Terminations	Percentage Terminations	
0	53	77	0
1	29	20	0
2	13	3	0
3	4	0	0
4	1	0	0
5	0	0	0
6	0	0	0

Table 6i.3 — Comparison of distribution of terminations by hypothetical gridpoints at locking for all 4×4 non-equivalent inexact fit target grids for a 2*1 and a 2*2*1 topology.

6i.3.3 Assessment of the H/H0 Paradigm

The tested implementation of the paradigm indicates that the theory is correct in that false locking does not occur. It also provides some insight into the stages of familiarisation at which convergence takes place.

The main problem with the technique is that if the global minimum error can already be guaranteed using a single direction, the usefulness of Mitchell is obscured, since the purpose of Mitchell's technique is to use two directions to recognise when the best fit has been found. If it is already known that the best fit is going to be found using either direction

singly since the training algorithm guarantees it, then there is no point in using Mitchell's bidirectional technique.

The technique does not use a partial ordering of concepts, either. This means that there is uncertainty in which grids can definitely be eliminated until termination is achieved. The only grids definitely eliminated are those which have already been used by H or H0 during the familiarisation sequence. Using a partial ordering enables the further elimination of all grids behind the two directions in the ordering.

The benefits of the symbolic technique have therefore not been fully transferred into the neural technique. In its current form, it is limited by being open to false locking unless the training algorithm is perfect. The locking guarantee relies on H and H0 having the global minimum error for the targets they have been trained on. If the global minimum error cannot be guaranteed by the training algorithm, then false locking is possible. Although a sub-optimum training algorithm may lock correctly, there are no measures in place to prevent false locking when a sub-optimum grid is returned by the training algorithm. There is a clear need for modifications which enable the use of conventional training algorithms. Conventional training algorithms are *imperfect* training algorithms in that the weight states they find are not always optimal.

6i.4 The B/W Paradigm

The B/W paradigm uses the partial ordering to keep the boundary representatives apart, but does not use target reversal. The purpose of this paradigm is to demonstrate the use of the partial ordering when learning using grids, and to illustrate the requirement for target reversal.

6i.4.1 Using the Partial Ordering

The B/W paradigm makes use of the partial ordering of the grids described in section 6i.1.2. The significance of the partial ordering for imperfect training *per se* is that it can be used to keep the two directions of the search apart, preventing false or premature locking due to the sub-optimum solutions returned by the training algorithms. The general significance of the partial ordering is that it enables candidate elimination

that goes beyond just the grids already visited by either of the two directions during learning. The partial ordering enables the representation of boundary members in the lattice. A boundary representative is a grid that lies at the head of a sub-lattice of grids where the other grids cannot be the goal concept. It will be seen that all grids lying in the sub-lattice behind any boundary representative can be eliminated without having been explicitly searched.

The partial ordering lays the foundation for candidate elimination and for using imperfect training. On this foundation, the notion of an *inherently better fit* is built. This concept is useful when the globally optimal fit cannot be guaranteed. A change of output grid is only made from one familiarisation to the next if it is certain that the proposed new output grid is an inherently better fit to the underlying target grid than the current output grid. This means that a move can be made without the need to be sure that the new output grid is a globally optimum fit to its target grid. Simultaneously, other unexplored grids may be eliminated since they are inherently worse fits than the new output grid. See figure 6i.13.

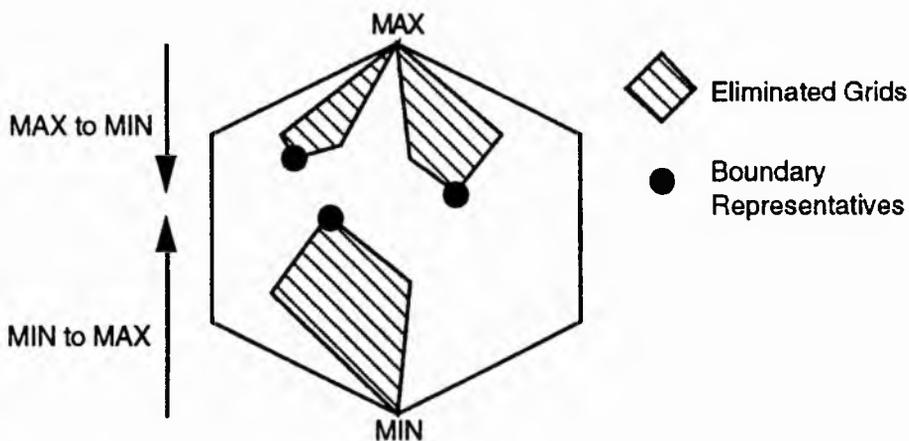


Figure 6i.13 — Elimination of grids using the partial ordering, which has maximal and minimal elements MAX and MIN, respectively. All grids behind the boundary representatives for each direction in the ordering can be eliminated since they are inherently worse fits to the underlying target grid than the boundary representatives.

The partial ordering and the concept of inherently better fit are used to try and make sure that any sub-optimum grid is behind in the ordering from the optimum grid. Thereby the possibility of false locking is prevented

whilst enabling candidate elimination. An additional feature of the grid technique for imperfect training algorithms is the design of inherent fit functions which connect a given output grid found by the training algorithm to its place in the partial ordering. Section 6i.4.2 will introduce the inherent fit functions used in the B/W paradigm.

Before a potential technique can be used with imperfect training, it is necessary to check that it works with perfect training. The technique must be checked under ideal training circumstances to see if it allows false locking under these circumstances. Whatever merits a proposed technique may have, they must take second place behind the necessity of avoiding false locking. The B/W paradigm will be shown in section 6i.4.3 to be a technique which allows false locking.

What is to be done about claims concerning the no-alternative situation when the training algorithm is imperfect? The aim is to try and retain these claims as much as possible, by relaxing the requirements on the training algorithm. (See sections 6i.4.2 and 6ii.1.1.)

In section 6i.2.1 the point was made that when all gridpoints are familiarised, the no-alternative situation must be reached, given perfect training. However, with imperfect training, this is not so, since the actual boundary representatives might not have caught up with the optimum grids in the ordering by the time all the gridpoints have been familiarised. This means that the effectiveness of a training algorithm in finding the globally optimum grid may be indicated by the number of trials which satisfy the locking conditions by the time all the gridpoints have been familiarised. (See section 6ii.1.2.)

6i.4.2 Inherently Better Fit

A condition is needed which makes use of the partial ordering of grids, and yet does not depend on achieving the globally optimum solution at every stage in the training sequence. This condition is that of inherently better fit, as mentioned earlier. The *predecessors* of a grid, G , relative to a root grid, R , is a term that will be used henceforth to describe the set of grids which belong on any chain in the partial ordering which goes from R to G .

The idea is that the grid chosen for one of the boundary representatives should always be an inherently better fit than any of its predecessors to the root grid, regardless of any future familiarisation. This enables any inherently worse fit than the grid chosen to be eliminated from the search, not just those which have been the current grids for earlier stages of learning. Using an inherently better fit is intended to enable boundary representatives to move without passing over the goal concept, even though they may not have the best fit to a target grid.

Two boundary representatives are used, as before, though they will be termed B and W. B has the all black grid as its root grid, R_B , and W has the all white grid as its root grid, R_W . The setting of the target grids for B and W is also different from the perfect training technique. The initial target grid for B, T_{B0} , is equal to R_B , and similarly for the initial target grid for W. Thus, hypothetical target gridpoints are always equal to the root gridpoints. The familiarisation of a gridpoint means setting the gridpoint in the target grids of both B and W to have the true target, determined by the patterns, rather than using the root value as a default.

Let the absolute fit of a grid measure the fit of an output grid designed to indicate members of version space (including the boundary members) through their being equal to or above a particular fit. Let ALF be a function which measures the absolute fit of a grid G , with root grid, R , to a target grid, T . The absolute fit, ALF, of G will be considered in three stages of relaxing restrictions on G . In the first stage, the only gridpoints in G which are of colour opposite to R are correctly classified real gridpoints. Let this number of gridpoints be ORC. 'O' represents a colour opposite to the root grid for G . 'R' is used to indicate that the variable pertains to real gridpoints. 'C' indicates that the variable counts the correctly classified gridpoints (i.e. those for which G is equal to T).

All predecessors of G only differ from G in having less of the gridpoints that G has of opposite colour to R . (See figure 6i.14.) Since G is given as correctly classifying all its points of opposite colour (which are real gridpoints), its predecessors must be poorer fits to any underlying target grid. This fact corresponds in this case to the predecessors having a less than perfect ORC score, which can therefore be used to eliminate them. That is, because the predecessors only differ from G in having more points

of known misclassification, such grids must be inherently worse than G , regardless of any future familiarisation. Therefore, for the first stage, we have:

$$ALF'' = ORC$$

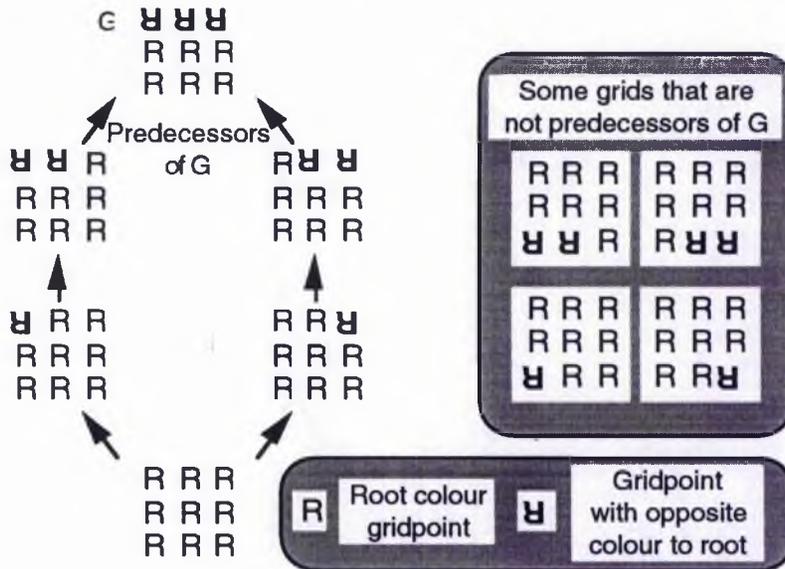


Figure 6i.14 — Grids that are predecessors of G have non-root gridpoints only where G has non-root gridpoints, and root-coloured gridpoints otherwise.

In the second stage, the restriction on G is relaxed, in that some of the opposite colour real gridpoints in G may misclassify their targets. Let ORM be this number of points, where 'M' is used to represent misclassified gridpoints. With this relaxation of restriction on G , the predecessors may have a worse, equal, or better degree of fit relative to G . However, G may be optimised to have the best degree of fit, by subtracting ORM from ALF , so that the predecessors cannot have a better fit. The degree of fit to the real gridpoints of opposite colour to R is now:

$$ALF' = ORC - ORM$$

Even after optimisation, there remains the possibility that a predecessor may have the same best degree of fit that G has. This fit is an integer and so the problem may be dealt with by adding a fractional score between 0 and 1 to reflect the closeness of each grid to the root grid. Optimisation

then yields the grid with the best degree of fit in terms of ORC – ORM that is the closest to the root grid.

In more detail, let the fractional closeness score be given by CLS. This term counts the number of gridpoints in G that have root-coloured output. The number is then expressed as a fraction between 0 and 1. We have:

$$CLS = \frac{n - \sum_{i=1,n} |R_i - G_i|}{n + 1} \quad [6i.14]$$

where n is the number of gridpoints, and $G_i \in \{1, 0\}$ be the value of the i th gridpoint of grid G .

The overall degree of fit is therefore given by:

$$ALF' + CLS \quad [6i.15]$$

In the third stage, the restriction on G is relaxed further, to allow the remaining possibility of hypothetical gridpoints in G which are opposite in colour to the root grid, R . Since the hypothetical gridpoints all have targets equal to R 's targets, all opposite colour hypothetical gridpoints in G are misclassified. Let OHM be this number of gridpoints, where 'H' is used to represent hypothetical gridpoints. OHM needs to be a penalty term since the hypothetical targets could have root values in reality. That is, OHM is subtracted from ORC – ORM, since this prevents the possibility of there being a grid behind G which might have a better absolute fit to the true (i.e. real) targets of the hypothetical gridpoints, if those targets turn out to be root targets.

The absolute fit is now given by:

$$ALF = ORC - ORM - OHM \quad [6i.16]$$

The possibility that a predecessor may have the same best degree of fit that G has is dealt with again by adding a CLS term. The overall inherent fit, IHF, which when maximised indicates the boundary members of version space, is given by:

$$IHF = ALF + CLS \quad [6i.17]$$

Figure 6i.15 summarises the terms used for any absolute fit formula.

As each gridpoint is familiarised, the grid with maximum inherent fit for B and W is selected, in the ideal case. Termination occurs when (i) the classification error of B on T_W is equal to the classification error of W on T_W , and (ii) the classification error of W on T_B is equal to the classification error of B on T_B . At this point, B and W have converged, and no grid on the same chain as B and W has a better inherent fit to the true target grid.

The satisfaction of conditions (i) and (ii) mean that there is no point in learning any further, for if B has the same error as W on W's targets, then there is no point in making B any whiter through familiarisation of the hypothetical gridpoints, since W's corresponding targets are already all white with no change in the degree of fit. Similarly, if W has the same error as B on B's targets, then there is no point making W any blacker, since B's targets are already all black with no change. This is therefore a locking point.

Real		Hypothetical		
Equal Colour to Root	Opposite Colour to Root	Equal Colour to Root	Opposite Colour to Root	
ERC	ORC	ERC	ORC	Correctly Classified
ERM	ORM	EHM	OHM	Misclassified

Figure 6i.15 — Summary of terminology for absolute fit formulae.

Since B is the blackest grid with the maximum inherent fit to B's targets, no grid behind B in the ordering will be better. Similarly, no grid behind W will be better, since W is the whitest grid with maximum inherent fit to W's targets. Therefore, no grid which lies in the same sub-lattice as B or W can be a better fit to the underlying target grid.

6i.4.3 False Locking Using the B/W Paradigm

Although B and W have the best inherent fit within their sub-lattices, this may not correspond to the best underlying fit. This is because the best inherent fit does not guarantee that there is no other grid in a different sub-lattice with lower classification error than B or W, given the true

targets of the hypothetical gridpoints. There may be grids in different sub-lattices from B and W with better performance on the true targets. This can be shown in practice through the exhaustive search method used in the H/H0 paradigm to test the performance of the B/W paradigm under ideal training conditions. An example of termination on grids which are not globally optimal is given in figure 6i.16.

B and W falsely lock in figure 6i.16 because the familiarisation order is such that B and W remain on the wrong sub-lattice in the ordering. The global minimum error grids lie on a different sub-lattice. This is indicated by the lattice in figure 6i.17, which indicates the sub-lattices, $B(l)$ and $W(l)$, on which B and W lock, and the sub-lattices, $B(o)$ and $W(o)$ which contain the best fit grids. Grid P is the grid on which B and W lock, which has an absolute fit of 1 for W, and an absolute fit of 5 for B. Grids Q and R are the global minimum error grids (see figure 6i.16), which lie on separate sub-lattices from the locking sub-lattices. Q has an inherent fit of 0 for W, and R has an inherent fit of 4 for B. This indicates that there are multiple boundary representatives for B and W, which lie on different sub-lattices in the ordering, with different absolute fit scores for the best inherent fit grids, which lie at the pinnacles of the sub-lattices.

In none of the runs that were done using this paradigm (under perfect training circumstances), did B or W ever go back in the ordering. This gives experimental support for the theory, that using inherently better fit guarantees that grids behind B and W in the ordering can be eliminated from consideration.

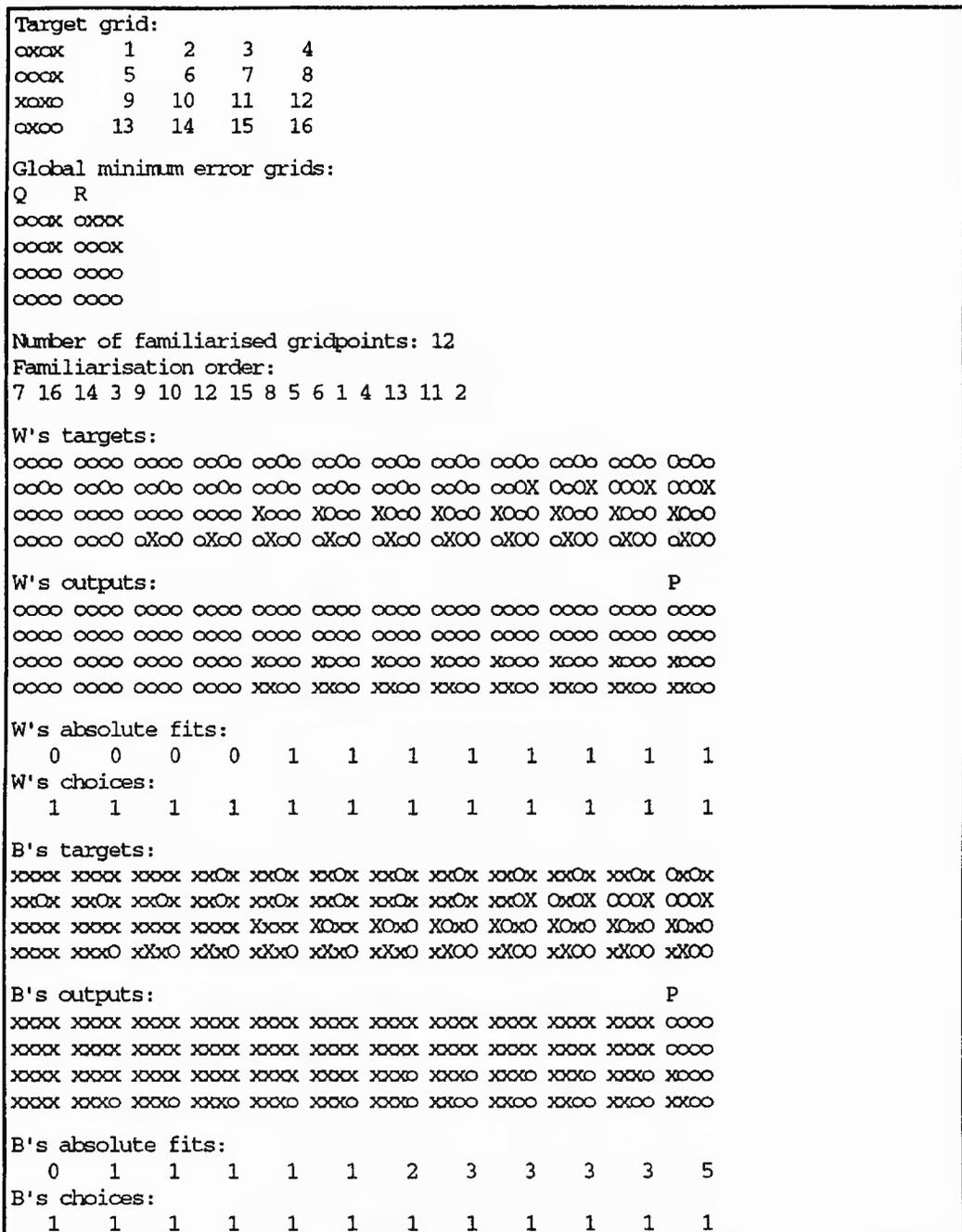


Figure 6i.16 — *B* and *W* converge, but on the wrong sub-lattice. They are the best inherent fit grids on their sub-lattice, but the global minimum error grid lies on a different sub-lattice. Upper case gridpoints in the targets indicate real gridpoints. The index numbers of the gridpoints are indicated next to the target grid, which are used to indicate the order in which the gridpoints are familiarised. The number of choices and the absolute fits of *B* and *W* are indicated for each familiarisation. *B* and *W* lock on grid *P*, shown in the lattice in figure 6i.17.

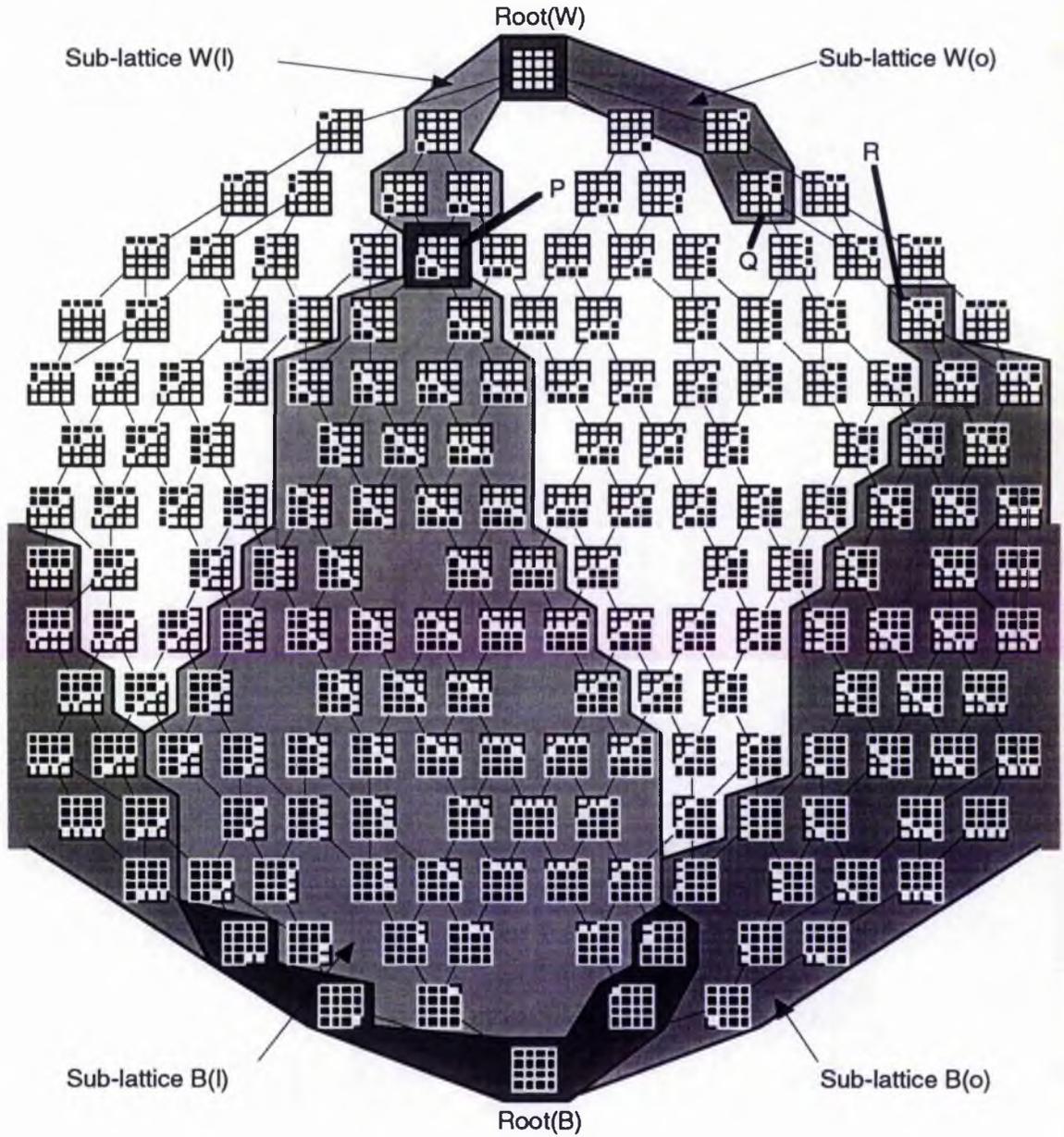


Figure 6i.17 — Complete lattice of 4×4 output grids, showing the sub-lattices of B and W at locking, B(l) and W(l), and the sub-lattices which contain the optimum grids for B and W, B(o) and W(o). Intersection of sub-lattices is indicated by the darkest shaded areas. Note that sub-lattice B(o) appears to be on both sides of the lattice as a whole. In order for a neat 2D representation of the whole lattice, certain grids are repeated on the left and right hand sides. Therefore the two sides of B(o) are actually all part of the one sub-lattice. Grids P, Q and R lie at the pinnacles of the sub-lattices in which they are contained.

6i.5 The I/IO Paradigm

The H/H0 paradigm, in section 6i.3, provided correct locking whilst the B/W paradigm provides candidate elimination, and a means for coping with imperfect training. The two paradigms are combined in the I/IO paradigm, which uses target reversal and the partial ordering to keep the boundary representatives from locking falsely. The I/IO paradigm is shown by the experiments in part (ii) of this chapter to have promising results for knowing when the no-alternative situation has been found.

The H/H0 paradigm had terminating conditions which were able to guarantee correct locking when it occurred. The key was that target reversal of H0's hypothetical gridpoints from H's classification made no difference to the classification of the hypothetical gridpoints by H0. However, for imperfect training, the H/H0 paradigm's defence against the possibility of false locking was limited to target reversal only. This defence was insufficient since false locking may occur in the H/H0 paradigm due to the two directions being imperfectly kept apart when the training algorithm returns sub-optimum results.

This target reversal, however, would also be useful in the context of the B/W paradigm, in that it could encourage the boundary representatives to lie on different sub-lattices in the ordering where possible. This would prevent the boundary representatives from falsely locking in the way that occurred in the B/W paradigm.

Essentially, the new paradigm uses the B/W paradigm for one boundary representative, and the H/H0 paradigm for the other, with a modified inherent fit formula.

Let I be the boundary representative based on the B/W paradigm. This will be a hypothesis for generalisation on the basis of inherent fit. The root grid of I , R_I , depends on the target of the first real gridpoint. If the first gridpoint to be familiarised has a target of 1, then the root grid of I is R_B . Otherwise the root grid of I is R_W . The targets for I are then set in accordance with the B/W paradigm, and I is found by maximising the inherent fit formula in [6i.17].

I0 is then the boundary representative for the opposite direction, and is taken from the H/H0 paradigm, which corresponds closely with H0. I0 represents the hypothesis that the generalisation of I to the hypothetical gridpoints is entirely incorrect, but with the opposite root grid, R_{I0} , to I. The targets of I0 are set in the same way as the targets of H0 — real gridpoint targets are set according to the underlying patterns, and hypothetical targets are set to be the opposite of the classification of I for those points. The absolute fit formula for I0 must be modified from [6i.16] to take into consideration the fact that the hypothetical targets have been made the opposite of the classification of I, whereas in [6i.16] the hypothetical targets all take values corresponding to those of the root grid.

The cases whereby I correctly classifies its hypothetical targets yield the same targets for I0 as in the B/W paradigm, i.e. the root values for I0. The difference is when I misclassifies any hypothetical targets. The corresponding hypothetical targets in I0's target grid will then be of the opposite colour to the root grid for I0, rather than the same colour as in the B/W paradigm. I0 must be encouraged to correctly classify these targets, in order that its outputs for the hypothetical gridpoints should be as different from I as possible. This retains consistency with the H/H0 paradigm, in that I0 is working on the assumption that the generalisation of I is incorrect. Therefore OHC is added to ALF from [6i.16], to give the absolute fit formula for the I0 direction, ALF_{I0} , in [6i.18].

$$ALF_{I0} = ORC - ORM - OHM + OHC \quad [6i.18]$$

and thus the inherent fit for the I0 direction, IHF_{I0} :

$$IHF_{I0} = ALF_{I0} + CLS \quad [6i.19]$$

I0 is then found by maximising the inherent fit formula in [6i.19]. This may well mean that I0 will go back and forth in the ordering for certain familiarisation sequences, as the hypothetical targets for I0 are changed in accordance with I's classification. This means that there is no certainty that grids which lie behind I0 in the ordering are necessarily discarded for good. However, there is this certainty for I.

As well as using the CLS term to make sure that I and I0 encourage minimal change along Mitchellian lines, a further heuristic is used.

Specifically, a change in output grid for I or IO is only accepted if that grid has a better inherent fit than the last output grid on the new target grid. Hence, any change in output grid is made only when it is necessary for an improved fit.

The terminating condition is the same as for the H/H0 paradigm: IO must misclassify all the reversed targets of the hypothetical gridpoints, with the same error on the real gridpoints as I. Thus, IO agrees with the generalisation hypothesis of I despite being trained with the opposite hypothesis so that there is no better fit to the underlying target grid.

The I/IO paradigm makes weaker demands on the training algorithm than the H/H0 paradigm. The H/H0 paradigm requires the globally optimum grid to be chosen for H and H0 at each stage. Here, a sub-optimum (inherently better) grid may be chosen, with a lower absolute fit than the optimum grid, which will not constitute a risk of false locking while it is further back in the ordering than the optimum grid. The CLS term in the inherent fit functions and the minimal change heuristic above are intended to ensure this as much as possible.

The following gives a high-level algorithm for the implementation of the I/IO paradigm (comments are given to the right of the vertical lines):

- 1 Get input grid and data sample.
- 2 Assign targets to the gridpoints using the data sample.
 (See section 6i.2.1.) Let T be the underlying target grid.
- 3 Let $T(I)(n)$ be the target grid for I at the nth familiarisation. Let $T(IO)(n)$ be the target grid for IO at the nth familiarisation. Let $O(I)(n)$ be the output grid for I at the nth familiarisation. Let $O(IO)(n)$ be the output grid for IO at the nth familiarisation.
- 4 Let $F(n)$ be the familiarisation sequence. For the nth familiarisation, $F(n)$ gives the index of the gridpoint to be changed from hypothetical to real.
- 5 Let $n = 1$
- 6 Set $T(I)(1)$ to be the target of gridpoint $F(1)$ from T for all gridpoints.

| Initialise I's target grid to be a root grid.

- 7 Repeat ...
- | The familiarisation loop begins here.
- 7.1 Obtain the real target, t , of gridpoint $F(n)$ from T .
- 7.2 Set the $F(n)$ th gridpoint of $T(I)(n)$ to be t .
 | This familiarises the n th gridpoint for I 's target grid.
- 7.3 Set the $F(n)$ th gridpoint of $T(I0)(n)$ to be t .
 | This familiarises the n th gridpoint for $I0$'s target grid.
- 7.4 Obtain $O(I)(n)$ using the training algorithm to find the best inherent fit to $T(I)(n)$ using equation [6i.17], starting from a new random initial weight state.
 | Find a new output grid for I using the training algorithm.
- 7.5 If $n > 1$ and $O(I)(n-1)$ has an equal or better inherent fit as per [6i.10] to $T(I)(n)$ than $O(I)(n)$, then $O(I)(n) = O(I)(n-1)$.
 | Do not accept a new output grid for I unless it is better than the previous one.
- 7.6 Set the hypothetical targets of $T(I0)(n)$ by using the reverse of the outputs of $O(I)(n)$.
- 7.7 Obtain $O(I0)(n)$ using the training algorithm to find the best inherent fit to $T(I0)(n)$ using equation [6i.19], starting from a new random initial weight state.
 | Find a new output grid for $I0$ using the training algorithm.
- 7.8 If $n > 1$ and $O(I)(n-1)$ has an equal or better inherent fit as per [6i.12] to $T(I0)(n)$ than $O(I0)(n)$, then $O(I0)(n) = O(I0)(n-1)$.
 | Do not accept a new output grid for $I0$ unless it is better than the previous one.

- 7.9 If $O(I0)(n)$ misclassifies all its hypothetical targets in $T(I0)(n)$ with the same fit as $O(I)(n)$ to the real targets in $T(I)(n)$ and $T(I0)(n)$, then locking has occurred.
- | Test for locking
- 7.10 Increment n .
- 8 ... While $n \leq$ the total number of gridpoints and locking has not occurred.
- | End of the familiarisation loop.
- 9 If locking has occurred then report this, else report that locking has not occurred before all gridpoints have been familiarised.

6i.6 Conclusion to Part (i)

The use of grids enables optimum classification boundaries to be found, whilst relating the performance on the grid to continuous generalisation, as each gridpoint is made to stand for its local area of input space. If more finely detailed boundaries are required, then a finer grid resolution or richer topology should be chosen.

A use of the grid technique not discussed here is for those cases where there are gridpoints whose targets are not determined by the data sample. The task is to see if these *uninstantiated* gridpoints can be found a suitable generalisation from the familiarisation of the other gridpoints. Learning using uninstantiated gridpoints will not be examined here, since the aim is to show that the techniques can reach the no-alternative situation. If the no-alternative situation is not reached when there are uninstantiated gridpoints, then it is not possible to tell whether this is a problem of the technique, or whether there are too many uninstantiated gridpoints. It is thought best to avoid this extra dimension of complexity in this initial development and exploration of techniques.

The H/H0 paradigm illustrates the correctness of the terminating condition. It shows the ability to recognise when H and H0 are grids that have globally optimum fits whatever the true targets of the hypothetical gridpoints. Given a nearly perfect training algorithm, it would rarely lock falsely at the early stages of learning.

In the discussion of the H/H0 paradigm it was pointed out that, given perfect training, the usefulness of Mitchell's no-alternative situation in neural networks, is obscured by there being no better alternative than an optimum solution by definition. This, coupled with the fact that (short of exhaustive search) no truly perfect training algorithm exists in practice, means that the usefulness of Mitchell's technique becomes clearer when considered for *imperfect* training algorithms — those which cannot provide a guaranteed global minimum misclassification error solution. Imperfect training algorithms may well arrive at optimum solutions some of the time, but it is difficult to be sure that this is indeed the case when it does occur, unless the optimum solution is known in advance.

The B/W paradigm shows the usefulness of the partial ordering and the concept of inherent fit as further means to keep the boundary representatives apart. This enhances the use of imperfect training algorithms. It has the further benefit of candidate elimination, which means that grids can be eliminated without needing to be inspected by either of the boundary representatives. The terminating condition of the B/W paradigm was not strong enough, since it used the partial ordering only as a basis. It recognised when B and W had locked, but was unable to prevent the possibility of the existence of grids outside the set of predecessors of B and W that were better overall fits. The H/H0 paradigm's stronger terminating condition is also needed.

Part (i) of this chapter has established a theoretical framework for recognising when a best fit grid has been found, in the form of the I/I0 paradigm. By combining target reversal, the partial ordering and the inherent fit functions, the I/I0 paradigm is designed to keep the boundary representatives apart until locking occurs on a grid with the optimum fit to the underlying target grid. This is explored empirically in part (ii).

The important theoretical contribution that Mitchell's technique has to make is that it can provide a method for recognising when the goal state has been reached in situations where there are no unique features by which the goal state can be distinguished from local minima. For inexact fit, the goal cannot be recognised by zero misclassification error, as is the case with exact fit. With inexact fit, if the global minimum misclassification error is unknown, then there is no possibility to directly

detect when the goal has been reached. Gradient descent methods are able to detect when a minimum has been reached, but the minimum could be local. Mitchell's technique provides a mechanism for recognising the goal state indirectly.

(ii) Experiments on the I/O Paradigm

6ii.1 Experiments and Results

In part (ii), some experiments are reported which show the I/O paradigm at work. One main object of interest was whether or not the I/O paradigm could falsely lock, in the same way as the B/W paradigm did. Were the modifications to the inherent fit formula and the H/H0-style target settings enough to prevent false locking? Tests were conducted using exhaustive search on a 4×4 grid. All 2^{16} possible target grids were tested with a single random presentation order each. There were no false locking examples in any of the trials.

Having thereby established some empirical support for the paradigm, tests were conducted using imperfect training. A GA was used as an imperfect training algorithm. Appendix 6.B describes the weight state encoding mechanism and the selection procedure. A population size of 100 was used, with a mutation probability of 1.0 and a crossover probability of 1.0. See appendix 6.C for an explanation of these probabilities.

An example trial is given in section 6ii.1.1. Further experiments were done which compared the performance of the GA with perfect training (6ii.1.2), and which compared the performance of the I/O paradigm against the H/H0 paradigm (6ii.1.3).

In section 6ii.1.4, there is a comparison with validation, which shows that validation regularly finds a poorer overall solution than the I/O paradigm, and that there is uncertainty, using validation, about when to stop training. Section 6ii.1.5 shows that the size of validation set makes little difference to the overall performance.

Section 6ii.1.6 explores the scalability of the technique, using a larger grid, and a topology with hidden units.

Finally, section 6ii.1.7 shows how the technique may be used to tackle a simple symbolic concept-learning problem.

6ii.1.1 Example Run

The purpose of the example run is to show how imperfect training can work with the I/I0 technique. A GA is used, and the example run will demonstrate how I0 may go back in the ordering due to the hypothetical targets changing back and forth between black and white, as I's classification of them changes. Also shown is how I and I0 may jump chains during the search — meaning that one boundary representative for each direction is sufficient for the neural technique. Finally, the recovery from a sub-optimum grid returned by the GA is illustrated, leading to correct termination with 16% of the gridpoints remaining hypothetical.

This experiment, and some of the following experiments, will be based on a simple problem, which is to be approached using a 2*1 topology. Figure 6ii.1(a) shows the underlying distribution from which the patterns are taken, and figure 6ii.1(b) shows the desired solution. The thin strip in figure 6ii.1(a) is not to be correctly classified by the neural network, with the simple topology finding a rule which matches with the majority of the data. The topology is deliberately kept simple so that its IO behaviour (a linear separation) can be easily seen.

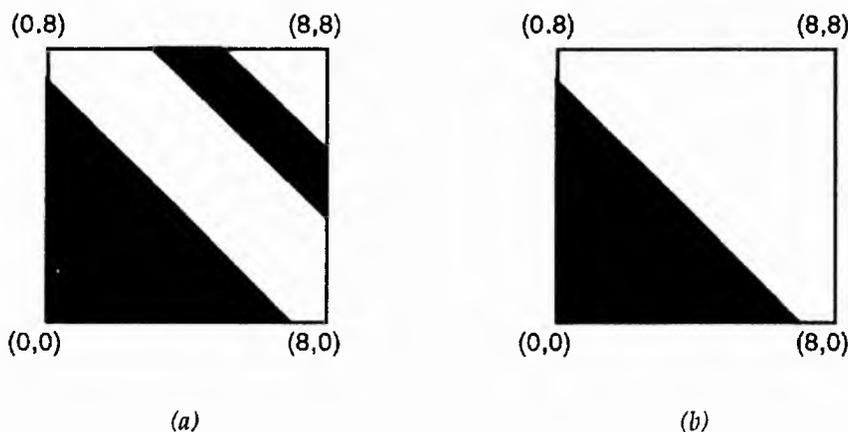


Figure 6ii.1 — (a) Underlying distribution of experiment. (b) Desired final solution to be found by 2*1 topology.

A sample of 200 patterns are taken from the underlying distribution, and a grid constructed with targets assigned in accordance with the patterns. The grid and patterns are shown in figure 6ii.2.

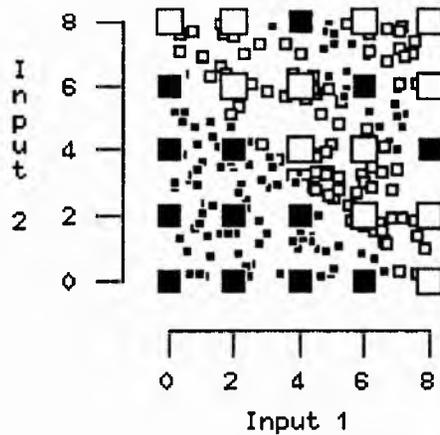


Figure 6ii.2 — *Gridpoints and patterns for a simple experiment. The patterns are indicated by the smaller squares, and the gridpoints by the larger squares. Black squares indicate a target of 1, and white squares indicate a target of 0.*

The example run given below used 100 generations to train each target grid for I or I0 to try to find the maximum inherent fit value in the allotted time. The trial below shows selected portions of the output generated by the software. The gridpoint index numbers are given first. The apparently random assignment of index numbers to gridpoints is based on the order in which the patterns in the sample from the underlying distribution appear in the file that contains them. The software creates the grid automatically, given a grid spacing and the co-ordinates of a single potential gridpoint. When a pattern is processed from the file, it must be assigned to a gridpoint. If a suitable gridpoint has not been created, then the software creates a new gridpoint. The order of familiarisation of the gridpoints is given below the gridpoint index numbers.

The trial itself shows, for each familiarisation, the number of the familiarisation, the index of the gridpoint being familiarised, and the target grid, output grid and absolute fit (ALF) of I and I0. Also shown is whether or not I or I0 have gone back in the ordering for the given familiarisation. This is indicated by the letter 'B' to the right of the output grid which has gone back in the ordering. The trial was also performed using perfect training with the same familiarisation sequence, in order to see how close the GA was to the optimum at each stage. The headings at

various stages through the trial indicate some of the features of learning using the I/I0 paradigm illustrated by the grids found. The output shown is from the GA trial.

Index numbers for gridpoints:

17	10	19	21	22
24	15	12	9	13
6	2	11	14	8
20	7	1	4	3
25	18	5	23	16

Familiarisation sequence:

21 7 6 3 9 17 2 23 14 18 4 13 24 20 15 22 11 12 1 10 16 19 5 25 8

- **Moving through the ordering.**

The perfect training and GA trials choose the same grids for I and I0 during the early familiarisations, as indicated by the following output from the GA trial.

Fam no.	GP no.	I target	I output	I ALF	I0 target	I0 output	I0 ALF
1	21	oooo	oooo	0	xxxOx	xxxOx	0
		oooo	oooo		xxxxx	xxxxx	
		oooo	oooo		xxxxx	xxxxx	
		oooo	oooo		xxxxx	xxxxx	
		oooo	oooo		xxxxx	xxxxx	

I and I0 remain on the root grid until familiarisation 6:

6	17	OooOo	OooOo	0	OxxOx	OxxOx	1
		ooXo	oooOo		xxxOx	xxxOx	
		Xoooo	Ooooo		Xxxxx	Xxxxx	
		oXooo	oOooO		xXxxO	xXxxO	
		oooo	oooo		xxxxx	xxxxx	

At familiarisation 6, I0 moves to correctly classify gridpoint 17 (top left), which has just been familiarised. The GA has found the optimum solutions so far for each familiarisation.

- **Staying back in the ordering when the GA does not find the optimum grid.**

On familiarisation 10, the perfect training trial makes a change for I and I0, to move to better fits. (See figure 6ii.3.) The GA does not find these fits, and I and I0 retain the grids from familiarisation 6, which are behind the optimum grids in the ordering. This ensures that the boundary

representatives are kept apart and prevented from locking prematurely due to the training algorithm not returning the optimum grid at each stage. This shows how the use of the partial ordering has enabled the I/I0 paradigm to cope with imperfect training algorithms better than the H/H0 paradigm would.

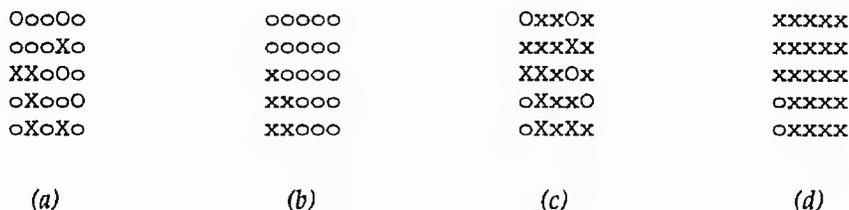


Figure 6ii.3 — Perfect training results for familiarisation 10. (a) Target grid for I at familiarisation 10. (b) Output grid for I using perfect training. (c) Target grid for I0 at familiarisation 10. (d) Output grid for I0 using perfect training.

- **Illustrating target reversal and chain jumping.**

The following output illustrates the state of I, I0 and their target grids at familiarisation 13 of the GA trial.

Fam no.	GP no.	I target	I output	I ALF	I0 target	I0 output	I0 ALF
13	24	OooOo	OooOo	2	OxxOx	XxxXx	2
		XooXO	XooOO		XxxXO	XxxXX	
		XXoOo	XOoOo		XXxOx	XXxXx	
		oXoOO	xXoOO		oXxOO	oXxXX	
		oXoXo	xXoOo		oXxXx	oXxXx	

On familiarisation 13 of the GA trial, I moves to a better absolute fit position than the root grid, and catches up with the perfect training trial, which has exactly the same grids for I and I0 at this point. The two hypothetical gridpoints in the bottom left (numbers 20 and 25) are now given the opposite classification to I's root grid. I0's target grid then has targets which are of opposite colour to I0's root grid for these gridpoints. I0 moves to a better absolute fit position which correctly classifies these targets. Note that I0 is no longer on the same chain in the partial ordering as it was in familiarisation 6.

- **I0 may go back in the ordering.**

The following output illustrates the state of I, I0 and their target grids at familiarisation 14 of the GA trial.

Fam no.	GP no.	I target	I output	I ALF	I0 target	I0 output	I0 ALF
14	20	OooOo	OooOo	4	OxxOx	XxxXx B	1
		XooXO	XooOO		XooXO	XooXX	
		XXoOo	XOoOo		XXxOx	XXxXx	
		XXoOO	XXoOO		XXxOO	XXxXX	
		oXoXo	xxoOo		oXxXx	oXxXx	

Gridpoint 20 is familiarised next. I's classification of it was correct previously, and hence there is no change to I. However, the change to I0's target grid means that a better absolute fit grid can be obtained by correctly classifying the now real gridpoint 20, which leads to I0 going back in the ordering. This is indicated by the letter B to the right of the output grid. The perfect training trial does exactly the same thing at this point. The I/I0 paradigm allows that I0 may go back in the ordering occasionally as the hypothetical targets are changed due to a change in I's classification or familiarisation occurs.

- **The GA does not always return grids that meet with the specifications of the I/I0 paradigm that are necessary in order to guarantee true locking.**

The grids of I and I0 in the imperfect training trial remain the same until familiarisation 19. The perfect training trial changes I0 at familiarisation 17. However, the GA does not find a better solution than the grid from familiarisation 14 until now:

19	1	OooOO	OooOO	6	OxxOO	XxxOO	3
		XOOXO	XOOOO		XCOXO	XXOOO	
		XXOOo	XXOOo		XXOOx	XXOOo	
		XXXOO	XXXOO		XXXOO	XOOOO	
		oXoXo	xxXOo		oXoXx	oOOOo	

Here, I moves to correctly classify the majority of real gridpoints — with only one real gridpoint misclassified. Target reversal means that the two gridpoints on the top row — numbers 10 and 19, are assigned targets that disagree with the final desired solution grid. These targets are correctly classified by I0, which moves to a better fit grid than it had after familiarisation 14.

However, though this grid is a better fit than that at familiarisation 14, it is not the optimum grid. The optimum grid is shown in figure 6ii.4(a). This does not matter. What is of more significance is that the grid found for IO is not the closest grid to IO's root grid with an absolute fit of 3. A grid which is on the same chain as the grid found for IO at familiarisation 19, lies behind it in the ordering, and has the same absolute fit of 3, is shown in figure 6ii.4(b). There is also a grid with a better absolute fit than the grid found for IO at familiarisation 19, which also lies behind it in the ordering. This is indicated in figure 6ii.4(c).

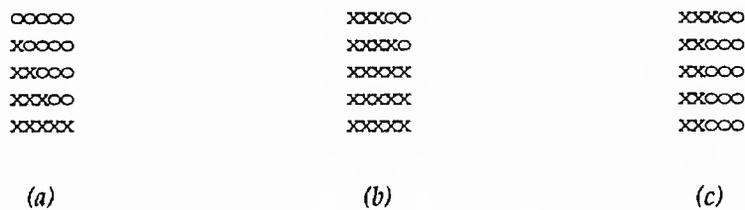


Figure 6ii.4 — (a) Optimum grid for IO at familiarisation 19, with an absolute fit of 6. (b) Grid with same absolute fit as the grid found for IO at familiarisation 19, but lies behind it, on the same chain in the ordering. (c) Grid with a higher absolute fit of 4, lying on the same chain behind the grid found for IO at familiarisation 19.

Thus, the GA, although an imperfect training algorithm, still does not meet the requirements of the I/IO paradigm necessary to always guarantee true locking. This is because as well as not being able to guarantee the optimum grid, a GA also cannot guarantee that any sub-optimum grid found is behind the optimum grid in the ordering. The CLS term in the inherent fit function can only encourage the GA to go for solutions which are closer to the root grid. It cannot guarantee the closest grids to the root grid, which would assure that any grid found by the GA would always be at or behind the optimum grid in the ordering. This means that it is possible for the GA to return grids which are ahead in the ordering. It is for this reason that the claims at locking must be watered down, as false locking could possibly occur when the GA returns grids that are ahead of the optimum grid in the ordering. It may be necessary in future to distinguish between *ideal* imperfect training algorithms, that meet the I/IO training specifications, and *sub-standard* imperfect training algorithms, that do not.

- **The imperfect training algorithm may catch up with the perfect training algorithm, locking just as quickly.**

The following output illustrates the state of I, I0 and their target grids for the last two familiarisations of the GA trial.

20	10	OOOO XOOXO XXOOO XXXOO oXoXo	OOOO XOOOO XXOOO XXXOO xXxXo	6	OOxOO XOOXO XXOOx XXXOO oXoXx	XXxOO XXOOO XXOOO XOOOO OOOOO	3
21	16	OOOO XOOXO XXOOO XXXOO oXoXO	OOOO XOOOO XXOOO XXXOO xXxXO	6	OOxOO XOOXO XXOOx XXXOO oXoXO	OOOOO XOOOO XXOOO XXXOO xXxXO	9

Finally, after familiarisation 21, I0 finds the solution grid, and I and I0 lock. There are four gridpoints remaining to be familiarised. Figure 6ii.5 shows the final solutions of I and I0 for this trial. The perfect training trial also terminates at familiarisation 21. This shows that it is possible for imperfect training to lock at the same time as perfect training.

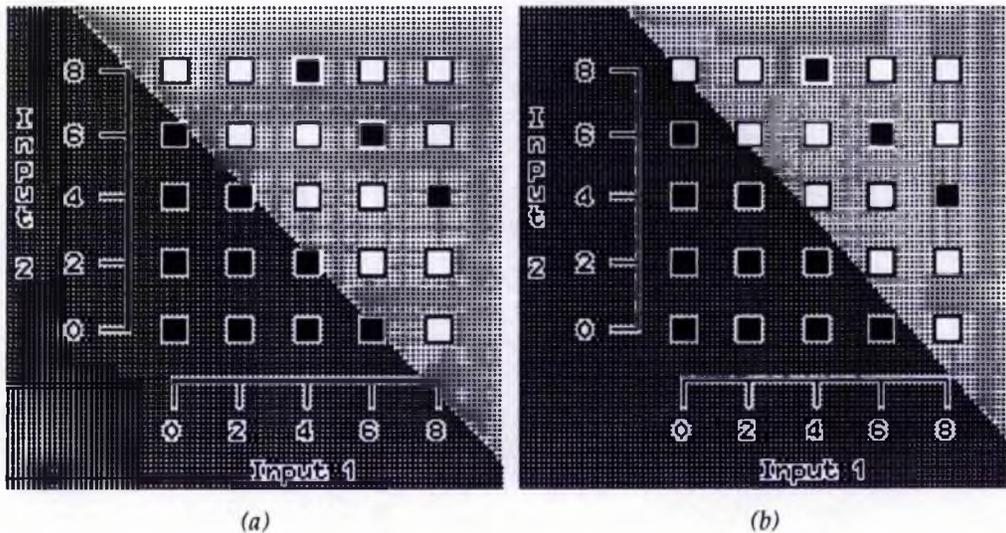


Figure 6ii.5 — Final solutions found by (a) I and (b) I0. The underlying target grid is represented by the black and white squares. The solutions found by I and I0 are represented by the light and dark grey areas. The dark grey areas represent an output of 1. The light grey areas represent an output of 0.

- **The accuracy of the generalisation is given by the resolution of the grid.**

Although the separating hyperplanes are in the correct position for the grid, it is interesting to note the precise positions of the hyperplanes found for I and I0 in figure 6ii.5. The hyperplane for I lies close to the black gridpoints, and the hyperplane for I0 lies close to the white gridpoints. This means that patterns from the original pattern set are likely to be misclassified in this borderline region. The grid philosophy, however, is that if greater accuracy is required on the borderlines between black and white gridpoints, then a more refined grid should be used, with a greater number of gridpoints.

In summary, this trial shows how it is possible to recover from the suboptimum solutions returned by the GA, and still terminate correctly with the same number of hypothetical gridpoints as a perfect training trial would. The partial ordering and inherent fit functions have discouraged I and I0 from falsely locking in this trial, and by familiarisation 21, I and I0 have locked on the best fit grid to the underlying target grid. The requirement for a globally optimum solution at all times during familiarisation has been successfully relaxed.

6ii.1.2 Comparison with Perfect Training

It is interesting to observe the difference in overall as well as individual performance between perfect and imperfect training. This is the purpose of this experiment, which is designed to show the effect of better and better training on the ability of the technique to find the no-alternative situation. The same underlying distribution and input grid as in section 6ii.1.1 may be used to compare perfect and imperfect training using the I/I0 paradigm. One hundred trials were performed, using different random presentation orders, for each training mechanism — exhaustive search and a GA with 1, 5, 10 and 50 generations allocated for each familiarisation, to compare performance. The results are given in table 6ii.1.

The number of false locking trials are indicated, where appropriate. False locking occurs only because the training algorithm does not meet the

requirements of the I/IO paradigm, not because there is anything wrong with the theory. The incidents of false locking are dramatically reduced as the training improves. The $N = 1$ trials falsely lock most of the time. Indeed, only two of the trials that locked were not false locking trials. It is rather unreasonable, however, to expect any kind of performance from using only a single generation to train the network. With $N = 5$, there are only three false locking trials — a huge reduction for only a few extra generations of training. None of the trials with 10 or 50 generations per familiarisation locked falsely.

Termination	Genetic Algorithm with N generations						Exhaustive Search
	N = 1		N = 5		N = 10	N = 50	
		false		false			
Unterminated	76%	N/A	39%	N/A	13%	2%	0%
0 hypothetical	10%	100%	33%	3%	38%	36%	29%
1 hypothetical	8%	88%	16%	6%	30%	33%	30%
2 hypothetical	2%	50%	4%	25%	11%	14%	19%
3 hypothetical	2%	100%	6%	0%	6%	8%	12%
4 hypothetical	0%	0%	2%	0%	2%	3%	5%
5 hypothetical	1%	100%	0%	0%	0%	3%	2%
6 hypothetical	1%	100%	0%	0%	0%	1%	3%

Table 6ii.1 — Comparison of perfect and imperfect training with various numbers of generations for 100 different random presentation orders on the underlying target grid in section 6ii.1.1. The percentages are of the number of trials (i.e. 100) for the training mechanism in that column, except for the "false" columns. The "false" columns for $N = 1$ and $N = 5$ indicate the percentage of false locking trials with the number of hypothetical gridpoints at termination in that row. There were no false locking trials for $N = 10$, $N = 50$, or for perfect training.

Also shown is that the ability to terminate with hypothetical gridpoints remaining increases as the training algorithm improves. Only 14% of the trials using a GA with 1 generation per familiarisation terminate with hypothetical gridpoints remaining. This rises through 28% for 5 generations, and 49% for 10 generations to 62% for 50 generations. The termination profile broadly matches that of perfect training with $N = 50$ generations.

The GA, although it has fewer terminations with hypothetical gridpoints remaining than exhaustive search, nevertheless shows the ability to reach

locking, with at least some hypothetical gridpoints remaining. It should be remembered that the ability to reach locking is far more important than terminating with hypothetical gridpoints remaining as it represents achieving the best fit to the underlying target grid. The general trend of fewer unterminated trials as the training improves also shows that the I/IO paradigm can also be used to indicate how well the training algorithm is doing in finding the optimum grids.

6ii.1.3 Comparison with H/H0

The H/H0 paradigm may be compared with the I/IO paradigm to observe the effects of using the inherent fit function and partial ordering on the ability to terminate with hypothetical gridpoints remaining. It would be desirable to compare the two paradigms using identical familiarisation sequences for each trial. As it happens, however, the software is unable to guarantee the same orders of familiarisation for any two batches of more than one trial at a time. To get round this, large batch sizes were used, which counteracted the effect of any bias due to the H/H0 and I/IO paradigms having different random presentation orders. Perfect training was used in two batches of 2 000 trials with different random presentation orders for each trial on the underlying target grid used in section 6ii.1.1, one batch for the H/H0 paradigm and the other for the I/IO paradigm. The results are given in table 6ii.2.

Termination	I/IO	H/H0
Unterminated	0%	0%
0 Hypothetical	33%	21%
1 Hypothetical	25%	20%
2 Hypothetical	18%	20%
3 Hypothetical	11%	15%
4 Hypothetical	8%	13%
5 Hypothetical	4%	8%
6 Hypothetical	1%	3%
7 Hypothetical	0%	1%
8 Hypothetical	0%	0%

Table 6ii.2 — Comparison of the H/H0 and I/IO paradigms using perfect training on 2 000 different random presentation orders of the underlying target grid used in section 6ii.1.1.

The results in table 6ii.2 show that the I/I0 paradigm is less likely to terminate with hypothetical gridpoints than the H/H0 paradigm. Since both paradigms represent potential goal grids at every stage, this may seem paradoxical.

Recall from chapter 6 part (i) that version space consist of those output grids which are globally optimum solutions for any of the possible target grids formed by reversing any combination of any number of the targets of the hypothetical gridpoints. As discussed in chapter 6 part (i) and proved in appendix 6.A, the locking condition of the H/H0 and I/I0 paradigms detects when H and H0 (or I and I0) have found output grids that are globally optimum fits to all of the possible target grids. These output grids are special members of version space that are the locking grids.

When there are locking grids in version space, there may nevertheless be other output grids in version space that are globally optimum fits to one or more, but not all of the possible target grids. The H/H0 paradigm may choose any grid in version space as the output grids for the two directions of the search. The I/I0 paradigm, however, must choose those output grids that are at the edge of version space in the partial ordering. When there is a locking grid that is not at the edge of version space, the H/H0 paradigm may lock sooner than the I/I0 paradigm if it happens to select this grid for both H and H0.

Consider, for example, the two runs on a 3×3 grid shown in figure 6ii.6, one for the H/H0 paradigm (a), and the other for the I/I0 paradigm (b). Both have the same underlying target grid and familiarisation sequence. However, the H/H0 paradigm terminates earlier than the I/I0 paradigm.

Figure 6ii.7 shows all the possible underlying target grids at familiarisation 6, when the H/H0 paradigm terminates. It will be seen that H and H0 are globally optimum fits to all possibilities. They have therefore found the best fit solution, and it is correct for the H/H0 paradigm to terminate.

Target grid (index 34):
 oXo 1 2 3
 oox 4 5 6
 ooo 7 8 9

Familiarisation order:
 2 9 8 1 4 6 7 5 3

H's targets:
 .X. .X. .X. OX. OX. OX.
 O.. O.X
O .OO .OO .OO .OO

H's outputs:
 xxx xxx xXo oXx oXx oXx
 xXo xxx xXo oox ooo oox
 xXo xXo xoo ooo ooo ooo

H's errors:
 0 0 0 0 0 0
 H's choices:
 29 17 14 2 2 1

H0's targets:
 oXo oXo oXo OXo OXo OXo
 oox ooo oox xXo Oxx OXX
 oox ooo ooo xOO xOO xOO

H0's outputs:
 oXx oXx oXx xXo xXo oXx
 oox ooo oox xXo xXo oox
 oox ooo ooo xoo xoo ooo

H0's errors:
 1 1 1 1 3 3
 H0's choices:
 2 3 1 1 8 8

(a)

Target grid (index 34):
 oXo 1 2 3
 oox 4 5 6
 ooo 7 8 9

Familiarisation order:
 2 9 8 1 4 6 7 5 3

I's targets:
 xXx xXx xXx OXx OXx OXx OXx OXx
 xxx xxx xxx xxx Oxx Oxx Oxx Oxx
 xxx xXo xOO xOO xOO xOO ooo ooo

I's outputs:
 xxx xxx xxx xxx xxx xxx xxx oXx
 xxx xxx xxx xxx xxx xxx oXx oXx
 xxx xXo xoo xoo xoo xoo ooo ooo

I's inherent fits:
 0 1 2 2 2 2 4 6
 I's choices:
 1 1 1 1 2 2 2 1

I0's targets:
 oXo oXo oXo OXo OXo OXo OXo OXo
 ooo ooo ooo ooo ooo oox oox OXX
 ooo ooo ooo ooo ooo ooo ooo ooo

I0's outputs:
 oXx oXx oXx oXx oXx oXx oXx oXx
 ooo ooo ooo ooo ooo oox oox oox
 ooo ooo ooo ooo ooo ooo ooo ooo

I0's inherent fits:
 2 2 2 2 2 3 3 3
 I0's choices:
 1 1 1 1 1 1 1 1

(b)

Figure 6ii.6 — Two runs using the same familiarisation order and underlying target grid, showing that the H/H0 paradigm (a) may terminate earlier than the I/I0 paradigm (b).

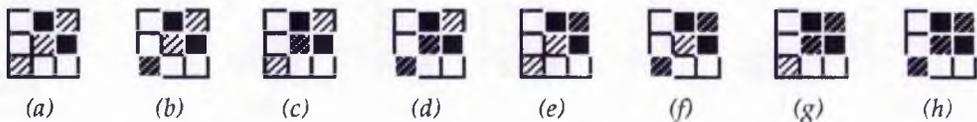


Figure 6ii.7 — (a)-(h) show the various possible underlying target grids at the sixth familiarisation for the runs in figure 6ii.6. The hypothetical gridpoints are indicated by the shaded cells.

Figure 6ii.8 shows version space at the 6th familiarisation. This contains more grids than just the grid found by H and H0. The I/I0 paradigm is restricted to the boundary grids in the partial ordering. It cannot terminate until both directions find a grid that is the best fit to all possible target grids. In the run in figure 6ii.6(b), the I direction must choose a blacker grid than the goal grid, since there are blacker grids on the same chain leading to I's root grid. I0 may choose the goal grid, since there is no whiter grid in version space that is on the same chain to the root grid for I0. The H/H0 paradigm may therefore lock earlier, since it has a wider set of possible choices for H and H0. The significance of this is that the choices include the special member of version space that is a globally optimum fit to all possible underlying target grids.

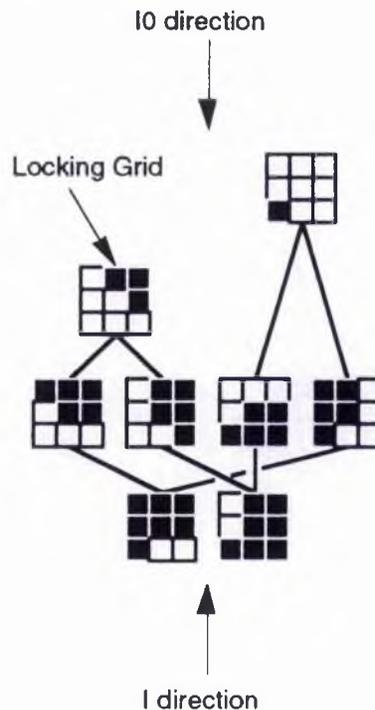


Figure 6ii.8 — Lattice of version space at familiarisation 6. The H/H0 paradigm may choose any of these. The locking grid is the grid which is a globally optimum fit to all possible target grids. I0 may choose the locking grid, since it is the closest to I0's root grid on that chain in the partial ordering. There are grids closer to I's root grid, however, and I must select one of these. Therefore I and I0 cannot lock.

The following summarises the general points to be made from this example:

- The choice of output grids for the H/H0 paradigm may include the locking grid at an earlier familiarisation than in the I/I0 paradigm.
- The wider choice for the H/H0 paradigm also includes pairs of grids that do not satisfy the locking condition, amongst which are the pair of grids chosen by I and I0 at the same stage.
- The I/I0 choice is restricted to pairs of grids that do not satisfy the locking condition until there is no alternative left but the locking grid or grids.

The apparent paradox mentioned earlier may now be resolved. Some potential goal grids in version space, X_i , can, in fact, be eliminated under a certain condition. This condition is that there exist other output grids in version space, Y_i , that are best fits to all the possible target grids that X_i are best fits to, as well as some other possible target grids that X_i are not best fits to. These other grids are locking grids.

The wider choice of available grids for H/H0 than I/I0 may be illustrated by counting the average numbers of alternatives for H, H0, I and I0 for each familiarisation. Table 6ii.3 shows the average number of choices for each of H, H0, I and I0 for each familiarisation during the 2 000 trials above. This shows that the H/H0 paradigm always has a greater choice of grid than the I/I0 paradigm.

Familiarisation	Average number of choices of grid			
	I	IO	H	H0
1	1.00	1.00	201.00	1.17
2	1.00	1.01	105.59	1.31
3	1.01	1.03	59.68	1.50
4	1.02	1.04	42.17	1.67
5	1.03	1.06	31.14	1.85
6	1.05	1.09	24.89	1.95
7	1.06	1.11	19.52	2.09
8	1.08	1.12	15.80	2.20
9	1.09	1.13	12.79	2.22
10	1.10	1.14	10.69	2.34
11	1.12	1.15	9.11	2.41
12	1.13	1.16	7.82	2.39
13	1.13	1.17	6.58	2.45
14	1.13	1.16	5.55	2.47
15	1.13	1.15	4.54	2.53
16	1.12	1.13	3.80	2.48
17	1.11	1.11	3.22	2.57
18	1.10	1.10	2.78	2.47
19	1.08	1.08	2.42	2.43
20	1.07	1.06	2.12	2.22
21	1.04	1.04	1.90	1.95
22	1.03	1.02	1.74	1.77
23	1.01	1.01	1.59	1.46
24	1.00	1.00	1.42	1.19
25	1.00	1.00	1.00	1.00

Table 6ii.3 — Average number of choices of grid for I, IO, H and H0 for each familiarisation during the 2 000 trials carried out in section 6ii.1.3.

6ii.1.4 Comparison with Validation

In order to compare the I/IO paradigm with validation, the validation technique was used in conjunction with a GA. Validation using a GA differs from using a gradient descent technique in that the latter aims at steadily reducing training error, rather than evolving a sample of weight space. However, by considering the best member of the population for each generation, a sequence of weight states is given which has a general trend of improvement in training error.

The validation technique is given the same underlying set of data as in the experiment in section 6ii.1.1. This is the set of 200 data points which has been used in the previous experiments to determine the targets of the gridpoints. For each trial, the data are divided at random into a training

set and a validation set, with 100 patterns in each. The same 2*1 topology is also used in each trial.

The training set is used to determine the chance a member of the population has of becoming a parent. Those members of the population with lower error on the training set are given a better chance of reproducing. The validation error is calculated for the member of the population in each generation with the best performance on the training set. This gives a single validation error value for each generation. A minimum of validation error is detected when the validation error rises in one generation, after having fallen in a previous generation.

At each minimum of validation error, the overall error on all 200 datapoints is recorded, as well as the generation number at which the minimum occurred. In contrast to the standard technique, the minimum of validation error with the best overall performance is taken to be the solution of the trial.

One hundred trials were performed, recording the overall error at each minimum of validation error during a period of 100 generations for each trial. The GA otherwise had the same parameters as used in the I/IO paradigm in all the other experiments, i.e. the population size was 100, the mutation probability was 1.0 and the crossover probability was 1.0.

To compare the performance of the validation technique with that of the I/IO paradigm, the IO behaviour of several candidate weight states with a known error on the set of 200 patterns was superposed onto the grid used in the preceding experiments in sections 6ii.1.1-3. (See figure 6ii.2.) It was found that an overall misclassification error of 45 or less was needed in order to achieve the equivalent of the desired separation of the gridpoints. This, then, is the threshold at which a solution will be considered to be a good solution. Solutions with a higher overall misclassification error will be considered to be bad solutions.

Of the 100 trials, 30 had minima that were all bad solutions, with the remaining 70 trials having at least one good solution. The information recorded at each minimum included the generation number at which the minimum occurred (see above). Using this information, it was possible to determine how many of the 100 trials had found a good solution in one of

their minima by 25 generations. The result was that 54 of the 100 trials had at least one good solution by the 25th generation.

Figure 6ii.9 shows some profiles of the overall performance at each minimum of validation for three of the 100 trials. Profile A shows a trial in which none of the minima were good solutions. Profile B shows a trial in which all of the minima were good solutions. Profile C shows a trial in which some of the minima were good, and some of the minima were bad.

These results indicate that, for a given fixed period of training, there is no certainty offered by validation that a good solution has been found. All but one of the 100 trials had at least one minimum of validation error, and of the 1464 minima of validation that occurred over all the trials, just over half (58%) were good solutions.

Therefore, there is no great degree of empirical certainty when using the validation technique that further training will not yield a better solution. The improvement may not be a matter of fine tuning either. This is supported by the observations made at 25 generations, which show that 16 trials found good solutions after 25 generations, having found only bad solutions beforehand. This is illustrated in the case of profile C, which does not find good solutions until after 25 generations.

In conclusion, these trials show clearly that the validation technique is not especially useful for deciding when to stop training, and that poor solutions are a regular occurrence. Contrasting this with the I/O paradigm, which can accurately determine the point at which learning is completed, the optimum solution is returned in the vast majority of the cases in which it terminates. (See table 6ii.1.)

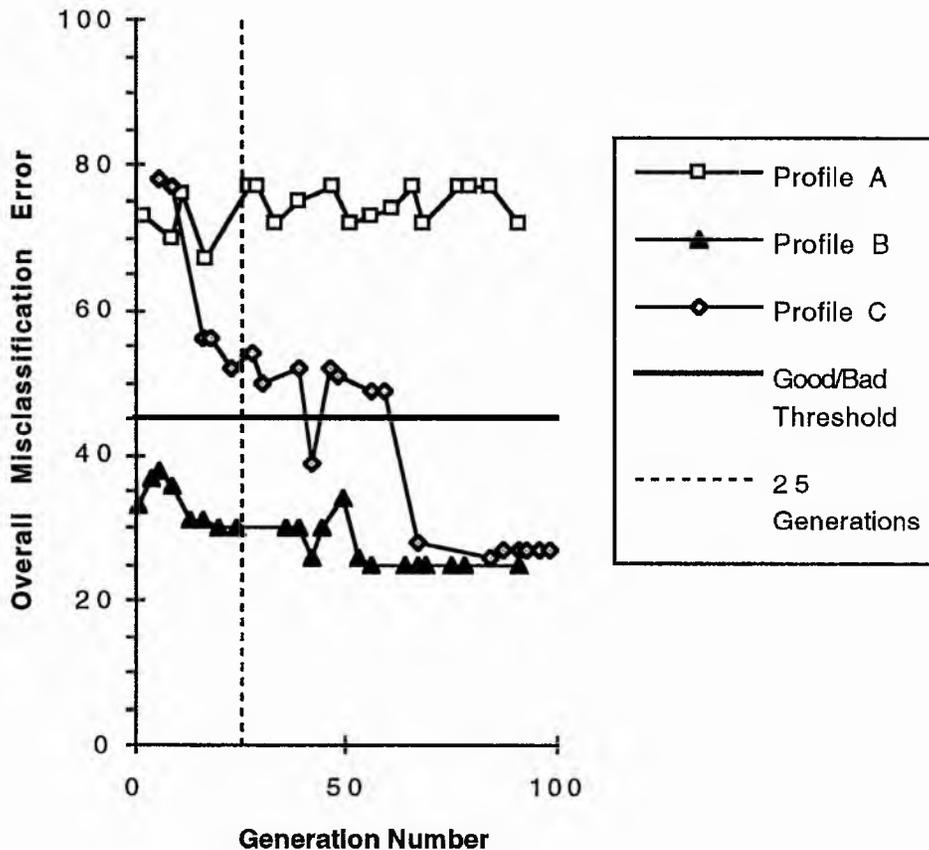


Figure 6ii.9 — Three training profiles, A, B and C. For each profile, the overall misclassification error is indicated by the markers, which are placed at each minimum of validation error during the trial from which the profile is taken. The threshold between a good solution and a bad solution (at a misclassification error of 45) is indicated by the solid line. The 25 generation point is indicated by the dashed line.

6ii.1.5 Validation Set Size

Denker et al observe “rather poor” overall performance even when most of the available data is used for training.¹ In this experiment, the effect of

¹Denker et al, 1987, p. 898

the size of the validation set on the solution found is explored in the context of the same problem discussed in section 6ii.1.4.

The same 200 patterns are used, with a 2*1 topology, and a GA as the training algorithm. The method for determining minima of validation error is the same as in section 6ii.1.4, and the same training parameters were used.

In this experiment, five sizes of validation set are considered. Each size is given 20 trials, and the overall performance on all 200 patterns is measured for each minimum of validation error. The average best minimum of validation error over the trials gives the final score for the given size of validation set. In all cases, any patterns not used for validation are used for training.

The following splits of data into validation set and training set respectively were considered: 1, 199; 5, 195; 10, 190; 50, 150; and 100, 100. The results are indicated in the histogram in figure 6ii.10. This shows that the validation set size has little effect on the ability to find a good solution. The variability is within 5% of the estimated maximum misclassification error of 175.

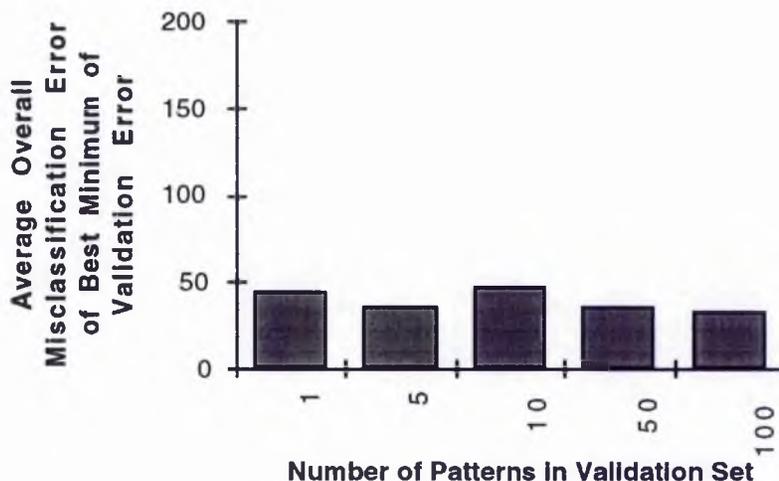


Figure 6ii.10 — *The effect of the size of validation set on the overall solution found.*

6ii.1.6 Scalability of the Technique

In this section the I/I0 paradigm is tested on a new problem. The purpose is to show that the I/I0 paradigm works using a more difficult problem which requires hidden units. The underlying distribution from which the patterns are taken is shown in figure 6ii.11(a). A $2 \times 3 \times 1$ topology will be used to attempt the problem, with the desired final decision region being one which ignores the two circles, and concentrates on the triangle, as shown in figure 6ii.11(b).

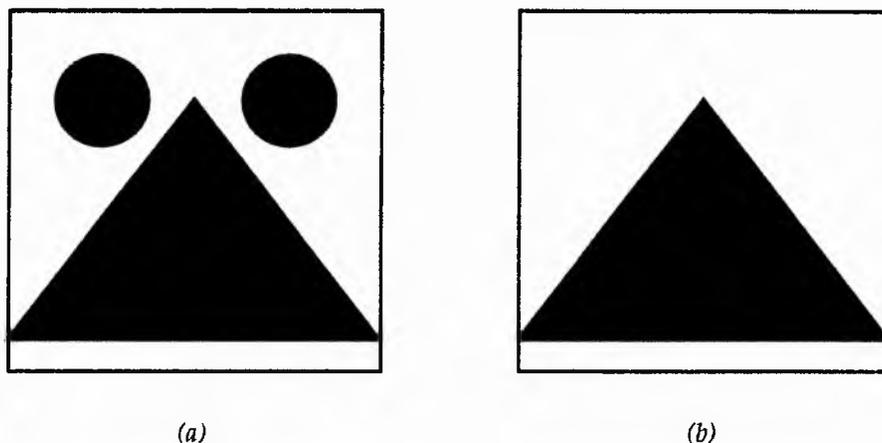


Figure 6ii.11 — (a) Underlying distribution of experiment. (b) Desired final solution to be found by $2 \times 3 \times 1$ topology.

A 7×7 grid is used, and its targets, based on a random sample of 500 patterns from the underlying distribution, are given in figure 6ii.12.

A single trial was done which used imperfect training with a GA with 2 000 generations per familiarisation, after preliminary trials with 200 and 300 generations per familiarisation. The preliminary trials had excessive numbers of runs whereby I and I0 had not locked after all gridpoints were familiarised. The results from experiment 6ii.1.2 indicate that more generations might yield better results. Figure 6ii.13 shows the final grids found for I and I0 with 2 000 generations per familiarisation.

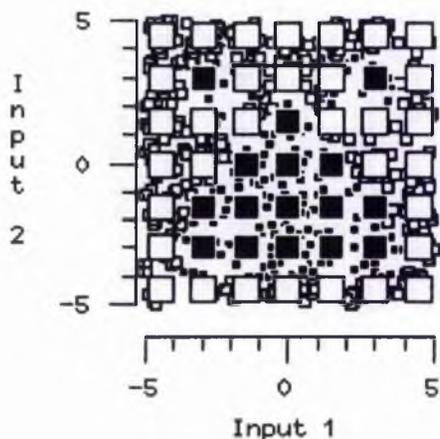
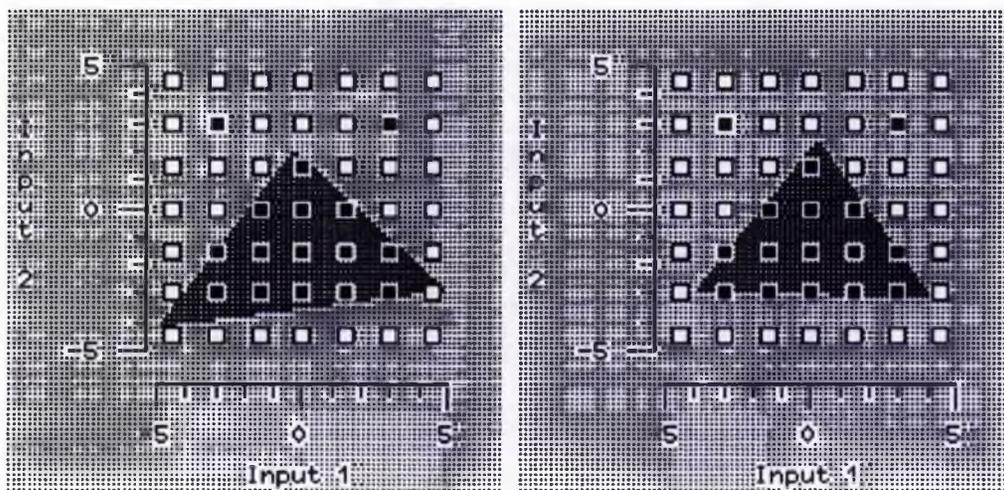


Figure 6ii.12 — Automatically produced diagram of the underlying target grid, based on a sample of 500 patterns from the underlying distribution shown in figure 6ii.11(a). Large squares are gridpoints, and small squares are patterns.



I

IO

Figure 6ii.13 — The final solutions found for I and IO in a single trial with 2000 generations per familiarisation. The gridpoints are shown slightly smaller than in figure 6ii.12, so they do not excessively obscure the decision regions of I and IO.

With the I/IO paradigm, there is certainty that, at termination of a single trial (though it may be a long one), either the best fit grid has been found, or improved training is known to be necessary. Other methods without

this certainty, such as validation, might use several training trials in any case, choosing the best solution found overall. Despite this, they still cannot be certain of having the globally optimum solution.

The single trial took over 24 hours to run. This is a long time to wait. The problem may be larger than other problems in this chapter, but it is not large enough to justify a learning time of over a day. However, the time taken is more due to the slowness of the GA rather than the fact that 48 familiarisations were used (with the trial having a single hypothetical gridpoint at locking).

6ii.1.7 A Symbolic Problem

Here a simple symbolic concept is attempted, using a sample of real-world data. The purpose is to show how a symbolic problem may be given an interpretation on a grid and that the I/I0 paradigm is able to cope with inconsistent data when learning the rule. Inconsistent data are data which do not agree with the rule to be learned. For Mitchell's symbolic technique this would mean, for example, presenting the instance {[Red, Fungus, Poisonous]} as positive when the concept to be learned is {[*, *, Edible]}, where '*' represents the wild-card. (See chapter 2.) For the neural technique, this means the stripe in figure 6ii.1(a) and the two circles in figure 6ii.11(a) are inconsistent data.

When learning to recognise trees, one method involves looking at the shape of the tree. One of the most basic classifications of trees is whether or not they are evergreen or deciduous. This is the task to be undertaken here. The shape of the tree will be represented rather crudely by its breadth and its height. This is on the basis that evergreens, especially the firs, tend to be tall and narrow, whereas deciduous trees tend to be shorter, and broader. Some exceptions to the rule are poplars, which are tall, thin deciduous trees, and the cedar of Lebanon, which can be broad and short, but is evergreen. Figure 6ii.14 indicates the general rule, and shows the exceptions.

A sample of rough measurements of the height and breadth of some adult trees in metres was taken during a walk in a forest, and is shown in figure 6ii.15(a). The idea is to have symbols which represent short, medium and

tall for the height dimension, and thin, medium and broad for the breadth dimension. This means using a 3×3 grid. However, the data are not widely spread, since there are not many trees which are very tall and very narrow, nor very short and very broad. The grid used is therefore not a regular horizontal/vertical grid, but a slanted grid, that fits the distribution as shown in figure 6ii.15(b), which also indicates the underlying target grid. Note that each gridpoint still represents the same area of input space.

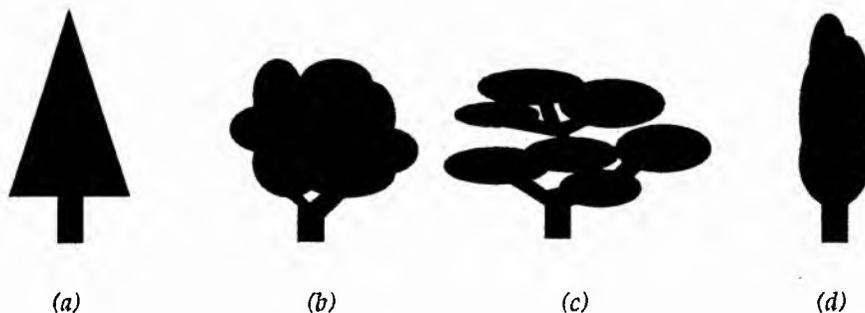


Figure 6ii.14 — (a) A typical evergreen tree, which is tall and thin. (b) A typical deciduous tree, which is shorter and broader. (c) A counter-example evergreen tree, the cedar of Lebanon, which is short and broad. (d) A counter-example deciduous tree, the poplar, which is tall and thin.

The cedar of Lebanon causes a spurious black gridpoint target (bottom right), which is a counter-example to the rule to be learned. The symbolic technique would be unable to learn the concept under these circumstances, since the underlying data are inconsistent with the overall concept to be learned.

Fifty trials were done using a GA with 100 generations per familiarisation. Forty of the trials found the best fit grid by the time all the gridpoints had been familiarised. The remaining ten trials did not terminate after all the gridpoints had been familiarised.

Figure 6ii.16 shows the entire search space of output 3×3 grids that can be generated by a 2×1 topology, and the chains in the partial ordering. The path of I and I0 through the ordering for one of the trials which terminated with 2 hypothetical gridpoints is also indicated.

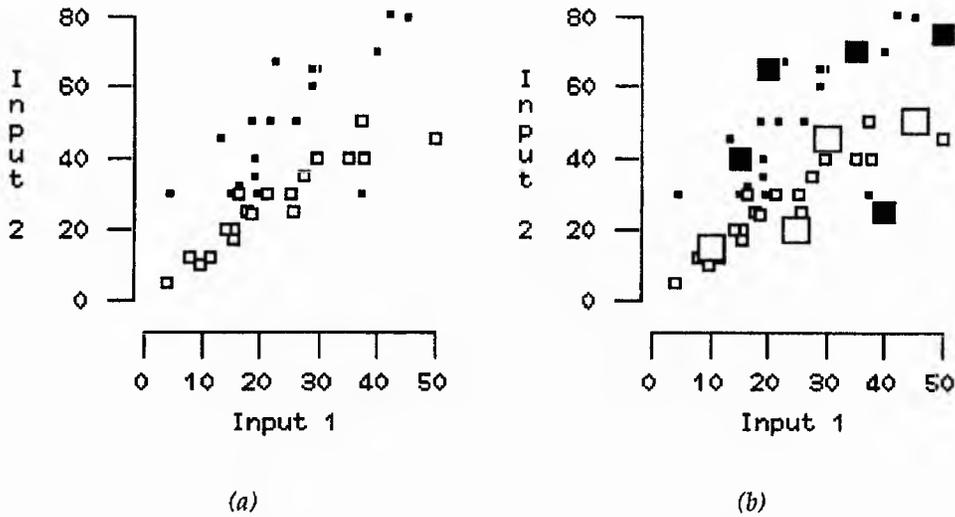


Figure 6ii.15 — (a) A sample of measurements of the heights and breadths of evergreen and deciduous trees. Input 1 is the breadth and input 2 is the height. Both are measured in metres. The evergreens are indicated by black points, the deciduous trees by white points. (b) The slanted target grid, designed to fit the distribution of trees.

Figure 6ii.17 shows the absolute fit values of I and I0 at the seventh familiarisation in the trial in figure 6ii.16, at which I and I0 lock using imperfect training. There is only one choice of highest absolute fit for I, which is the goal grid. For I0, there are three choices of absolute fit, but only one choice of inherent fit, since one of the three choices of absolute fit is closer to the root grid for I0 than any of the others. In the imperfect training trial in figure 6ii.16, I0 does not choose this grid, choosing the goal grid instead, and I and I0 lock.

The search spaces and absolute fits indicated in figure 6ii.17 show why false locking is such a rare event, even though it is, in theory, possible. For example, suppose a somewhat arbitrary selection of grids is made, such as those with an absolute fit greater than zero (i.e. better than the root grid). I has a choice of 15 grids with an absolute fit greater than zero, and I0 has a choice of 12 such grids. The estimated odds of I and I0 choosing the same sub-optimum grid with a value greater than zero are therefore 1 in 180.

This estimate does not take into consideration the following two points, which make it difficult to give an accurate estimate. Firstly, target reversal

of the sub-optimum grids I chooses will encourage I0 not to choose a grid which will allow the satisfaction of the locking conditions. Secondly, I and I0 do not necessarily have to be the same grid for locking to occur.

The probability of false locking would nevertheless seem in crude terms to be low, which would indicate why it is so rare for I and I0 to falsely lock. The experiments in section 6ii.1.2 also show that as the quality of the training algorithm improves, the chance of false locking goes down further. This leads to the general point that even if the training algorithm is a sub-standard imperfect training algorithm, the chances of false locking are somewhat remote.

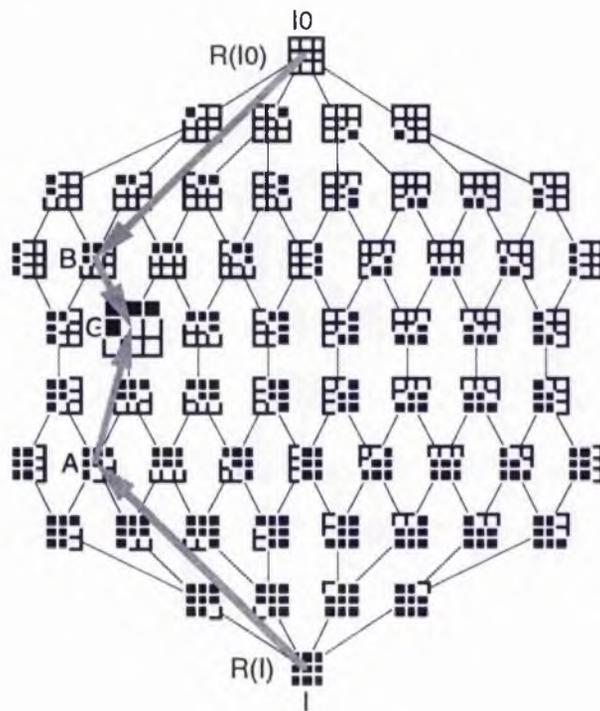
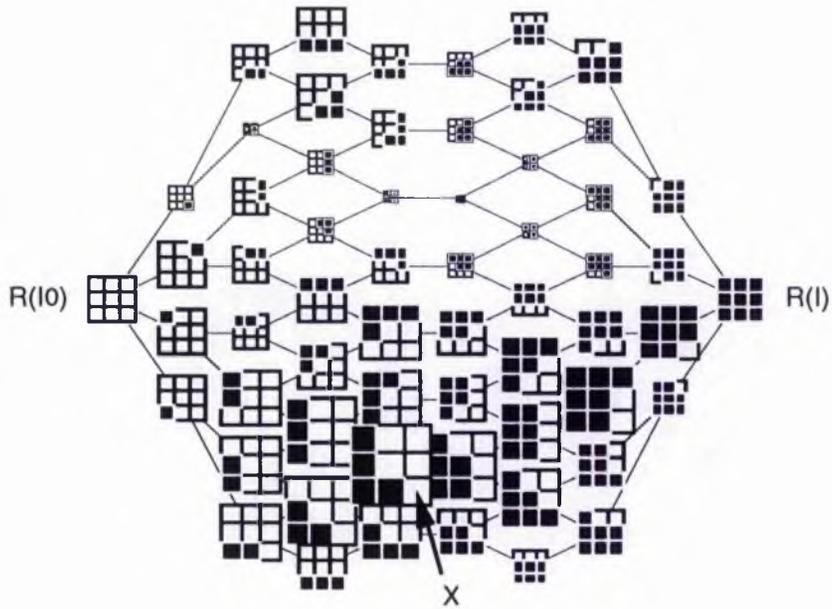
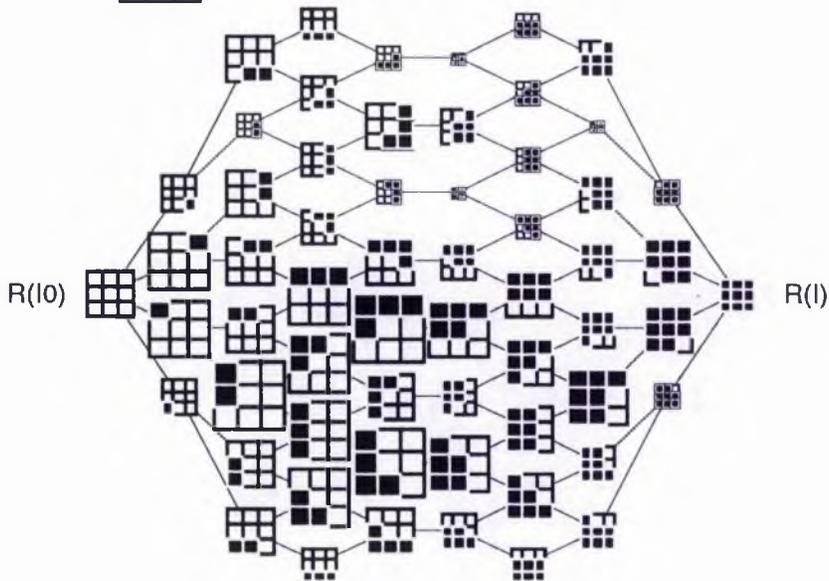
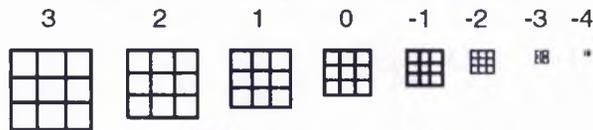


Figure 6ii.16 — The partial ordering of all the 3×3 grids. The chains in the ordering are indicated by the thin lines. The goal grid (G) is emphasised by being larger than all the other grids. In the trial indicated in this diagram, I starts at the all black grid, R(I), and I0 starts at the all white grid, R(I0). On the fourth familiarisation, I moves from R(I) to grid A, which lies on a chain between R(I) and G in the ordering. Also, I0 moves from R(I0) to grid B. On the sixth familiarisation, I0 moves from B to the goal grid, G. On the seventh familiarisation, I moves from A to G, and locking happens to occur with two hypothetical gridpoints remaining.



(a) Absolute fit values for I

Guide to absolute fit values for I and I0.



(b) Absolute fit values for I0

Figure 6ii.17 — Absolute fit values for (a) I and (b) I0 on the seventh familiarisation of the trial in figure 6ii.16 are indicated by the size of the grid. The I representative chosen to determine the hypothetical targets for I0 is grid X. The absolute fit values in (b), for I0, are calculated on this basis.

6ii.2 Assessment of the I/IO Paradigm

The I/IO paradigm has managed to combine the benefits of the H/H0 paradigm in terms of a guaranteed locking condition for perfect training, and the benefits of the B/W paradigm in terms of the use of a partial ordering. The partial ordering enables the boundary representatives to be kept apart in the ordering as much as possible to cope with sub-optimum training, whilst still enabling convergence on the correct grid.

No false locking trials occurred with perfect training, though section 6ii.1.3 showed that the I/IO paradigm tends to lock a little later than the H/H0 paradigm. Section 6ii.1.2 showed that using imperfect training rather than perfect training further exaggerates this effect, and that when the training algorithm does not meet the specifications, there is the possibility of false locking.

The experiment in section 6ii.1.1 demonstrates the I/IO paradigm at work with imperfect training, and shows how a sub-optimum grid at one stage during training can be recovered from at a later stage. Therefore there is no longer the requirement that with every familiarisation, the optimum grid must be found, as is the case with the H/H0 paradigm.

The ideal is to be able to find a grid which is at or behind the optimum grid in the ordering. This is a weaker requirement for training because there is no longer the need to find the optimum grid at every stage of learning. The demands on the training algorithm therefore have been relaxed, to a certain extent. However, they have not been relaxed enough such that the actual training algorithms used are able to meet the conditions necessary to ensure the prevention of false locking.

Nevertheless, there were no false locking trials out of the fifty trials conducted in section 6ii.1.7 (though in section 6ii.1.2 there were 22 false locking trials when the GA had 1 generation per familiarisation, and 3 false locking trials when the GA had 5 generations per familiarisation). The reason that false locking is typically such a rare event is that there is a greater choice of sub-optimum grids than optimum grids for I and IO, and it is unlikely that any two such grids chosen will lock. This is indicated in

section 6ii.1.7. Despite this, false locking is nevertheless a possibility, and thus the locking guarantee must be watered down a little when a GA is used as the training algorithm.

Validation's weakness as a unidirectional process is highlighted in section 6ii.1.4. It was shown here that there is a fundamental problem with knowing when to terminate, and that poor solutions occur regularly. Section 6ii.1.5 showed that the size of validation set makes little difference to the overall performance.

Section 6ii.1.6 illustrated the extra time required when the problem is made more complex than a linear separation. A much larger number of generations is required to get a GA to be close enough to the optimum such that it terminates with hypothetical gridpoints. However, this is more of a training problem than a problem that specifically relates to the I/O paradigm. Note also that a single trial using a GA provides either a guarantee of the optimum solution or the indication that better training is needed. Other techniques may use several trials, in the hope that they have sampled weight space enough to find the best solution.

A symbolic problem was the focus of section 6ii.1.7. The display of the entire search space enabled the illustration of how single boundary representatives can move through the partial ordering, but are not constrained to a particular chain. The need for multiple boundary representatives — a costly problem with the symbolic technique — is thereby shown empirically as well as theoretically to be unnecessary in the neural implementation. Section 6ii.1.7 also showed that the grids need not be constrained to strictly regular horizontal and vertical lines of gridpoints.

The strengths and weaknesses of the I/O paradigm may now be summarised.

- The I/O paradigm offers the ability to recognise when a globally optimum solution has and has not been found for an inexact fit even if training is imperfect.
- The extended training sequence during the learning process means that concepts may take longer to learn than with

conventional training methods. The differences in training times between the multiple serial training stages of the I/I0 paradigm and the multiple initial weight states of techniques such as cross-validation needs investigation. If the differences prove to be excessive, the I/I0 paradigm should be evolved further (see chapter 7).

- There are not yet any training algorithms which fully meet the specifications an ideal paradigm using a partial ordering would demand in order to guarantee that no false locking occurs — in particular that sub-optimum grids selected for the two directions must be at or behind the optimum grid in the ordering. However, despite this, false locking is nevertheless unlikely until learning is effectively complete.
- The I/I0 paradigm may be used to indicate the effectiveness of a training algorithm, in terms of whether or not learning should continue and in general how well the training algorithm is able to find a best fit solution.
- The I/I0 paradigm is able to cope with noisy and inconsistent data, and may also be used to tackle symbolic problems.
- Unlike Mitchell's symbolic technique, the I/I0 paradigm requires only a single boundary representative for each direction.
- Unlike Schwarz et al's technique² there is no need to use exhaustive search in order to estimate the generalisation ability.
- Unlike VC theory, there is no need to saturate input space with patterns in order to guarantee generalisation. Generalisation follows from having found the best fit given the user's assumptions.
- Unlike validation and other unidirectional processes, there is the certainty provided by bidirectional convergence as to when to stop learning.

²Schwarz et al, 1990

6ii.3 Conclusion to Part (ii)

Overall, the grid technique in the form of the I/I0 paradigm has overcome the difficulties of weight symmetries that were present in the angle technique in chapter 5. The problems with weight space symmetries are effectively dealt with by sampling IO space at regular intervals, as discussed in section 6i.1. More significantly, the grid technique has set up a theoretical framework for being able to recognise when the best fit solution has been found using conventional training algorithms.

Two main techniques have been presented. Firstly, the H/H0 paradigm shows how grids may be used during learning to enable a guarantee of correct locking. This paradigm does not use a number of core Mitchellian principles. Nevertheless, the H/H0 technique also reveals a problem to be overcome in combining the symbolic technique with neural networks: that of having to assume perfect training of the neural network.

Mitchell does not have to worry about whether or not the optimum concepts are found at each stage for the boundary representatives. The correct concepts for the boundary representatives are deduced using pattern matching, on the basis of the instances and the bias of the concept and instance representation language. There is no question of not finding these representatives, as there is in this neural implementation. This is therefore an extra problem for a neural implementation to consider.

The I/I0 paradigm is an attempt to overcome such difficulties, by an increased closeness to Mitchell. The use of a partial ordering ensures that the fact that training algorithms cannot necessarily guarantee optimal boundary representatives is accommodated. The algorithms are instead encouraged to keep the boundaries apart from one another as much as possible through the use of appropriate potential functions.

The I/I0 technique puts the user in control of the biasing factors, in the grid and the topology, which affect the kind of generalisation that will be found, just as Mitchell's symbolic technique does through the user-specified concept and instance languages. The ability to recognise locking, also inherited from Mitchell's technique, is an advantage over standard

generalisation techniques such as validation. The terminating condition for the validation technique does not offer the same certainty, as discussed in chapter 3, because it is not known when to terminate.

Both H/H0 and I/I0 are also able to deal with inexact fit, which means there is a tolerance of inconsistent and noisy data. Such tolerance is not present in the symbolic technique without extensive and expensive alterations. This also represents a further improvement on the angle technique, which relied on consistent data as well. Section 6ii.1.7 showed the I/I0 paradigm learning a simple concept which would not be learnable by Mitchell's symbolic technique.

There is also the possibility to represent disjunctive symbolic concepts, if need be, through appropriate choice of the grid and topology. The symbolic concept of $\{[* , lorry] \vee [blue , van]\}$ is represented as shown in figure 6ii.18.³ Thus, there are gains for Mitchell's technique through the grid technique being useful for symbolic problems.

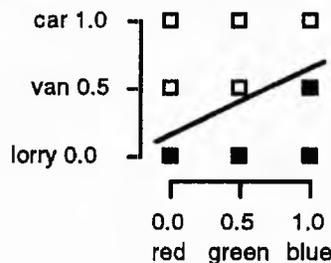


Figure 6ii.18 — Through the representation of symbols by the ordinates of the gridpoints on each axis of input space, concepts such as $\{[* , lorry] \vee [blue , van]\}$ can be learned. The separation is represented by the line through the grid.

Hence there has been considerable success in merging a symbolic technique with neural networks to mutual benefit. The neural networks are able to cope with noise, and the partial ordering adapted from the symbolic technique enables a directed search which results in a guarantee of generalisation, given training algorithms which meet the required specifications. The guarantee must be watered down a little in the face of

³Weir & Polhill, 1995

training algorithms that do not meet these specifications. However, false locking is unlikely until learning is effectively complete due to the large choice of sub-optimum grids for I and I_0 that would not satisfy the locking conditions.

The grid technique with inherent fit gives a theoretical framework for providing guaranteed learning of the concepts. It provides a guaranteed terminating condition, provided the training algorithm meets the required specifications.

Appendices

6.A Proof of Locking for the H/H0 Paradigm

The goal of this proof is to show that, at locking, H0 has the best fit whatever the true targets of the hypothetical gridpoints. This provides the basis for the idea that target reversal of the hypothetical gridpoints makes no difference to the classification boundaries.

Let Z be the set of hypothetical target gridpoints at locking. Let A be the set of real target gridpoints at locking. It is assumed that H0 has the global minimum error fit to its target grid, $H0_T = A + Z$, at locking. Thus, for any arbitrarily chosen neural network, N :

$$E_N(A + Z) \geq E_{H0}(A + Z) \quad [6.A.1]$$

where $E_F(G)$ is the misclassification error of a neural network F on the set of gridpoints G .

The set Z will be split into two arbitrary subsets, X and Y . Let the set X^* be the set of input gridpoints from X but with opposite targets from X for those points. The set $Z^* = X^* + Y$ is then the same set of hypothetical gridpoints as Z but with some of the targets of those gridpoints reversed. The goal of the proof is to show that, given [6.A.1], for any arbitrarily chosen neural network, N :

$$E_N(A + Z^*) \geq E_{H0}(A + Z^*) \quad [6.A.2]$$

If [6.A.2] holds, then since the split of Z is arbitrary, H0 gives the best fit regardless of the true targets of Z . The proof has three main steps. The first step is to relate $E_{H0}(Z)$ to $E_{H0}(Z^*)$. The second step is to relate $E_N(Z)$ to $E_N(Z^*)$. The third step is to substitute for $E_{H0}(Z)$ and $E_N(Z)$ in [6.A.1] using the expressions found in the first two steps.

H0 misclassifies all members of Z at locking. This is necessary in order to satisfy the terminating conditions. Writing the cardinality of Z as $\#Z$, therefore, $E_{H0}(Z) = \#Z$. Since Z is comprised of X and Y , $E_{H0}(Z) = \#X + \#Y$. Since X^* has reversed targets from X , all members of X^*

are correctly classified. Therefore $E_{H0}(Z^*) = \#Y$. This gives the following expression relating $E_{H0}(Z)$ to $E_{H0}(Z^*)$:

$$E_{H0}(Z) = \#X + E_{H0}(Z^*) \quad [6.A.3]$$

For the second stage, X and Y will be considered separately. Since X^* has reversed targets from X , the error of N on X is related to the error of N on X^* as follows:

$$E_N(X) = \#X - E_N(X^*) \quad [6.A.4]$$

Since when $G_1 \cap G_2 = \emptyset$, $E_F(G_1 + G_2) = E_F(G_1) + E_F(G_2)$ (as is the case here), $E_N(Z^*)$ may be re-written as per [6.A.5]:

$$E_N(Z^*) = E_N(X^*) + E_N(Y) \quad [6.A.5]$$

Hence,

$$E_N(Y) = E_N(Z^*) - E_N(X^*) \quad [6.A.6]$$

Since $E_N(Z) = E_N(X) + E_N(Y)$, from [6.A.4] and [6.A.6]:

$$E_N(Z) = \#X - E_N(X^*) + E_N(Z^*) - E_N(X^*) \quad [6.A.7]$$

Simplifying:

$$E_N(Z) = \#X + E_N(Z^*) - 2E_N(X^*) \quad [6.A.8]$$

Expanding [6.A.1] gives:

$$E_N(A) + E_N(Z) \geq E_{H0}(A) + E_{H0}(Z) \quad [6.A.9]$$

Substituting for $E_{H0}(Z)$ and $E_N(Z)$ in [6.A.9] using [6.A.3] and [6.A.8] respectively gives, for the third step:

$$E_N(A) + E_N(Z^*) + \#X - 2E_N(X^*) \geq E_{H0}(A) + E_{H0}(Z^*) + \#X \quad [6.A.10]$$

Simplifying [6.A.10] gives:

$$E_N(A + Z^*) - 2E_N(X^*) \geq E_{H0}(A + Z^*) \quad [6.A.11]$$

Since $E_N(X^*) \geq 0$:

$$E_N(A + Z^*) \geq E_{H0}(A + Z^*) \quad [6.A.12]$$

Thus, [6.A.2] is proved, and since N and the split of Z into X and Y were arbitrary, $H0$ has global minimum error fit whatever the true targets of Z , the hypothetical gridpoints.

6.B Genetic Encoding of Neural Network and Selection Mechanism

6.B.1 Genetic Encoding of Neural Network Weight State

The coding mechanism uses a binary string for each member of the population. This binary string is used to specify the entire weight state of the neural network. The topology is fixed, and hence there is no need for the coding mechanism to include this.

Each member of the population is assigned the same number of bytes, y , for each weight value of a single connection in the network. A network with a $2 \times 2 \times 1$ topology has 9 connections, for example. The length of the binary string for each member of the population would then be $9y$ bytes.

Of the y bytes, a number of bits, $i + 1$, is specified to be the integer part of the weight value, with the remaining bits barring the last bit of y being used to represent the rational part of the weight value. The last bit of y specifies the sign. If the last bit is 1, then the weight is negative, otherwise it is positive. For the GA used in this chapter, y was set at 2, and i was set at 5. The maximum and minimum weight values are then +63.998046875 and -63.998046875 respectively.

The weights are Gray coded,⁴ to avoid Hamming cliffs. A Hamming cliff occurs between, for example, 01111 (15) and 10000 (16). Five simultaneous mutations are required to go from one consecutive value to the next. Gray coding ensures that there is always a single mutation path between any two consecutive numbers.

⁴Hamming, 1986, pp. 97-99

If G is a Gray coded binary number, and B is a binary number, then to encode from B to G do the following: for each bit, j , of B , where j reads from left to right, if $j = 1$ then $G_j = B_j$, otherwise if $B_j = B_{j-1}$, $G_j = 0$ and if $B_j \neq B_{j-1}$, $G_j = 1$. For example, 01111 becomes 01000 under the Gray code, and 10000 becomes 11000. This means there is a difference of only 1 bit between the Gray coded binary numbers for 15 and 16. To decode from G to B : for each bit j of G , if $j = 1$ then $B_j = G_j$, otherwise if $G_j = B_{j-1}$, $B_j = 0$, and if $G_j \neq B_{j-1}$, $B_j = 1$.

Having decoded the binary string, the leftmost bit represents 2^i , with the next bit representing 2^{i-1} . In general, the n th bit from the leftmost bit represents the $(i-n)$ th power of 2.

For example, consider a neural network with a single weight, with $y = 1$ and $i = 3$. A member of the population consists of the following binary string:

1 0 1 1 1 0 1 0

The binary string is decoded to give the following:

1 1 0 1 0 0 1 1

Each bit represents the following number, or sign in the case of the last bit:

2^3 2^2 2^1 2^0 2^{-1} 2^{-2} 2^{-3} $\pm ve$

And hence the decoded binary string represents the following sum:

$2^3 + 2^2 + \quad 2^0 + \quad \quad 2^{-3} \quad -ve$

The weight value of the single weight in the neural network represented by the above member of the population is therefore -13.125 .

6.B.2 Selection Mechanism

The selection mechanism works as follows.⁵ The members of the population are sorted in order of preference, with the best members last

⁵Harvey, 1994

and the worst first. A number of tickets is then assigned to each member of the population, which will be used to buy their way into propagating into the next generation. The number of tickets for each member of the population is equal to its position in the sorted preference list. The worst member gets 1 ticket, and the best member gets a number of tickets equal to the size of the population, s .

The members of the population of the next generation are then chosen by picking a random member of the population, p , and choosing a random number, r , between 1 and s . If p has a number of tickets greater than r , then it is copied into the next generation, and its number of tickets is reduced by 1. This process is repeated until s offspring have been selected.

Sometimes it is desirable to bias the selection process in favour of the better members. This is achieved in this implementation by raising the number of tickets assigned above to the power of a constant, k , giving a new number of tickets. The selection process continues in the same way, except that the random number r is chosen between 1 and s^k . This gives the better members of the population considerably more chance of being chosen to be in the next generation. After several trials with various values, the value for k used in the experiments above was 10. This is high relative to the norm, but the poorer members of the population still have a chance (albeit an extremely small chance) of passing their genes on to the next generation.

6.C Crossover and Mutation Probabilities

The crossover probability represents the probability that a crossover operation will be performed on a member of the population selected for the gene pool. Two alternatives were considered for crossover. One possibility was to perform a crossover on the bit string for each weight belonging to a member of the population. The other was to perform a single crossover on the entire bit string of the member of the population. The latter alternative was chosen.

Each bit of the bit string of a member of the population selected for the gene pool was given a chance of mutating. Yet the mutation probability specified should indicate the average probability of a single mutation

occurring in the member of the population. This means the probability of mutation of each bit in the bit string is the specified mutation probability divided by the number of bits in the gene.

7 Conclusions and Further Work

7.1 Achievements

The central aim of the thesis is to abstract the tenets of Mitchell's technique, and evolve and implement it in a neural environment. This was with a view to guaranteeing generalisation in neural networks without resorting to comprehensible rule extraction, by finding the no-alternative classification situation. The work discussed in chapters 5 and 6 showed four attempts to achieve this, culminating in the I/I0 paradigm of the grid technique in chapter 6. The I/I0 paradigm showed the following points, theoretically and empirically:

- A close implementation of Mitchell's technique, which uses a grid and a partial ordering of the grids, can guarantee generalisation given a training algorithm that meets the required standard. This standard is that when the optimum grid is not found by an imperfect training algorithm, the grid found must be behind the optimum in the ordering. The generalisation guarantee rests on the user's assumptions given that the implementation has found the best fit to the underlying target grid.

These principles were demonstrated in the example run in section 6ii.1.1, which showed how an imperfect training algorithm may find the best fit output grid to the underlying target grid. The experiments in sections 6ii.1.4 and 6ii.1.5 showed that validation is regularly unable to find the best fit grid, and that there is no certainty of termination as there is with the I/I0 paradigm.

- If there is imperfect training which does not meet the above standards, how close the training algorithm is to the optimum may be indicated by the number of trials which do not lock after all the gridpoints are familiarised. This was shown in the experiment in section 6ii.1.2.
- Unlike Mitchell's symbolic technique, the neural implementation does not require fully explicitly represented sets of boundary

representatives, but instead may rely on a single explicit representative of each set at each stage. In addition, there is the possibility to tolerate noise in the data. This was shown in the experiment in section 6ii.1.7.

This has shown that there have been gains for neural networks through the borrowing of ideas from Mitchell's technique. The ability to recognise when the best fit has been found to a target grid is a useful contribution to the discipline.

This work also indicates that Mitchell's technique does indeed have a wider applicability than just symbolic AI. This view may well apply to other techniques in symbolic AI, and thus it may be the case that in general symbolic AI and neural networks may gain by developing ideas taken from one another.

An underlying theme, which first arose in chapter 2, was the concept of bias. This, in many ways, underpins the core of Mitchell's technique. Good generalisation is not feasible without *a priori* assumptions which enable the generaliser to provide a concept which fits the data.

Bias arose in a neural network context in chapter 3, where the work of Geman et al¹ shows that bias can be found in the number of parameters in the generaliser and hence, in neural networks, in the topology. Approaches such as the average generalisation and VC theories indicate that the only way to guarantee generalisation in an unbiased fashion is to saturate input space with patterns, such that no other alternative exists. Such approaches are not practical, since a learner may typically not have the luxury of exposure to a near-exhaustive set of data with which to learn the concepts. It is for this reason that Mitchell emphasises the essential role of biases in concept learning.²

The capabilities of various topologies was discussed in chapter 4, which also provided guidelines for determining the number of units to be used in the first and second hidden layers.

¹Geman et al, 1992

²Mitchell, 1980

Chapter 5 attempted to find the no-alternative situation for an exact fit to a simple topology capable of a linear separation. The weight states with the most extreme separation in terms of angle were found as patterns were familiarised. The technique could not be generalised to networks with hidden units, though, due to weight space symmetries making it difficult to measure the angle. Neither could it be generalised to an inexact fit.

In addition to the choice of topology, a bias may be found in the choice of grid in the grid technique. The grid chosen determines the accuracy to which the neural network may be expected to generalise. This was discussed in chapter 6, which resulted in the I/I0 paradigm. The I/I0 paradigm is capable of finding the best inexact fit for any feed-forward topology given a suitable training regime.

7.2 Further Work

There are several possibilities for further work which arise from the work done in this thesis. The following sections briefly describe some thoughts for ways in which the I/I0 paradigm could be improved.

7.2.1 Keeping I0 Strictly Behind in the Ordering

In the I/I0 paradigm in chapter 6, it is possible for I0 to go back in the ordering, for certain hypothetical target familiarisations. This leads to the possible theoretical concern that there may be false locking, even with perfect training. However, when I0 does go ahead in the ordering, there is a different classification of the hypothetical gridpoints. This means the locking conditions cannot be satisfied, and hence there is no possibility of false locking using the I/I0 paradigm and a training algorithm that meets the standards required.

However, keeping I0 strictly behind in the ordering would be useful, since more information could be gained from cases in which I and I0 do not lock. This is because the grids that are behind I0 in the ordering could be eliminated from consideration, as well as those behind I.

In essence, this would mean the development of other inherent fit functions for the I0 direction.

7.2.2 Using the Network Trained from the Previous Familiarisation

One of the problems with the technique is training from multiple random initial weight states as new output grids are searched for with each familiarisation. There is the positive point that the use of a grid will reduce training times for a single target grid in comparison to training with the underlying patterns. For example, the experiment in section 6ii.1.6 had 500 underlying patterns, but a grid of size 49. This represents more than a ten-fold reduction in the amount of data used for training. Another positive point is that the grid technique does not have to have multiple runs to stochastically sample for a best fit. Such sampling occurs in Oppen and Haussler's Bayesian approach,³ and might also be used in conjunction with validation. However, such reduction in the grid technique's training time may not always outweigh the cost of multiple training. There is also the point that more conventional techniques could use a grid as well.

Currently, the training mechanism does not use any information from the grid found at the previous familiarisation. It is often the case that the subsequent output grid may differ only slightly from the grid of the previous familiarisation. This is especially true for the I direction where there is only one change of target for each familiarisation. (An ideal IO direction would have this property too.) Hence the consecutive weight states could be found more easily by using the weight state resulting from the previous familiarisation.

One possibility is to restrict the search of the GA to a given hypersphere of weight space, which is centred around the output grid that was found at the last familiarisation. This would increase the chance of finding good grids which are close to the current output grid, and thus reduce the chance of I going ahead in the ordering due to sub-standard training. The radius of the hypersphere could be increased during training if nothing better than the current output grid could be found.

³Oppen & Haussler, 1991

7.2.3 Relaxing the Training Requirements Further

Currently, if the training algorithm cannot guarantee the globally optimum grid over the whole lattice, it is asked to be able to guarantee a grid which is at least not ahead of the optimum grid in the ordering. Since the optimum grid is not known, this requirement effectively means that the grid found must be the closest grid to the root grid with the inherent fit found. This is because there is currently no other way to encourage the grid found to be behind the optimum grid in the ordering when there is more than one output grid with the same potential function value, some of which are ahead of the optimum grid in the ordering. This means that the condition on the training algorithm is stricter than that required by the framework. Requiring output grids to be as close as possible to the root grid is more restrictive than requiring output grids to be at or behind the optimum grid in the ordering.

There may be methods for improving training by searching intervening weight states between a candidate output grid and its root grid to try and find output grids which are closer to the root grid but have the same inherent fit as the candidate. The closest such output grid to the root grid should then be returned from the training algorithm in place of the original candidate. However, this strategy would still not be able to guarantee that the final grid found was as close as possible to the root grid. It would merely improve the chances of finding the grid.

7.2.4 Proposal of New Instances

The ability to propose the familiarisation of gridpoints that are significant in advancing locking would lead to faster locking. This has the consequence of fewer familiarisations, and therefore less time spent training. The familiarisation order would be chosen on an informed basis, rather than at random, as it is in the current implementation.

Although not investigated in this thesis, the I/I0 paradigm could also be used to generalise to gridpoints whose targets are not determined by patterns *a priori*. These *uninstantiated* gridpoints would always be hypothetical gridpoints. If the technique was unable to lock with a given set of uninstantiated gridpoints, then this would indicate either that there

are too many uninstantiated gridpoints, or that the training algorithm is sub-standard.

Assuming that the training algorithm meets the requirements of the I/I0 paradigm, then if I and I0 could not lock, the familiarisation of some of the uninstantiated gridpoints might enable locking. It would be useful if the technique could indicate which of the uninstantiated gridpoints are the most likely to enable locking.

Whatever the truth underlying the indication is, the ability to propose which gridpoints should be familiarised next would be desirable. A possible rule is to familiarise those hypothetical gridpoints for which I and I0 give different classifications. This could equivalently be used to suggest those uninstantiated gridpoints which should be found real targets. The benefit of familiarising such gridpoints is that it would at least partially resolve the differences between I and I0.

7.3 Conclusion

A better understanding of generalisation in neural networks is essential if they are to rival existing non-neural techniques. This thesis has made an initial step based on bidirectional convergence into providing a practical technique which can give some guarantees about generalisation. The biasing factors, in the form of the grid and topology are put in the control of the user.

There are three properties that Mitchell outlined as being the key properties that learning techniques in symbolic AI lacked before the advent of his concept and version spaces technique:⁴

- (i) Knowledge of when the concept has been learned.
- (ii) Proposal of informative new instances.
- (iii) Detection of and recovery from inconsistent data.

⁴Mitchell, 1978, p. 8

Properties (i) and (iii) have been transferred into a neural technique. The no-alternative situation is recognisable in the technique in chapter 5, and in the H/H0 and I/I0 paradigms of chapter 6. The I/I0, B/W and H/H0 paradigms are also all able to cope with inconsistent data, as indicated in the experiment in section 6ii.1.7. The blurring effect of the grid also helps cope with inconsistent underlying data in that it reduces the effect of noise. Proposal of new instances has not been addressed in this thesis, though it has been argued above in section 7.2.4 that there is scope for fruitful research in this area.

Thus, two out of the three properties outlined above have been transferred, which shows the extent to which Mitchell's technique has been transferred into neural networks. Neural networks now have some ability to shout "Eureka!" (or perhaps "Neureka!") when learning concepts, without resorting to exhaustive search, excessively large training sets, or multiple training runs through version space. Press et al, as mentioned at the end of chapter 3, acknowledge the difficulty of reaching a globally optimum fit:

In some cases, we may be interested in global, rather than local questions. Not, "how good is this fit?", but rather, "how sure am I that there is not a *very much better* fit in some other corner of parameter space?" ... This kind of problem is generally quite difficult to solve.⁵

The ability to reach the no-alternative situation in neural networks means that there is knowledge of when to stop training, and that the optimum fit has been found, from which generalisation is guaranteed, given the biases of the user. This is not possible with standard neural training algorithms.

The generalisation framework outlined in chapter 6 indicates that stronger training is needed. Although it is not possible to prove a general rule from the single approach developed in this thesis, it nevertheless seems reasonable to say that one cannot expect good generalisation without good training algorithms.

⁵Press et al, 1988, pp. 517-518

Training is not an issue for Mitchell, since the concept representation is modified exhaustively through pattern matching to maintain the two sets of boundary representatives. Neural networks do not need exhaustive pattern matching in order to be able to generalise, however, and they are also able to tolerate inconsistent data. This benefit has a cost in that finding the best fit neural network is heuristic rather than algorithmic — though the heuristic has been shown to be a powerful one.

Learning of the sort described in this thesis integrates generalisation and training. Existing neural training algorithms aim to find the best fit to the training data, but are unable to recognise when it has occurred over all the data. Generalisation occurs on the basis of the assumptions of the user once training ceases. Neither training nor validation performance alone guarantee that the best fit grid has been found. Learning here is the process of finding the best fit by combining generalisation, through testing the hypotheses of the boundary representatives during learning, with training, which finds the weight states that make these hypotheses in the first place. Having found the best fit output grid to the underlying target grid, the assumptions and biases of the user may be reliably known to have been fully exploited.

A Software Manual

A large amount of software was written during the creation of this thesis. This appendix demonstrates the usage of the most important applications used for the work in this thesis.

A.1 General Points and File Formats

In general, all the applications display a message indicating their usage when the command to run them is entered to the C-shell. This usage will indicate the files that are expected, and any options the software has. All the applications are command-line driven from C-shell, rather than using menus and user-friendly interfaces. This is because it makes it easier to run the processes in the background, without having to worry about whether the process will ask for any input from the user at any stage. (To run a process in the background, add an ampersand after the command. If the process is likely to take a long time, then prefix the whole command with "nohup nice".)

Some of the applications use library archives which I have written, and will not compile unless the compiler knows where to find these libraries, and their header files. The libraries are contained in `/user/rsch/gary/lib`, and it is necessary to add a line to your `.cshrc` file in your home directory to tell the compiler where to look for them:

```
setenv LD_LIBRARY_PATH `printenv LD_LIBRARY_PATH`:/user/rsch/gary/lib
```

This adds the directory containing the extra libraries to your library path, and any applications which use them should now compile successfully.

In the event that `/user/rsch/gary` has been removed, the current directory containing these library files should replace `/user/rsch/gary/lib` in the above command. When changing the directory of the libraries, please note that the software expects the header files to be in the specified location, and it will be necessary to change any `#include` statements which include header files in `/user/rsch/gary/lib`.

There are three main file formats which are used by the software. Other file formats which are unique to the application concerned will be specified in the description of that application. The file formats specify the topology and learning constants, the weights, and the patterns.

A.1.1 Format of Topology Files

All topology files must have a ".net" suffix. They are text files, and the following illustrates a topology file for a 2*3*1 topology:

```
number_of_weights: 13
number_of_neurons: 6
number_of_layers: 3
number_of_input_units: 2
units_in_next_layer: 3
number_of_output_units: 1
max_init_weight: 1.0
min_init_weight: -1.0
learn_by_epoch: true
high_target: 0.9
low_target: 0.1
output_tolerance: 0.1
bias_unit_output: 1.0
learning_rate: 0.5
momentum: 0.0
sigmoid_coefficient: 1.0
full
```

If more layers are required, then the number of units in these layers should be specified on additional lines which start "units_in_next_layer: ", all of which must precede the line beginning "number_of_output_units: ".

The other constants should be self-explanatory. The maximum and minimum values for choosing the initial weights are specified on the "max_" and "min_init_weight" lines. The "high_" and "low_" target lines specify the sigmoid output which is to be treated as 1 and 0 respectively.

The word "full" on the final line is used to imply that a strictly layered topology file with full connectivity of the weights is to be specified. For other (feed-forward) topologies, the word "full" should be replaced by a list in square brackets [] of connections which are to be added to or taken away from full connectivity, or a list in curly brackets {} specifying the only connections to be made. (Bias weights are always connected. If no bias weights are required, the bias_unit_output constant should be set to 0.0.) A connection to be added should be written a+b, where a and b are

the unit indexes of the units to be connected (which start at 0 for the first input unit). A connection to be removed should be written a-b. Each connection is separated by a comma, with no white space. E.g. 2*1*1 XOR:¹ [0+3,1+3]. This could also be written {0+2,1+2,0+3,1+3,2+3}

A.1.2 Format of Weight Files

All weight files must have a ".wgt" suffix. They are text files, with each line specifying the information for a single connection. The order in which the weights are specified is important, and it is best to let the software create a weight-file for the topology you wish to use, and then edit the weight-file to set the weights manually. This can be done using the library of procedures, using the following set of commands:

```
#include "/user/rsch/gary/lib/oldnn/nn.h"

char * _Topology_file_name_;
char * _Weights_file_name_;

load_topology(_Topology_file_name_);
total_pnum = 1;
connect_net();
save_weights(_Weights_file_name_);
```

The complete program must be compiled with "-lnn" at the end of the compiling command to C-shell. The weights file can then be edited. Do not edit the first two numbers of each line. These specify the unit in the higher layer, followed by the unit in the lower layer of the connection. The next number is the weight value. The last two numbers should be ignored. Bias weights are at the end of the file, and the bias unit is given the highest index of any unit. Otherwise, the units are numbered incrementally beginning at 0 for the input unit. Here is an example of a weight file for a 2*3*1 topology:

```
2 0 54.897461 0.000000 0.000000
3 0 -19.791016 0.000000 0.000000
4 0 47.112305 0.000000 0.000000
2 1 -15.149414 0.000000 0.000000
3 1 -3.985352 0.000000 0.000000
4 1 21.898438 0.000000 0.000000
5 2 50.956055 0.000000 0.000000
5 3 49.203125 0.000000 0.000000
```

¹Rumelhart et al, 1986, p. 321

```
5 4 -26.395508 0.000000 0.000000
2 6 29.316406 0.000000 0.000000
3 6 53.229492 0.000000 0.000000
4 6 61.515625 0.000000 0.000000
5 6 -60.447266 0.000000 0.000000
```

The two input units are numbered 0 and 1; the three hidden units 2, 3 and 4; the output unit 5; and the bias unit 6. The layer of the bias unit is considered to be below all other layers. This is because the bias unit receives no input, and its output is always fed forward to other units.

A.1.3 Format of Pattern Files

All pattern files must have a ".pat" suffix. Pattern files are text files, with the following format: The first line contains the string "number_of_patterns:" followed by a number which specifies the number of patterns in the file. The subsequent lines specify the patterns, with one line dedicated to each pattern. The input is specified first. The input values for each unit must be specified, separated by a white space. Then a vertical bar "|" follows, and then the target values for the input are specified. Target values may be written "hi" and "lo", in which case, they will take their values from high_target and low_target in the topology file, respectively. The following is an example of a pattern file for the XOR problem:

```
number_of_patterns: 4
0.0 0.0 | lo
0.0 1.0 | hi
1.0 0.0 | hi
1.0 1.0 | lo
```

A.2 Grid Technique Simulators

In this section the applications which were used to obtain the results for the grid technique are discussed. There are two. The first uses exhaustive search in weight space to find all the possible grids, and then finds the optimum grid each familiarisation. It has a variety of paradigms. The second uses a GA, and is only for the I/I0 paradigm in chapter 6.

A.2.1 Perfect Training Version Using Exhaustive Search

The exhaustive search application is contained in the directory /user/rsch/gary/sim/grid/exhaustive. You will need "mexhstv7.c" in that directory to compile it. The "v7" at the end of the filename specifies the version number. At the time of writing, version 7 is the latest version. If there are higher versions, compile the highest one. Higher version numbers have more options, and also may correct any bugs in the lower versions. To compile version 7, type "acc -o mexhstv7 mexhstv7.c" to the C-shell.

Having compiled it, type "mexhstv7" at any time to get the following help message, which explains the usage of the application:

```
Usage: mexhstv7 [options] <ws hypercube size> <ws hypercube spacing>
      <grid origin 1> <grid origin 2> <grid points 1> <grid points 2>
      <grid spacing 1> <grid spacing 2> <seed> <runs>
```

Options are:

- a Use a 2-2-1 topology instead of a 2-1 topology
- b Print runs which go back in the ordering only
- B n Print n runs which go back in the ordering
- d Printed runs must have different output grids at locking
- e Ignore equivalent target grids
- E Send e-mail when process finished
- f Print false locking runs only
- F n Print n false locking runs
- g Print number of optimal grids for each target grid.
- G Ditto, but no other information printed.
- h x Use a 1 hidden layer topology with x hidden units
- i When printing, show all best grids each familiarisation
- I When printing, show all grids each familiarisation
- j Just print one example, then stop
- m Print each run which exceeds maximum unfamiliar so far
- M Print the max no of unfamiliars for each target grid
- n u Print all runs with u unfamiliar gridpoints at locking
- N u n Print n runs with u unfamiliar gridpoints at locking
- o f Send run outputs to file f
- O ... Specify familiarisation order
- p p Use paradigm p, where p =
 - 0 => old EW
 - 1 => new EW
 - 2 => new EW blackest/whitest
 - 3 => as above, but only if increase in E value
 - 4 => as above, but using new better fit measure
 - 10 => H/H0
 - 11 => blackest/whitest H/H0
 - 20 => best fit H/H0
 - 21 => new best fit H/H0
 - 22 => all new best fit H/H0
- q Quiet output for background process running
- r Print the end report only
- s Print successful runs only
- S n Print n successful runs
- t p q Use targets between indexes p and q inclusive

```

-T g Use target grid g, represented by o's and x's
-u   Print runs which do not terminate only
-U n Print n runs which do not terminate
-x   Inexact fit grids only
-X   Exact fit grids only
Maximum grid size: 32 gridpoints

```

The options will be explained later. The rest of the command informs the program of various constant settings. <ws hypercube size> specifies the size of one edge of the hypercube (centred at the origin) which will be searched to find possible grids. Note that since the neural networks used here have threshold activation functions, the size of the hypercube is not especially important. As the weight values are scaled up, the output of each unit will always be the same. By default, a 2*1 topology is used, which has a total of three weights. If 1.0 is entered here, then the program will search the unit cube centred at the origin, with vertices at ± 0.5 in each dimension.

<ws hypercube spacing> specifies the interval at which weight states will be sampled for every dimension of weight space. A value of 0.1 would mean 11 samples per dimension, at $-0.5, -0.4, -0.3, -0.2, -0.1, 0.0, 0.1, 0.2, 0.3, 0.4$ and 0.5 , which for a 3 dimensional weight space would mean 11^3 or 1331 samples. For a 3×3 grid, and a weight space hypercube size of 1.0, 0.1 is a small enough spacing to find all 54 output grids possible with a 2*1 topology. For a 4×4 grid, this figure is 0.05, and for a 5×5 , 0.01 is needed before all the grids are found.

The program assumes 2D input. The next constants to be specified in the command give details of the input grid to be used. The first two numbers specify the co-ordinates of the bottom left hand corner of the grid (the "grid origin"). The next two numbers specify the number of gridpoints in each direction. The next two numbers specify the spacing between the gridpoints in each direction. For example, a 4×3 grid with (0.0, 0.0) as the origin, a horizontal spacing of 1.0, and a vertical spacing of 0.4 would be specified by the sequence of numbers: 0.0 0.0 3 4 0.4 1.0. The program can cope with any grid size up to a maximum of 32 gridpoints in total (i.e. a 4×8 grid). Runs using more than 25 gridpoints in total are likely to take a very long time, however (i.e. more than 1 week) unless the number of target grids explored is restricted using the -t, or -T options (see later).

The next number is the seed. The seed should be an integer if a deterministic run is required. Any two runs with the same value for the seed will have the same seed of the random number, and therefore, all else being equal, will produce the same output. Otherwise the word "time" may be used instead, in which case, the run is seeded by the amount of time the computer has been on for.

Finally, it is necessary to specify the number of runs. The program automatically goes through every possible target grid for the input grid specified. The "runs" constant specifies the number of different random pattern presentation orders that will be tried for each target grid. A value of 1 for this constant means that a single presentation order will be tried for each target grid.

Here is an example using a 2 by 1 grid. The following command is typed to the C-shell: `mexhstv7 1.0 0.1 0.0 0.0 1 2 1.0 1.0 99 1`. The output is given below.

```
Total number of grids: 4
Running all target grids 1 times...
Termination after all gridpoints familiarised
Target grid (index 0):
∞    1    2
```

```
Global minimum error grids:
∞
```

```
Number of familiarised gridpoints: 2
Familiarisation order:
1 2
```

```
H's targets:
∞ ∞
```

```
H's outputs:
∞ ∞
```

```
H's errors:
0 0
```

```
H0's targets:
∞ ∞
```

```
H0's outputs:
∞ ∞
```

```
H0's errors:
0 0
```

```
H went back in the ordering
Termination after all gridpoints familiarised
```

Target grid (index 1):

xo 1 2

Global minimum error grids:

xo

Number of familiarised gridpoints: 2

Familiarisation order:

1 2

H's targets:

x. xo

H's outputs:

xx xo

H's errors:

0 0

H0's targets:

xo xo

H0's outputs:

xo xo

H0's errors:

0 0

Termination after all gridpoints familiarised

Target grid (index 2):

ox 1 2

Global minimum error grids:

ox

Number of familiarised gridpoints: 2

Familiarisation order:

1 2

H's targets:

o. ox

H's outputs:

oo ox

H's errors:

0 0

H0's targets:

ox ox

H0's outputs:

ox ox

H0's errors:

0 0

Termination after all gridpoints familiarised

Target grid (index 3):

xx 1 2

Global minimum error grids:

xx

Number of familiarised gridpoints: 2

Familiarisation order:

2 1

H's targets:

.x xx

H's outputs:

ox xx

H's errors:

0 0

H0's targets:

xx xx

H0's outputs:

xx xx

H0's errors:

0 0

Unfam	Term	FalseH	FalseH0	H!=H0
0	4	0	0	0
1	0	0	0	0
2	0	0	0	0

Number of times H went back in ordering: 1

Number of times H0 went back in ordering: 0

Number of times unterminated after all gridpoints presented: 0

The output is interpreted in the following way: For each run, comments are printed, such as "H went back in the ordering", or "Termination after all gridpoints familiarised". Then the target grid is printed, along with a unique identifying number, its index. The grid is printed with an 'x' representing a target or output of 1, and a 'o' representing a target or output of '0'. The gridpoint numbers are then printed.

Subsequently, the set of globally optimal grids is printed for that target grid. A false locking run is a run in which either of the boundary representatives does not belong to the set of globally optimal grids once the terminating condition is satisfied for that run.

Then the number of gridpoints familiarised before the terminating condition is satisfied is displayed, followed by the order of familiarisation of the gridpoints. The order of familiarisation uses the index numbers for the gridpoints, which are displayed next to the target grid.

Then the run itself is printed. Familiarisation proceeds horizontally across the screen, with the target, output and error (which is the fit for other paradigms), for each boundary representative.

At the end of all the runs, a report is printed, which shows the number of terminations for a given number of hypothetical gridpoints. Also shown is the number of false locking runs, and the number of runs for which the boundary representatives were not equal at termination. At the bottom of the table, further information is printed, which shows the number of times each boundary representative goes back in the ordering. (This is meaningless in the H/H0 paradigm.) Also displayed is the number of runs which did not satisfy the terminating criteria even after all the gridpoints were familiarised.

There are various options which may be used, which affect the information that is displayed, the target grids used, which paradigm to test, and the topology.

- a This option uses a $2*2*1$ topology rather than the default $2*1$ topology. Be aware that many more grids are possible, even though only one extra hyperplane has been drawn. Runs will therefore take rather longer, as there are a larger number of grids to search to find the optimum for each target grid.
- b Rather than printing all the runs, as is the default, this option displays only those runs in which one or both of the boundary representatives goes back in the ordering at least once.
- B n This limits the number of going back in the ordering runs displayed to the first n .
- d This option only displays those runs in which the boundary representatives are different grids at locking. This is an entirely legitimate possibility for inexact fit target grids with more than one optimum fit.
- e In this option, any target grids which are equivalent to previous target grids which have been tried are ignored. Target grids are equivalent by various symmetries — see chapter 6.

- E This option sends e-mail informing the user when the program has finished running. This is useful for lengthy process which are run in the background.
- f Only runs in which either of the boundary representatives does not belong to the set of optimal fits to the target grid are displayed. These are false locking runs.
- F *n* The number of false locking runs displayed is restricted to the first *n*.
- g With this option, each run printed also shows the number of choices that a boundary representative has for each familiarisation. When the runs are completed, a table follows the final report, which displays the average number of choices for each boundary representative at each familiarisation.
- G For each run, the number of choices only is printed, and other information is not printed.
- h *x* This option enables the user to set the number of hidden units in a single hidden layer. There is currently no possibility to use two hidden layer topologies with this program. You are warned that with numbers of hidden units above 3, you will be facing a very long wait whilst the program finds all the possible grids.
- i This option enables more information to be printed for each familiarisation. It is similar to the -g option, with the exception that the actual optimal grids are displayed each familiarisation, rather than the number of optimal grids.
- I All the possible grids are printed with their errors/inherent fits, depending on the paradigm. Be warned that this may generate a great deal of output.
- j The program stops after the first run printed.
- m A run is only printed if the number of hypothetical gridpoints at locking exceeds the previous run printed.

- M This really applies when several runs are being done per target grid. The program prints the maximum number of hypothetical gridpoints at locking over all the runs done on each target grid.
- n *u* Only print those runs which have *u* hypothetical gridpoints at locking.
- N *u n* Only print *n* runs which have *u* hypothetical gridpoints at locking.
- o *f* The output of the program is saved in the file with name *f*. At the top of the file, the whole of the command is printed. This informs the user of the settings used to achieve the output in the rest of the file.
- O ... Specify the familiarisation order. The order in which the gridpoints are to be familiarised each run may be specified. Very little in the way of checking is done here, so be sure that the order is entered correctly. The numbering of the gridpoints is as specified by the numbers displayed next to the target grid. The familiarisation order must be the first option on the command line. At least one other option must be used, such as options -t or -T.
- p *p* Change the paradigm. The default is the H/H0 paradigm (*p* = 10). *p* should be a number, which corresponds to the desired paradigm. The following values of *p* are of interest with relevance to the paradigms discussed in chapter 6:
 - 4 The B/W paradigm, as described in chapter 6.
 - 10 The H/H0 paradigm, as described in chapter 6.
 - 22 The I/I0 paradigm, as described in chapter 6.
- q No output is sent to the terminal. This is for background process running.
- r Only print the end report. This option just produces the table of results at the end of all the runs.
- s Only those runs with one or more hypothetical gridpoints at locking are printed. These are the successful runs.

- S *n* The first *n* successful runs are printed.
- t *p q* Rather than running through all the possible grids, as is the case by default (unless the -e, -x, or -X options are selected), this runs through all grids from *p* to *q*, inclusive. *p* and *q* are the index numbers of the target grids, and *p* must be less than or equal to *q*.
- T *g* Use target grid *g*. Rather than using index numbers to represent the grid, *g* is a string of 'o' and 'x' characters, which specify the underlying target grid for all runs. They must be in the order specified by the numbers printed next to the target grid in a run using the same size of grid as *g*. This order starts at 1 for the top left hand gridpoint, and increases across the grid before going down.
- u Only print those runs which do not terminate. These are the runs for which the terminating condition of the paradigm is not satisfied even after all the gridpoints have been familiarised.
- U *n* Print *n* runs which do not terminate.
- x Rather than using all the target grids, the only target grids selected are those which are not realisable exactly by the current topology.
- X As -x, but the only target grids selected are those which are exactly realisable.

The following example shows the software being used with a 2*2*1 topology, and a specified 3 × 2 target grid. Information about the errors for all the grids is displayed for each familiarisation, and the H/H0 paradigm is used. Also displayed are the number of choices for H and H0 at each familiarisation. A single successful run out of 30 is to be displayed. For the displayed run, the entire run and familiarisation sequence is printed first. Then, the extra information is printed, which shows the misclassification error of each output grid on the target grid of each boundary representative at each familiarisation. Real gridpoints are indicated by upper case characters. Seed 99 is used.

```
mexhstv7 -a -T xxxxxx -g -I -p 10 -S 1 1.0 0.1 0.0 0.0 2 3 1.0 1.0 99 30
Getting IHL grids...
Combining IHL grids to output grids...
```

Total number of grids: 62
 Extra grids from 2-2-1 topology: 34
 Number of separations of IHL units: 14
 Running target grids between 21 and 21 30 times...

Target grid (index 21):

```

xox  1  2  3
oxo  4  5  6

```

Global minimum error grids:

```

oax xoo xxx xox xox xox
oxo oxo oxo ooo xxo oxx

```

Number of familiarised gridpoints: 5

Familiarisation order:

```
1 2 4 5 3 6
```

H's targets:

```

x.. xo. xo. xo. xox
... ... o.. ox. ox.

```

H's outputs:

```

xox xoo xox xoo xox
oxx xox oax oxo oxx

```

H's errors:

```
0 0 0 0 0
```

H's choices:

```
31 15 7 3 1
```

H0's targets:

```

xox xox xoo xox xox
xoo oxo oxo oxx oxo

```

H0's outputs:

```

xox xox xoo xox xox
xoo ooo oxo oxx oxx

```

H0's errors:

```
0 1 0 0 1
```

H0's choices:

```
1 6 1 1 6
```

Familiarisation 1: Gridpoint 1

Target grids: (H, H0)

```
Xoo Xxo
```

```
ooo xoo
```

Output grids:

```

Ooo Oax Oax Oax Ooo Ooo Oxx Oxx Ooo Ooo Oax Oxx Oxx Ooo Xoo Xxx Xxx Xxx Xoo Xxx
ooo ooo oax oxx oax oxx oax oxx xoo xxx xxx xxx ooo xoo ooo ooo oax oxx xoo xxx

```

Errors on H's target grid:

```
1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
```

Errors on H0's target grid:

```
3 4 5 6 4 5 4 5 2 4 5 4 3 3 2 2 3 4 1 3
```

```

Xoo Xoo Xxo Xxo Xxo Xxo Xxx Xxx Ooo Oax Oxo Oxo Oxo Oxo Oxx Oax Oax Oxo Oxo Oxo
xoo xxx ooo xoo xoo xxx xoo xxx oxo oxo ooo oax oxo oxx oxo xoo xxx xoo xxx xxx

```

Errors on H's target grid:

0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
 Errors on H0's target grid:
 2 3 1 0 1 2 1 2 4 5 2 3 3 4 4 3 4 1 2 3

OxOx OxOx Xoo Xxo Xoo Xoo Xxo Xxo Xox Xox Xox Xox Xox Oox Xox Xox Xox Xox Ooo Xoo
 xoo xxo oox oox oxo oox oxo oox oxo ooo xoo xxo xxx xox oox xox xox oox xox xox

Errors on H's target grid:
 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0
 Errors on H0's target grid:
 2 3 3 2 3 4 2 3 3 3 2 3 4 4 4 3 2 5 3 2

Xxo OxOx
 xox xox

Errors on H's target grid:
 0 1
 Errors on H0's target grid:
 1 3

Output grids chosen: (H, H0)
 Xox Xxo
 oox xoo

Familiarisation 2: Gridpoint 2
 Target grids: (H, H0)
 Xoo Xox
 ooo oxo

Output grids:
 Ooo Oox Oox Oox Ooo Ooo Oxx Oxx Ooo Ooo Oox Oxx Oxx Ooo Xoo Xxx Xxx Xxx Xoo Xxx
 ooo ooo oox oox oox oox oox oox xoo xxx xxx xxx ooo xoo ooo ooo oox oox xoo xxx

Errors on H's target grid:
 1 1 1 1 1 1 2 2 1 1 1 2 2 1 0 1 1 1 0 1
 Errors on H0's target grid:
 3 2 3 2 4 3 4 3 4 4 3 4 3 3 2 2 3 2 3 3

Xoo Xoo Xxo Xxo Xxo Xxo Xxx Xxx Ooo Oox Oxo Oxo Oxo Oxo Oxx Oox Oox Oxo Oxo Oxo
 xoo xxx ooo xoo xoo xxx xoo xoo oox oox ooo oox oox oox oox xoo xoo xoo xoo xxx

Errors on H's target grid:
 0 0 1 1 1 1 1 1 1 2 2 2 2 2 1 1 2 2 2
 Errors on H0's target grid:
 2 3 3 4 3 4 3 2 2 1 4 5 3 4 2 3 2 5 4 5

Oxx Oxx Xoo Xxo Xoo Xoo Xxo Xxo Xxx Xox Xox Xox Xox Oox Xox Xox Xxx Xox Ooo Xoo
 xoo xoo oox oox oox oox oox oox ooo xoo xoo xxx xox oox xox xox oox xox xox

Errors on H's target grid:
 2 2 0 1 0 0 1 1 1 0 0 0 0 1 0 0 1 0 1 0
 Errors on H0's target grid:
 4 3 3 4 1 2 2 3 1 1 2 1 2 4 2 3 4 1 5 4

Xxo Oxx
 xox xox

Errors on H's target grid:
 1 2

Errors on H0's target grid:
5 5

Output grids chosen: (H, H0)
X0o X0x
x0x 00o

Familiarisation 3: Gridpoint 4
Target grids: (H, H0)
X0o X0o
0oo 0xo

Output grids:
00o 00x 00x 00x 00o 00o 0Xx 0Xx 00o 00o 00x 0Xx 0Xx 00o X0o XXx XXx XXx X0o XXx
0oo 0oo 0ox 0xx 0ax 0ax 0ax 0ax Xoo Xxx Xxx Xxx 0oo Xxo 0oo 0oo 0ax 0ax Xoo Xxx

Errors on H's target grid:
1 1 1 1 1 1 2 2 2 2 3 2 2 0 1 1 1 1 2
Errors on H0's target grid:
2 3 4 3 3 2 5 4 3 3 4 5 4 2 1 3 4 3 2 4

X0o X0o XXo XXo XXo XXo XXx XXx 00o 00x 0Xo 0Xo 0Xo 0Xo 0Xx 00x 00x 0Xo 0Xo 0Xo
Xxo Xxx 0oo Xoo Xxo Xxx Xoo Xxo 0xo 0xo 0oo 0ax 0xo 0xx 0xo Xoo Xxo Xoo Xxo Xxx

Errors on H's target grid:
1 1 1 2 2 2 2 2 1 1 2 2 2 2 2 2 3 3 3
Errors on H0's target grid:
1 2 2 3 2 3 4 3 1 2 3 4 2 3 3 4 3 4 3 4

0Xx 0Xx X0o XXo X0o X0o XXo XXo XXx X0x X0x X0x X0x 00x X0x X0x XXx X0x 00o X0o
Xoo Xxo 0ox 0ox 0xo 0xx 0xo 0xx 0xo 0oo Xoo Xxo Xxx Xax 0ax Xax Xax 0xx Xax Xax

Errors on H's target grid:
3 3 0 1 0 0 1 1 1 0 1 1 1 2 0 1 2 0 2 1
Errors on H0's target grid:
5 4 2 3 0 1 1 2 2 2 3 2 3 5 3 4 5 2 4 3

XXo 0Xx
Xox Xox

Errors on H's target grid:
2 3
Errors on H0's target grid:
4 6

Output grids chosen: (H, H0)
X0x X0o
0ox 0xo

Familiarisation 4: Gridpoint 5
Target grids: (H, H0)
X0o X0x
0Xo 0Xx

Output grids:

OOO OOX OOX OOX OOO OOO OXX OXX OOO OOO OOX OXX OXX OOO XOO XXx XXx XXx XOO XXx
 OOO OOO OOX OXX OOX OXX OOX OXX XOO XXx XXx XXx OOO XXO OOO OOO OOX OXX XOO XXx

Errors on H's target grid:
 2 2 2 1 2 1 3 2 3 2 2 3 3 2 1 2 2 1 2 2
 Errors on H0's target grid:
 4 3 2 1 3 2 3 2 5 3 2 3 4 4 3 3 2 1 4 2

XOO XOO XXO XXO XXO XXO XXx XXx OOO OOX OXO OXO OXO OXO OXx OOX OOX OXO OXO OXO
 XXO XXx OOO XOO XXO XXx XOO XXO OXO OXO OOO OOX OXO OXx OXO XOO XXO XOO XXO XXx

Errors on H's target grid:
 1 1 2 3 2 2 3 2 1 1 3 3 2 2 2 3 2 4 3 3
 Errors on H0's target grid:
 3 2 4 5 4 3 4 3 3 2 5 4 4 3 3 4 3 6 5 4

OXX OXX XOO XXO XOO XOO XXO XXO XXx XOX XOX XOX XOX OOX XOX XOX XXx XOX OOO XOO
 XOO XXO OOX OOX OXO OXx OXO OXx OXO OOO XOO XXO XXx XOX OOX XOX XOX OXx XOX XOX

Errors on H's target grid:
 4 3 1 2 0 0 1 1 1 1 2 1 1 3 1 2 3 0 3 2
 Errors on H0's target grid:
 5 4 2 3 2 1 3 2 2 2 3 2 1 3 1 2 3 0 4 3

XXO OXX
 XOX XOX

Errors on H's target grid:
 3 4
 Errors on H0's target grid:
 4 4

Output grids chosen: (H, H0)
 XOO XOX
 OXO OXX

Familiarisation 5: Gridpoint 3
 Target grids: (H, H0)
 XOX XOX
 OXO OXO

Output grids:
 OOO OOX OOX OOX OOO OOO OXX OXX OOO OOO OOX OXX OXX OOO XOO XXX XXX XXX XOO XXX
 OOO OOO OOX OXX OOX OXX OOX OXX XOO XXx XXx XXx OOO XXO OOO OOO OOX OXX XOO XXx

Errors on H's target grid:
 3 2 2 1 3 2 3 2 4 3 2 3 3 3 2 2 2 1 3 2
 Errors on H0's target grid:
 3 2 3 2 4 3 4 3 4 4 3 4 3 3 2 2 3 2 3 3

XOO XOO XXO XXO XXO XXO XXX XXX OOO OOX OXO OXO OXO OXO OXX OOX OOX OXO OXO OXO
 XXO XXx OOO XOO XXO XXx XOO XXO OXO OXO OOO OOX OXO OXx OXO XOO XXO XOO XXO XXx

Errors on H's target grid:
 2 2 3 4 3 3 3 2 2 1 4 4 3 3 2 3 2 5 4 4
 Errors on H0's target grid:
 2 3 3 4 3 4 3 2 2 1 4 5 3 4 2 3 2 5 4 5

```
OXX OXX XOO XXO XOO XOO XXO XXO XXX XOX XOX XOX XOX OOX XOX XOX XXX XOX OOO XOO
XOO XXO OOX OOX OXO OXX OXO OXX OXO OOO XOO XXO XXO XOX OOX XOX XOX OXX XOX XOX
```

```
Errors on H's target grid:
4 3 2 3 1 1 2 2 1 1 2 1 1 3 1 2 3 0 4 3
Errors on H0's target grid:
4 3 3 4 1 2 2 3 1 1 2 1 2 4 2 3 4 1 5 4
```

```
XXO OXX
XOX XOX
```

```
Errors on H's target grid:
4 4
Errors on H0's target grid:
5 5
```

```
Output grids chosen: (H, H0)
XOX XOX
OXx OXx
```

Unfam	Term	FalseH	FalseH0	H!=H0
0	25	0	0	18
1	5	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0

```
Number of times H went back in ordering: 29
Number of times H0 went back in ordering: 46
Number of times untermiated after all gridpoints presented: 0
Average number of choices for H and H0 each familiarisation:
```

Fam	H	H0
1	31	1.66667
2	15	1.33333
3	7	1.16667
4	3	1.5
5	1	6
6	6	6

Finally, the following example shows the maximum number of hypothetical gridpoints at locking for the I/I0 paradigm for 20 runs on each non-equivalent, inexact fit 4×4 target grid between indexes 6 and 15 inclusive. Seed 77 is used. Only the report is printed, along with the maximum number of hypothetical gridpoints for each target grid tested. The total number of grids is the total number of output grids found for the weight states sampled.

```
mexhstv7 -e -M -p 22 -r -t 6 15 -x 1.0 0.05 0.0 0.0 4 4 1.0 1.0 77 20
Total number of grids: 174
Running target grids between 6 and 15 20 times...
Ignoring equivalent target grids.
Inexact fit target grids only.
Max. unfamiliar gridpoints for target grid 6: 3 (1 times)
Max. unfamiliar gridpoints for target grid 9: 4 (1 times)
Max. unfamiliar gridpoints for target grid 10: 3 (2 times)
```

Max. unfamiliar gridpoints for target grid 11: 3 (1 times)

Unfam	Term	FalseI	FalseI0	I!=I0
0	37	0	0	29
1	23	0	0	18
2	15	0	0	8
3	4	0	0	3
4	1	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0
9	0	0	0	0
10	0	0	0	0
11	0	0	0	0
12	0	0	0	0
13	0	0	0	0
14	0	0	0	0
15	0	0	0	0
16	0	0	0	0

Number of times I went back in ordering: 0

Number of times I0 went back in ordering: 3

Number of times unterminated after all gridpoints presented: 0

Target grids 7, 8, 12, 14 and 15 are ignored because they are all realisable. Target grid 13 is ignored because it is symmetrically equivalent to target grid 11. Figure A.1 shows the target grids.

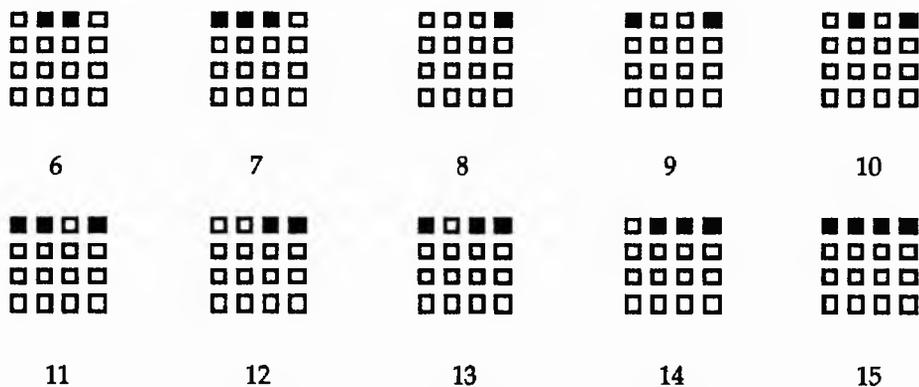


Figure A.1 — The 4×4 target grids with indexes from 6 to 15 inclusive.

A.2.2 Imperfect Training Version Using a GA

The imperfect training application is contained in the directory /user/rsch/gary/sim/grid/imperfect. You will need the library files and the files makefile and mmgav9.c to compile it. There may be later versions, as for the exhaustive search technique. Version 9 is the most up to date version at the time of writing. However, the makefile should be set up such that you can just type "make" to C-shell, and the most up to date

version will be compiled. Having done so, the following help message can be obtained at any time by typing "mmgav9":

```
Usage: mmgav9 [options] <X origin> <Y origin> <X spacing> <Y spacing>
      <Maximum number of generations>
      <GA constants> <Topology> <Patterns> <Weights> <Seed> <Trials>
```

Options are:

- b Breakdown of inherent fit displayed
- d o Restricted display option, o, which may be one of:
 - report Print report only
 - nofams Do not print familiarisation sequence
- f Constant familiarisation sequence for all trials
- g f Save gridpoints to pattern file f (with .pat suffix)
- G Just train the grid using a GA and MCE
- i f Use (irregular) grid from pattern file f
- m n Terminate validation after n generations
- M Send mail when runs complete
- o f Output to file f
- p x Progress report interval for GA = x
- t f Test saved solutions on samples from files whose names
 - are of the following format:
 - fNN.pat
 - where f is specified on the command line, and NN is a two digit
 - number starting at 01 and increasing in units of 1.
 - T distribution data are available for values of NN up to 16.
- T Just show the target grid
- v Compare with validation on grid
- V Compare with validation on patterns
- w n Just do validation on grid with n gridpoints in v. set
- W n Just do validation on patterns with n patterns in v. set

The options will be explained later. The program is designed to work for 2D input and 1D output only, with the I/I0 paradigm, as described in chapter 6. <X origin> is the co-ordinate of the origin of the grid on the horizontal axis. It is akin to <grid origin 2> in mexhstv7. <Y origin> is the co-ordinate of the origin of the grid on the vertical axis. <X spacing> and <Y spacing> are again similar to <grid spacing 2> and <grid spacing 1> in mexhstv7. Here, however, the grid is (by default) calculated on the basis of the patterns file specified. The grid origin specifies the co-ordinate at which to start the grid. For each pattern, the nearest gridpoint is found and defined by making increments of the appropriate spacing value on each dimension until the nearest gridpoint is found. The only gridpoints that will be defined are those which have been found on the basis of the patterns. This means that each gridpoint can be potentially familiarised.

The <maximum number of generations> specifies the maximum number of generations allowed to find the best output grid for I or I0 with each familiarisation.

The next four items on the command line are files. The format of the patterns, weights and topology files are as shown in section A.1. The patterns file contains the patterns from which the grid will be constructed. The weights file specifies the filename which the final weights will be saved to. The topology file specifies the topology which is to be used on the given grid. The user is warned that the more weights there are in the topology, the more generations will be needed to find a good fit each familiarisation. The GA constants file is of a format indicated by the following example:

```
population_size: 100
crossover_probability: 1.0
mutation_probability: 1.0
number_of_bytes_per_chromosome: 2
number_of_bits_before_binary_point: 5
best_gene_selection_bias: 10.0
```

All GA constants files must end with the suffix “.gac”.

The first three constants in the GA constants file are self-explanatory.

The `number_of_bytes_per_chromosome` constant specifies the number of bytes that will be allocated for each weight. This means that for a $2 \times 2 \times 1$ topology, with 9 weights in total, the following file will use a string of length $18 \times 8 = 144$ bits to represent the entire weight state. There was some confusion about genes and chromosomes and which was a smaller DNA sequence than which when writing the software. I thought that there were many chromosomes on a gene, and thus it seemed appropriate to label a single weight as being a chromosome. Naturally, it turned out to be the other way round, and thus it would have been better to make a correspondence between a gene and a weight. Since it is all completely meaningless, and I had already created several files with this format, the convention stuck, however.

The `number_of_bits_before_binary_point` constant specifies how many bits out of the number of bytes allocated for each weight will count towards the integer part of the weight value when the binary string is decoded. The value 5 here means that the maximum integer part weight value is 63. The remaining 11 bits are used for the rational part of the weight value.

The `best_gene_selection_bias` constant gives the strength of the bias that will be assigned to the best gene in the population when it comes to selecting genes for the next generation. This line of the file is optional, and if not included, the default value is 2.0. The higher the value, the more likely the next generation is to contain genetic material from the best gene in the current generation.

After all the files have been specified, in the order indicated, the seed may be specified, in the same way as `mexhstv7`. Finally, the number of trials to be run on the target grid based on the pattern file is indicated. This specifies the number of different random presentation orders that will be tried on the target grid. A full sequence of familiarisation is tested for each trial, until the terminating condition is satisfied, or all the gridpoints have been familiarised.

Thus, before the program can be run, a pattern file, topology file and GA constants file must be prepared. The following pattern file may be used to deliberately create the 3×3 grid shown in figure A.2:

```
number_of_patterns: 9
0.0 0.0 | hi
1.0 0.0 | hi
2.0 0.0 | hi
0.0 1.0 | lo
1.0 1.0 | hi
2.0 1.0 | hi
0.0 2.0 | hi
1.0 2.0 | lo
2.0 2.0 | hi
```

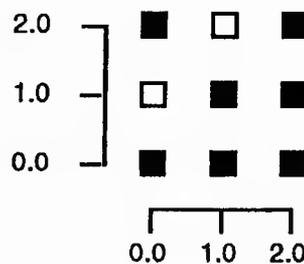


Figure A.2 — The grid to be used in the following examples.

A topology file is also created for a 2×1 topology, and the GA constants file illustrated above will also be used. In `/user/rsch/gary/sim/grid/imperfect`, the directory in which `mmgav9` is to be found, there are also subdirectories containing various topology, pattern, GA constants and

weight files. These directories are given single letter names, which are "a", "p", "g" and "w" respectively. The topology file is "a/2-1.net", the GA constants file is "g/wild.gac" and the pattern file is "p/appdx.pat". Thus, if "mmgav9 0.0 0.0 1.0 1.0 50 g/wild.gac a/2-1.net p/appdx.pat w/appdx.wgt 99 1" is typed to C-shell, with /user/rsch/gary/sim/grid/imperfect as the current working directory, the output is as follows:

Index numbers for gridpoints:

```

7     8     9
4     5     6
1     2     3
    
```

Familiarisation sequence:
8 9 2 6 3 1 7 4 5

Fam no.	GP no.	I target	I output	I ihf	I0 target	I0 output	I0 ihf
1	8	oOo ooo ooo	oOo ooo ooo	0	xOx xxx xxx	xOx xxx xxx	0
2	9	oOX ooo ooo	oOo ooo ooo	0	xOX xxx xxx	xOX xxx xxx	0
3	2	oOX ooo oXo	oOo ooo oOo	0	xOX xxx xXx	xOX xxx xXx	0
4	6	oOX oaX oXo	oOX oaX oXx	2	xOX xOX xXo	xOX xOX xXo	1
5	3	oOX oaX oXX	oOX oaX oXX	4	xOX xOX xOX	xOX B xOX xOX	0
6	1	oOX oaX XXX	oOX oaX OXX	4	xOX xOX XXX	xOX xOX XXX	0
7	7	XOX oaX XXX	OXX oaX OXX	4	XOX xOX XXX	XXX xOX XXX	0
8	4	XOX OaX XXX	OXX OaX OXX	4	XOX OaX XXX	OXX OaX XXX	1

```

9      5      XOX      OOX      6      XOX      OOX      1
          OXX      OXX
          XXX      XXX      XXX      XXX

```

Termination achieved with 0 unfamiliar gridpoints

Underlying target grid

```

XOX
OXX
XXX

```

I's final output grid

```

OXX
OXX
XXX

```

I0's final output grid

```

OXX
OXX
XXX

```

Report: -----

Trial	Unfam	I mce	I0 mce	I != I0	I back	I0 back
1*	0	1	1	0	0	1

* - Saved weight state of I and I0.

The first part of the output informs the user of the index numbers of the gridpoints. (An index number of 0 means that the gridpoint corresponding to that position is undefined. This is not the case here.) The familiarisation sequence is then given, followed by a table, which, for each familiarisation, prints the index number of the gridpoint familiarised, and the target output and inherent fit for I and I0. (Note that the CLS term is ignored in the inherent fit value.) If either I or I0 go back in the ordering, a B is printed next to the output grid. (See I0 in familiarisation 5.)

The same convention is used for printing the grids as in mexhstv7. An upper case letter means the gridpoint is real, with an "x" (or "X") representing a target or output of 1, and an "o" (or "O") representing a target or output of 0.

When the terminating condition is satisfied, or all gridpoints have been familiarised, whichever comes first, the underlying target grid, and final output grids of I and I0 are printed. If many trials are being done, the

software then proceeds to the subsequent trial. Only 1 trial was used in the above example.

Once all trials are finished, a report is printed. This shows, for each trial, the number of hypothetical gridpoints at locking (which is -1 if I and IO had not locked after all gridpoints were familiarised), the misclassification errors of I and IO's locking grids, whether or not I and IO were the same grid at termination, and how many times I and IO went back in the ordering during the trial.

Although the asterisk indicates the saved weight state, version 9 actually saves the weight states of all trials. The run above creates the following files in the directory w:

```
w/appdx.I-t01.wgt      w/appdx.IO-t01.wgt
w/appdx.I.wgt         w/appdx.IO.wgt
```

The weight state for I and IO each trial is saved as \$.I-tNN.wgt and \$.IO-tNN.wgt respectively, where \$ represents the filename entered on the command line minus the .wgt suffix ("w/appdx" here) and NN is the number of the trial. The starred trial in the report is also saved as \$.I.wgt and \$.IO.wgt for the weight states of I and IO. This is always the trial with the greatest number of hypothetical gridpoints at locking.

There are various options which may be used, to enable comparisons with validation, training using just a GA, and the information that is displayed:

- b Display a breakdown of the inherent fit of I and IO each familiarisation. All the possible inherent fit values are displayed: OFC, OFM, OUM, OUC, SFC, SFM, SUM, SUC, and CLS. See chapter 6 for an explanation of the terminology.
- d o Restricted display options. Less information is printed. This is especially good for situations whereby many trials are to be done, which would otherwise result in a great deal of output. o is a string which may be one of:

"report" Only the report is printed.

- "nofams" The familiarisation sequence is not printed, but the underlying target grid, and output grids of I and IO are printed at the end of each trial.
- g *f* The gridpoints are saved to a file *f*. *f* must have a ".pat" suffix.
- G The grid is used as the set of patterns for a GA to train, with misclassification error as the function to be minimised. No Mitchellian processes are used. This is basically treated as a scenario in which validation is to be done on the grid, only with no points in validation set. It is useful for deciding how many generations may be needed to find a near-optimum solution on the grid based on the given patterns.
- i *f* File *f* (with a ".pat" suffix) is used as the set of gridpoints in the grid. The targets of the gridpoints are decided using the patterns in the patterns file entered after the GA constants file on the command line. Each pattern has a vote for the target of its nearest gridpoint. Each gridpoint assumes the target of the majority of the patterns who have a vote for it. Equal votes are cast in favour of the nearest pattern to the gridpoint. Each gridpoint must have a target assigned to it by at least one of the patterns, otherwise an error message is displayed, and the program terminates.
- m *n* By default, validation continues until the validation error goes up from one generation to the next. This option enables the setting of a maximum number of generations, *n*, to be used, in case validation takes too long.
- o *f* The output is sent to file *f*, rather than to the terminal. The first line of *f* contains the command entered to C-shell, so that you can tell how the output was generated.
- p *x* A progress report is printed during training every *p* generations. This takes the following format:

```
verr=0 gen=99 glob=4 max=4 (freq=95) min=2 mode=4 (freq=95) diff=3
```

"verr" is the validation error. It is not printed if the GA is being used to train I or IO. "gen" is the number of the generation. "glob"

is the best solution found so far. (N.B. The GA maximises a function. When training using the `-G` option above, the value printed here is the fit, rather than the misclassification error.) "max" is the best solution in the current population, and in parentheses after it is "freq" which is the total number of members of the population which have this solution. "min" is the worst solution in the current population, and "mode" is the most popular, with the frequency of the most popular solution in parentheses. "diff" is the number of different solutions in the population.

`-t f` A paired *t*-test is done using the best validation trial and the best Mitchellian trial (the grid found by I is used), to see which has the better validation performance on a number of samples of untrained patterns. There is a bug, in that when validation and the I/I0 paradigm have the same solution, the *t* statistic is displayed as "NaN" which means "not a number". The bug is not crucial, and could be fixed by checking that validation and the I/I0 paradigm do not have the same classification of the patterns in the untrained sample. If they do, a message should be printed saying so, and that there is therefore no point in comparing their performance using a *t*-test.

The number of samples in the test depends on how many files there are with the required filename. *f* is a prefix to the files, with the remainder of the filenames being a 2 digit number and the ".pat" suffix. The 2 digit number must start at "01" and increase in steps of 1 to the last sample.

For example, if 5 samples are being used to compare the performance of validation and the I/I0 paradigm, then a set of pattern files would be: example01.pat, example02.pat, example03.pat, example04.pat and example05.pat. In this case, *f* would be the string "example".

`-T` The underlying target grid only is shown. No training takes place. This is useful for checking that the underlying grid is close to the one you are looking for on the set of patterns being used to

determine the grid. If the grid is not desired, then you know that a different pattern sample should be chosen.

- v A comparison is made with validation on the grid. The best Mitchellian trial (which is the trial with the maximum number of hypothetical gridpoints at locking) is used to determine the number of gridpoints which are to be in the validation set. Validation is then done using a GA as the training algorithm until the first minimum of validation error on the validation set.
- V The same as -v, except that the underlying patterns are used, rather than the grid. The proportion of patterns in the validation set is the same as the proportion of gridpoints which are hypothetical at locking in the best Mitchellian trial.
- w *n* The same as -v, except that the Mitchellian trials are not done, and *n* specifies the number of gridpoints which are to be in the validation set.
- W *n* The same as -V, except that the Mitchellian trials are not done, and *n* specifies the number of patterns which are to be in the validation set.

A.3 IO Visualisor

The IO visualisor is a useful facility for observing the behaviour of neural networks with 1 or 2D input, and one output unit. It is to be found in the directory /user/rsch/gary/bin and the command is "io". The source code is in /user/rsch/gary/bin/source. To compile it, you will need the makefile in /user/rsch/gary/bin. Type "make io" to the C-shell. The source code requires the library archives in /user/rsch/gary/lib. Having compiled the program, type "alias io /user/rsch/gary/bin/io" to C-shell, and then you can use io in any current working directory. Type io at any time to get the following help message:

```
Usage : io [options] <topology> <weights> <asdata> <raster>
Options are:
  -a <xmrk> <ymrk> [<xtck> <ytck> <size>]
                        Draw axes
  -d [ex] ++           Add double hyperplanes [at <ex>]
                        Default for <ex> = +/-4.595 (cut = 0.01/0.99)
```

```

-g          Use grey level rasterfile
-G <gridpoints> [[S]size] ++
            Add gridpoints from <gridpoints> file
-h ++      Add hyperplanes
-n         Don't plot IO of network
-o         Overwrite rasterfile if it exists
-p <pattern> [[S]size]
            Add patterns from <pattern> file
-P <greylevel> +
            Draw patterns in <grey level>
-q         Quit if rasterfile exists
-r         Don't add a report
-s <style> ++ Style of IO graph plot
            <style> may be one of:
            bw      Black and white
            BW *    Dark grey & Light grey
            cNN     NN contours
            gl0     10 grey level contour map
            gcNN    NN contours filled with grey
            smooth  Shade of grey proportional to output
            smcNN   As smooth, but with NN contours
-S <greylevel> <thickness> +
            Grey level and thickness of 1D plot
-x <func>   Use activation function <func>
-X <func>   As -x but applies to output only
            <func> may be one of:
            sig *   Sigmoid function
            lin     Linear function
            pct     Threshold function
            act     No function (output only)

* -- Default
+ -- 1D input only
++ -- 2D input only

```

The options will be discussed later. The command requires four files to be specified. The topology and weights files are of the format specified in section A.1. <rasdata> is a text file which must end in a ".rsd" suffix, and is of a format illustrated by the following example:

```

-10.0  -10.0
20.0   20.0
192    192

```

The first two numbers specify the co-ordinate in input space of the bottom left hand corner of the IO plot for 2D input space. For 1D input, the first number specifies the input value on the left hand side of the IO plot, and the second number specifies the output value at the bottom of the IO plot. The second two numbers give the amount of input space that is to be covered on each axis for 2D input. For 1D input, the first number specifies the length of input space to be covered. The second number specifies the height of output space.

Thus, for the above example, if the graph were for 2D input, the bottom left hand corner of the rasterfile would be at (-10, -10) in input space, and the top right hand corner would be at (10, 10). If the input was 1D, then the left hand side of the rasterfile would represent an input of -10, and the right hand side an input of 10. The bottom of the rasterfile would represent an output of -10, and the top of the rasterfile would represent an output of 10.

Finally the last two numbers give the width and height of the rasterfile in pixels, respectively.

<raster> is the name of the Sun format rasterfile (cf. man rasterfile) to be created, which must have a ".ras" suffix.

To import a rasterfile into Microsoft Word, use Converter to convert it into a PICT format file, and then load it into a drawing application which can read PICT format. Please note that Converter can only convert monochrome rasterfiles. For colour rasterfiles, convert the rasterfile to a postscript file using the C-shell command "ras2ps". For example, the rasterfile "example.ras" is converted to a postscript file by typing "ras2ps example.ras > example.ps" to C-shell. Postscript files should have a ".ps" suffix. MS Word can then read the postscript file, but be warned that it may take a long time to load in, and when it has, you will not be able to see the image in MS Word. Instead, the screen will show some of the image data in the frame set aside for the picture, such as the date and time of creation, and the original filename. When the Word document is finally printed, however, the original image will be there.

To view the rasterfile, type /user/rsch/gary/bin/loadscrn <raster>, and to print it, type lpr -v -Plw <raster> where *lw* is the name of a printer with postscript capability. In the absence of /user/rsch/gary/bin/loadscrn, the rasterfile can be converted to postscript as indicated above, and then viewed using a standard viewer such as ghostview, or pageview. The image quality may be unsatisfactory, however. Failing this, you'll just have to waste paper.

The example in figure A.3 shows the 2D IO space from training a 2*10*1 topology for 165 cycles of standard back-propagation with a learning rate of 0.2 and a momentum of 0.7 on a single revolution of Lang and

Witbrock's 2 spirals problem.² The command entered to C-shell, with /user/rsch/gary/bin/source as the current working directory was:

```
io bp/a/2-10-1.net bp/w/2spir.wgt bp/r/2spir.rsd bp/io/2spir.ras
```

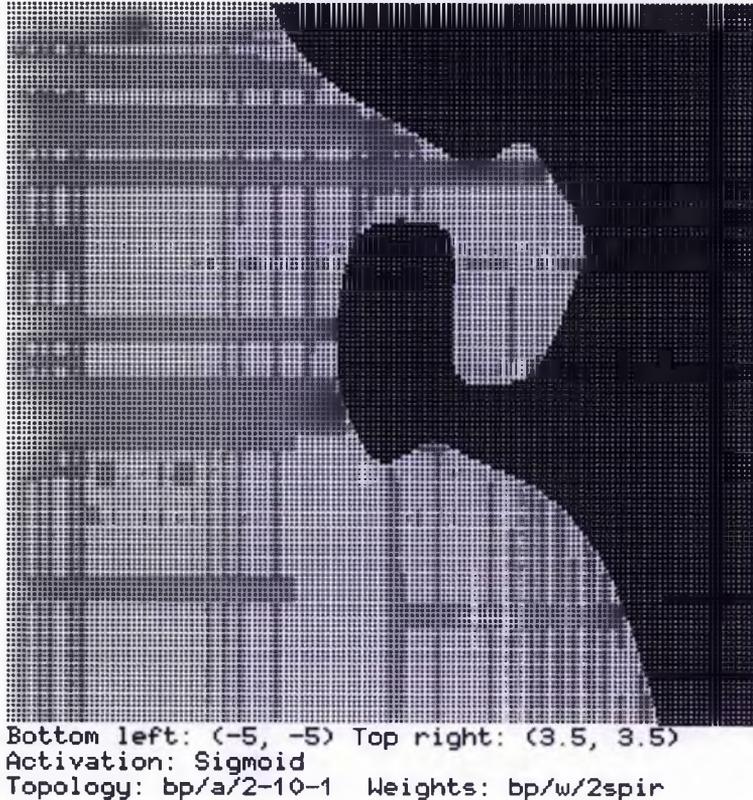


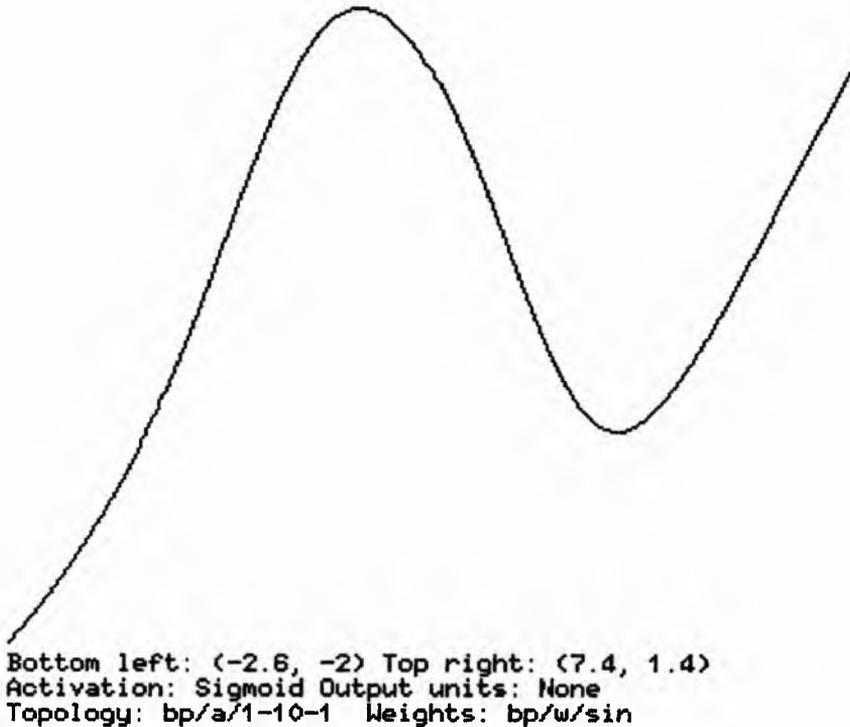
Figure A.3 — Rasterfile produced for the solution weight state found for the 2 spirals problem. Note the addition of the report at the bottom of the rasterfile. This is to inform the user of the files that were used to create the rasterfile.

Figure A.4 shows a 1D input space example. This is a fit to a single cycle of $\sin(x)$, using a $1 \times 10 \times 1$ topology. The problem was trained using a variable learning rate, for 20 000 cycles. The excitation of the output unit is treated as the output of the network. This means there is no sigmoid on the output unit during training. To reflect this, when printing the IO graph, it is necessary to use the $-X$ option (see later). The command entered to C-

²Lang & Witbrock, 1988

shell, with the same directory as before being the current working directory was:

```
io -X act bp/a/1-10-1.net bp/w/sin.wgt bp/r/sin.rsd bp/io/sin.ras
```



*Figure A.4 — Approximation to the sin function using a 1*10*1 topology. The excitation of the output unit is treated as the output of the network, and hence there is no activation function on the output unit. This information is reflected in the report beneath the IO graph.*

These graphs are rather uninformative, and the options enable more information to be printed on the graphs. The options available are:

```
-a  $x_m$   $y_m$  [ $x_t$   $y_t$   $s$ ]
```

Axes are displayed, with mark intervals x_m and y_m for the horizontal and vertical axes, respectively. Optionally, the number of tick intervals between the marks on the horizontal and vertical axes may be specified (x_t and y_t respectively), along with the size, s , of the axes. x_t , y_t and s must all be specified together, if any of these values is to be changed. The size, s , should be a whole

number which specifies the amount by which the size of the axes is to be multiplied. The effect is to give thicker lines, and larger lettering, which is useful if the rasterfile is to be printed onto a slide.

- d [*ex*] Double hyperplanes are drawn at excitation values $\pm ex$ for each first hidden layer unit. The default value draws double hyperplanes when the output of each hidden unit is 0.01 and 0.99. The double hyperplanes are drawn with dashed lines. See chapter 4 for more on double hyperplanes. This facility is available for 2D input only.
- g Use a colour rasterfile. The default is a monochrome rasterfile. With colour rasterfiles, there are 256 shades of grey, rather than the 17 shades simulated on monochrome rasterfiles by using different patterns of pixels. However, colour rasterfiles take up eight times more memory, and are not so easy to import into a Word document.

-G *f* [[*S*]*s*]

Gridpoints are plotted on the rasterfile from a pattern file, *f* specifying the location and targets of the gridpoints. The pattern file must have a ".pat" suffix. The gridpoints are indicated by squares, centred at the appropriate position in input space. Black squares indicate a target of 1, white squares indicate a target of 0. The size of the squares plotted may be changed optionally. If larger squares are required, a whole number is specified which indicates the amount by which the default size is to be multiplied. If smaller squares are required, a whole number, preceded by the letter 'S' is entered, which indicates the number of pixels to be subtracted from the default size. For example if gridpoints are to be plotted from the gridpoint file "example.gridpoints.pat" and they are to be twice the normal size, the option would be written -G example.gridpoints.pat 2. If, however, they are to be 4 pixels smaller than the normal size, the option would be written -G example.gridpoints.pat S4, with no white space between the S and the 4.

- h The hyperplanes are drawn, for each hidden unit in the first hidden layer. This option applies to 2D input only.
- o If the rasterfile specified on the command line exists already, then it is automatically over-written. This is useful for running *io* in the background. If this option is not specified, and the rasterfile exists, then you are asked if you want to replace the old rasterfile with the new one. Type 'y' if you do, and 'n' if you do not.

-p *f* [[S]*s*]

This option works in exactly the same way as *-G*, except that it is for patterns, rather than gridpoints. When patterns and gridpoints are to be plotted on the same rasterfile, it is a good idea to make the patterns smaller, so that they can be discriminated from the gridpoints.

- P *g* The patterns are printed in grey level, *g*, which should be an integer between 0 (white) and 255 (black). In monochrome rasterfiles, grey levels are simulated by various combinations of pixel patterns. This option is available for 1D input only, and is useful to enable the discrimination of the patterns from the IO function.
- q The program terminates if the rasterfile exists already. This is useful for running *io* in the background, when you want to be sure that any existing rasterfiles will not be over-written.
- r The report at the bottom of the rasterfile is not added. This is useful when the rasterfile is to be put into a document such as a PhD thesis!
- s *style* The kind of graph drawn can be varied. There are seven possible values for *style*:
 - bw A black and white plot. A black point is plotted where the output is 1 and a white point where the output is 0.
 - BW As for *bw*, except that dark grey and light grey are used in place of black and white. This is the default.

- cNN* *NN* contours are plotted as the output increases from 0 to 1. Note that there must be no white space between the 'c' and the number of contours to be plotted.
- g10* Ten grey levels are plotted, as the output increases from 0 to 1. Five lighter shades of grey are plotted from 0 to 0.5, and darker shades are used from 0.5 to 1. Thus the black/white boundary is shown clearly.
- gcNN* As for *cNN*, except that the regions between the contours are filled with a uniform shade of grey, which is light for outputs close to 0, and becomes darker as the output goes to 1. This style works best when a colour rasterfile is plotted using the *-g* option.
- smooth* A shade of grey proportional to the output is plotted for each point in input space. This style works best in conjunction with the *-g* option.
- smcNN* This style combines the smooth plot above with a contoured plot. It differs from the *gcNN* style in that the shade of grey between the contours will not be uniform. Again, the best effects are observed with the *-g* option.
- S g t* For 1D input, the shade of grey and thickness of the line drawn for the IO function may be changed to *g* and *t* respectively. If either are to be changed, both must be specified when using this option.
- x f* The activation function for all non-input units in the network may be specified by *f*, which is one of the following strings:
- "sig" A sigmoid activation function. This is the default.
 - "lin" A linear activation function. (See chapter 1.)
 - "pct" A threshold activation function.
- X f* The activation function for the output unit only may be specified. *f* may be any of the functions as for *-x*, but with the following additional possibility:

“act” No activation function. The excitation of the output unit is treated as the output. This is useful for 1D input, when the output may be have a different range than [0, 1].

To illustrate some of these options, figure A.5 shows a colour rasterfile version of figure A.3 generated using style *smcNN*, complete with patterns and axes. Figure A.6 shows a colour rasterfile version of figure A.4, with the patterns (in a shade of grey) and axes marked. The command entered for figure A.5 was:

```
io -a 1.0 1.0 5 5 1 -g -p bp/p/2spir.pat -s smc10 bp/a/2-10-1.net
bp/w/2spir.wgt bp/r/2spir.rsd bp/io/2spir.ras
```

The command entered for figure A.6 was:

```
io -a 1.0 1.0 -g -p bp/p/sin.pat -P 100 -S 255 2 -X act bp/a/1-10-1.net
bp/w/sin.wgt bp/r/sin.rsd bp/io/sin.ras
```

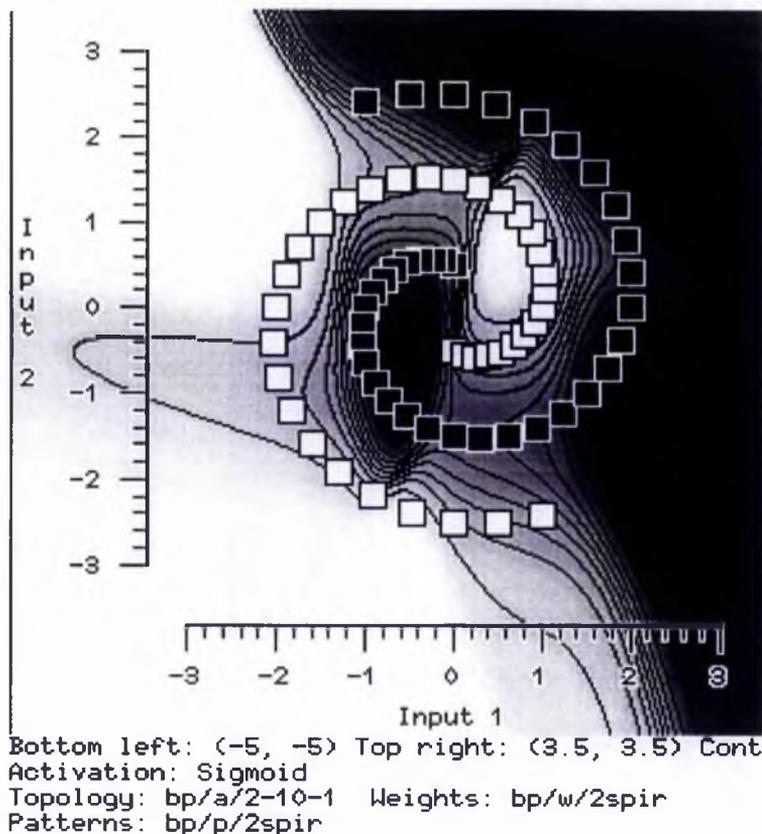


Figure A.5 — Grey level plot of solution to simple 2 spirals problem

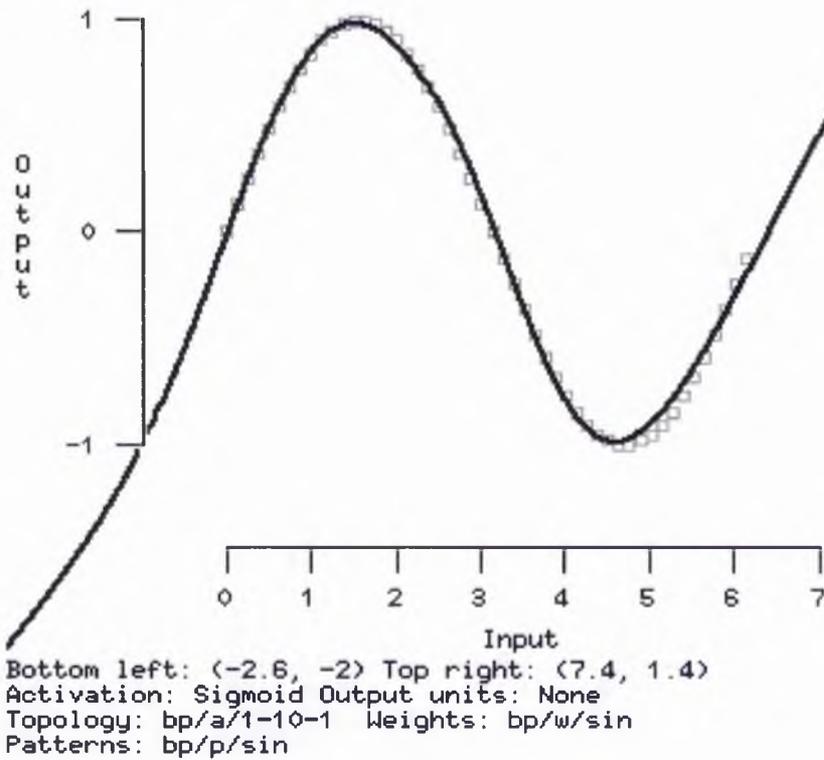


Figure A.6— Plot of fit to $\sin(x)$ with axes and patterns.

Bibliography

- Abu-Mostafa, Y. S. (1989) "The Vapnik-Chervonenkis Dimension: Information Versus Complexity in Learning" *Neural Computation* Vol. 1 No. 3 pp. 312-317
- Aha, D. W. (1992) "Tolerating Noisy, Irrelevant and Novel Attributes in Instance-Based Learning Algorithms" *International Journal of Man-Machine Studies* Vol. 36 No. 2 pp. 267-287
- Aha, D. W. and Kibler, D. (1989) "Noise-Tolerant Instance-Based Learning Algorithms" *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, Detroit*, pp. 794-799, Morgan Kaufmann
- Baba, N. (1989) "A New Approach for Finding the Global Minimum of Error Function of Neural Networks" *Neural Networks* Vol. 2 No. 5 pp. 367-373
- Baba, N., Mogami, Y., Kohzaki, M., Shiraishi, Y. and Yoshida, Y. (1994) "A Hybrid Algorithm for Finding the Global Minimum of Error Function of Neural Networks and Its Applications" *Neural Networks* Vol. 7 No. 8 pp. 1253-1265
- Bahrami, A. and Dagli, C. H. (1994) "Hybrid Intelligent Packing System (HIPS) Through Integration of Artificial Neural Networks, Artificial-Intelligence, and Mathematical-Programming" *Applied Intelligence* Vol. 4 No. 4 pp. 321-336
- Balchin, M. (1993) *Personal Communication*
- Barron, A. R. (1994) "Neural Networks: A Review from a Statistical Perspective — Comment" *Statistical Science* Vol. 9 No. 1 pp. 33-35
- Baum, E. B. (1990) "When are K-Nearest Neighbor and Back Propagation Accurate for Feasible Sized Sets of Examples" *Lecture Notes in Computer Science* Vol. 412 pp. 2-25
- Baum, E. B. and Haussler, D. (1989) "What Size Net Gives Valid Generalisation?" *Neural Computation* Vol. 1 pp. 151-160

Bibliography

- Baum, E. B. and Lang, K. J. (1991) "Constructing Hidden Units Using Examples and Queries" in Lippmann, R. P., Moody, J. E. and Touretzky, D. S. (Eds.) *Advances in Neural Information Processing Systems: Proceedings of the 1990 Conference* pp. 904-910, Morgan Kaufmann
- Beale, R. and Jackson, T. (1990) *Neural Computing: An Introduction*, Adam Hilger
- Boden, M. (1987) *Artificial Intelligence and Natural Man (Second Edition)* MIT Press
- Booker, L. B., Goldberg, D. E. and Holland, J. H. (1989) "Classifier Systems and Genetic Algorithms" *Artificial Intelligence* Vol. 40 No. 1-3 pp. 235-282
- Borowski, E. J. and Borwein, J. M. (1989) *Dictionary of Mathematics*, Collins
- Brady, M., Raghavan, R. and Slawny, J. (1988) "Gradient Descent Fails to Separate" *Proceedings of the IEEE International Conference on Neural Networks 1988* Vol. 1 pp. 649-656
- Brent, R. P. (1991) "Fast Training Algorithms for Multilayer Neural Nets" *IEEE Transactions on Neural Networks* Vol. 2 No. 3 pp. 346-354
- Bryson, A. E. and Ho, Y.-C. (1969) *Applied Optimal Control*, Blaisdell, NY
- Burton, R. M. and Faris, W. G. (1991) "Reliable Evaluation of Neural Networks" *Neural Networks* Vol. 4 No. 3 pp. 411-415
- Carbonell, J. G., Michalski, R. S. and Mitchell, T. M. (1983) "An Overview of Machine Learning" in Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (Eds.) *Machine Learning: An Artificial Intelligence Approach*, Tioga, Palo Alto
- Charniak, E. and McDermott, D. (1985) *Introduction to Artificial Intelligence*, Addison-Wesley
- Chauvin, Y. (1989) "A Back-Propagation Algorithm with Optimal Use of the Hidden Units" in Touretzky, D. (Ed.) *Advances in Neural Information Processing Systems: Proceedings of the 1988 Conference*, Morgan Kaufmann

Bibliography

Chauvin, Y. (1990) "Generalisation Performance of Overtrained Backpropagation" in Almedia, L. B. and Wellekens, C. J. (Eds.) *Proceedings of the EURASIP Workshop on Neural Networks*, pp. 46-55, Springer-Verlag

Cheng, B. and Titterton, D. M. (1994) "Neural Networks: A Review from a Statistical Perspective" *Statistical Science* Vol. 9 No. 1 pp. 2-30

Chester, D. L. (1990) "Why Two Hidden Layers are Better than One" *Proceedings of the International Joint Conference on Neural Networks, 15-19 January 1990 Washington DC*, Vol. 1 pp. 265-268

Church, A. (1936) "An Unsolvable Problem of Elementary Number Theory" *American Journal of Mathematics* Vol. 48 pp. 345-363

Coombs, M. J., Pfeiffer, H. D. and Hartley, R. T. (1992) "e-MGR: An Architecture for Symbolic Plasticity" *International Journal of Man-Machine Studies* Vol. 36 pp. 247-263

Cosnard, M., Koiran, P. and Paugam-Moisy, H. (1993) "A Step Towards the Frontier Between One-Hidden-Layer and Two-Hidden-Layer Neural Networks" *Proceedings of the International Joint Conference on Neural Networks 25-29 October 1993 Nagoya, Japan*, Vol. 3 pp. 2292-2295

Cybenko, G. (1989) "Approximation by Superposition of a Sigmoidal Function" *Mathematics of Control, Signals and Systems* Vol. 2 No. 4 pp. 303-314

Dasgupta, B. and Schnitger, G. (1993) "The Power of Approximating: A Comparison of Activation Functions" Hanson, S. J., Cowan, J. D. and Giles, C. L. (Eds.) *Advances in Neural Information Processing Systems: Proceedings of the 1992 Conference* pp. 615-622, Morgan Kaufmann

Denker, J., Schwarz, D., Wittner, B., Solla, S., Howard, R., Jackel, L. and Hopfield, J. (1987) "Large Automatic Learning, Rule Extraction, and Generalization" *Complex Systems* Vol. 1 pp. 877-922

Draghici, S. (1995) "Using Constraints to Improve Generalisation and Training of Feed-Forward Neural Networks: Constraint Based Decomposition and Complex Backpropagation" *PhD Thesis, Department of Mathematical and Computational Sciences, University of St. Andrews*

Bibliography

Drucker, H. and Le Cun, Y. (1991) "Double Backpropagation Increasing Generalisation Performance" *Proceedings of the International Joint Conference on Neural Networks 1991, Seattle*, Vol. 2 pp. 145-150

Duda, R. O. and Shortliffe, E. H. (1983) "Expert Systems Research" *Science* Vol. 220 No. 4594 pp. 261-268

Duda, R. O., Gaschnig, J. G. and Hart, P. E. (1980) "Model Design in the PROSPECTOR Consultant System for Mineral Exploration" in Michie, D. (Ed.) *Expert Systems in the Microelectronic Age*, pp. 153-167, Edinburgh University Press

Dyer, M. G. and Lee, G. (1995) "Goal Plan Analysis Via Distributed Semantic Representations in a Connectionist System" *Applied Intelligence* Vol. 5 No. 2 pp. 165-197

Eisenstein, E. and Kanter, I. (1993) "Generalisation Performance of Complex Adaptive Tasks" *Physical Review Letters* Vol. 70 No. 23 pp. 3667-3670

Fahlman, S. E. and Lebiere, C. (1991) "The Cascade-Correlation Learning Architecture" *Technical Report CMU-CS-90-100, August 1991, Carnegie-Mellon University*

Fang, H. L., Ross, P. and Corne, D. (1993) "A Promising Genetic Algorithm Approach to Job-Shop Scheduling, Rescheduling, and Open-Shop Scheduling Problems" in Forrest, S. (Ed.) *Proceedings of the Fifth International Conference on Genetic Algorithms, San Mateo*, pp. 375-382, Morgan Kaufmann

Fernandez, A. (1994) *Personal Communication*

Fogel, D. B., Fogel, L. J. and Porto, V. W. (1990) "Evolving Neural Networks" *Biological Cybernetics* Vol. 63 No. 6 pp. 487-493

Fu, L. M. (1994) "Rule Generation from Neural Networks" *IEEE Transactions on Systems, Man and Cybernetics* Vol. 24 No. 8 pp. 1114-1124

Fujita, O. (1992) "Optimization of the Hidden Unit Function in Feedforward Neural Networks" *Neural Networks* Vol. 5 No. 5 pp. 755-764

Bibliography

- Funahashi, K. (1989) "On the Approximate Realisation of Continuous Mappings by Neural Networks" *Neural Networks* Vol. 2 No. 3 pp. 183-192
- Geman, S., Bienenstock, E. and Doursat, R. (1992) "Neural Networks and the Bias/Variance Dilemma" *Neural Computation* Vol. 4 No. 1 pp. 1-58
- Gibson, G. J. (1992) "Constructing Functions Using Multilayer Perceptrons — Towards a Theory of Complexity" *Journal of Systems Engineering* Vol. 2 pp. 263-271
- Gibson, G. J. (1993) "A Combinatorial Approach to Understanding Perceptron Capabilities" *IEEE Transactions on Neural Networks* Vol. 4 No. 6 pp. 989-992
- Gibson, G. J. (1994) "Some Results on the Exact Realisation of Decision Regions Using Feed-Forward Networks with a Single Hidden Layer" *Proceedings of the IEEE International Conference On Neural Networks 28 June -2 July 1994, Orlando, Florida* Vol. 2 pp. 912-917
- Gibson, G. J. and Cowan, C. F. N. (1990) "On the Decision Regions of Multilayer Perceptrons" *Proceedings of the IEEE* Vol. 78 No. 10 pp. 1590-1594
- Goldberg, D. E. (1989) *Genetic Algorithms*, Addison-Wesley
- Guo, H. and Gelfand, S. B. (1991) "Analysis of Gradient Descent Learning Algorithms for Multilayer Feedforward Neural Networks" *IEEE Transactions on Circuits and Systems* Vol. 38 No. 8 pp. 883-894
- Hamming, R. W. (1986) *Coding and Information Theory: Second Edition*, Prentice-Hall
- Hanson, S. J. and Burr, D. J. (1990) "What Connectionist Models Learn: Learning and Representation in Connectionist Networks" *Behavioural and Brain Sciences* Vol. 13 pp. 471-518
- Hanson, S. J. and Pratt, L. Y. (1989) "Some Comparisons of Constraints for Minimal Network Construction with Backpropagation" in Touretzky, D. (Ed.) *Advances in Neural Information Processing: Proceedings of the 1988 Conference*, Morgan Kaufmann

Bibliography

- Hartigan, J. A. (1975) *Clustering Algorithms*, John Wiley and Sons
- Harvey, I. (1994) *Personal Communication*
- Hasegawa, A. Matoba, O., Itoh, K. and Ichioka, Y. (1992) "Learning Generalization by Validation Set" *Japanese Journal of Applied Physics* Vol. 31 Part 1 No. 8 pp. 2459-2462
- Hausler, D., Kearns, M. and Schapire, R. E. (1994) "Bounds on the Sample Complexity of Bayesian Learning Using Information Theory and the VC Dimension" *Machine Learning* Vol. 14 No. 1 pp. 83-113
- Hertz, J., Krogh, A. and Palmer, R. G. (1991) *Introduction to the Theory of Neural Computation*, Addison-Wesley
- Hinton, G. E. (1989) "Connectionist Learning Procedures" *Artificial Intelligence* Vol. 40 pp. 185-234
- Hirose, Y., Yamashita, K. and Hijiya, S. (1991) "Back-Propagation Algorithm which Varies the Number of Hidden Units" *Neural Networks* Vol. 4 No. 1 pp. 61-66
- Hirsh, H. (1990) "Learning from Data with Bounded Inconsistency" *Proceedings of the Seventh International Conference on Machine Learning* pp. 32-39
- Hofstadter, D. R. (1980) *Gödel, Escher, Bach: An Eternal Golden Braid*, Penguin
- Holland, J. H. (1975) *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor
- Holland, J. H. (1986) "Escaping Brittleness: The Possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems" in Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (Eds.) *Machine Learning: An Artificial Intelligence Approach Volume II*, Morgan Kaufmann
- Hopfield, J. J. (1982) "Neural Networks and Physical Systems with Emergent Collective Computational Abilities" *Proceedings of the National Academy of Sciences USA*, Vol. 79 pp. 2554-2558

Bibliography

- Hornik, K. Stinchcombe, M. and White, H. (1989) "Multilayer Feedforward Networks are Universal Approximators" *Neural Networks* Vol. 2 No. 5 359-366.
- Hrycej, T. (1990) "A Modular Architecture for Efficient Learning" *Proceedings of the IEEE INNS International Joint Conference on Neural Networks 1990, San Diego* Vol. 1 pp. 557-562
- Huang, W. M. and Lippmann, R. P. (1988) "Neural Net and Traditional Classifiers" in Anderson, D. Z. (Ed.) *Neural Information Processing Systems*, pp. 387-396, American Institute of Physics.
- Ishikawa, M. (1989) "A Structural Learning Algorithm with Forgetting of Weight Link Weights" *Proceedings of the International Joint Conference on Neural Networks 1989 Washington DC*, Vol. 2
- James, R. C. and James, G. (1992) *Mathematics Dictionary (Fifth Edition)*, Van Nostrand Reinhold
- Jean, J. S. N. and Wang, J. (1994) "Weight Smoothing to Improve Generalisation" *IEEE Transactions on Neural Networks* Vol. 5 No. 5 pp. 752-763
- Joerding, W. H. and Meador, J. L. (1991) "Encoding A-Priori Information in Feedforward Networks" *Neural Networks* Vol. 4 No. 6 pp. 847-856
- Kameyama, K. and Kosugi, Y. (1991) "Neural Network Pruning by Fusing Hidden Layer Units" *IEICE Transactions on Communications, Electronics, Information and Systems* Vol. 74 No. 12 pp. 4198-4204
- Kanaya, F. and Miyake, S. (1991) "Bayes Statistical Behavior and Valid Generalization of Pattern Classifying Neural Networks" *IEEE Transactions on Neural Networks* Vol. 2 No. 4 pp. 471-475
- Khanna, T. (1990) *Foundations of Neural Networks*, Addison-Wesley
- Kitano, H. (1990) "Empirical Studies on the Speed of Convergence of Neural Network Training Using Genetic Algorithms" *AAAI 1990* pp. 789-795, MIT Press

Bibliography

- Kolen, J. F. and Pollack, J. B. (1991) "Back Propagation is Sensitive to Initial Conditions" in Lippmann, R. P., Moody, J. E. and Touretzky, D. S. (Eds.) *Advances in Neural Information Processing Systems: Proceedings of the 1990 Conference* pp. 860-867
- Kurkova, V. (1992) "Kolmogorov's Theorem and Multilayer Neural Networks" *Neural Networks* Vol. 5 No. 3 pp. 501-506
- Lang, K. J. and Witbrock, M. J. (1988) "Learning to Tell Two Spirals Apart" in Touretzky, D. (Ed.) *Proceedings of the 1988 Connectionist Models Summer School, San Mateo, California*, pp. 52-59, Morgan-Kaufmann
- Lang, K. J., Waibel, A. H. and Hinton, G. E. (1990) "A Time-Delay Neural Network Architecture for Isolated Word Recognition" *Neural Networks* Vol. 3 pp. 23-43
- Lansley, S. and Clark, A. (1993) "The Minimum Topology Finder" *SH Project, Department of Mathematical and Computational Sciences, University of St. Andrews*
- Lee, T. and Chung, F. L. (1990) "Determining the Number of Hidden Nodes by Progressive Training" *Electronics Letters* Vol. 26 No. 16 pp. 1318-1320
- Levin, E., Tishby, N. and Solla, S. A. (1990) "A Statistical Approach to Learning and Generalization in Layered Neural Networks" *Proceedings of the IEEE* Vol. 78 No. 10 pp. 1568-1574
- Li, L. K. (1991) "On Computing Decision Regions with Neural Nets" *Journal of Computer and System Sciences* Vol. 43 pp. 509-512
- Lippmann, R. P. (1987) "An Introduction to Computing with Neural Nets" *IEEE ASSP Magazine*, April 1987, pp. 4-22
- Lippmann, R. P. (1989) "Pattern-Classification Using Neural Networks" *IEEE Communications Magazine* Vol. 27 No. 11 pp. 47-64
- MacKay, D. J. C. (1992a) "Bayesian Interpolation" *Neural Computation* Vol. 4 No. 3 pp. 415-447

Bibliography

- MacKay, D. J. C. (1992b) "A Practical Bayesian Framework for Backpropagation Networks" *Neural Computation* Vol. 4 No. 3 pp. 448-472
- MacKay, D. J. C. (1992c) "Information-Based Objective Functions for Active Data Selection" *Neural Computation* Vol. 4 No. 4 pp. 590-604
- MacKay, D. J. C. (1992d) "The Evidence Framework Applied to Classification Networks" *Neural Computation* Vol. 4 No. 5 pp. 720-736
- Makhoul, J., El-Jaroudi, A. and Schwartz, R. (1989) "Formation of Disconnected Decision Regions with a Single Hidden Layer" *Proceedings of the International Joint Conference on Neural Networks, June 1989, Washington DC* Vol. 1 pp. 455-460
- Mato, G. and Parga, N. (1992) "Generalization Properties of Multilayered Neural Networks" *Journal of Physics A: Mathematical and General* Vol. 25 No. 19 pp. 5047-5054
- McClelland, J. L. and Rumelhart, D. E. (1988) *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs and Exercises*, MIT Press
- McClelland, J. L., Rumelhart, D. E. and Hinton, G. E. (1986) "The Appeal of Parallel Distributed Processing" in Rumelhart, D. E., McClelland, J. L. and the PDP Research Group (Eds.) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* Vol. 1 Ch. 1, MIT Press
- McClintock, D. and Fitter, R. S. R. (1956) *The Pocket Guide to Wild Flowers*, Collins
- McCulloch, W. S. and Pitts, W. (1943) "A Logical Calculus of Ideas Immanent in Nervous Activity" *Bulletin of Mathematical Biophysics* Vol. 5 pp. 115-133
- Mel, B. W. and Omohundro, S. M. (1991) "How Receptive Field Parameters Affect Neural Learning" in Lippmann, R. P., Moody, J. E. and Touretzky, D. S. (Eds.) *Advances in Neural Information Processing Systems: Proceedings of the 1990 Conference* pp. 757-763

Bibliography

- Mezard, M. and Nadal, J. P. (1989) "Learning in Feedforward Layered Networks: The Tiling Algorithm" *Journal of Physics A: Mathematical and General* Vol. 22 pp. 2191-2203
- Minor, J. M. (1993) "Parity with Two Layer Feedforward Nets" *Neural Networks* Vol. 6 pp. 705-707
- Minsky, M. L. and Papert, S. A. (1969) *Perceptrons*, MIT Press
- Minsky, M. L. and Papert, S. A. (1988) *Perceptrons: An Introduction to Computational Geometry (Expanded Edition)* MIT Press
- Mirchandani, G. and Cao, W. (1989) "On Hidden Nodes For Neural Nets" *IEEE Transactions on Circuits and Systems* Vol. 36 No. 5 pp. 661-664
- Mitchell, T. M. (1977) "Version Spaces: A Candidate Elimination Approach to Rule Learning" *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Vol. 1 pp. 305-310, MIT Press
- Mitchell, T. M. (1978) "Version Spaces: An Approach to Concept Learning" *PhD Thesis STAN-CS-78-711, Stanford University, December 1978*
- Mitchell, T. M. (1980) "The Need for Biases in Learning Generalizations" *Technical Report CBM-TR-117 Computer Science Department, Rutgers University, May 1980*
- Mitchell, T. M. (1982) "Generalization as Search" *Artificial Intelligence* Vol. 18 pp. 203-226
- Mozer, M. C. and Smolensky, P. (1989) "Skeletonisation: A Technique for Trimming the Fat from a Network Via Relevance Assessment" in Touretzky, D. (Ed.) *Advances In Neural Information Processing Systems: Proceedings of the 1988 Conference* pp. 349-357, Morgan Kaufmann
- Musavi, M. T., Ahmed, W., Chan, K. H., Faris, K. B. and Hummels, D. M. (1992) "On the Training of Radial Basis Function Classifiers" *Neural Networks* Vol. 5 No. 4 pp. 595-603
- Newell, A. and Simon, H. A. (1976) "Computer Science as Empirical Inquiry: Symbols and Search" *Communications of the ACM* Vol. 19 No. 3

Bibliography

- Nilsson, N. J. (1965) *Learning Machines: Foundations of Trainable Pattern-Classifying Systems*, McGraw-Hill
- Ochiai, K., Toda, N. and Usui, S. (1994) "Kick-Out Learning Algorithm to Reduce the Oscillation of Weights" *Neural Networks* Vol. 7 No. 5 pp. 797-807
- Opper, M. and Haussler, D. (1991) "Generalization Performance of Bayes Optimal Classification Algorithm for Learning a Perceptron" *Physical Review Letters* Vol. 66 No. 20 pp. 2677-2680
- Parker, D. B. (1985) "Learning Logic" *Technical Report TR-47, Centre for Computational Research in Economics and Management Science, MIT*
- Parrondo, J. M. R. and Van den Broeck, C. (1993) "Vapnik-Chervonenkis Bounds for Generalization" *Journal of Physics A: Mathematical and General* Vol. 26 No. 9 pp. 2211-2223
- Pople, H., Myers, J. and Miller, R. (1975) "DIALOG: A Model of Diagnostic Logic for Internal Medicine" *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, pp. 848-855
- Preparata, F. P. and Shamos, M. I. (1985) *Computational Geometry: An Introduction*, Springer-Verlag
- Press, W. H., Flannery, B. P., Teukolsky, S. A. and Vetterling, W. T. (1988) *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge University Press
- Radcliffe, N. J. (1993) "Genetic Set Recombination and its Application to Neural Network Topology Optimisation" *Neural Computing and Applications* Vol. 1 pp. 67-90
- Rich, E. (1983) *Artificial Intelligence*, McGraw-Hill
- Rich, E. and Knight, K. (1991) *Artificial Intelligence (Second Edition)*, McGraw-Hill
- Richard, M. D. and Lippmann, R. P. (1991) "Neural Network Classifiers Estimate Bayesian A Posteriori Probabilities" *Neural Computation*, Vol. 3 pp. 461-483

Bibliography

- Romaniuk, S. G. and Hall, L. O. (1993) "SC-Net — A Hybrid Connectionist, Symbolic System" *Information Sciences* Vol. 71 No. 3 pp. 223-268
- Rosenblatt, F. (1958) "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain" *Psychological Review* Vol. 65 No. 6 pp. 386-408
- Rosenblatt, F. (1962) *Perceptrons and the Theory of Brain Mechanisms* Spartan Books, Washington DC
- Roy, A., Kim, L. S. and Mukhopadhyay, S. (1993) "A Polynomial Time Algorithm for the Construction and Training of a Class of Multilayer Perceptrons" *Neural Networks* Vol. 6 No. 4 pp. 535-545
- Rumelhart, D. E. and Zipser, D. (1986) "Feature Discovery by Competitive Learning" in Rumelhart, D. E., McClelland, J. L. and the PDP Research Group (Eds.) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* Vol. 1 Ch. 5, MIT Press
- Rumelhart, D. E., Hinton, G. E. and Williams, R. J. (1986) "Learning Internal Representations by Error Propagation" in Rumelhart, D. E., McClelland, J. L. and the PDP Research Group (Eds.) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* Vol. 1 Ch. 8, MIT Press
- Schmidt, W. A. C. and Davis, J. P. (1994) "Generation of Neural Network Decision Surfaces: A Pattern Classification Example" *Optical Engineering* Vol. 33 No. 10 pp. 3388-3397
- Schwartz, D. B., Samalam, V. K., Solla, S. A. and Denker, J. S. (1990) "Exhaustive Learning" *Neural Computation* Vol. 2 pp. 371-382
- Shavlik, J. W. (1994) "Combining Symbolic and Neural Learning" *Machine Learning* Vol. 14 No. 3 pp. 321-331
- Shonkwiler, R. (1993) "Separating the Vertices of N-Cubes by Hyperplanes and its Application to Artificial Neural Networks" *IEEE Transactions on Neural Networks* Vol. 4 No. 2 pp. 343-347

Bibliography

- Shortliffe, E. H. (1976) *Computer-Based Medical Consultations: MYCIN*, American Elsevier
- Sietsma, J. and Dow, R. J. F. (1991) "Creating Artificial Neural Networks That Generalize" *Neural Networks* Vol. 4 No. 1 pp. 67-79
- Sikora, R. (1992) "Learning Control Strategies for Chemical Processes — A Distributed Approach" *IEEE Expert* Vol. 7 No. 3 pp. 35-43
- Smalz, R. and Conrad, M. (1994) "Combining Evolution with Credit Apportionment: A New Learning Algorithm for Neural Nets" *Neural Networks* Vol. 7 No. 2 pp. 341-352
- Smieja, F. J. (1993) "Neural Network Constructive Algorithms — Trading Generalization for Learning Efficiency" *Circuits Systems and Signal Processing* Vol. 12 No. 2 pp. 331-374
- Smieja, F. J. and Muhlenbein, H. (1990) "The Geometry of Multi-Layer Perceptron Solutions" *Parallel Computing* Vol. 14 pp. 261-275
- Sontag, E. D. (1989) "Sigmoids Distinguish More Efficiently than Heavisides" *Neural Computation* Vol. 1 pp. 470-472
- Sontag, E. D. and Sussmann, H. J. (1988) "Back-Propagation Separates When Perceptrons Do" *Technical Report SYCON-88-12, Rutgers Center for Systems and Control, December 1988*
- Sontag, E. D. and Sussmann, H. J. (1989) "Backpropagation Separates When Perceptrons Do" *Proceedings of the International Joint Conference on Neural Networks, June 1989, Washington DC* Vol. 1 pp. 639-642
- Storer, J (1994) "ISLAND: Interactive Single-layer Linear Activation Network Display" *SH Project, Department of Mathematical and Computational Sciences, University of St. Andrews*
- Stork, D. G. and Allen, J. D. (1992) "How to Solve the N-Bit Parity Problem with Two Hidden Units" *Neural Networks* Vol. 5 pp. 923-926
- Sun, R. and Bookman, L. (1993) "How Do Symbols and Networks Fit Together: A Report from the Workshop on Integrating Neural and Symbolic Processes" *AI Magazine* Vol. 14 No. 2 pp. 20-23

Bibliography

- Sussmann, H. J. (1992) "Uniqueness of the Weights for Minimal Feedforward Nets with a Given Input-Output Map" *Neural Networks* Vol. 5 No. 4 pp. 589-593
- Telfer, B. A. and Szu, H. H. (1994) "Energy Functions for Minimizing Misclassification Error with Minimum-Complexity Networks" *Neural Networks* Vol. 7 No. 5 pp. 809-818
- Turing, A. M. (1936) "On Computable Numbers with an Application to the Entscheidungsproblem" *Proceedings of the London Mathematical Society* Vol. 42 pp. 230-265
- Utgoff, P. E. (1986) "Shift of Bias for Inductive Concept Learning" in Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (Eds.) *Machine Learning: An Artificial Intelligence Approach Volume II*, Morgan Kaufmann
- Vapnik, V. N. Chervonenkis, A. Y. (1971) "On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities" *Theory of Probability and its Applications* Vol. 16 pp. 264-280
- Voronoi, G. (1908) "Nouvelles Applications des Parametres Continus à la Theorie des Formes Quadratiques. Deuxième Mémoire: Recherches sur les Paralléloèdres Primitifs" *J. Reine Angew. Math.* Vol. 134, pp. 198-287
- Watkin, T. L. H. Rau, A. and Biehl, M. (1993) "The Statistical Mechanics of Learning a Rule" *Reviews of Modern Physics* Vol. 65 No. 2 pp. 499-555
- Weigend, A. S., Rumelhart, D. E. and Huberman, B. A. (1991a) "Back-Propagation, Weight Elimination, and Time Series Prediction" in Touretzky, D. S. (Ed.) *Proceedings of the 1990 Summer School* pp. 105-116, Morgan Kaufmann
- Weigend, A. S., Rumelhart, D. E. and Huberman, B. A. (1991b) "Generalisation by Weight-Elimination with Application to Forecasting" in Lippmann, R. P., Moody, J. E. and Touretzky, D. S. (Eds.) *Advances in Neural Information Processing Systems: Proceedings of the 1990 Conference* pp. 875-882, Morgan Kaufmann

Bibliography

- Weir, M. K. (1991) "A Method for Self-Determination of Adaptive Learning Rates in Back Propagation" *Neural Networks* 1991 Vol. 4 pp. 371-379
- Weir, M. K. (1993) *Personal Communication*
- Weir, M. K. and Polhill, J. G. (1993) "A Neural Implementation of Mitchell's Concept and Version Spaces Technique" *Technical Report CS/93/12, Department of Mathematical and Computational Sciences, University of St. Andrews*
- Weir, M. K. and Polhill, J. G. (1994a) "Implementing Mitchell's Concept and Version Spaces Technique in Neural Networks Using Weight Space Angle as the Partial Order Analogue" *Technical Report CS/94/10 Department of Mathematical and Computational Sciences, University of St. Andrews*
- Weir, M. K. and Polhill, J. G. (1994b) "Bidirectional Convergence: A Cognitive Approach to Generalisation" *Proceedings of the IEEE International Conference on Neural Networks, 28 June-2 July 1994, Orlando, Florida, Vol. IV* pp. 2285-2290
- Weir, M. K. and Polhill, J. G. (1995) "Neural Bidirectional Convergence: A Method for Concept Learning in Neural Networks and Symbolic AI" *Proceedings of the International Conference on Brain Processes, Theories and Models, Canary Islands, To Appear*
- Werbos, P. J. (1974) "Beyond Regression: New Tools for Prediction and Analysis in the Behavioural Sciences" *PhD Thesis, Harvard University*
- Wieland, A. and Leighton, R. (1987) "Geometric Analysis of Neural Network Capabilities" *IEEE First Initial Conference on Neural Networks 1987* pp. 111-385
- Winston, P. H. (1975) "Learning Structural Descriptions from Examples" in Winston, P. H. (Ed.) *The Psychology of Computer Vision*, Ch. 5, McGraw-Hill
- Wolpert, D. H. (1989) "A Benchmark for How Well Neural Nets Generalize" *Biological Cybernetics* Vol. 61 No. 4 pp. 303-313

Bibliography

- Wolpert, D. H. (1990) "Constructing A Generalizer Superior to NetTalk Via a Mathematical-Theory of Generalization" *Neural Networks* Vol. 3 No. 4 pp. 445-452
- Wynne-Jones, M. (1993) "Node-Splitting: A Constructive Algorithm for Feed-Forward Neural Networks" *Neural Computing and Applications* Vol. 1 pp. 17-22
- Xu, L., Klasa, S. and Yuille, A. (1992) "Recent Advances on Techniques of Static Feedforward Network Training Techniques with Supervised Learning" *International Journal of Neural Systems* Vol. 3 pp. 253-290
- Zadeh, L. A. (1973) "Outline of a New Approach to the Analysis of Complex Systems and Design Processes" *IEEE Transactions on Systems, Man and Cybernetics*, Vol. SMC-3, pp. 28-44
- Zupan, J. and Gasteiger, J. (1991) "Neural Networks — A New Method for Solving Chemical Problems or Just a Passing Phase" *Analytica Chimica Acta* Vol. 248 No. 1 pp. 1-30