Inductive Machine Learning Bias in Knowledge-Based Neurocomputing

Sean Snyders

Thesis presented in partial fulfilment
of the requirements for the degree of
Master of Science
at the University of Stellenbosch

Supervisor: Prof. Dr. Christian W. Omlin

April 2003

Declaration

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

Signature: Date: 10 March 2003

Abstract

The integration of symbolic knowledge with artificial neural networks is becoming an increasingly popular paradigm for solving real-world problems. This paradigm named *knowledge-based neurocomputing*, provides means for using prior knowledge to determine the network architecture, to program a subset of weights to induce a learning bias which guides network training, and to extract refined knowledge from trained neural networks. The role of neural networks then becomes that of knowledge refinement. It thus provides a methodology for dealing with uncertainty in the initial domain theory.

In this thesis, we address several advantages of this paradigm and propose a solution for the open question of determining the strength of this learning, or inductive, bias. We develop a heuristic for determining the strength of the inductive bias that takes the network architecture, the prior knowledge, the learning method, and the training data into consideration.

We apply this heuristic to well-known synthetic problems as well as published difficult real-world problems in the domain of molecular biology and medical diagnoses. We found that, not only do the networks trained with this adaptive inductive bias show superior performance over networks trained with the standard method of determining the strength of the inductive bias, but that the extracted refined knowledge from these trained networks deliver more concise and accurate domain theories.

Opsomming

Die integrasie van simboliese kennis met kunsmatige neurale netwerke word 'n toenemende gewilde paradigma om reëlewêreldse probleme op te los. Hierdie paradigma genoem, *kennis-gebaseerde neurokomputasie*, verskaf die vermoë om vooraf kennis te gebruik om die netwerkargitektuur te bepaal, om a subversameling van gewigte te programeer om 'n leersydigheid te induseer wat netwerkopleiding lei, en om verfynde kennis van geleerde netwerke te kan ontsluit. Die rol van neurale netwerke word dan die van kennisverfyning. Dit verskaf dus 'n metodologie vir die behandeling van onsekerheid in die aanvangsdomeinteorie.

In hierdie tesis adresseer ons verskeie voordele wat bevat is in hierdie paradigma en stel ons 'n oplossing voor vir die oop vraag om die gewig van hierdie leer-, of induktiewe sydigheid te bepaal. Ons ontwikkel 'n heuristiek vir die bepaling van die induktiewe sydigheid wat die netwerkargitektuur, die aanvangskennis, die leermetode, en die data vir die leerproses in ag neem.

Ons pas hierdie heuristiek toe op bekende sintetiese probleme so wel as op gepubliseerde moeilike reëlewêreldse probleme in die gebied van molekulêre biologie en mediese diagnostiek. Ons bevind dat, nie alleenlik vertoon die netwerke wat geleer is met die adaptiewe induktiewe sydigheid superieure verrigting bo die netwerke wat geleer is met die standaardmetode om die gewig van die induktiewe sydigheid te bepaal nie, maar ook dat die verfynde kennis wat ontsluit is uit hierdie geleerde netwerke meer bondige en akkurate domeinteorië lewer.

Acknowledgements

I am in debt to many people for their support, help, and understanding for guiding my efforts to the completion of this work.

I want to thank my supervisor, Prof. Christian W. Omlin, for unlimited intuition and excitement in attacking all aspects of this work. I am greatly in debt for his support in wanting to immerse his students in the research environment. Thank you to Reg Dodds for helpful tips on document preparation. I would like to thank the following people for helpful discussions: Dr. Margarita Sordo Sanchéz, Anwar Vahed, and to my friend Wian de Jongh for amazing cross-discipline understanding and knowledge[1].

Then, boundless appreciation for all my friends and family, keeping me *un*sane within the *in*saneness of this world. And, finally, but definitely not in the least, to my wife Gayle, for unnerving compassion and sacrifices she had to make for my passion of the sciences of computers.

[1]This always occurred over lunch at our usual place.

Articles

Parts of this thesis are contained in the following articles:

1. S. Snyders and C.W. Omlin. What Inductive Bias Gives Good Neural Network Training Performance? In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks*, volume 3, pages 445–450, 2000.

2. S. Snyders and C.W. Omlin. Rule Extraction from Knowledge-based Neural Networks with Adaptive Inductive Bias. In *Proceedings of the 8th International Conference on Neural Information Processing*, volume 1, pages 143–148, 2001.

3. S. Snyders and C.W. Omlin. Inductive Bias in Recurrent Neural Networks. In *Proceedings of the 6th International Work-Conference on Artificial and Natural Neural Networks (LCNS 2084)*, volume I, pages 339–346, 2001.

4. S. Snyders and C.W. Omlin. Inductive Bias Strength in Knowledge-Based Neural Networks: Application to Magnetic Resonance Spectroscopy of Breast Tissues. *Artificial Intelligence in Medicine*, expected to appear 2003. To be published in the Special Issue on Knowledge-Based Neurocomputing.

5. S. Snyders and C.W. Omlin. Inductive Bias in Knowledge-Based Neural Networks. *Neural Computation*. To be submitted for review, 2003.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Neural networks which can solve difficult non-linear *and* every day problems have established themselves as standard tools in the toolbox of machine learning methods. Straight-forward connectionist models have contributed to many successes; however, researchers have found that to model non-trivial learning problems, some prior structure is sorely needed. Parallel to the pure connectionist viewpoint on solving learning problems, symbolic processing of information has also attracted attention. This was mainly due to the meaningful representation of the learning problem and the knowledge that could be gained transparently from the system after learning.

The combination of these two paradigms has featured on many occasions in the past decade. This combination exploits the advantages of each paradigm and avoids their respective weaknesses. Effectively combining these two paradigms is an open research area.

This thesis explores this combination and focuses on specific architectures proposed in the literature. We propose a novel method for determining a good combination which biases the hybrid architecture for increased performance above the current standard method.

1.2 A Historical Perspective

Symbolic artificial intelligence uses symbol manipulation and formal languages in an attempt to model intelligence. This basis of representing and manipulating symbolic knowledge has always been a focal point of artificial intelligence; it led to the development of expert systems and knowledge-based systems. The advantages of these systems are that they can represent human comprehensible knowledge and are able to reason with it.

Neural networks, on the other hand, do not offer this explicit symbol manipulation; they have the appeal of acquiring knowledge from learning from examples and have demonstrated good results, e.g. in the areas of computer vision and speech recognition. The internal knowledge representation of neural networks is incomprehensible to humans and thus the knowledge is not easily manipulated.

Boundaries between these two paradigms are becoming less distinct as researchers are exploring the combination of symbolic and connectionist approaches.

1.3 Neurocomputing: Opportunities and Challenges

The capability of neural networks to adapt has attributed to long-standing successes. The strength of biologically inspired computing paradigms such as neural networks lies in their capability to adapt to model a certain situation. Although traditional multilayer perceptrons are able to solve a variety of tasks, new and innovative networks are needed to solve more complex problems. Different approaches to neural network design and learning algorithms exploiting the benefits of new developments are of great importance.

Current systems provide opportunities for application to a large variety of real-world problems. This will lead to better understanding of the benefits of intelligent computing and create wider user acceptance. The main goal of learning systems is to acquire, manipulate and effectively use knowledge. Thus, challenges include better knowledge representation, manipulation and transfer to other similar learning problems.

1.4 Problem Statement

Prior information is needed for the successful modelling of difficult non-trivial problems [54]. There exist various means and architectures to achieve this combination in the neurocomputing paradigm for feedforward and recurrent neural networks.

The quality of prior information for many real-world problems is suspect. In order to combine this partially correct knowledge with a neurocomputing architecture, a measure of the *quality* of the prior knowledge is needed:

1. How good is the prior knowledge actually?
2. Does the prior knowledge explain the given data sufficiently well?
3. Will the prior knowledge improve or impede the learning process?

Combining the prior information with the neurocomputing architecture introduces a bias that can be adjusted for increased performance. Current existing architectures indiscriminately choose a fixed standard bias for all applications. Determining this bias for effective use of the prior information, for finding a good solution, is the central theme of this thesis.

1.5 Premises

We conducted our research in the hybrid symbolic/neural learning paradigm. We assert that inductive bias—and machine learning bias in general—is an important aspect of machine learning. Training neural networks with bias is typically achieved through the prestructuring of a neural network architecture with prior knowledge. An optimal solution for a specific set of parameters for a specific problem generally exists; whether such a solution can be found through training depends on the chosen parameters. This optimal solution can be approximated through the use of some heuristic which delivers a solution of acceptable quality. This heuristic for determining the inductive bias, and thus the method for measuring the quality of or confidence in the prior knowledge, is intimately tied to the training method.

1.6 Technical Objectives

Highlighting important issues in the combination of symbolic and neural learning, we address the importance of bias in knowledge-based neurocomputing and consequently have the following objectives:

1. To develop a measure of confidence in the prior knowledge that can be computed, *prior* to training, that takes the following factors into consideration:
    - the prior symbolic knowledge,
    - the training data,
    - the training method, and
    - the network architecture.
2. To use this measure to determine a good combination of the prior symbolic knowledge and the neural network architecture.
3. To show the impact of this choice of combination, as compared to the current standard method, on
    - the training time,
    - the generalisation performance, and
    - the quality of the extracted refined knowledge.
4. Apply this heuristic to synthetic and difficult real-world problems.

1.7 Methodology

To gain background knowledge on the knowledge-based neurocomputing paradigm, we first sought an understanding of the importance of using knowledge to prestructure neural network architectures. We thus investigated the role of bias and its adjustment in other machine learning paradigms.

We then identified the predominant neural network architecture used for the insertion of prior knowledge. Knowledge-based Artificial Neural Networks (KBANNs) [91] were chosen as our test-bed for feedforward neural networks and the knowledge insertion method; for recurrent neural networks, we chose the method proposed in [58]. Although we chose these two specific architectures, we still endeavoured to develop a measure of confidence in the prior knowledge that is independent of the specific architecture[1], but that is tied to the learning method. We decided to use the error function as a guide to determine this measure. The error function intrinsically takes the different aspects into account that are mentioned in the first objective. The prior knowledge is taken into account through the insertion methods of the respective architectures examined. These insertion methods typically structure the physical network prior to training.

We analysed the confidence measure and gave justifications for the use of this measure in order to achieve a good combination of symbolic and neural learning. The measure is based on decreasing the error function with 'optimal' use of the prior knowledge. The inductive bias was chosen such that the method for optimising the network parameters converged to a local minimum most rapidly.

We compared the impact of using our measure for determining this combination with the standard method by measuring the training time and generalisation performance, as well as the quality of the extracted refined knowledge after training. Cross-validation techniques were used in all experiments for feedforward neural networks. Recurrent neural networks were trained with an incremental training strategy described in [53]. Different measures of quality were defined and used as benchmarks for the extracted knowledge.

We not only ventured to apply our measure for determining a good combination of symbolic and neural learning to well-known synthetic problems in the literature, but we also targeted difficult, published real-world problems.

1.8 Accomplishments

We were successful in developing a measure for determining a good symbolic/neural learning combination. To achieve this, we implemented the various neural network architectures and training algorithms as well as their knowledge insertion and extraction algorithms. Our measure takes into account all the items stated in the first objective. We were able to determine *prior* to training the best method for using the prior information based on the developed heuristic. This resulted in better overall neural network

[1]This does not refer to the physical network architecture, but the neural network methodology.

performance compared to the current standard way of combining the prior knowledge with the neural network architectures. Our method not only works for feedforward neural networks, but also applies to recurrent neural networks. This is due to the basis, contained in all gradient learning systems, from which the heuristic was developed, i.e. the error function. To be able to quantify this measure we had to derive our heuristic for both domains, taking the different architectures into account. Extracting refined knowledge from the networks, we delivered more concise and accurate knowledge compared to standard knowledge encoding methods for the networks. We thus demonstrated the importance of effective use of an explicit inductive bias.

The following summarises and lists the important accomplishments of this thesis:

- We developed a heuristic which measures a neural network's confidence in the quality of the prior knowledge; this heuristic results in a good combination of symbolic and neural learning.
- The measure takes into account the prior knowledge, the training data, the network architecture, and the learning method.
- Quantifying this measure made it possible to evaluate the heuristic.
- The heuristic can be applied to feedforward *and* recurrent neural networks.
- We successfully applied our heuristic to synthetic problems, as well as difficult, published real-world problems in the domain of molecular biology and medical diagnosis.

1.9 Thesis Outline

In Chapter 2, we describe examples of bias in machine learning in general. We explore the role of bias in learning algorithms and discuss the trade-off between bias and variance in the learning context. We look at the impact of prior knowledge as a type of bias on the complexity of learning algorithms and the advantages of using hints whilst learning.

In Chapter 3, we discuss the basic neurocomputing architectures and the basic processing elements of the neural network paradigm. We describe the process of combining symbolic knowledge with the connectionist model of neural networks and stress its importance in the general neurocomputing paradigm. We conclude this chapter with examples of hybrid systems which combine symbolic information with a neural network paradigm.

Chapter 4 explores the representation of symbolic knowledge for knowledge-based artificial neural networks and recurrent neural networks. Algorithms for the insertion of prior information, training of the networks, and extraction of the refined symbolic knowledge are discussed in detail for the two network architectures.

In Chapter 5, we develop a heuristic for determining a good combination of the symbolic knowledge and the network architecture. We address the question of why it is important to do this and determine a computable measure that is used to analyse this heuristic. This measure is used to bias the learning algorithm and adjust the network's confidence in the prior knowledge for finding a good solution. We test our heuristic on well-known published synthetic problems, measure the network performance compared to the current standard method for combining the prior information with the network architecture and compare the accuracy of the extracted refined knowledge. We conclude with a discussion of the results.

In Chapter 6, we apply our heuristic to two difficult real-world problems. The first problem is from the domain of molecular biology. It concerns the classification of promoters in sequenced DNA strings. The second problem is from the domain of medical diagnoses: the classification of the different stages of the menstrual cycle through magnetic resonance spectroscopy (MRS) of the normal breast tissue. We discuss the benefits of using our heuristic for determining the inductive bias above the standard inductive bias. We show that our method not only performs better for synthetic problems as discussed in Chapter 5, but achieves good results for these difficult real-world problems as well.

Chapter 7 concludes with an overview of our achievements and directs attention to possible future areas of research.

Chapter 2

Bias In Machine Learning

2.1 Introduction

Every type of learning algorithm within machine learning has some inherent bias toward finding a solution in a hypothesis space. During learning, this hypothesis space is formed and then adjusted and modified, typically with generalisation or specification techniques, to form new hypotheses. Bias is used to exhibit a *preference* when there is a choice among hypotheses. Choosing an appropriate bias influences the learning method: it either inhibits or enhances system performance. Many bias selection and evaluation systems have been studied in literature; see [26] for an overview. In this chapter, we will discuss a few existing methodologies which establish the bias selection premise and their explicit choice of bias. We will look at different examples of bias in machine learning and conclude with a discussion of the impact of prior knowledge as a type of bias on the complexity of learning algorithms and the trade-off between bias and variance which is inherent in every learning method. We will also present the advantages of learning with hints. In the next section, we will discuss the role of bias in machine learning in more detail.

2.2 The Role of Bias

We can distinguish between two types of inductive bias [25]: *Representational bias* defines the states in the hypothesis search space; it can be introduced through some language, e.g. propositional logic, or structure, e.g. decision trees, neural network architectures; *procedural bias* defines the manner in which the hypothesis space is searched, e.g. high information gain attributes close to the root in decision trees, gradient-descent search in the weight space of neural networks.

As we have mentioned, all machine learning methods have some inherent bias toward finding a solution in hypothesis space. For instance, the inductive bias of the ID3 algorithm [70] for building decision trees is toward shallow trees that place high information gain attributes close to the root; the error backpropagation algorithm [73] for feedforward neural networks is biased toward finding a smooth interpolation between data points. However, these implicit biases are often not sufficient and an explicit bias must be introduced to achieve acceptable training and generalisation performance.

The explicit bias shifts a learning algorithm's inherent preference for a solution to a preferred, domain-specific solution. This explicit inductive bias can come in several forms depending on the learning algorithm and architecture of the system. The hypothesis chosen as a result of this explicit inductive bias will then favorably influence the learning as well as the generalisation performance of the system.

2.3 Bias Selection and Evaluation in Machine Learning

2.3.1 PREDICTOR

PREDICTOR [39] is an implementation of a system that uses one of three major techniques used for biasing the induction method during learning. The first technique *restricts* the *hypothesis language* which limits the number of possible hypotheses that can be formed as not all hypotheses are expressible. [96] developed a system that adds terms to a restricted hypothesis language, thus altering its bias. A second technique used in COBWEB [28], called *testing*, evaluates already generated hypotheses according to some measure. The third method, also used by PREDICTOR, *screens* hypotheses before generating a hypothesis, thus reducing the number of possible hypotheses. In PREDICTOR, bias is both represented explicitly as assumptions that are used to screen hypotheses and as procedures for testing those assumptions. Generalisation heuristics are used for the screening methods. These heuristics are condition/action pairs; a condition consists of a procedure for testing bias assumptions and the action consists of the application of a generalisation operator, e.g. the elimination of a feature, to the current and future hypotheses. This elimination of a feature can be seen as restricting the hypothesis language by removing a feature from the initial hypothesis language. This screening process has the effect of a bias shift by incrementally restricting the hypothesis language.

PREDICTOR presents new generalisation heuristics; its explicit use of bias offers advantages beyond the obvious computational improvement due to the screening of hypotheses before they are generated. Generalisation heuristics have been developed for the three assumptions, *irrelevance*, *independence*, and *cohesion*. The authors considered these assumptions to be significant for any system learning empirically from examples using features. One advantage is that the bias is a meta-level hypothesis itself; therefore, it can be tested and confirmed or rejected. Through the testing of its own hypothesis generator, PREDICTOR is able to anticipate future hypotheses. It is thus possible to reduce the hypothesis search space for current and future hypotheses. Testing of its bias can be done in different degrees of thoroughness and at different times during the learning process. A cautious learning approach will test the assumptions thoroughly before hypothesis generation. This will assure that the inductive hypothesis will be consistent with previous instances but does not guarantee consistency with future instances. These generalisation heuristics are called *consistency-preserving* and avoid the re-examination of instances and other consistency checks after generalising.

With a less cautious approach, learning may proceed faster, but errors become more likely. When a generalisation error occurs, error resolution methods are used to retest biasing assumptions and any violated assumptions are retracted though *directed backtracking* to a previous hypothesis. This shifts the bias and may result in the removal of hypothesis language assumptions made earlier.

The use of multiple learning methods is another advantage of using explicit biasing assumptions.

The authors tested and discussed the different generalisation heuristics. They compared their method of using an explicit inductive bias with the `CEA1`[1] algorithm that uses an implicit bias. They found that PREDICTOR is able to learn certain concepts that `CEA1` cannot cope with.

[1]`CEA1` implements the well-known Candidate Elimination Algorithm (CEA) [55].

2.3.2 Model Class Selection (MCS)

[14] implemented a system that automatically and recursively selects a bias for the construction of a classifier. For a given data set, it is often not clear which learning algorithm will deliver the best results. The authors constructed a system that, through feedback from the learning process, uses characteristics of the data set to guide a search for a tree-structured hybrid classifier.

Each learning algorithm depicts a selective superiority for the learning of different tasks. The problem of selecting an appropriate learning bias for a task then introduces a new problem: no single bias is best for all learning tasks. [14] developed a system to search for the best hypothesis space (model class) and search bias for a given data set. The system incorporates heuristic knowledge about the characteristics of the different representational languages, thus deciding through feedback from those learning algorithms the best bias to choose. MCS builds a classifier for a given data set, using these heuristic rules to guide a hill-climbing search for the best representation language for the different nodes in a hybrid classifier. The authors chose three basic representation languages used in statistical and machine learning algorithms: instance-based classifiers, linear discriminant functions, and decision trees.

The authors tested the system on an array of problems with the goals to (1) find out whether domain independent knowledge about characteristics can effectively guide an automatic algorithm selection search and (2) test whether the resulting hybrid classifier performs at least as well as any homogeneous classifier produced by its primitive components. It was found that MCS is more accurate than each of the primitive components, thus demonstrating the effectiveness of the algorithm selection search technique for solving the selective superiority of the individual algorithms. The authors also found that MCS performed better than three other hybrid classifiers, indicating that the simple increase of the search space through the introduction of hybrid classifiers is not sufficient to enhance overall performance. MCS's knowledge-based approach to choosing the structure and algorithms of the hybrid classifier proved to be more accurate and less time-consuming.

2.3.3 Inductive Policy for Bias Selection

Through the separation of the strategy for selecting an inductive bias—the *inductive policy*—from the actual bias itself, a framework is established that is flexible and can effectively obtain different results, on the same data, based on different considerations. This framework described in [69] selects bias in a learning system by considering different relations between the learning program and the user. These relations are called *pragmatic considerations.*

An inductive policy considers trade-offs in a specific domain and with a specific learning algorithm. These trade-offs, or pragmatic considerations, include preference of performance such as for time, space, accuracy, and the cost of errors. By addressing the relationship between the actual inductive bias choices and the goals of the users of the learning algorithm, the bias can be shifted to deliver user-specific performance. In the system developed by the authors, the inductive policies are represented as a set of bias choices and bias evaluation functions. It is thus possible to construct policies for selecting bias in any dimension of the underlying learning algorithm that are explicitly represented.

Through a search-based bias selection method, their system can learn differently under different pragmatic constraints. The system can perform term selection, parameter selection, and example selection. The authors have shown that, by representing the inductive policy *explicitly*, a variety of policies can be implemented in a single system. They have found that simple inductive policies can guide the search in bias space effectively; however, it is possible to implement more complex policies, such as simulated annealing.

2.3.4 Baldwin Effect: Insight into Bias Shifts

Most classical learning algorithms have an implicit bias fixed in their design. More recent algorithms can dynamically shift their bias as learning progresses. [94] states that lessons can be learned from the Baldwin effect on how to design and analyse bias shifting algorithms. The Baldwin effect was proposed to explain how Lamarckian evolution (the inheritance of acquired characteristics) can arise from purely Darwinian evolution and is concerned with the evolution of populations of individuals that can, during their lifetime, learn certain characteristics. A computational model of this effect was established by [43] in 1987: Learning smooths the fitness landscape and learning has a cost. There is an evolutionary pressure to find replacements instinctively for learned behaviors. Thus, in the early phase of the Baldwin effect, a selective pressure for learning will be exhibited; in the later phase, there will be a selective pressure in favor of instinct. The author argues that strong bias is analogous to instinct and weak bias to learning. Therefore the Baldwin effect predicts that, under certain conditions, a bias shifting algorithm will shift from a weak bias to a stronger bias during the learning process. He argues that the Baldwin effect does not merely describe the behavior of the shifting algorithm, but that the predicted trajectory, from a weak to a strong bias, is superior to alternative trajectories.

[94] introduced a variation of the computational model of the Baldwin effect, similar to [43]. The bias is separated into a strength and direction component and therefore the strength could be set to continuous values. The Baldwin effect promotes certain trajectories in bias search space above others. These trajectories seem to form a wide band of paths from a weak bias to a strong bias. Empirical results have shown that some paths that are outside of this *Baldwinian band* perform worse. The author suggests that thinking of the Baldwin effect as a continuum of bias strength, rather than the dichotomy of learning or not learning [43], attributes to a richer thinking of learning.

2.4 The Bias/Variance Dilemma

The basis of this dilemma [36] rests on the choice of an appropriate model for learning of a specific task. Too many free parameters in the model (i.e. a model-free inference) leads to high *variance* in the estimation error, whereas a model-based inference is usually *biased*.

When a model-free method is used to infer a complex task, a large data set must be used for it to converge to an acceptable measure as the model has a large number of parameters to estimate. A model which is restricted is inherently biased, because it has less parameters to model a certain function. Such complex models are usually difficult to attain, thus leading to increased bias for an *incorrect* model. The dilemma is this: Reduction of the variance in an estimation increases the bias, and *vice versa*. Thus, any attempt to minimise the variance by using a model-based inference restricts the estimation of the function to be learned to that specific model; less restrictive models increase the variance.

This is easily seen in neural networks as the number of weights (through the number of hidden neurons), used in the network, that contributes to this problem. For a complex task, a small network will lead to a high bias; the approximation of the function is limited by the small number of free parameters, effectively ignoring the data in favor of the constrained model. Over-parameterisation through a large number of hidden neurons—thus more weights—leads to a lower bias, but a definitely higher variance. The network might interpolate/overfit the data. These effects have been demonstrated by several authors [36].

2.5 Learning with Hints

Learning from hints is an advancement on learning from examples alone. Learning from examples deals with the learning of a concept represented by examples of this concept which is inferred by some learning algorithm. Learning from hints generalises this paradigm through the introduction of some known information about the target concept to be learned. This information is used in conjunction with the examples to deliver a better learned concept with increased performance. Hints can come in a variety of forms including invariance properties, symmetries, correlated functions, explicit rules, minimum-distance properties, or any information that reduces the search for the target concept.

The VC dimension [11, 98] is an established method for analysing the learning from examples. The VC dimension $VC(G)$ determines an upper boundary for the number of examples needed to learn a certain target concept or function f starting with a set of hypotheses G about the function f. The learning process uses the examples to guide the search for a hypotheses $g \epsilon G$ that is a close approximation of f. To enlarge the chances of finding a good hypothesis for f, the initial set G is large. This large set G, however, requires more examples of f to find a good hypothesis. This is reflected in a bigger value of $VC(G)$.

[2] derived a new quantity that defines a VC dimension for the hints and investigated how learning from hints influences the VC dimension. Through the use of the hints about the function f, it is possible to reduce the size of the hypothesis set G without losing good hypotheses. The author established the VC dimension for the hint H as

$VC(G;H)$ and the VC dimension given the hint H, thus the number of examples needed to learn f with the given hint, as $VC(G|H)$.

The author has found a relationship between $VC(G|H)$ and $VC(G;H)$, in many cases, to be as follows: For stronger hints H, $VC(G;H)$ is larger which reflects the smaller number of examples needed to learn the target concept f, thus a smaller $VC(G|H)$. For weak hints, the situation is reversed.

The result of this research concludes that a smaller example set is needed to learn a specific concept when hints are used. These hints can be introduced in various forms; [2] gives a brief outline on how to incorporate any type of hint in this framework.

2.6 Sample Complexity in Hint-Biased Neural Networks

The use of hints as bias, during a learning procedure, is well known. Hints in the form of symbolic rules have been used in neural networks to increase performance. These expert neural networks have empirically proven to perform better than normal multilayer perceptrons. See Chapter 3 for details of this paradigm.

[30] uses the VC dimension to formally explore the learning capacity and sample complexity of expert neural networks. The author establishes several theorems which prove that using hints, in the form of symbolic rules as bias in neural networks, reduces the generalisation dimensionality. This reduction results in a smaller number of training examples needed by an expert network for valid generalisation compared to an ordinary multilayer perceptron, effectively reducing the sample complexity.

2.7 Summary

The presence of bias in all machine learning algorithms is apparent. We have explored a few examples of methods used to evaluate and select bias according to various criteria. The correct choice of an algorithm and thus of bias was shown to be very important. Explicitly defining the bias has proven to be useful and fruitful to enhance the performance of learning algorithms.

Bias presents itself in many forms. Multiple forms of hints as bias exist; using these hints in a beneficial way is a non-trivial task. We discussed the use of hints to reduce sample complexity and to improve generalisation performance of learning systems. In subsequent chapters we will further explore the use of prior information about a specific domain as hints, and thus bias, to increase performance of neural networks. We will define bias in neural networks explicitly, discuss its benefits, and develop heuristics to quantify that bias for improved performance.

Chapter 3

Knowledge-Based Neurocomputing

3.1 Introduction

Neurocomputing methods and systems are partially based on the biological counterpart, i.e. the human/animal brain. Since biological computing is so powerful, researchers mimick the processing elements (neurons) of the brain to build information processing systems [40, 72] with the main advantage that they can adapt to a changing environment. This engineering method has been dubbed with several other terms, such as parallel distributed processing (PDP) [74], connectionist processing [79], artificial neural systems [104], massively parallel architectures, self-organising systems [46], and neuromorphic systems [3].

Knowledge-based systems refer to systems either mainly concerned with the actual knowledge as in expert systems, or where knowledge is used advantageously with existing architectures to improve the performance of such a system. Knowledge-based neurocomputing falls into the latter category. It involves the application of problem-specific knowledge within the neurocomputing paradigm; the representation, refinement and finally extraction of knowledge from an artificial neural network.

[18] states that: "Knowledge-based neurocomputing (KBN) concerns methods to address the explicit representation and processing of knowledge where a neurocomputing system is involved."

This is a necessarily vague statement because of a variety of approaches to this paradigm. For an excellent review of past and current methods in this area see [18].

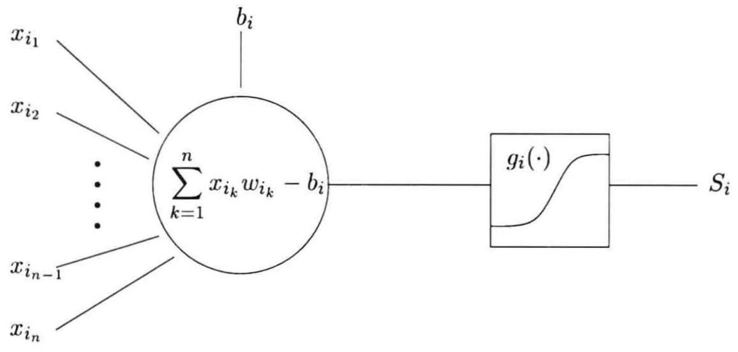


Figure 3.1: **A Neuron (basic processing element):** The weighted sum of the inputs, x_{i_k}, is used as input to the discriminant function, $g_i(\cdot)$, which calculates the output of the neuron, S_i.

In this chapter we are going to have a brief look at the basic artificial neural network paradigm, the basic processing elements and network architectures, typical learning algorithms and how integrating knowledge with these systems are possible. We will conclude with some example hybrid systems combining knowledge in some form with a neurocomputing architecture.

3.2 Neural Network Fundamentals

3.2.1 Processing Elements

A neural network consists of a group of single processing units (artificial neurons) (Figure 3.1) that are interconnected via weighted connections. Each neuron S_i is capable of processing some information, which is typically the weighted sum, $\sum_{i_k=1}^{n} x_{i_k} w_{i_k}$, of some other neurons' outputs or inputs given to the network. This input is then mapped through a discriminant function, $g_i(\cdot)$, to produce the output of the neuron. Common discriminant functions include linear, step, radial basis, and sigmoidal functions. The discriminant function maps the input into a specified output range denoted by the mathematical function. An internal threshold term, b_i, is typically associated with a neuron; the input to the neuron must exceed this threshold for the neuron to become active. Thus, mathematically the processing of a specific neuron is described as follows:

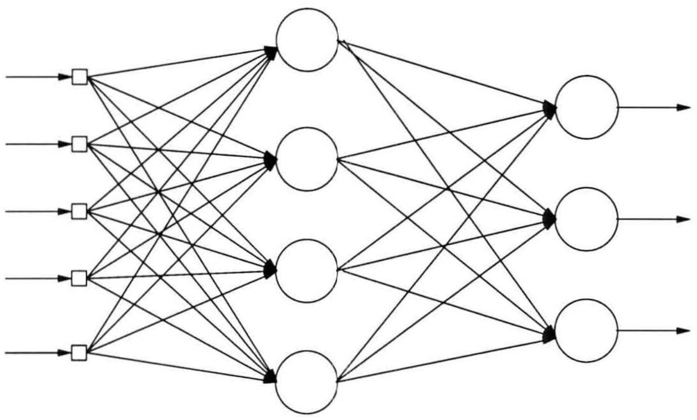


Figure 3.2: **A Feedforward Network:** A network with a single hidden layer. The neurons in a preceding layer are *fully* connected to neurons in the current layer via weighted connections. The first hidden layer is connected to the inputs to the network. The summation of the weighted inputs and discriminant functions are not showed.

$$S_i = g_i\left(\sum_{i_k=1}^{n} x_{i_k} w_{i_k} - b_i\right) \tag{1}$$

3.2.2 Network Topologies

Neural networks as directed graphs can be classified into two basic categories: acyclic graphs (feedforward networks, see Figure 3.2) and cyclic graphs (recurrent networks, see Figure 3.3). Feedforward networks have the capability to learn input/output mappings through static examples. They can also approximate arbitrary functions when sufficient neurons/weights are present in the network architecture [24]. Feedforward networks, in contrast to recurrent networks, can not remember state information over an indefinite time; however, they are able to store small finite amounts of state information through time-delay input neurons which stores past input features [78].

3.2.3 Learning Algorithms

Many different algorithms exist for training neural networks. The algorithms vary through the way they interpret the typical problem of optimisation for that specific

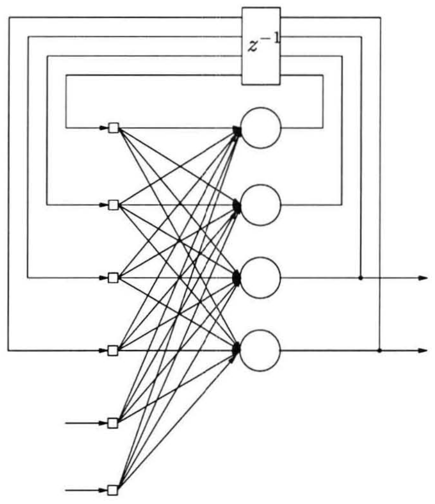


Figure 3.3: **A Recurrent Network:** A fully interconnected network. The network states are routed through a unit delay before the next input to the network are presented. The summation of the weighted inputs and discriminant functions are not shown.

network architecture. The goal in training a neural network is to find a set of weights that sufficiently delivers some solution. This is commonly measured by some error function, e.g. mean squared error. The minimisation of this error through the adjustment of the weights is achieved differently for each algorithm and network architecture.

The most common and popular methods of optimisation for neural network training are gradient-based algorithms [73]. Gradient-descent optimisation is a local searching method which deterministically converges towards a minimum in weight space. This minimum is typically a local minima in the region of the initial weight configuration; it has a small chance of being the global minima of the optimisation problem. This together with slow convergence, depending on the parameters chosen for the optimisation method and geometry of the error surface, is the major disadvantage of gradient-based learning methods. Heuristics to improve training and generalisation performance have been proposed [87].

A variation of the gradient-descent method is conjugate gradient learning; it finds the next weight configuration by using conjugate gradients to traverse weight space.

Other learning methods include simulated annealing and genetic algorithms. The former is another local search technique allowing stochastic exploration of the neighbourhoods of a current weight configuration whose size diminishes. The latter are global search techniques derived from the analogy of evolution [44]. Solutions are viewed as populations on which the basic evolutionary operations are applied: mutation, recombination, and then selection according to a fitness function specific to the problem, in this case, the minimisation of the error. An excellent survey for evolutionary neural networks can be found in [102].

3.3 Combining Symbolic Knowledge and Connectionist Learning

3.3.1 The General Paradigm

Combining symbolic and neural learning has become a well-established paradigm [1, 10, 29, 31, 33, 41, 50, 51, 52, 60, 68, 80, 86, 89, 92, 93]. There are different ways in which neural and symbolic learning can be combined to solve a given learning task. An excellent collection of a variety of approaches can be found in [95, 18].

The different methods can be classified according to the amount of prior knowledge available for the use with the neurocomputing architecture. Where an approximately complete and correct domain theory exists, the method is categorised as a *knowledge-intensive* or translational technique. The main problem with this technique is that most domain theories are not correct or do not describe the full domain attributes. *Knowledge-primed* or hint-based techniques involve the development of neural networks where partial and not necessarily correct domain knowledge exists. These "hints" are used to prime a neural network. Hints can vary from the structure of the network, global constraints on the function to be learned, to characteristics of the learning task that are useful, a priori, to the training of such a network. On the other end of the spectrum are *knowledge-free* or search-based techniques. These methods involve the use of scarce or unusable domain knowledge and rely mainly on guided search for the development of the neural network. In this category, networks can be statically defined before training or dynamically expanded/pruned during training.

The initialisation of feedforward networks with Horn clauses has been the predominant

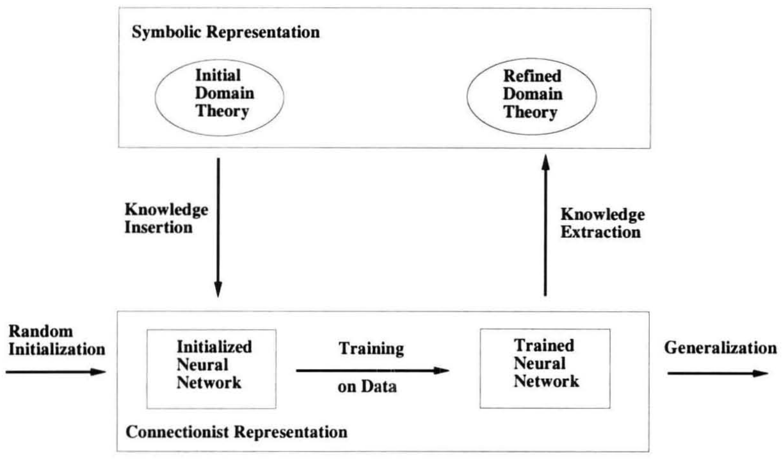


Figure 3.4: **A Framework for Combining Symbolic and Neural Learning:** The use of a neural network for knowledge refinement consists of three steps: (1) Insertion of prior symbolic knowledge (initial domain theory) into a neural network (2) Refinement of knowledge through training a neural network on examples, and (3) Extraction of symbolic of learned knowledge (refined domain theory) from a trained network.

paradigm for prior knowledge in the neural networks community. More recent work has shown how recurrent neural networks can be initialised with prior knowledge about a finite-state process [62]. Other examples of using a domain theory for initialising a feed-forward neural network have been proposed in the literature [35, 91]. Prior knowledge can also be used to alter the objective of the hypothesis search space. TangentProp provides explicit knowledge about the derivatives of the function to be learned [81]; it thus overrides the backpropagation learning algorithm's bias toward a smooth interpolation between points with explicit training derivatives. Explanation-based neural networks use previously trained neural networks as initial domain theories, and compute training derivatives from each observed training sample that describes the relevance of each input feature [56]. They are then trained using the TangentProp learning algorithm which minimises the network output error and the error in network derivatives.

In the following sections, we will limit our discussion to the paradigm illustrated in Figure 3.4.

3.3.2 The Importance of Prior Knowledge

The paradigm sketched in Figure 3.4 can include symbolic knowledge in the following way ('symbolic representation'): Prior knowledge about a task (initial domain knowledge) is used to initialise a network before training. The translation of the information from a symbolic into a connectionist representation is essential and the particular method for converting the symbolic representation of knowledge into its equivalent connectionist representation depends on the kind of symbolic knowledge, the learning task, and the network model used for learning.

There are advantages to making effective use of prior knowledge that is common to all learning tasks: (1) The learning performance may lead to faster convergence to a solution, (2) networks trained with hints may generalise better to future examples, (3) explicit rules may be used to generate additional training data which are not present in the original data set, and (4) learning leads to revision and extraction of a more concise domain knowledge.

Most recent efforts are directed towards encoding prior knowledge by programming some network weights to specified values instead of choosing small random values. A starting point, for the search of a solution in weight space, is defined by these programmed weights. The premise is that a better solution will be found faster compared to starting the search from a random point in weight space. The prior knowledge presumably defines a good starting point in the space of adaptable parameters and leads to faster learning convergence. This introduces an explicit inductive bias that draws a network's attention to relevant input features or favours a desirable connectionist knowledge representation. Prestructuring of feedforward networks with Boolean concepts (see e.g. [34, 91]) and imposing rotation invariance in neural networks for image recognition [7] are examples of this approach. We should point out that other types of prior knowledge encoding are possible. Rotation invariance can also be achieved through training, by presenting examples of rotated objects as inputs to a network. The choice of a network architecture itself represents an implicit use of prior knowledge about an application.

Fidelity of the translation of the prior knowledge from a symbolic form into a neural network is very important since a network may not be able to take full advantage of poorly encoded prior knowledge or, if the encoding alters the essence of the prior knowledge, the prior knowledge may actually hinder the learning process.

3.3.3 Knowledge Refinement

Using neural networks in the traditional simple way is shown in the bottom part of Figure 3.4 ('connectionist representation'). A network's weights are initialised with random values drawn according to some distribution. Using numerical optimisation methods (e.g. gradient-based techniques, simulated annealing), the network is trained on some known data to perform a certain task (e.g. pattern classification) until some training criterion is met. After successful training, a network can take advantage of its generalisation capabilities to perform the intended task on arbitrary data. Notice that during the entire process, the knowledge remains *hidden* in a network's adaptable connections, hence the name 'connectionist representation'.

3.3.4 The Significance of Knowledge Extraction

Once a network has succeeded in learning a task as measured by its performance on the training data, it may be useful to extract the learned knowledge. The question arises whether it is possible to extract an adequate symbolic representation of the knowledge learned by a network, i.e. a representation that captures the essence of the learned knowledge.

Of particular concern are *fidelity* of the extraction process, i.e. how accurately the extracted knowledge corresponds to the knowledge stored in the network, *accuracy* of the extracted knowledge, i.e. how well do extracted rules explain the given training data, and *comprehensibility* of the rules, i.e. the ease with which the rules can be interpreted and verified by an expert. Unfortunately, rule extraction is a computationally very hard problem. It has been shown that there do not exist polynomial-time algorithms for concise knowledge extraction [38]. The merits of rule extraction include discovery of unknown salient features and non-linear relationships in data sets, explanation capability leading to increased user acceptance, improved generalisation performance, and possible transfer of knowledge to new, yet similar learning problems.

Extraction algorithms can broadly be divided into three classes [5]: *Decompositional* methods infer rules from the internal network structure (individual nodes and weights). *Pedagogical* methods view neural networks as black boxes, and use some machine learning algorithm for deriving rules which explain the network input/output behaviour. Algorithms which do not clearly fit into either class are referred to as *eclectic*, i.e.

they may have aspects of decompositional and pedagogical methods. For feedforward networks, that knowledge has typically been in the form of Boolean and fuzzy if-then clauses [33, 42, 90]; excellent overviews of the current state-of-the-art can be found in [4, 5]. For recurrent networks, finite-state automata have been the main paradigm of temporal symbolic knowledge extraction [16, 29, 37, 61, 99, 103]. Clearly, neural networks are no longer black boxes.

In many cases, the extracted knowledge may only approximate a network's true knowledge; however, it is also possible for the extracted symbolic representation to exceed the accuracy of the knowledge stored in a trained network [61]. For feedforward neural networks, it has been shown that knowledge extracted *during* training can be useful for dynamically adapting a network's topology, i.e. the extracted knowledge can be used to guide the search for a solution. Methods for both feedforward and recurrent neural networks have been proposed [65, 97].

3.4 Examples of Hybrid Systems

3.4.1 Constructing Networks from If-Then Rules

As mentioned earlier, prior information in the form of Horn clauses (propositional rules) has been the main format for encoding knowledge into feedforward networks. In knowledge-based artificial neural networks, an initial domain theory in the form of propositional rules is used to construct a feedforward neural network [91, 88] (refer to Section 4.2 for more details). The backpropagation learning algorithm is then used to refine that initial domain theory. The paradigm provides a generalisation bias such that networks are more likely to generalise as predicted by the initial domain theory; backpropagation provides a generalisation bias such that networks are more likely to converge toward a solution with small weights. The authors have applied this technique to several domains including problems from molecular biology. They have shown good improvements in using this architecture above other encoding methods and the use of no prior information [90, 88].

Another method developed in [31], also uses symbolic rules to establish a neural network architecture. The author uses a non-differential discriminant function for representing disjunctive rules in the network and thus uses different learning mechanisms to solve this non-differentialability for training. Because of these different learning mechanisms, the system can not evolve with current trends in learning algorithms.

VL_1ANN is another system developed by [17] which uses symbolic rules in the form used by a variety of machine learning algorithms to encode a neural network. The algorithm converts symbolic rules to represent the same domain theory. It incorporates tunable fuzziness in the decision by adjusting the rule representation together with the representation of the input data. All input data have to be converted to real-valued data for this technique; it includes a method for treating nominal attributes. This overcomes the limitations of binary values for attributes in typical propositional rule domain theories.

3.4.2 Neuro-Fuzzy Combinations

When real-world problems are to be solved, basic and classical approaches fall short due to their basis on Boolean logic, analytical models, crisp classifications, and deterministic search techniques. Real-world problems are typically ill-defined, difficult to model and possess large solution spaces. Soft computing methods are well suited for the solving of such problems. They include architectures such as fuzzy logic, neurocomputing, evolutionary computing and probabilistic computing. For a detailed collection and taxonomy including examples of industrial and commercial hybrid soft computing systems, see [12]. In this section, we will focus on the combination of fuzzy logic and neurocomputing architectures.

Combining fuzzy methods with traditional neurocomputing architectures spawns hybrid systems that are capable of overcoming the precise modelling of the input and output data. [66] introduced a fuzzy neural network model based on a typical multilayer neural network. See [67] for a detailed variety of other methods applied to pattern recognition. [66]'s model converts numerical and linguistic inputs to linguistic terms and provides output in terms of class membership values; it is thus capable of fuzzy classification of patterns. The network is trained using a modified version of the backpropagation algorithm where the errors propagated back through the network are assigned appropriate weights according to the membership values at corresponding outputs. The authors only used three linguistic terms (*low, medium, high*) for the input to the network. They propose the use of fuzzy hedges (*more or less, nearly, very*) as additional properties for increased performance.

They applied this model to the problem of vowel recognition in the consonant-vowel-consonant context. The effectiveness of this model compared favourably with conventional neural network models and with a Bayes classifier trained on the same data.

3.4.3 Mapping Hidden Markov Models into Neural Networks

Hidden Markov models (HMM) probabilistically link an observed signal of a finite-state process - either discrete or continuous - to the state transitions of such a system that generated this signal. HMMs learn temporal models faster than recurrent neural networks while the latter is better at generalising on unseen input. [100] proposed to combine these two paradigms to alleviate the negative and amplify the positive aspects of these different methods.

The authors showed how first-order HMMs can be mapped into recurrent neural networks by using the structural similarities of the different architectures. This allow HMMs to be encoded into recurrent neural networks and be refined through subsequent training. This overcomes the first-order assumption of fixed number of states for the HMM and the slow and difficult training of recurrent networks initialised without prior knowledge.

Applications are widespread involving HMMs, thus attaining a further refined model through extraction of a HMM from the trained network will be of great benefit. Applications range from various speech recognition systems [9, 13], through automatic extraction of bibliographic information from scientific articles on the world wide web [20], to genetic and molecular biology data manipulation/classification in bioinformatics [45, 6, 47].

3.4.4 Data Mining from Time Series

Rule discovery from data to use as prior knowledge, for the encoding of neural networks, can benefit most solutions to real-world problems. Time series prediction is often a difficult process and the incorporation of knowledge for such a classification/prediction system is of great importance. Prior knowledge for time series is often found in the form of time window size, sampling frequencies or dimension information. These attributes of a time series are typically extracted using methods such as mutual information and false nearest neighbours. Information involving the dynamical process can be delivered through methods such as Lyapunov exponent extraction, power density spectrum analysis, correlation dimension determination, and non-stationarity detection [27].

[105] proposed a method for extracting rules as prior knowledge directly from the time series. These rules are then used to initialise a time delay neural network. The rule extraction process from the time series data is based upon the creation of a rule when a real value in the data is above a threshold α. These rules correspond to *positive* rules, and emphasises to the network that its output must be high when it should be high. The encoding of Boolean rules necessitates the transformation of the real-valued data to Boolean values. For this, another threshold β is used. Values above this threshold become *one* and values below *zero*. These rules are then encoded into the network and the biases of the neurons are scaled to facilitate the correct combination of rules to activate the neurons. This scaling was mentioned to be closely dependent on the threshold β.

The authors tested this method of knowledge extraction, and subsequent knowledge insertion, on time series such as the Lorenz time series as well as a time series obtained from seismic events in gold mines. The correspondingly initialised networks performed well; for the Lorenz time series a 50% reduction in training time was observed as well as a good reduction in the mean squared error, and generalisation performance, for one-step prediction. The seismic time series showed good improvements as well; this is beneficial for this difficult real-world problem of prediction in mines through the monitoring of seismic events.

3.4.5 Deterministic Finite-state Automata Encoding in Recurrent Neural Networks

Recurrent neural networks are appropriate tools for modelling time-variant systems, e.g. speech recognition, dynamical systems and the financial stock market. Models such as finite-state automata and their corresponding language can be viewed as a general paradigm of temporal, symbolic knowledge. [58] developed a method for encoding these automata into recurrent neural networks as prior knowledge. DFA extraction through clustering techniques reduces the network to the task of knowledge refinement in the case of partial/incorrect prior knowledge [59, 62]. For further detailed explanations see [61]. Recurrent networks are typically trained using either real-time recurrent learning

(RTRL) or backpropagation through time (BPTT).

3.5 Summary

In this chapter we have focused on the basics of neurocomputing. We have looked at the merits of combining knowledge with traditional neurocomputing methods. This combination favourably affects the performance and scope of application for neural networks. The knowledge-based neurocomputing paradigm can be broken down into three distinct steps: (1) the encoding of prior symbolic knowledge into neural networks, (2) the refinement of that knowledge through traditional neural learning methods, and (3) the extraction of the refined knowledge from the network.

This three-step process has been applied to a large variety of symbolic knowledge and neural architectures. We have looked at a few of these architectures and conclude that neural networks have definitely shed the notorious title of "black boxes". In the following chapter, we will elaborate on feedforward and recurrent networks, and discuss the insertion and extraction of symbolic knowledge for those networks.

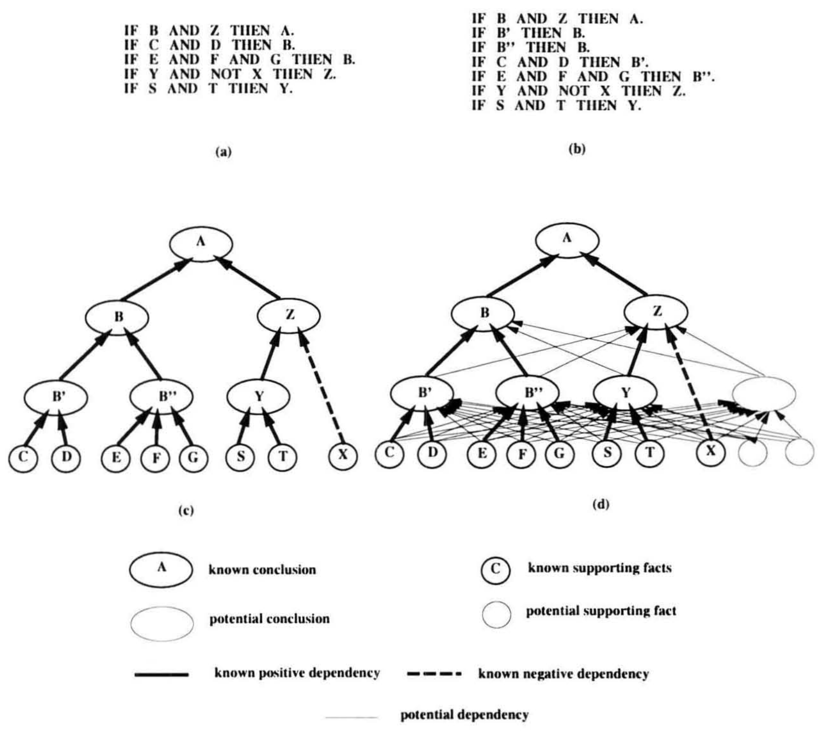
Chapter 4

Knowledge Representation and Neurocomputing

4.1 Introduction

Representing knowledge in a neurocomputing architecture depends closely on the quality of the knowledge available, the type of knowledge and the neurocomputing model used. Examples of hybrid systems and the combination of different knowledge platforms and architectures have been discussed in the previous chapter.

In this chapter, we focus on the insertion of propositional rules, the predominant format for the neural network community, into feedforward neural networks and the insertion of deterministic finite-state automata into recurrent neural networks. In particular we discuss the knowledge-based neural network architecture, developed in [91, 88] and the architecture described in [58] for recurrent neural networks. We will also discuss some extraction algorithms used for the extraction of refined knowledge from those networks.



```
IF B AND Z THEN A.
IF C AND D THEN B.
IF E AND F AND G THEN B.
IF Y AND NOT X THEN Z.
IF S AND T THEN Y.
```

(a)

```
IF B AND Z THEN A.
IF B' THEN B.
IF B'' THEN B.
IF C AND D THEN B'.
IF E AND F AND G THEN B''.
IF Y AND NOT X THEN Z.
IF S AND T THEN Y.
```

(b)

Figure 4.1: **Construction of KBANNs:** (a) Original knowledge base (b) rewritten knowledge base (c) network constructed from rewritten knowledge base (d) network augmented with additional neurons and weights.

4.2 Knowledge Representation in Feedforward Neural Networks

4.2.1 Knowledge Insertion

We use the method proposed in [88, 89, 91] to illustrate how Horn clauses can be encoded into feedforward networks. Other methods only differ in the way neuron inputs are combined (e.g. [48]). The construction of an initial network is based on the correspondence between entities of the knowledge base and neural networks, respectively. Supporting facts translate into input neurons, intermediate conclusions are modelled as hidden neurons, output neurons represent final conclusions; dependencies are expressed as weighted connections between neurons. The neuron outputs are computed by a sigmoidal function which takes as its argument a weighted sum of inputs.

Given a set of if-then rules (Figure 4.1 a), disjunctive rules are rewritten as follows: The consequent of each rule becomes the consequent of a single antecedent; it in turns becomes the consequent of the original rule (Figure 4.1 b). This rewriting step is necessary in order to prevent combinations of antecedents from activating a neuron when the corresponding conclusion cannot be drawn from such combinations. These rules are then mapped into a network topology as shown in Figure 4.1 c. A neuron is connected via weight H to a neuron in a higher level if that neuron corresponds to an antecedent of the corresponding conclusion. The weight of that connection is $+H$ if the antecedent is positive; otherwise, the weight is programmed to $-H$. For conjunctive rules, the neuron bias[1] of the corresponding consequent is set to $-(P - \frac{1}{2})H$ where P is the number of positive antecedents; for disjunctive rules, the neuron bias is set to $-\frac{H}{2}$. This guarantees that neurons have a high output when all or any one of their antecedents have a high output for conjunctive and disjunctive rules, respectively. If the given initial domain theory is incomplete or incorrect, a network may be supplemented with additional neurons and weights which correspond to rules still to be learned from data (Figure 4.1 d).

If an initial domain theory is sparse, the network constructed from the prior knowledge may be too small for a given learning task. In particular, the number of hidden neurons which along with their weights corresponding to intermediate conclusions may be insufficient. A heuristic search technique for dynamically creating hidden neurons during the learning process has been proposed [65]. After initial training, a set of tuning examples is used to identify poorly performing hidden units; new hidden units are added as long as a performance improvement can be observed.

4.2.2 Network Dynamics

Prior knowledge can be used to derive an initial hypothesis from which to start the search for a solution. In knowledge-based artificial neural networks, an initial domain theory in the form of propositional rules is used to construct a feedforward neural network [91]. The backpropagation learning algorithm [73] is then used to refine that initial domain theory. The encoding provides an inductive bias which is more likely to generalise as predicted by the initial domain theory; backpropagation provides a generalisation bias such that networks are more likely to converge toward a solution

[1]The neuron bias offsets the sigmoidal discriminant function; it is not to be confused with the inductive bias of the learning process.

with small weights.

The dynamics of a typical knowledge-based feedforward network can be described by the following equation:

$$S_j^l = g_j \left(\sum_{i=1}^{m} S_i^{l-1} w_{j_i}^l - b_j^l \right) \tag{2}$$

where S_j^l is the output of the neuron, j, in layer l. g_j is the discriminant function, typically a sigmoidal function. S_i^{l-1} is the output of neuron i in layer $l-1$ (containing m neurons) and $w_{j_i}^l$ the weight associated with that connection to neuron j. b_j^l is the internal threshold/bias of the neuron.

4.2.3 Learning Algorithm

Weight updates for a specific pattern are done according to the quadratic error function[2]

$$E = \frac{1}{2} \sum_{j=1}^{m} \left(d_j - S_j^l \right)^2 \tag{3}$$

where d_j is the desired output for neuron j in the output layer (containing m neurons), and S_j^l the actual output of the neuron j in layer l, where l is the output layer.

The weight updates are computed by

$$\triangle w_{j_i}^l = \eta \left(\triangle w_{j_i}^l \right)' + \alpha \delta_j^l S_i^{l-1} \tag{4}$$

where α is the *learning rate* constant and η usually a positive constant called the *momentum rate* constant. $\left(\triangle w_{j_i}^l \right)'$ is the previous weight update. The local gradient for neuron j, δ_j^l, can be calculated by

[2]Other error functions have also been proposed in the literature [91].

$$\delta_j^l = \begin{cases} (d_j - S_j^l)S_j^l(1 - S_j^l) & \text{when layer } l \text{ is an output layer} \\ S_j^l(1 - S_j^l)\sum_{k=1}^{m} \delta_k^{l+1} w_{k_j}^{l+1} & \text{when layer } l \text{ is a hidden layer} \end{cases} \qquad (5)$$

4.2.4 Training Method

The network is trained on examples chosen from the available data set. This set is typically divided into a training and a testing set. The ratio of this division depends on the method used for refining the network. The N-fold cross-validation methodology for training and testing, divides the data set into N sets; where N is the total number of examples in the data set, in the extreme case. The network is then trained on $N-1$ sets and tested on the remaining set. The average of these N tests then constitutes the resulting accuracy of the trained network. Commonly, a 10-fold cross-validation is used for the refinement of a neural network.

A network can be trained using either pattern- or batch-mode training. In the former, the weight updates are calculated and applied to the network after each example has been fed to the network. In the latter, weight updates are accumulated through feeding several examples to the network before applying this accumulated weight update to the network's weights. A trained network correctly classifies an example within a certain threshold. The thresholds ϵ and ψ are used for training and testing an example, respectively. Both of these parameters can be adjusted according to the problem domain, the values are typically $\epsilon = 0.25$ and $\psi = 0.5$.

4.2.5 Knowledge Extraction

Subset Algorithm

One of the extraction algorithms [84] we used is based on the subset algorithm and shares many of its characteristics [32, 90]; other algorithms have been proposed in the literature [22, 32, 75, 90][3]. It assumes that the sigmoidal neurons of a trained network operate near their saturation regions and that training does not significantly alter the knowledge representation. Most rule extraction algorithms search for combinations of weighted inputs that exceed a neuron's bias, resulting in the neuron operating near

[3]See the following subsection for details of one such an algorithm.

its upper saturation region; otherwise, the neuron output is close to zero. Thus, the neuron activation depends only on the size of its incoming weights. Extraction of rules is therefore reduced to finding subsets of weights whose sum exceeds the neuron's threshold. A lower limit on the size of the weights to be considered further reduces the set of candidate weights and thus the extracted rules. An ill-chosen lower weight threshold value may cause some learned rules to be overlooked.

Methods have been proposed for avoiding the combinatorial explosion [32, 75, 90]. We limit the number of weights per neuron that need to be considered. If the number of candidate weights exceeds that limit, the smallest weights are removed as possible antecedents until the limit is reached. This method has the disadvantage that different domains may require different limits and different number of weights to be considered. The modified subset algorithm is shown in Table 4.1.

```
With each hidden and output unit $U$:
A. (1) Find the maximal number (not more than $ceiling$)
       of positively weighted links greater than
       $threshold$, create a set of those links.
   (2) Extract subsets from that set whose summed weight
       is greater than the bias on the unit.
B. With each subset $P$ found in A.2:
   (1) Find number (not more than $ceiling$) of negatively
       weighted links greater than $threshold$, create a
       set of those links.
   (2) Extract subsets from that set whose summed weight
       is greater than the sum of $P$ less the bias
       on the unit.
   (3) Let $Z$ be a new predicate used nowhere else.
   (4) With each subset $N$ of the subsets found in B.2
       form the rule:  $Z$ :- $N$.
   (5) Form the rule:  $U$ :- $P$, !$Z$.
```

Table 4.1: **Subset Algorithm:** Pseudo-code for the extraction algorithm. *ceiling* and *threshold* values should be chosen for each application. We used *ceiling* = 15 to reduce combinatoric problems and a *threshold* = 0.4 to extract rules corresponding to more weighted connections.

The algorithm extracts a set of rules where some of the rules may be subsumed by other rules; thus, the size of the extracted rule set can further be reduced. These redundant rules only affect the size of the rule set and thus the degree of comprehension; they do not affect the classification performance.

TREPAN Algorithm

Another system TREPAN, developed by [21, 22, 23], involves the extraction of symbolic decision trees from trained neural networks. The system makes few assumptions about the architecture of the network and thus has a general appeal. TREPAN is similar to other decision-tree algorithms but instead of learning a target concept from training instances alone, TREPAN uses the neural network to classify/label all instances. It can thus learn from arbitrarily large samples to try and learn the concept function induced by the neural network.

TREPAN tries to progressively refine an extracted representation of a neural network by incrementally adding nodes to a decision tree that characterises the target function of the neural network. The extracted decision tree starts with a leaf node that predicts the class that network predicts most often. The tree is expanded by iteratively selecting a leaf node and converting it to an internal node with children nodes. Selection is based on an evaluation function that rates the leaf nodes, depending on their potential to increase the fidelity of the decision tree. The best node according to this criteria is then chosen for expansion. The node is expanded by determining a logical test to be inserted at that specific node. The partitioning of the input space are determined using *information gain* as a measure. The classes of the leaves of the newly expanded node are then determined. To ensure that a good logical split is made, a large sample of instances is used. These instances are drawn form the training examples of the network that reach that node, and from a model of the underlying distribution of the data in the domain.

The authors have applied this method of knowledge extraction to many problems [21] including the noisy time series of the US dollar/Deutsch mark exchange rate [23]. They've found that TREPAN extracted concise symbolic descriptions in the form of decision trees from trained neural networks. The extracted trees nearly match the accuracies of the networks, and are more comprehensible than trees produced by conventional decision-tree algorithms executed directly on the training data.

We used this extraction algorithm to extract knowledge from our trained neural networks obtained in Chapter 5 and in Chapter 6.

4.2.6 Knowledge Refinement

The neural network is reduced to the task of knowledge refinement, or revision in the case of incorrect prior information, through the combination of knowledge insertion and extraction methods. It has generally been observed that networks initialised with correct prior knowledge train faster and generalise better compared to networks trained without the benefits of an initial domain theory.

The impact of training feedforward neural networks with prior knowledge on the computational learning complexity has been discussed in [1]; the sample complexity for valid generalisation has been investigated in [30]. The results show that knowledge-based neural networks require a smaller sample size for valid generalisation compared to networks trained without prior knowledge.

4.3 Knowledge Representation in Recurrent Neural Networks

4.3.1 Knowledge Insertion

We used a method proposed by [58] for encoding prior knowledge in the form of Deterministic Finite-state Automata (DFAs) into recurrent neural networks.

The DFA encoding algorithm follows directly from the similarity of state transitions in a DFA and the dynamics of a recurrent neural network: Consider a state transition $\delta(q_j, a_k) = q_i$. We arbitrarily identify DFA states q_j and q_i with state neurons S_j and S_i, respectively. One method of representing this transition is to have state neuron S_i have a high output (≈ 1) and state neuron S_j have a low output (≈ 0) after the input symbol a_k has entered the network *via* input neuron I_k. One implementation is to adjust the weights w_{jjk} and w_{ijk} accordingly: setting w_{ijk} to a large positive value will ensure that $S_i(t+1)$ will be high and setting w_{jjk} to a large negative value will guarantee that the output $S_j(t+1)$ will be low. All other weights are set to zero in the case of full DFA encoding or to random initialised values when partial information is encoded. In addition to the encoding of the known DFA states, we also need to program the response neuron S_0, indicating whether or not a DFA state is an accepting state. We program the weight w_{0jk} as follows: If state q_i is an accepting state, then we set the weight w_{0jk} to a large positive value; otherwise, we will initialise the weight w_{0jk} to a large negative value. We define the values for the programmed weights as a rational number H, and let large *programmed* weight values be $+H$ and small values $-H$. We will refer to H as the *strength* of a rule. We set the value of the biases b_i of state neurons that have been assigned to known DFA states to $-H/2$. This ensures that all state neurons which do not correspond to the previous or the current DFA state have a low output. Thus, the rule insertion algorithm defines a nearly *orthonormal internal representation* of all known DFA states. We assume that the DFA generated the example strings starting in its initial state. Therefore, we can arbitrarily select the output of one of the state neurons to be 1 and set the output of all other state neurons initially to zero.

The summary of the DFA encoding algorithm is as follows:

$$w_{ijk} = \begin{cases} +H & \text{if} & \delta(q_j, a_k) = q_i \\ 0 & \text{otherwise, for} & \textit{full} \text{ DFA encoding} \\ \textit{random initialised} & \text{otherwise, for} & \textit{partial} \text{ DFA encoding} \end{cases}$$

$$w_{jjk} = \begin{cases} +H & \text{if} & \delta(q_j, a_k) = q_j \\ -H & \text{otherwise} \end{cases}$$

$$w_{0jk} = \begin{cases} +H & \text{if} & \delta(q_j, a_k) \text{ is an accepting state} \\ -H & \text{otherwise} \end{cases}$$

$$b_i = -H/2 \quad \text{for all state neurons } S_i$$

The initial state $\boldsymbol{S}(0)$ of the network is, thus

$$\boldsymbol{S}(0) = (S_0(0), 1, 0, 0, ..., 0),$$

where initial value of the response neuron $S_0(0)$ is 1 if the DFA's initial state q_0 is an accepting state and 0, otherwise.

4.3.2 Network Dynamics

We use a typical second-order recurrent neural network architecture as shown in Figure 4.2. The network is trained using either backpropagation through time (BPTT) [73] or real-time recurrent learning (RTRL) [101].

The continuous network dynamics are described by the following equations:

$$S_i(t+1) = g_i(\text{net}_i(t)) = \frac{1}{1+e^{-\text{net}_i(t)}} \tag{6}$$

$$\text{net}_i(t) = b_i + \sum_{j,k} w_{ijk} S_j(t) I_k(t), \tag{7}$$

where S_i is the activation of the hidden recurrent state neurons, I_k is the k-input, w_{ijk} is the corresponding weight and b_i is the bias for neuron i. The product $S_j(t)I_k(t)$ directly corresponds to the state transition of the automaton: $\delta(q_j, a_k) = q_i$ where a_k is the k^{th} symbol, represented by the input $\boldsymbol{I}$ using unary encoding, i.e. $I_k(t) \in \{0,1\}$. A special neuron S_0 represents the output and decides whether or not the string is accepted.

4.3.3 Learning Algorithm

The weight updates for a specific string is done according to the quadratic error function

$$E = \frac{1}{2}(\tau_0 - S_0(f))^2 \tag{8}$$

where τ_0 is the desired output for the string and $S_0(f)$ the output of neuron S_0 after time-step f, thus after the f^{th} input has entered the network. The weight updates for the RTRL algorithm are computed by

$$\triangle w_{lmn} = \alpha(\tau_0 - S_0(f))\frac{\partial S_0(f)}{\partial w_{lmn}} + \eta \triangle w'_{lmn} \tag{9}$$

where α is the *learning rate*, η the *momentum rate* and $\triangle w'_{lmn}$ the previous weight update. $\frac{\partial S_0(f)}{\partial w_{lmn}}$ can be computed recursively with equations

$$\frac{\partial S_i(t+1)}{\partial w_{lmn}} = g'_i(t)\left(\delta_{il} S_m(t) I_n(t) + \sum_{jk} w_{ijk} \frac{\partial S_j(t)}{\partial w_{lmn}} I_k(t)\right) \tag{10}$$

$$\frac{\partial S_i(t+1)}{\partial b_l} = g'_i(t)\left(\delta_{il} + \sum_{jk} w_{ijk} \frac{\partial S_j(t)}{\partial b_l} I_k(t)\right) \tag{11}$$

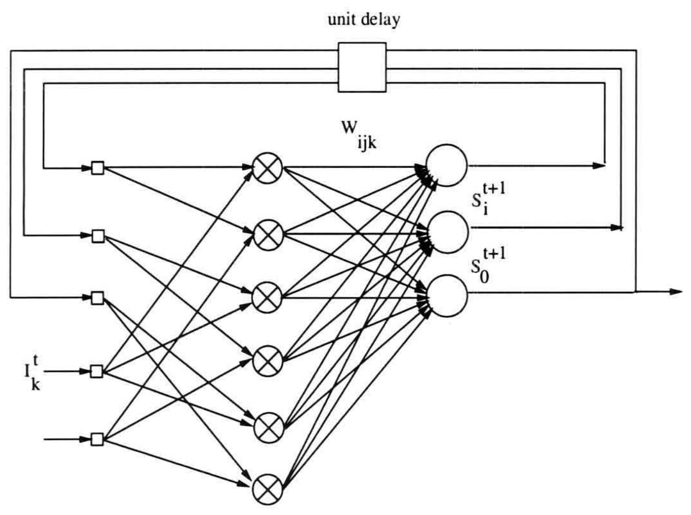


Figure 4.2: **Second-order Recurrent Neural Network.**

where $g_i'(t)$ is the derivative of the sigmoidal discriminate function and δ_{il} is the Kronecker-delta; δ_{il} is equal to 1 if $i = l$, 0 otherwise.

An aspect of the second-order recurrent neural network is that the product $S_j(t)I_k(t)$ in the recurrent network directly corresponds to the state transition $\delta(q_j, a_k) = q_i$ in the DFA. The effect of order in recurrent neural networks has been studied in [53]. After a string has been processed, the output of a designated neuron S_0 decides whether the network accepts or rejects a string.

4.3.4 Training Method

Recurrent neural networks are typically trained using a set of strings. [53] found that arranging the strings in lexicographic order and training on a small set of shorter strings first, facilitated easier training of the recurrent network. The initially small training set is then incrementally expanded through the addition of the next couple of strings in the lexicographically-arranged original data set. This process is repeated until the network classifies all of the strings correctly.

4.3.5 Knowledge Refinement

Methods have been proposed for extracting symbolic knowledge in the form of deterministic finite-state automata from trained networks [62]. The extracted knowledge is a concise representation of the refined initial domain theory.

4.4 Summary

We have looked specifically at the knowledge-based neural network architecture for combining prior information in the form of Horn clauses with feedforward neural networks. We discussed the algorithm for the insertion of this knowledge and then typical extraction algorithms for extracting the refined knowledge from those trained networks. We also looked at the insertion and extraction algorithms for recurrent networks and specifically looked at how to combine prior information in the form of deterministic finite-state automata with second-order recurrent networks. This enabled the refinement of the initial domain knowledge as well and could be extracted by the typical algorithm mentioned.

We have discussed the methods for combining prior information with two neural network architectures. Effectively encoding this information is of great concern and in the next chapter we will look at how to adjust the strength of this prior information encoding, or *inductive bias*, for the two neural network architectures.

Chapter 5

Quantifying Inductive Bias

5.1 Introduction: Why Quantify Learning Bias?

We have shown how to encode information into neural networks prior to training. This integration of prior knowledge provides means to determine the network architecture, to program a subset of weights to induce a learning bias which guides network training, and to extract refined knowledge from trained networks. While good empirical results have been achieved using the framework which combines neural and symbolic learning described in the previous chapter, the merits underlying the symbolic/connectionist approach are not yet well understood. Gaining that insight remains an important open research problem.

In this chapter, we address the following open question: How should this explicit inductive bias H be chosen? If we give too little weight to the inductive bias, then it may not be very helpful in finding a solution. If we assign too much importance to it, then the network might not be able to find a solution, particularly when the prior knowledge and the training data do not represent similar concepts.

It is conceivable that the choice of this inductive bias depends on the application, the training data, and the network architecture. By finding a good heuristic for choosing this inductive bias, we can combine the representational and procedural biases (refer to Section 2.2). Bias interactions have also been studied in [15, 19].

We proposed a novel heuristic for determining the strength of the inductive bias for feedforward neural networks encoded with prior information [82]. In Section 5.3, we present the details of this algorithm. We have also applied this heuristic to recurrent neural networks [83]; the details of which are described in Section 5.4. We also show the performance of our heuristic on synthetic problems for both network architectures and discuss the results achieved using our heuristic to determine the strength of the inductive bias.

5.2 Premises

Consider an error function E used to train a network. The idea for determining a good value for the inductive bias H is to start the search for a solution at a point in weight space where the gradient $\partial E/\partial H$ is maximal, i.e. we choose H such that the search starts at a point where the error function in the "direction" of the inductive bias H—the direction of the prior knowledge—is steepest: $\max(|\partial E/\partial H = 0|)$. This avoids the need for determining H through trial-and-error or traversing flat regions of the weight space during the initial training phase. Furthermore, the value H which achieves good performance depends on the prior knowledge, the training data, the network architecture, and the learning algorithm. The function $\partial E/\partial H$ takes all these dependencies into consideration. The more prior knowledge that is available and the more accurate that knowledge is, the more the function $\partial E/\partial H$ influences the gradient-descent search for a solution in weight space. Steepest descent makes fast convergence possible; furthermore, it is a reasonable premise that good local minima in weight space are more likely to be found at the bottom of steep ravines than in shallow valleys.

5.3 Feedforward Neural Networks

Based on empirical investigations, it has been suggested that all weights which reflect prior knowledge about a learning task be set to $H = 4$. This indiscriminate choice of the inductive bias has two major drawbacks: (1) It is conceivable that different applications require different choices of the inductive bias H which leads to fast convergence and good generalisation performance, and (2) it does not provide a mechanism for dealing with uncertainty about the initial domain theory. This section proposes a method for choosing the strength of the inductive bias which takes these two objections into account: The choice of H depends on the application represented by the initial domain theory, the network architecture, the training data, and the learning algorithm; it adjusts its confidence in the prior knowledge according to the amount and the quality of the available prior knowledge.

5.3.1 Algorithm

We will now derive a recursive procedure for evaluating the gradient $\partial E(H)/\partial H$ prior to training which is similar to the error backpropagation learning algorithm [1]. Refer to Section 4.2.2 for equations.

Consider the commonly used quadratic error function [2] for a specific pattern

$$E(H) = \frac{1}{2}(d_0 - S_0^l(H))^2 \tag{12}$$

where d_0 is the desired network output and $S_0^l(H)$ is the actual network output for a specific pattern p [3], where l is equal to the output layer. Notice that S_0^l depends on the particular choice of H [4]. Then, the derivative $\partial E/\partial H$ is given by

$$\frac{\partial E}{\partial H} = -(d_0 - S_0^l)\frac{\partial S_0^l}{\partial H} \tag{13}$$

where l is equal to the output layer. We can compute $\partial S_0^l/\partial H$ recursively as follows:

$$\frac{\partial S_0^l}{\partial H} = S_0^l(1 - S_0^l)\sum_{j=1}^{m}(\frac{\partial w_{0_j}^l}{\partial H}S_j^{l-1} + w_{0_j}^l\frac{\partial S_j^{l-1}}{\partial H}) \tag{14}$$

where $w_{0_j}^l$ is the weight connecting the output of neuron j in the hidden layer (containing m neurons) immediately preceding the network output layer with the output neuron, S_0^l. The derivative $\partial w_{0_j}^l/\partial H$ can easily be calculated by

$$\frac{\partial w_{0_j}^l}{\partial H} = \begin{cases} +1 & \text{if } w_{0_j}^l = +H \\ -1 & \text{if } w_{0_j}^l = -H \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

[1]The value of the error function E depends on the particular choice of H, thus $E(H)$. For simplicity, we omit the argument H in the equations for the computation of $\partial E(H)/\partial H$.

[2]Other error functions have also been proposed in the literature. The derivation of the function $\partial E(H)/\partial H$ can be adjusted accordingly.

[3]Normalisation of the error function according to the number of patterns is necessary for a comparable value.

[4]For reasons of simplicity, we only consider networks with a single output; the generalisation to networks with multiple outputs is straightforward.

The derivative $\partial S_j^l / \partial H$ for neurons in the hidden layers can be recursively computed similarly:

$$\frac{\partial S_j^l}{\partial H} = S_j^l(1 - S_j^l) \sum_{i=1}^{m} (\frac{\partial w_{ji}^l}{\partial H} S_i^{l-1} + w_{ji}^l \frac{\partial S_i^{l-1}}{\partial H}) \tag{16}$$

where w_{ji}^l connects neuron i, in layer $l-1$, with neuron j in the next hidden layer, l. The derivative $\partial w_{ji}^l / \partial H$ can easily be calculated by

$$\frac{\partial w_{ji}^l}{\partial H} = \begin{cases} +1 & \text{if } w_{ji}^l = +H \\ -1 & \text{if } w_{ji}^l = -H \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

We need a "bootstrap" equation in the case where node j is in the first hidden layer ($l = 0$), i.e. S_i^{l-1} does not depend on H since it is equal to the value of input neuron i. We then have $\partial S_i^{l-1} / \partial H = 0$ and Equation 16 simplifies to

$$\frac{\partial S_j^l}{\partial H} = S_j^l(1 - S_j^l) \sum_{i=1}^{m} \frac{\partial w_{ji}^l}{\partial H} S_i^{l-1} \tag{18}$$

The same equations also apply to the neuron biases.

5.3.2 Performance on Synthetic Problem

In order to illustrate our heuristic, we used a simple initial domain theory consisting of Boolean rules (Figure 5.1). These rules corresponds to the Winston's Cup domain theory [92, 76].

```
        Cup :- Stable, Liftable, Open Vessel.
               Stable :- bottom flat.
           Liftable :- Graspable, light.
              Graspable :- has handle.
 Open Vessel :- has concavity, concavity up.
```

Table 5.1: **Winston's Cup Domain Theory:** This set of rules in PROLOG notation was used to illustrate our heuristic for determining the strength of the inductive bias which leads to good performance.

The initial domain theory was encoded using the knowledge-based neural network architecture. We used networks with neurons with sigmoidal discriminant functions and used the standard quadratic error function E (refer to Equation 3 in Section 4.2.2) for training. A network correctly classified an example if its output was within $\epsilon = 0.25$

	Cups					Non-Cups				
Feature	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}
has handle	y			y	y		y			
handle on top							y			
handle on side	y			y	y					
bottom is flat	y	y	y	y	y	y	y		y	y
has concavity	y	y	y	y	y		y	y	y	y
concavity up	y	y	y	y	y		y	y	y	
light	y	y	y	y	y	y	y		y	
expensive	y		y		y		y		y	
material: ceramic	y							y		
material: paper				y	y		y		y	
material: styrofoam		y	y			y				y
fragile	y	y			y	y		y		y

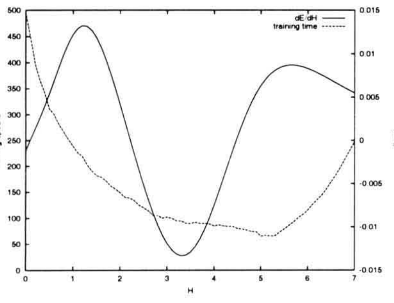
Table 5.2: **Data for the Winston's Cup Domain Theory:** The data indicates the features present for each example for this domain problem.

of the desired output for training data and to within $\psi = 0.5$ from the desired output for testing data. We chose the learning rate $\alpha = 0.1$ and the momentum $\eta = 0.1$ and trained all networks until one of the three following stopping criteria was satisfied:

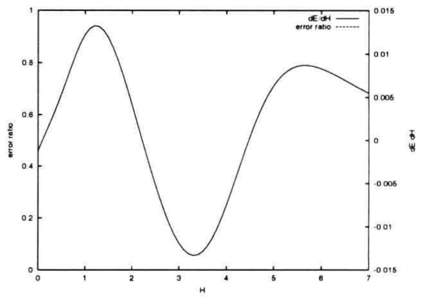
- On 99% of the training examples, the activation of every output unit was within ϵ of correct, or
- a network has been trained for 5,000 epochs, or
- a network classified at least 90% of the training examples correctly, but has not improved its ability to classify the training examples for five epochs.

We used a data set consisting of 5 positive and 5 negative examples (Table 5.2). The representative results in Figure 5.1 show the training times (number of epochs) and the generalisation (error percentage on test set) for a particular run as a function of the inductive bias H; we chose the value of H from the interval $[0, 7]$ in increments of 0.1. The training and test set size as well as the amount of prior knowledge are varied to show the general performance of our heuristic. We also show the graph of the function $\partial E/\partial H$.

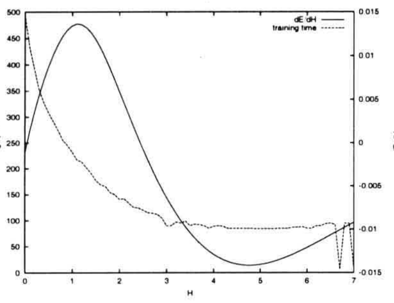
We observe that the value of H for which the function of $|\partial E/\partial H = 0|$ has a maximum corresponds reasonably well with the value of H for which the network achieves good training time as well as generalisation performance. Figure 5.1a,b show training and



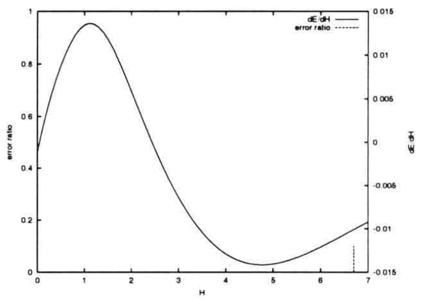
(a)



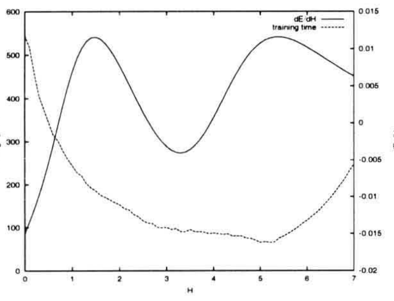
(b)



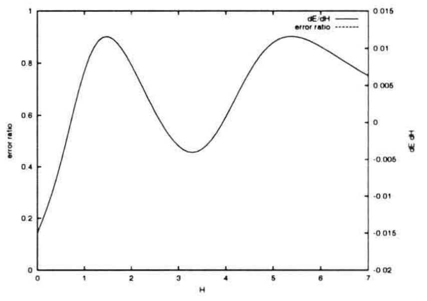
(c)



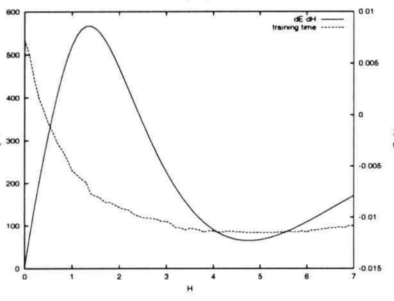
(d)



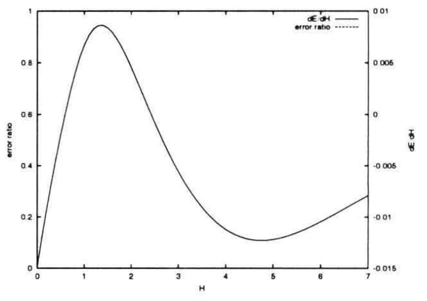
(e)



(f)



(g)



(h)

Figure 5.1: **Winston's Cup Illustrative Results:** These figures show typical training times and the corresponding generalisation performance for networks trained with different values of the inductive bias H, varied training and testing set sizes and different amounts of prior knowledge. It plots the function $\partial E/\partial H$ as a function of the inductive bias strength H.

generalisation performance, respectively, of the network trained with 100% of the data for training and testing sets and with full prior knowledge ($|\partial E/\partial H = 0|_{max} = 0.01334$ for $H = 3.3$; 95 epochs; 0% error). Figure 5.1c,d show training and generalisation performance, respectively, of the network trained with 100% of the data for training and testing sets with rule 1,2,4, and 5 of the domain theory ($|\partial E/\partial H = 0|_{max} = 0.01416$ for $H = 4.8$; 85 epochs; 0% error). Figure 5.1e,f show training and generalisation performance, respectively, of the network trained with 90% of the data for training and 10% of the data for testing with full prior knowledge ($|\partial E/\partial H = 0|_{max} = 0.01160$ for $H = 5.4$; 75 epochs; 0% error). Figure 5.1g,h show training and generalisation performance, respectively, of the network trained with 90% of the data for training and 10% of the data for testing with rule 1,2,4, and 5 of the domain theory ($|\partial E/\partial H = 0|_{max} = 0.01229$ for $H = 4.8$; 84 epochs; 0% error).

5.3.3 Discussion

Encoding prior information using our heuristic to determine the inductive bias H, results in good performance. Although the Winston's Cup is a small problem, it is apparent that there are merits in choosing the inductive bias H well. Comparing results for using our heuristic to choose the inductive bias H with the standard inductive bias $H = 4$, we note similar performances. Our heuristic is able to point out a good inductive bias even when partial information is encoded and subsets of the data set are used for training.

These encouraging results led to the application of our heuristic to difficult real-world problems as presented in Chapter 6.

5.4 Recurrent Neural Networks

We also applied our heuristic to recurrent neural networks, specifically second-order networks used for the encoding of deterministic finite-state automata. Again the algorithm takes the prior information, the network architecture, the training data, and the learning algorithm into consideration for choosing the strength of the inductive bias H.

5.4.1 Algorithm

We will now derive a recursive procedure for evaluating the gradient $\partial E(H)/\partial H$ prior to training which is similar to the derivation of the real-time recurrent learning algorithm[5]. Refer to Section 4.3.2 for equations.

Consider the quadratic error function

$$E = \frac{1}{2}(\tau_0 - S_0(f))^2 \tag{19}$$

where τ_0 is the desired output for a string and $S_0(f)$ the output of neuron S_0 after time-step f, i.e. after the final input of the string. Note that $S_0(f)$ depends on the particular choice of H. Then, the derivative $\partial E/\partial H$ for a specific string[6] is given by

$$\frac{\partial E}{\partial H} = -(\tau_0 - S_0(f))\frac{\partial S_0(f)}{\partial H} \tag{20}$$

We can compute $\partial S_i(t)/\partial H$ recursively as follows:

$$\frac{\partial S_i(t)}{\partial H} = g_i'(t)\left[\frac{\partial b_i}{\partial H} + \sum_{jk}\frac{\partial w_{ijk}}{\partial H}S_j(t-1)I_k(t-1) + \sum_{jk} w_{ijk}I_k(t-1)\frac{\partial S_j(t-1)}{\partial H}\right] \tag{21}$$

where S_i is the activation of the hidden recurrent state neurons, $g_i'(t)$ is the derivative of the sigmoidal discriminant function, I_k is the k-input, w_{ijk} is the corresponding weight, b_i is the bias for neuron i, and

$$\frac{\partial W_{ijk}}{\partial H} = \begin{cases} +1 & \text{if } w_{ijk} = +H \\ -1 & \text{if } w_{ijk} = -H \\ 0 & \text{otherwise} \end{cases} \quad \textit{and} \quad \frac{\partial b_i}{\partial H} = \begin{cases} +1/2 & \text{if } b_i = +H/2 \\ -1/2 & \text{if } b_i = -H/2 \\ 0 & \text{otherwise} \end{cases} \tag{22}$$

When $t = 0$, $\boldsymbol{S}$ does not depend on H since it is the initial state of the network, thus

[5]The value of the error function E depends on the particular choice of H, thus $E(H)$. For simplicity, we omit the argument H in the equations for the computation of $\partial E(H)/\partial H$.

[6]$\partial E(H)/\partial H$ is calculated for a specific string. Normalisation according to the number of strings in the training set is necessary for a comparable value.

$$\frac{\partial S_i(t)}{\partial H} = 0. \tag{23}$$

5.4.2 Performance on Synthetic Problem

To demonstrate our heuristic, we used a published 10-state DFA (Figure 5.2a) from literature [63] for the initial domain theory.

Network Performance

We encoded the DFA into a second-order recurrent neural network according to the encoding algorithm described in Section 4.3. The neurons had sigmoidal discriminant functions and were trained with the standard quadratic error function E (refer to Equation 8 in Section 4.3.2). A network correctly classified an example, during training, if its output was within $\epsilon = 0.2$ of the desired output and for testing, to within $\psi = 0.5$ from the desired output. We used a learning rate of $\alpha = 0.5$ and a momentum rate of $\eta = 0.5$.

The training set consisted of all strings[7] up to and including length 10 generated by the DFA, in lexicographic order. The networks were not trained *wholesale* on all the strings, but incrementally [53]; the initial working set contained the first 30 strings. Training was subdivided into cycles. In each cycle, the network was trained on the working set up to a maximum of 300 epochs or until all strings in the working set were correctly classified. After such a cycle, the network was evaluated on the *whole* training set. If all the strings were correctly classified then the training was stopped, otherwise the next 10 strings from the original training set were added to the working set and a new training cycle was started. Training was also stopped when the networks trained for a total of 10300 epochs. After training the networks were tested on all strings[8] up to and including length 15 generated by the original DFA.

To test our heuristic, we used varying amounts of prior information in the form of partial DFAs of the original DFA (Figure5.2b-h) and for malicious information we used the DFAs in Figure 5.3. All the networks were trained to learn the original DFA through the use of strings generated from that DFA. Figures 5.4, 5.5, 5.6, and 5.7 show

[7]The training set consisted of 1023 strings.

[8]The test set consisted of 65534 strings.

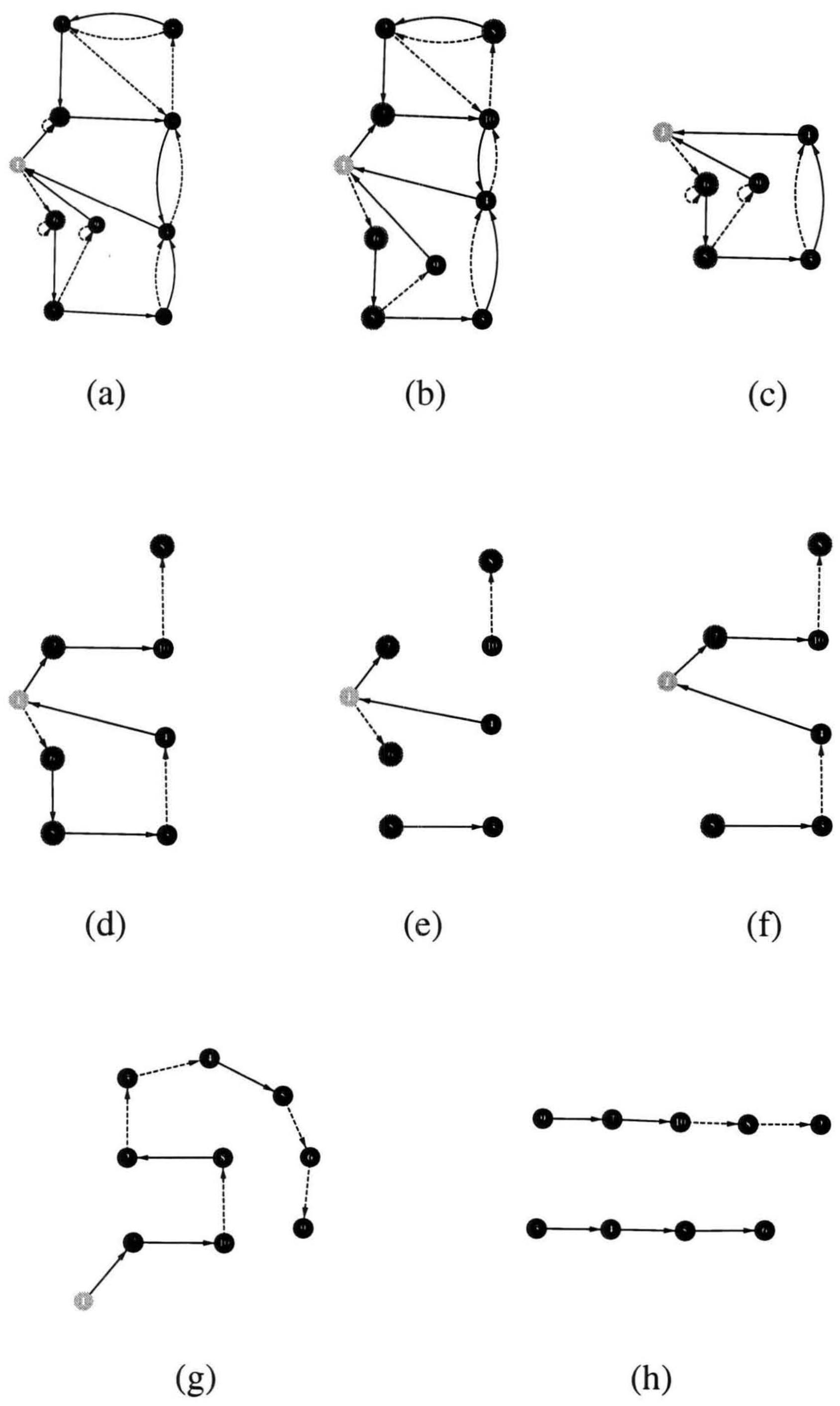


Figure 5.2: **Initial Domain Theory:** Shown are the DFAs used to encode the recurrent networks before training. State 1 is the start state and state transitions on input symbols '0' and '1' are shown respectively as solid and dashed arcs. Accepting states have double-edged circles. (a) all prior information (the entire DFA), (b) all rules except self-loops, (c) partial DFA, (d) rules for string '(10010)*001', (e) rules for disjointed transitions, (f) rules that do not start with a start state, (g) rules for string '001011011' without programming a loop, (h) rules for separate strings '000' and '0011'.

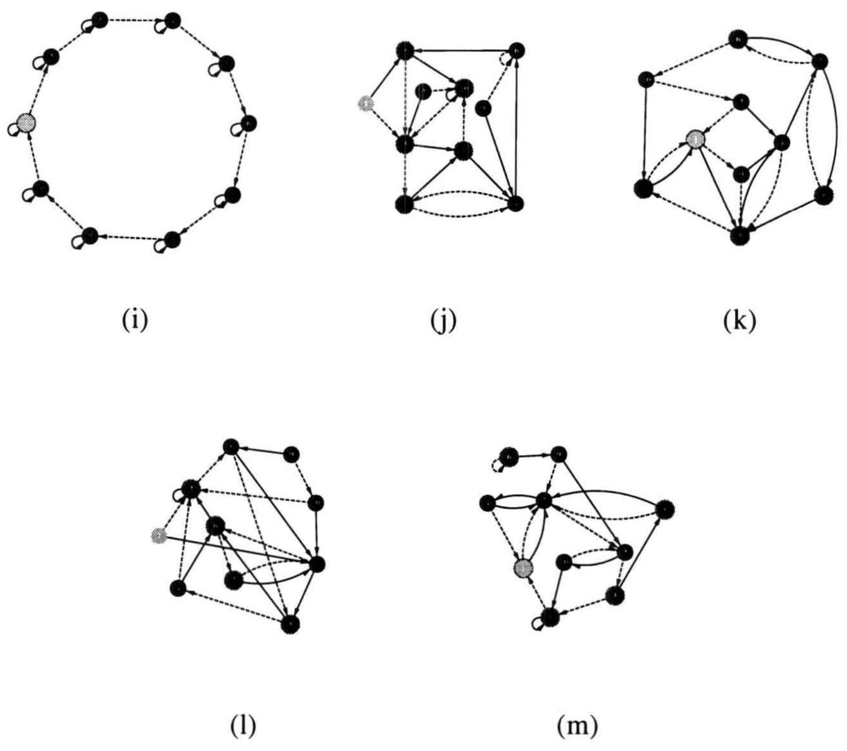
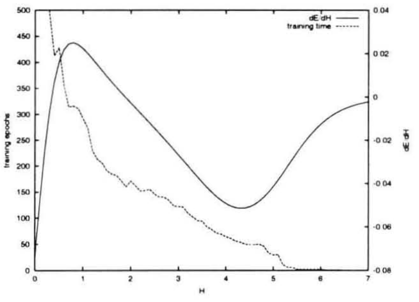


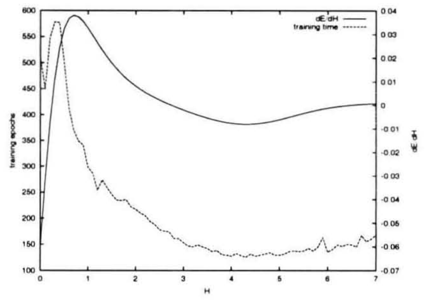
Figure 5.3: **Initial Domain Theory:** Shown are the DFAs used to encode the recurrent networks, with *malicious* information, before training. State 1 is the start state and state transitions on input symbols '0' and '1' are shown respectively as solid and dashed arcs. Accepting states have double-edged circles. (i) DFA accepting all strings where the number of 1's is a multiple of 10, (j-m) randomly generated DFAs with 10 states.

typical results obtained for the corresponding DFAs in Figures 5.2 and 5.3. The training performance (number of epochs) and generalisation performance (error percentage on test set) for a particular network are shown as a function of the inductive bias H; we chose the value of H from the interval $[0, 7]$ in increments of 0.1. We also show the graph of the function $\partial E/\partial H$.

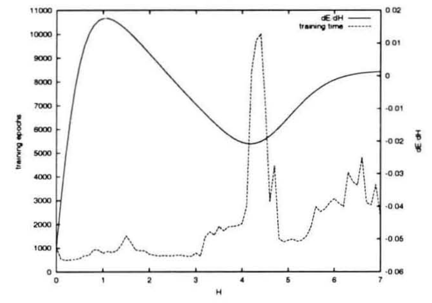
We ran 10 runs for each DFA, varying the random initialised weights for each network. The DFA encoding algorithm (refer to Section 4.3) only allows certain weights for partial information encoding to be set to small random initialised values; for full DFA encoding, the weights not corresponding to prior information are set to zero. The random weights could only be varied for the partial DFAs from Figure 5.2b-h. Networks encoded with DFAs from Figure 5.2a and Figures 5.3 only needed one run, as the weights not corresponding to prior information was set to zero; all the networks were the same, for a particular DFA. The average and standard deviation of the training



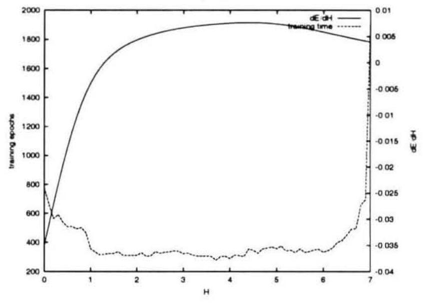
(a)



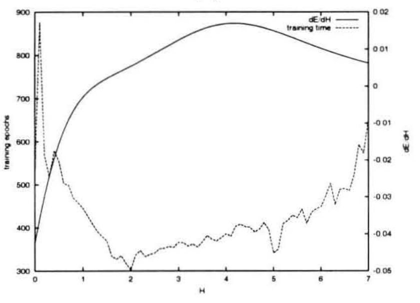
(b)



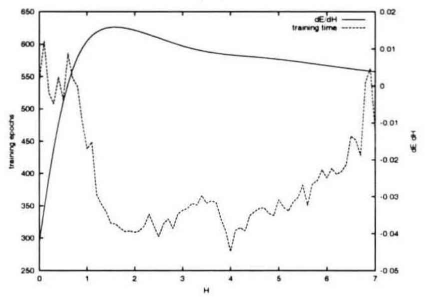
(c)



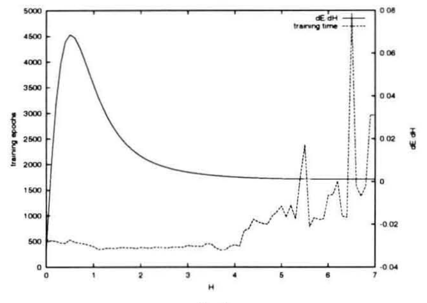
(d)



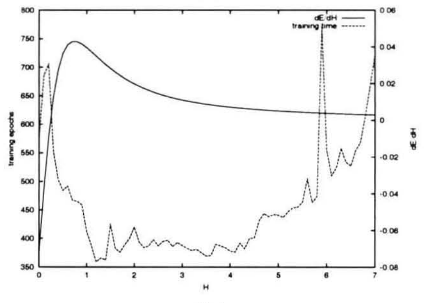
(e)



(f)

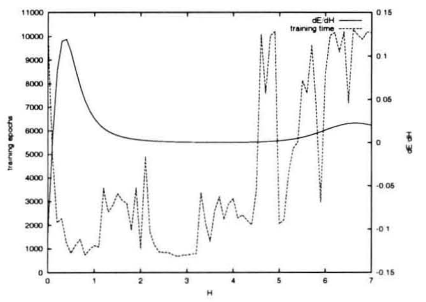


(g)

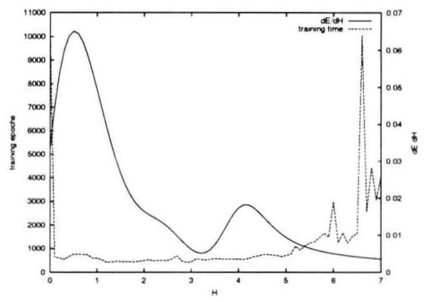


(h)

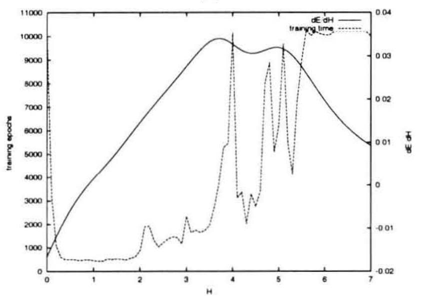
Figure 5.4: **Training Performance:** These figures show typical training times for networks trained with the respective DFAs from Figure 5.2 as prior knowledge. Networks were trained with different values of the inductive bias H. It plots the function $\partial E/\partial H$ as a function of the inductive bias strength H.



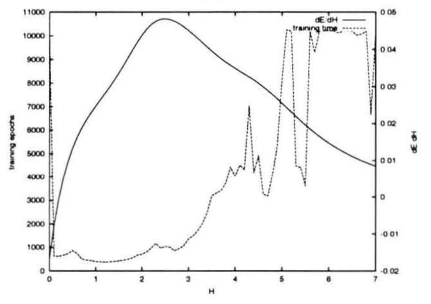
(i)



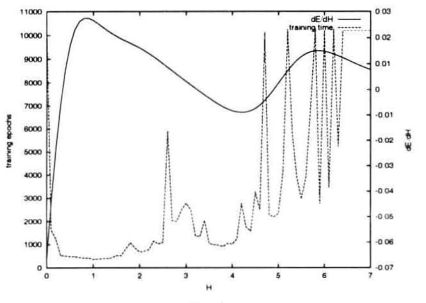
(j)



(k)



(l)



(m)

Figure 5.5: **Training Performance:** These figures show typical training times for networks trained with the respective DFAs from Figure 5.3 as prior knowledge. Networks were trained with different values of the inductive bias H. It plots the function $\partial E/\partial H$ as a function of the inductive bias strength H.

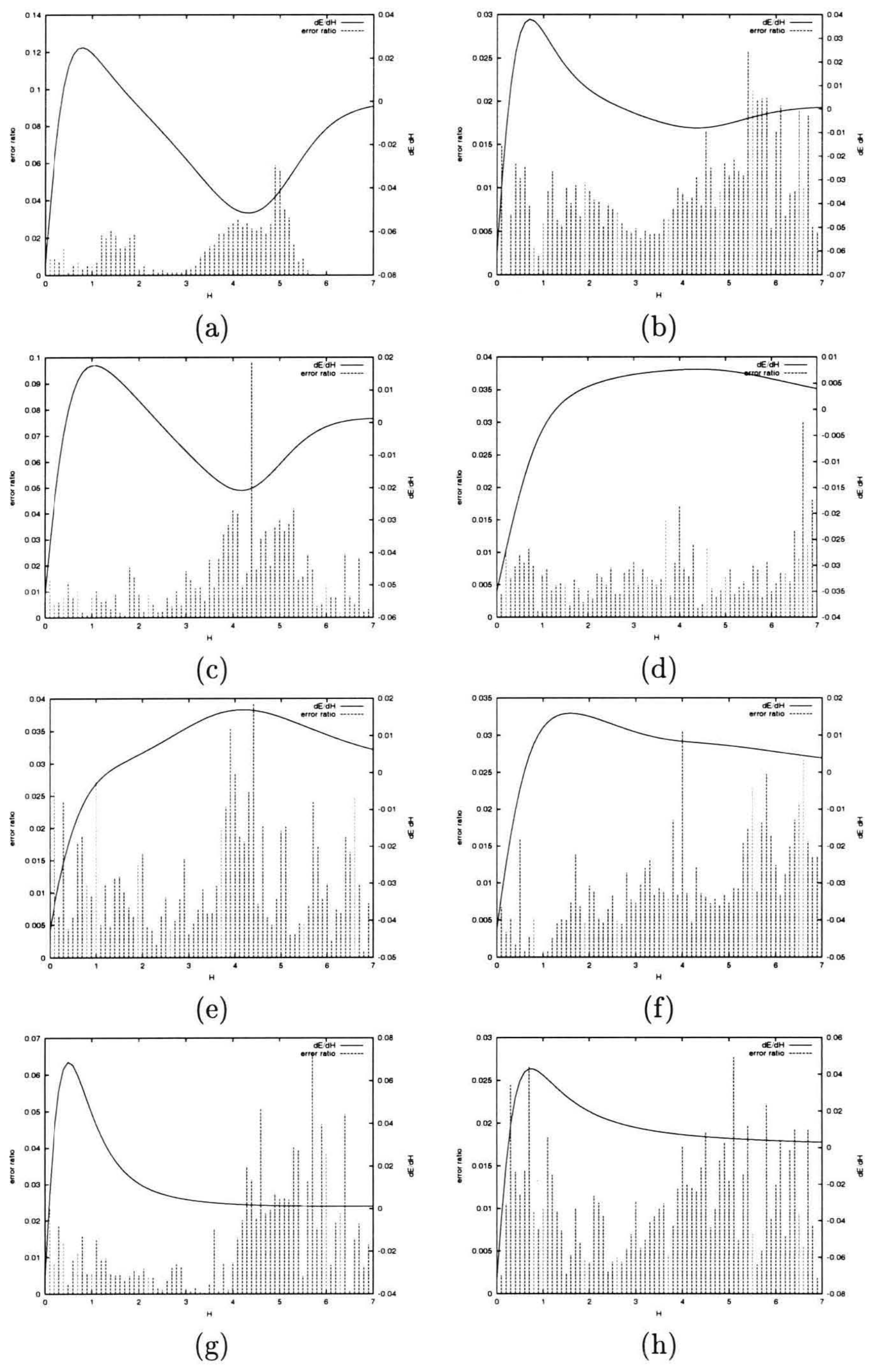
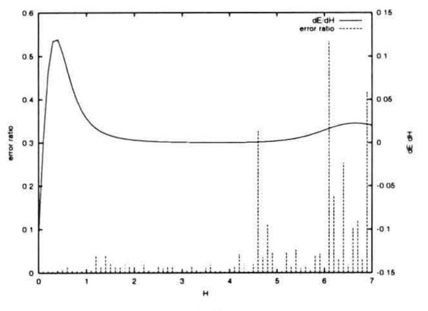
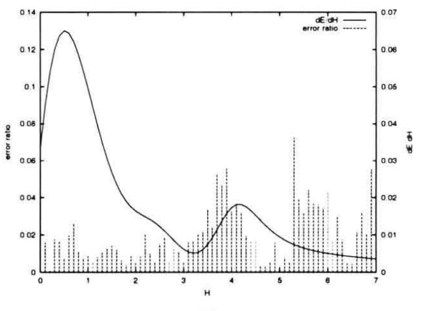
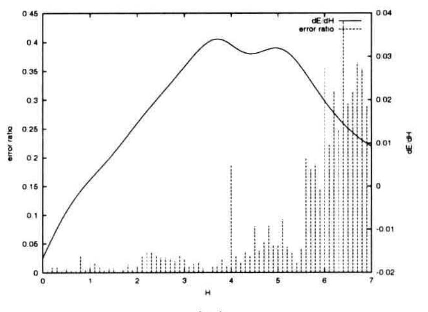
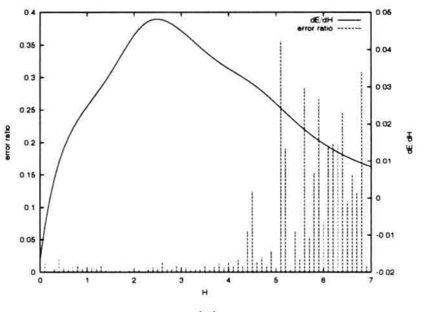


Figure 5.6: **Generalisation Performance:** These figures show typical generalisation performances for networks trained with the respective DFAs from Figure 5.2 as prior knowledge. Networks were trained with different values of the inductive bias H. It plots the function $\partial E/\partial H$ as a function of the inductive bias strength H.


(i)


(j)


(k)


(l)

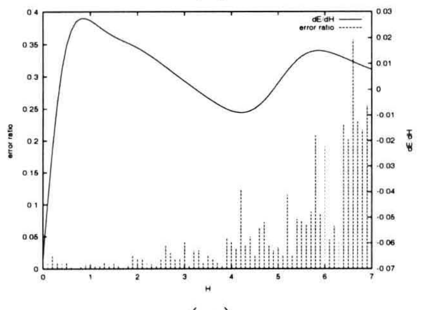

(m)

Figure 5.7: **Generalisation Performance:** These figures show typical generalisation performances for networks trained with the respective DFAs from Figure 5.3 as prior knowledge. Networks were trained with different values of the inductive bias H. It plots the function $\partial E/\partial H$ as a function of the inductive bias strength H.

DFA	Inductive Bias H	training epochs		generalisation error	
		μ	σ	μ	σ
b	$H = 4$	136.60	5.06	0.817%	0.125%
	$H_{heuristic}$	350.20	16.62	0.814%	0.245%
c	$H = 4$	2197.50	570.31	3.088%	1.357%
	$H_{heuristic}$	3078.50	1948.40	2.157%	1.020%
d	$H = 4$	274.70	10.06	1.754%	0.439%
	$H_{heuristic}$	287.90	19.12	1.331%	0.710%
e	$H = 4$	358.50	23.63	1.136%	0.812%
	$H_{heuristic}$	358.70	22.83	1.291%	0.933%
f	$H = 4$	334.80	15.01	1.450%	0.743%
	$H_{heuristic}$	335.30	10.60	0.724%	0.368%
g	$H = 4$	504.30	47.06	0.472%	0.267%
	$H_{heuristic}$	477.30	29.99	1.066%	0.322%
h	$H = 4$	387.00	24.27	0.771%	0.259%
	$H_{heuristic}$	457.00	49.48	1.194%	0.542%

Table 5.3: **Results for Partial DFA Encoding:** The table shows the average and standard deviation for the training time and generalisation performance, respectively, of multiple runs of the neural networks encoded with the partial prior information from Figure 5.2b-h, as a function of the inductive bias H for the standard choice $H = 4$ and our heuristic $H_{heuristic}$ for choosing H.

and generalisation performance for these networks are shown in Table 5.3 and 5.4.

The trained networks encoded with partial correct knowledge (Figure 5.2b-h) using our heuristic as a means of determining the inductive bias H delivers comparable results with networks encoded with the standard inductive bias $H = 4$ (see Table 5.3). Although no significant performance increase could be seen in using partial correct information, for malicious rules the situation proved quite the contrary (see Table 5.4). Our heuristic was able to gain as much information from the malicious rules as possible to consistently deliver better generalisation performances and significantly better training times. In most cases our heuristic determined a small inductive bias H (see Figures 5.5 and 5.7), suggesting that it does not have sufficiently large confidence in the initial domain theory explaining the training data or that the concept described by the theory and data are not similar.

DFA	Inductive Bias H	training epochs		generalisation error	
		μ	σ	μ	σ
a	$H = 4$	64.00	0	2.800%	0%
	average	268.66	0	1.376%	0%
	optimal training	0.00	0	0.000%	0%
	$H_{heuristic}$	54.00	0	2.809%	0%
i	$H = 4$	3117.00	0	0.909%	0%
	average	4212.25	0	5.030%	0%
	optimal training	677.00	0	1.459%	0%
	$H_{heuristic}$	1263.00	0	0.783%	0%
j	$H = 4$	557.00	0	3.249%	0%
	average	1185.44	0	2.101%	0%
	optimal training	410.00	0	1.099%	0%
	$H_{heuristic}$	761.00	0	0.769%	0%
k	$H = 4$	10155.00	0	18.830%	0%
	average	4351.39	0	8.024%	0%
	optimal training	426.00	0	1.524%	0%
	$H_{heuristic}$	3731.00	0	1.274%	0%
l	$H = 4$	4041.00	0	1.390%	0%
	average	4101.30	0	5.549%	0%
	optimal training	367.00	0	0.597%	0%
	$H_{heuristic}$	1032.00	0	0.586%	0%
m	$H = 4$	1015.00	0	4.340%	0%
	average	3345.24	0	5.848%	0%
	optimal training	364.00	0	0.897%	0%
	$H_{heuristic}$	417.00	0	0.261%	0%

Table 5.4: **Results for Full DFA Encoding:** The table shows the average and standard deviation for the training time and generalisation performance, respectively, of multiple runs of the neural networks encoded with the full prior information from Figure 5.2a and Figure 5.3, as a function of the inductive bias H for the standard choice $H = 4$, the average over values of H ranging from 0 to 7 in increments of 0.1, the optimal training performance choice, and our heuristic $H_{heuristic}$ for choosing H. Note that all the standard deviations are zero because all the networks, for a specific DFA, are the same.

5.5 Summary

We have introduced a heuristic for determining the strength of the inductive bias that result in good training performance. The method takes into account the prior knowledge, the available training data, the network architecture and the learning algorithm. We tested the heuristic on two synthetic problems using feedforward and recurrent neural networks, respectively. Preliminary results show that the heuristic gives a good indication into a network's confidence in the initial domain theory. In the next chapter, we will apply the heuristic to feedforward networks that are trained to solve difficult real-world problems.

Chapter 6

Applications

6.1 Molecular Biology

The Human Genome Project is generating a rapidly growing database of DNA sequences. For our purposes, we represent DNA as a linear sequence of characters from the set {`A`,`G`,`T`,`C`} (referred to as nucleotides). Human DNA consists of approximately $3 * 10^9$ nucleotides and the DNA of *E. coli* about $5 * 10^6$ nucleotides, in contrast. Knowing the DNA sequence for any organism and location of its genes in the DNA sequence will lead scientists to the treatment and classification of genetic disorders and improve understanding of the basic units of life. A gene is a portion of the DNA sequence that can be transcribed into a protein. Proteins are the actual workers in cells. Thus, different genes transcribe into different proteins that perform a specific task in the cell.

This wealth of data introduced the need for computer-based algorithms to support and enhance biologists findings and to maybe replace certain experiments typically done in the laboratory.

6.1.1 Promoter Recognition: Problem Statement

We are applying our heuristic method for choosing the *inductive bias* in knowledge-based neurocomputing to the published problem [91, 88] of identifying prokaryotic promoter sites in sequenced DNA [82]. Prokaryotes are single-celled organisms that do not have a nucleus and promoters are short sequences of DNA which precede genes.

```
promoter :- contact, conformation.
contact :- minus-35, minus-10.
minus-35 :- @-37 'CTTGAC-'.  minus-10 :- @-14 'TATAAT--'.
minus-35 :- @-37 '-TTGACA'.  minus-10 :- @-14 '-TATAAT-'.
minus-35 :- @-37 '-TTG-CA'.  minus-10 :- @-14 '-TA-A-T-'.
minus-35 :- @-37 '-TTGAC-'.  minus-10 :- @-14 '--TA---T'.
conformation :- @-45 'AA--A'.
conformation :- @-45 'A---A', @-28 'T---T-AA--T-' ,@-04 'T'.
conformation :- @-49 'A----T', @-27 'T----A--T-TG', @-01 'A'.
conformation :- @-47 'CAAT-TT-AC', @-22 'G---T-C',
@-08 'GCGCC-CC'.
```

Table 6.1: **Knowledge Base for Promoter Recognition:** The rules, in `PROLOG` notation, specify where a sequence of DNA is likely to occur relative to a reference point. This reference point occurs 7 nucleotides to the left of the end of the DNA sequence. The notation `@-40 'AT-C'` means that an `'A'` must appear 40 nucleotides to the left of the reference point, a `'T'` must appear 39 nucleotides to the left of the reference point. The `'-'` indicates that any nucleotide will suffice.

The end of genes are easily located through character sequences known as *stop codons*. The beginning of the gene sequences are not so easily found. Thus, identification of promoters aids in locating genes in uncharacterised DNA sequences. Researchers have developed an understanding of the structure of promoters but not a fool-proof way of classifying promoters without *wet* biological experiments. In these experiments, the protein *RNA polymerase* is used to locate promoters; if the protein binds to that specific sequence then that sequence is a promoter. This forms the basis of biological classification of promoters.

6.1.2 Data and Initial Domain Theory

The data for the recognition of promoters were used from the machine learning repository of the University of California [57]. The data set consisted of 106 examples (53 positive and 53 negative).

The rules for the promoter recognition task in Table 6.1 were derived from the biological literature [64]. They use a notation to specify where a sequence of DNA is likely to occur relative to a reference point[1]. This reference point occurs 7 nucleotides to the left of the end of the DNA sequence. So the notation `@-40 'AT-C'` means that a `'A'`

[1]The reference point for promoter recognition identifies the site at which gene transcription begins.

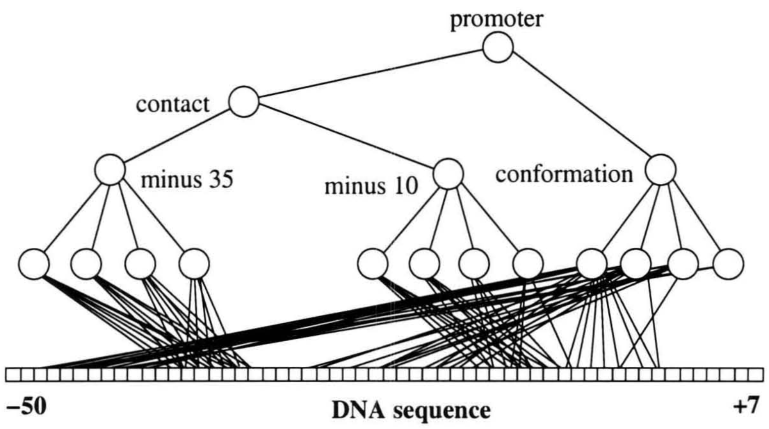


Figure 6.1: **KBANN for Promoter Recognition:** The structure of the knowledge-based neural network derived from the rules in Table 6.1. Random-initialised, low-weighted links are not shown.

must appear 40 nucleotides to the left of the reference point, a `'T'` must appear 39 nucleotides to the left of the reference point, *etc.* The `'-'` indicates that any nucleotide will suffice.

According to the rule set, there are two sites at which the *RNA polymerase* binds to the DNA, `minus-10` and `minus-35` [2]. The `conformation` rule attempts to simulate the three-dimensional structure of DNA and to make sure that the `minus-10` and `minus-35` sites are spatially aligned.

6.1.3 Knowledge Encoding

The initial domain theory in Table 6.1 was encoded using the KBANN architecture. The structure of the network, before the addition of low-weighted random-initialised weights, are shown in Figure 6.1. For a sequence location, four input units were programmed to represent the set {`A,G,T,C`}.

6.1.4 Network Performance

We used KBANNs with neurons with sigmoidal discriminant functions and the standard quadratic error function E (refer to Equation 3 in Section 4.2.2). A network correctly classified an example, during training, if its output was within $\varepsilon = 0.25$ of the desired

[2]These two rules are named according to their position from the reference point.

output and for testing, to within $\psi = 0.5$ from the desired output. We chose the learning rate $\alpha = 0.1$ and the momentum $\eta = 0.1$[3], and trained all networks until one of the three following stopping criteria was satisfied:

- On 99% of the training examples, the activation of every output unit was within ε of correct, or
- a network has been trained for 5,000 epochs, or
- a network classified at least 90% of the training examples correctly, but has not improved its ability to classify the training examples for five epochs.

In general, networks stopped training on the first criterion.

We performed a 10-fold cross-validation on the data. Each fold contained 96 of the examples from the data set (except the last fold which had 90 examples); the remaining examples were used for testing. We ran 10 experiments for each fold with different random initialised weights from the interval $[-0.1, 0.1]$. We measured the training and generalisation performance for values of H ranging from 0 to 7 in increments of 0.1 [4].

Figures 6.2a,c,e,g represent typical training performances for each of the different folds, respectively. Figures 6.2b,d,f,h represent the corresponding generalisation performances. From the graphs of the function $\partial E/\partial H$, we observe that the function $|\partial E/\partial H = 0|$ has a maximum near the inductive bias $H \approx 1.9$. Choosing the inductive bias H such that the gradient of the error function, in the direction of the prior knowledge, to be a maximum value ($H \approx 1.9$ for this particular case), result in very good performances for this difficult real-world problem.

Average and standard deviation results of the cross-validation for the training and generalisation performances, respectively, are shown in Table 6.2 as a function of the inductive bias H for the standard choice $H = 4$, the average over values of H ranging from 0 to 7 in increments of 0.1, the optimal training performance choice, and using our heuristic $H_{heuristic}$ to determine the strength of the inductive bias H.

[3]We found that networks encoded with this prior knowledge were most likely to converge to a solution with these parameters.

[4]The number of networks trained for this problem totalled 7100.

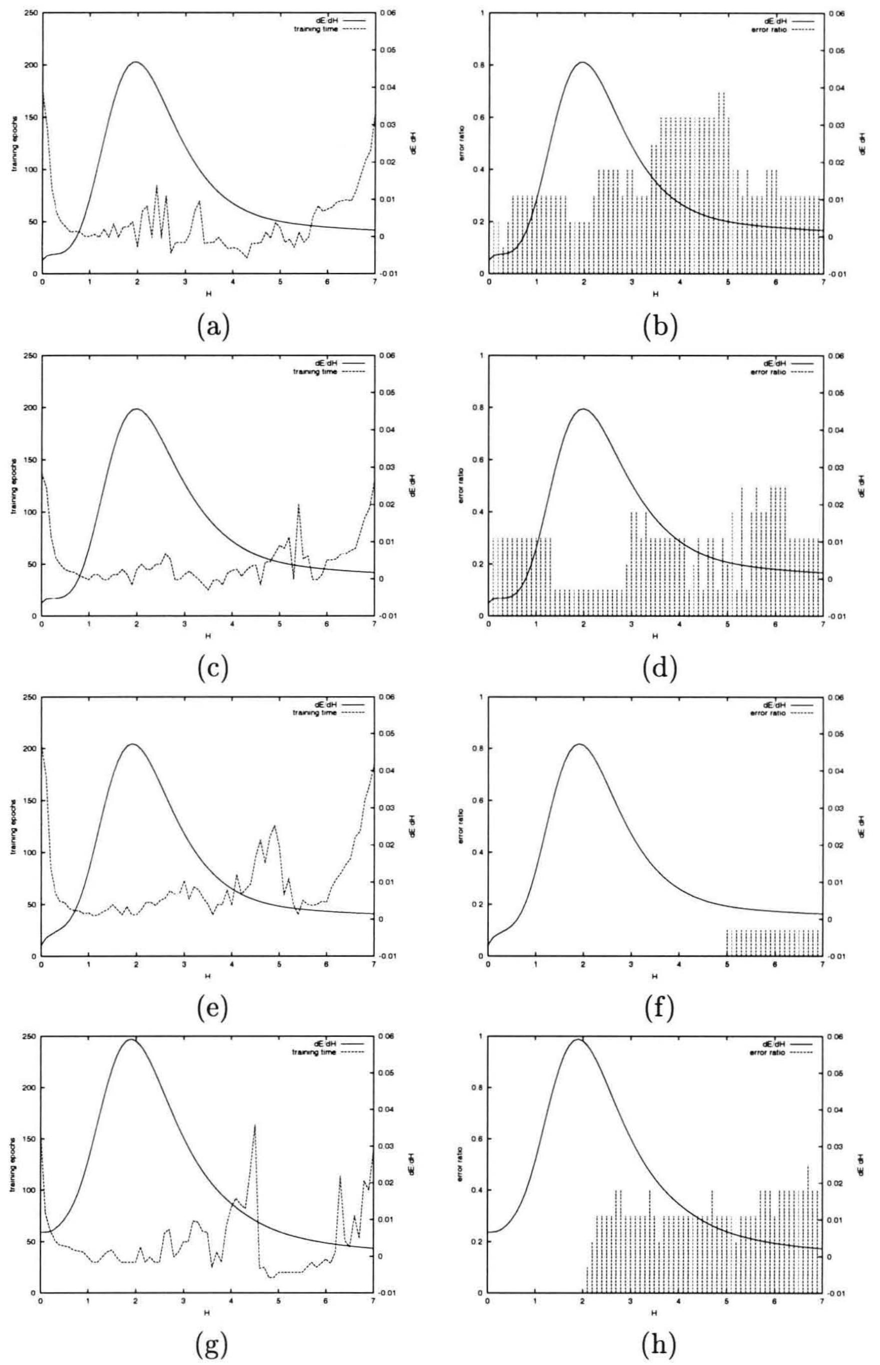


Figure 6.2: **Cross-validation Results for Promoter Recognition:** These figures show typical training times and the corresponding generalisation performance for networks trained with different values of the inductive bias H. It plots the function $\partial E/\partial H$ as a function of the inductive bias strength H. Choosing H such that the function $|\partial E/\partial H = 0|$ is maximal results in good performance.

Inductive Bias H	training epochs		generalisation error	
	μ	σ	μ	σ
$H = 4$	41.95	20.10	11.5%	17.9%
average	70.28	27.71	12.2%	11.1%
optimal training	23.03	6.16	14.4%	18.9%
$H_{heuristic}$	41.04	9.67	7.8%	9.5%

Table 6.2: **Results of Cross-validation for Promoter Recognition:** The table shows average and standard deviation for the training time and generalisation performance of the neural networks, respectively, as a function of the inductive bias H for the standard choice $H = 4$, the average over values of H ranging from 0 to 7 in increments of 0.1, the optimal training performance choice, and our heuristic $H_{heuristic}$ for choosing H.

Our results show a reduction in training time of 2% over the standard choice of $H = 4$ for the inductive bias and are within 78% of the optimal training time. Generalisation performance improved with 32.2% over the standard inductive bias $H = 4$. Using our heuristic for choosing the strength of the explicit inductive bias H over the average choice of the inductive bias, resulted in an improvement of 41.6% and 36.1% for training and generalisation performances, respectively.

6.1.5 Knowledge Extraction

We used the TREPAN algorithm described in section 4.2.5 to extract knowledge in the form of decision trees from the trained neural networks. The algorithm was used *per se* with its default settings. The data sample size was limited to a 1000 examples and the tree size to 15 nodes.

Decision trees where extracted for all the networks trained by the cross-validation method. For each of these decision trees, the fidelity and accuracy was measured on the training and test set, as determined by the cross-validation, as well as its comprehensibility. *Fidelity* is defined as the percentage of examples on which the classification made by a tree corresponds with its neural network counterpart and *accuracy* is defined as the percentage of examples that are correctly classified by a tree. A decision tree's *comprehensibility* was measured by counting the number of internal nodes (after the simplification of the tree), the number of leaves, and the number of features used in the logical tests in the nodes.

We compared the performance of the decision trees extracted from neural networks

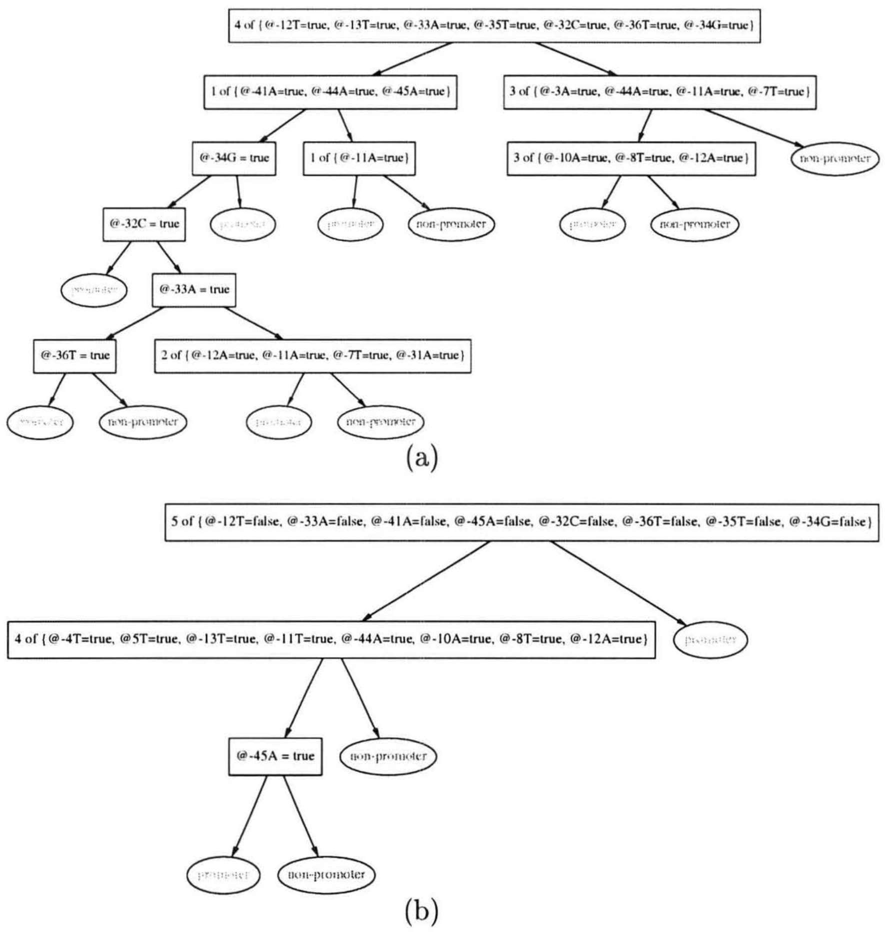

4 of { @-12T=true, @-13T=true, @-33A=true, @-35T=true, @-32C=true, @-36T=true, @-34G=true }
1 of { @-41A=true, @-44A=true, @-45A=true }
3 of { @-3A=true, @-44A=true, @-11A=true, @-7T=true }
@-34G = true
1 of { @-11A=true }
3 of { @-10A=true, @-8T=true, @-12A=true }
non-promoter
@-32C = true
non-promoter
non-promoter
@-33A = true
@-36T = true
2 of { @-12A=true, @-11A=true, @-7T=true, @-31A=true }
non-promoter
non-promoter
(a)
5 of { @-12T=false, @-33A=false, @-41A=false, @-45A=false, @-32C=false, @-36T=false, @-35T=false, @-34G=false }
4 of { @-4T=true, @5T=true, @-13T=true, @-11T=true, @-44A=true, @-10A=true, @-8T=true, @-12A=true }
promoter
@-45A = true
non-promoter
non-promoter
(b)

Figure 6.3: **Extracted Decision Trees for Promoter Recognition:** (a) Typical extracted decision tree for neural networks encoded with the standard inductive bias $H = 4$ (b) typical extracted decision tree for neural networks encoded using our heuristic to determine the inductive bias. Left branches in a tree corresponds to *true* conditions and branches to the right with *false* conditions.

Inductive Bias H	training set fidelity		testing set fidelity	
	μ	σ	μ	σ
$H = 4$	92.66%	3.07%	90.64%	10.51%
average	91.16%	3.98%	87.91%	11.94%
$H_{heuristic}$	93.51%	2.39%	88.74%	9.88%

Table 6.3: **Extracted Decision Trees for Promoter Recognition - Fidelity Results:** The table shows average and standard deviation for the training and testing set fidelity of the extracted decision trees, respectively, as a function of the inductive bias H for the standard choice $H = 4$, the average over values of H ranging from 0 to 7 in increments of 0.1, and our heuristic $H_{heuristic}$ for choosing H.

Inductive Bias H	training set accuracy		testing set accuracy	
	μ	σ	μ	σ
$H = 4$	91.72%	2.99%	85.66%	17.20%
average	89.84%	5.00%	79.86%	16.92%
$H_{heuristic}$	93.41%	2.46%	84.66%	13.64%

Table 6.4: **Extracted Decision Trees for Promoter Recognition - Accuracy Results:** The table shows average and standard deviation for the training and testing set accuracy of the extracted decision trees, respectively, as a function of the inductive bias H for the standard choice $H = 4$, the average over values of H ranging from 0 to 7 in increments of 0.1, and our heuristic $H_{heuristic}$ for choosing H.

Inductive Bias H	# internal nodes		# leaves		# feature references	
	μ	σ	μ	σ	μ	σ
$H = 4$	5.39	2.77	6.39	2.77	22.88	10.19
average	5.06	3.24	6.06	3.24	25.92	15.72
$H_{heuristic}$	2.13	1.78	3.13	1.78	16.39	8.21

Table 6.5: **Extracted Decision Trees for Promoter Recognition - Comprehensibility Results:** The table shows average and standard deviation for the comprehensibility of the extracted decision trees, respectively, as a function of the inductive bias H for the standard choice $H = 4$, the average over values of H ranging from 0 to 7 in increments of 0.1, and our heuristic $H_{heuristic}$ for choosing H.

encoded through the use of our heuristic to determine the strength of the inductive bias H, the standard inductive bias $H = 4$, and the average choice of the inductive bias. Figure 6.3 shows typical examples of the extracted decision trees delivered by the TREPAN algorithm. It can be seen that decision trees extracted from the neural networks encoded using our heuristic (Figure 6.3b) are more comprehensible than decision tree extracted using the standard inductive bias (Figure 6.3a). The full results are shown in Tables 6.3, 6.4, and 6.5.

Not only are the trees more comprehensible (see Table 6.5) than the trees extracted using the standard inductive bias or trees extracted using the average inductive bias, they achieve 1.84% higher accuracy on the training set, than the standard choice and are 3.97% and 6.01% more accurate, respectively on the training and test set, than the average choice of the inductive bias. Although the extracted trees using the standard inductive bias perform better on the fidelity and accuracy of the testing set, the drop in fidelity for the trees extracted using our heuristic can be pinned to the trade-off between fidelity and comprehensibility. Using our heuristic for determining the strength of the inductive bias produces decision trees that have 60.48% and 57.91% less internal nodes than trees extracted using the standard inductive bias and trees extracted using the average choice of the inductive bias, respectively; leaves are 51.02% and 48.35% less, respectively; and feature references are reduced by 28.37% and 36.77%, respectively.

Refinement of the initial domain theory, for the classification of promoters in unrecognised DNA sequences, using our heuristic, produced a more comprehensible domain theory with a minimal degradation in testing accuracy. The higher comprehensibility of the refined domain theory lends itself to being more easily understood and are thus easier to apply. We think the small trade-off of accuracy for comprehensibility are justified in this particular case.

6.2 Medical Diagnoses

Medical decision making is well-suited for the application of artificial intelligence techniques [49, 77]. Expert knowledge in the medical field is often incomplete due to the variability and the complexity of disease processes. Most practical learning problems lie somewhere between the two extremes of plentiful data without prior knowledge and perfect prior knowledge with scarce data. Combining inductive with analytical learning methods holds the promise of exploiting the strengths of the two approaches while alleviating their respective weaknesses. This hybrid approach is applicable to many practical problems including computer-assisted medical diagnosis.

For an overview of neural network applications in medicine, see e.g. [8, 71]. For a brief summary of neural network methods applied to clinical diagnosis and medical imaging, see [76]. An overview of other data mining techniques with selected medical applications can be found in [49].

6.2.1 ^{31}P MRS of Normal Breast Tissue: Problem Statement

We applied our heuristic method for choosing the *inductive bias* in knowledge-based neurocomputing to the published problem [76] of classifying the different stages of the menstrual cycle through magnetic resonance spectroscopy (MRS) of the normal

		metabolites						
volunteer	phase	PDE	PCr	PME	Pi	γ-ATP	α-ATP	β-ATP
2	ef	0.6323	0.1467	0.1588	0.5002	0.1723	0.2115	0.2587
2	lf	0.4324	0.0047	0.2658	0.1763	0.1632	0.2230	0.2074
2	el	0.6133	0.0061	0.1229	0.2855	0.2010	0.2676	0.1649
2	ll	0.6300	0.0799	0.1226	0.3451	0.2200	0.3750	0.2025
3	ef	0.9466	0.0849	0.3191	0.3622	0.3447	0.5459	0.2817
3	lf	0.6604	0.0060	0.2177	0.1159	0.3330	0.2998	0.1959
3	el	0.6429	0.0704	0.0234	0.2234	0.1150	0.2834	0.3028
3	ll	0.9270	0.0077	0.0664	0.3381	0.3571	0.4899	0.2868
4	ef	0.6100	0.0827	0.3381	0.1255	0.2466	0.2817	0.1207
4	lf	0.5504	0.1298	0.1907	0.1723	0.2388	0.3781	0.3413
4	el	0.5660	0.0046	0.0984	0.0972	0.3028	0.3099	0.2811
4	ll	0.3833	0.0941	0.2126	0.0694	0.2453	0.3498	0.1958
5	ef	1.0000	0.0099	0.2012	0.4226	0.2298	0.4669	0.2817
5	lf	0.5651	0.0690	0.2543	0.4362	0.2562	0.2832	0.2167
5	el	0.7325	0.0512	0.1199	0.1711	0.2467	0.1846	0.2740
5	ll	0.6381	0.0060	0.1993	0.2011	0.1993	0.1365	0.1827

Table 6.6: **Data for MRS of Breast Tissue:** Metabolic changes during the four phases of the menstrual cycle. Values correspond to the normalised peak area of seven metabolites extracted from each spectrum.

breast tissue [85]. Fluctuations in hormone levels during the different phases[5] of the menstrual cycle produce variations in metabolite levels of the breast tissue in women. This well-established observation [76] can be monitored by means of *in vivo* ^{31}P magnetic resonance spectroscopy (MRS). The complexity of the test results requires expert knowledge for their analysis. For a detailed discussion of this complex real world problem and the knowledge acquisition methods, see [76].

6.2.2 Data and Initial Domain Theory

^{31}P MRS is a non-invasive technique for observing phosphorus-containing metabolites and intracellular pH. It allows the observation of metabolic activity in cells as it detects the magnetic resonance emitted by cells when exposed to a magnetic field and radio signals. A ^{31}P spectrum of the sampled breast tissue is the result of this method. Peaks in the spectrum correlates to different metabolites (PME, PDE, PCr, Pi, α-ATP, β-ATP and γ-ATP) of the cells. The area under such a peak corresponds to the intensity of the resonance signal for specific nuclei of the cells in the tissue sample. These intensities are used as the data for analysing the different stages of women's menstrual cycles.

[5]Four menstrual phases: early follicular (ef), late follicular (lf), early luteal (el), late luteal (ll).

```
ef :- prolif rate                     lf :- prolif rate
ef :- pi level                        el :- metab act
ll :- metab act                       el :- pme level
ll :- pme level                       el :- pi level
ll :- pi level                        ll :- prolif rate
lf :- pde level                       el :- bio changes
ef :- bio changes                     ll :- bio changes
prolif rate :- pme level              lf :- bio changes, pme level
bio changes :- pde level, pme level
```

Table 6.7: **Knowledge Base for MRS of Breast Tissue:** The rules, in PROLOG form, of the knowledge extracted from experts for the classification of women's menstrual cycle using ^{31}P MRS.

The data[6] contains 16 *in vivo* ^{31}P MR spectra obtained from four female pre-menopausal volunteers ranging in age from 21 to 45 (see Table 6.6). They all had regular menstrual cycles and none were using the contraceptive pill. Four spectra from each volunteer were taken, one at each of the different stages of the menstrual cycle. Seven values were extracted from each spectrum. Each specific normalised value corresponds to a peak area of a specific metabolite present in the spectrum.

The initial domain theory (see Table 6.7) was extracted from experts. For a detailed explanation of the knowledge acquisition process, see [76].

6.2.3 Knowledge Encoding

The prior knowledge encoded KBANN are shown in Figure 6.4. We used the real-value encoding of [76] instead of the input encoding method proposed in [91] for the purpose of comparison.

6.2.4 Network Performance

We performed a 4-fold cross-validation on the data. Each fold contained data from 3 volunteers; the remaining volunteer's data was used for testing. We ran 10 experiments for each fold with different random initialised weights from the interval $[-0.1, 0.1]$. We measured the training and generalisation performance for values of H ranging from 0 to 7 in increments of 0.1 [7]. All KBANN networks were trained until one of the three

[6]Data provided by the CRC Clinical Magnetic Resonance Research Group, Royal Marsden Hospital, Sutton.
[7]The number of networks trained for this problem totalled 2840.

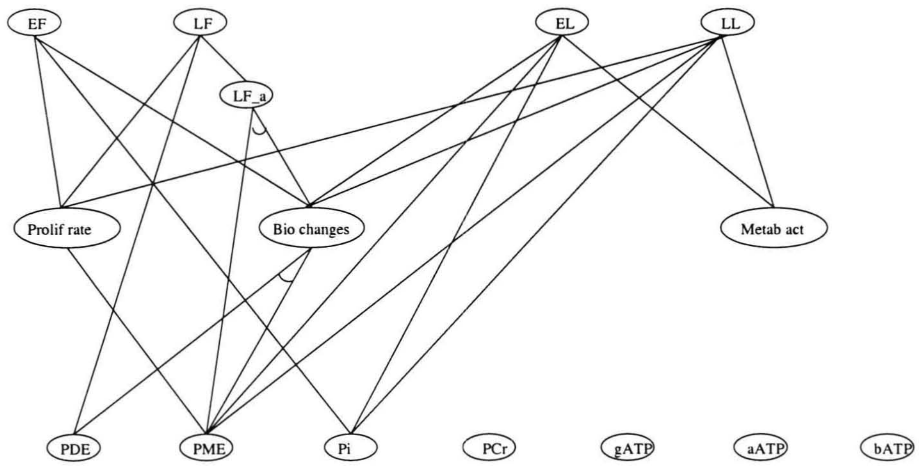


Figure 6.4: **KBANN for MRS of Breast Tissue:** The network structure after the prior information have been encoded into the feedforward network according to the KBANN method. Low-weighted, random-initialised connections are not shown.

following stopping criteria was satisfied:

- On 99% of the training examples, the activation of every output unit was within $\varepsilon = 0.25$ of the desired output, or
- a network had been trained for 15,000 epochs, or
- a network classified at least 90% of the training examples correctly, but had not improved its ability to classify the training examples for five consecutive epochs.

Neurons had sigmoidal discriminant functions and all networks were trained using the standard quadratic error function E (refer to Equation 3 in Section 4.2.2). A network correctly classified an example if its output was within $\varepsilon = 0.25$ and $\psi = 0.5$ of the desired output, for training and testing respectively. We chose the learning rate $\alpha = 0.5$ and momentum $\eta = 0.7$ [8].

Figure 6.5 represent typical simulation results for each of the different folds, respectively. The scarce data for this complex medical domain poses a big challenge. We observe that our heuristic for choosing an *explicit inductive bias* yields good generalisation and training time performance. Variations from fold to fold in training and generalisation performance are due to the limited data set, as for each fold, 25% of the data is set aside for testing.

[8]These parameters are not necessarily optimal for the networks.

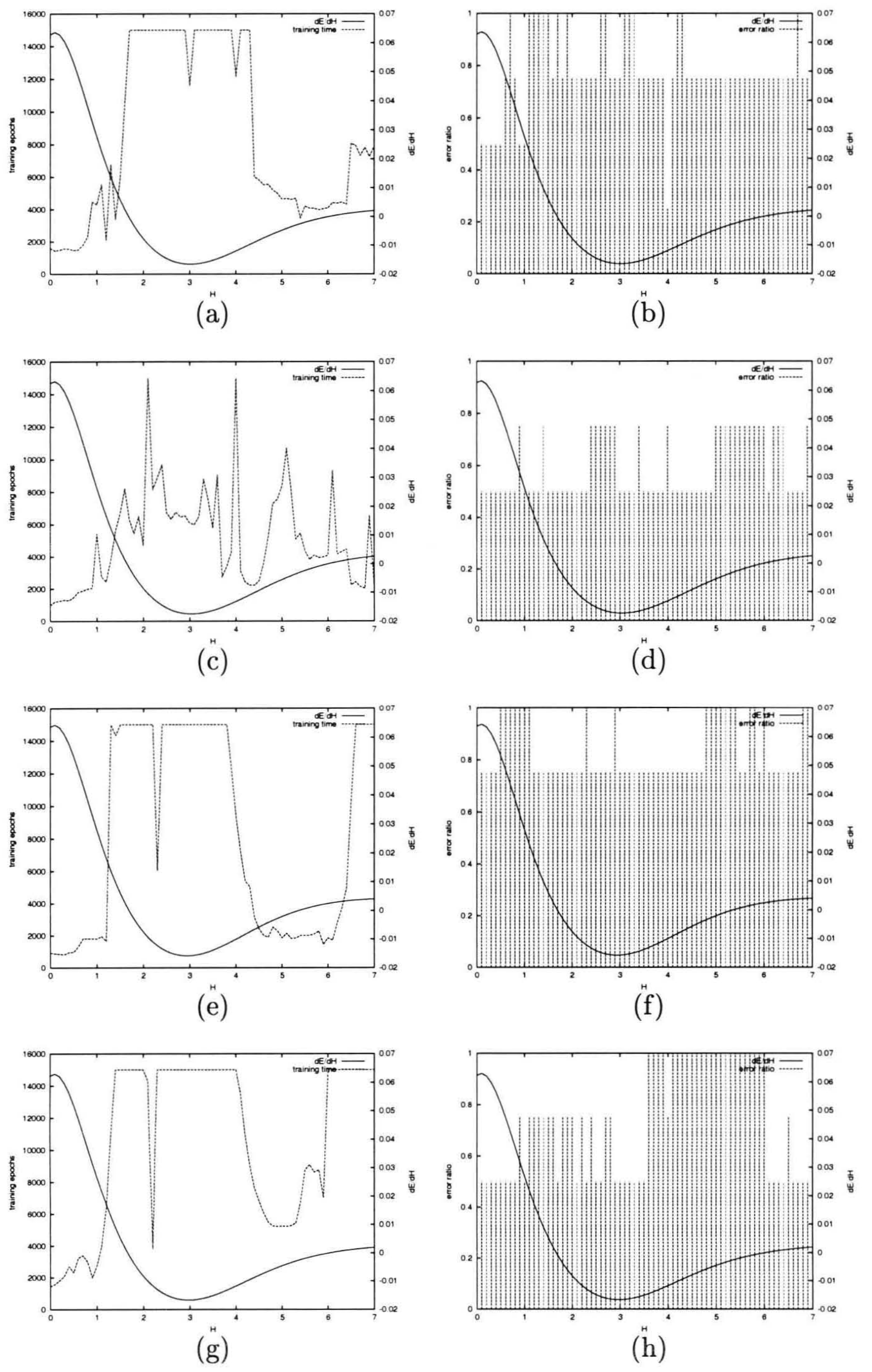


Figure 6.5: **Cross-validation Results for MRS of Breast Tissue:** The figures show typical training times and the corresponding generalisation performance for networks trained with different values of the inductive bias H, for the four different folds, respectively. It plots the function $\partial E/\partial H$ as a function of the inductive bias strength H. Choosing H such that the function $|\partial E/\partial H = 0|$ is maximal results in good performance.

From the graph of the function $\partial E/\partial H$, we observe that the function $|\partial E/\partial H = 0|$ has a maximum near the inductive bias $H \approx 0.1$. This confirms that the initial domain theory does not fully explain the given training data. A weak inductive bias seems to indicate the programmed network's low confidence in the prior knowledge. We speculate that it is the small training data set and the small overlap between the initial domain theory and the data that leads our heuristic to choose a weak inductive bias. In applications where the initial domain theory and the training data represent similar concepts, we have observed that they have a synergistic effect on the training and generalisation performance of neural networks [82].

Average and standard deviation results of the cross-validation for the training and generalisation performances, respectively, are shown in Table 6.8 as a function of the inductive bias H for the standard choice $H = 4$, the average over values of H ranging from 0 to 7 in increments of 0.1, the optimal training performance choice, and using our heuristic $H_{heuristic}$ to determine the strength of the inductive bias H.

The initial domain theory only explains 20% of the data. Thus, an average error close to 60% for the cross-validation experiment using our heuristic to encode the networks can be seen as a very good result for this difficult domain. Our heuristic for determining the strength of the explicit inductive bias resulted in almost 17% improvement of the generalisation performance compared to the average choice of the inductive bias. This also exceeds the performance for the standard inductive bias $H = 4$ by almost 9%.

Our results show a good relative reduction in training time where the inductive bias is chosen according to our heuristic. Training times are reduced by more than 82% compared to the average choice of the inductive bias H and reduced by 88% compared to the standard inductive bias $H = 4$; our training times are within 16% of optimal training times. Note that we made no efforts to optimise the training parameters.

6.2.5 Knowledge Extraction

We extracted decision trees using the TREPAN algorithm described in Section 4.2.5. The algorithm was used in a similar way as for the Promoter Recognition problem. The TREPAN algorithm was limited to a sample set of a 1000 examples and the extracted trees were limited to 15 nodes.

Each of the neural networks trained through the cross-validation method as described

Inductive Bias H	training epochs		generalisation error	
	μ	σ	μ	σ
$H = 4$	11955	3804	66.3%	24.1%
average	8075	1842	72.8%	9.6%
optimal training	1198	263	60.6%	15.7%
$H_{heuristic}$	1396	235	60.6%	12.3%

Table 6.8: **Results of Cross-validation for MRS of Breast Tissue:** The table shows average and standard deviation for the training time and generalisation performance, respectively, as a function of the inductive bias H for the standard choice $H = 4$, the average over values of H ranging from 0 to 7 in increments of 0.1, the optimal training performance choice, and our heuristic $H_{heuristic}$.

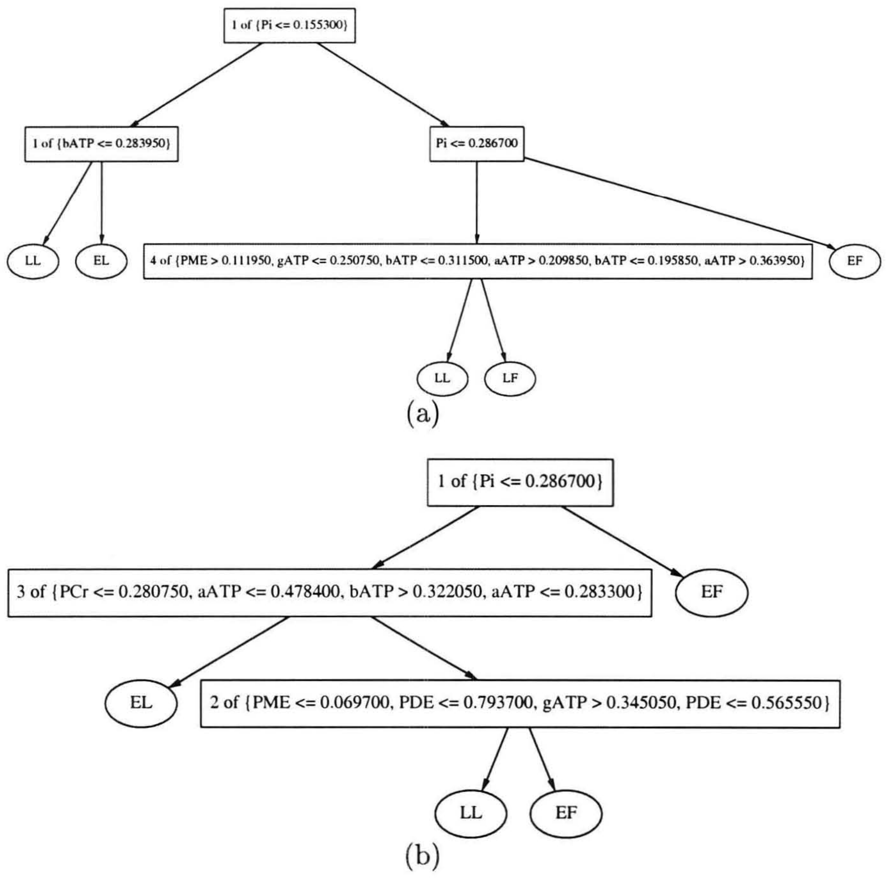

1 of {Pi <= 0.155300}
1 of {bATP <= 0.283950}
Pi <= 0.286700
LL
EL
4 of {PME > 0.111950, gATP <= 0.250750, bATP <= 0.311500, aATP > 0.209850, bATP <= 0.195850, aATP > 0.363950}
EF
LL
LF
(a)
1 of {Pi <= 0.286700}
3 of {PCr <= 0.280750, aATP <= 0.478400, bATP > 0.322050, aATP <= 0.283300}
EF
EL
2 of {PME <= 0.069700, PDE <= 0.793700, gATP > 0.345050, PDE <= 0.565550}
LL
EF
(b)

Figure 6.6: **Extracted Decision Trees for MRS of Breast Tissue:** (a) Typical extracted decision tree for neural networks encoded with the standard inductive bias $H = 4$ (b) typical extracted decision tree for neural networks encoded using our heuristic to determine the inductive bias. Left branches in a tree corresponds to *true* conditions and branches to the right with *false* conditions.

Inductive Bias H	training set fidelity		testing set fidelity	
	μ	σ	μ	σ
$H = 4$	61.87%	16.75%	58.75%	24.08%
average	59.70%	15.36%	57.53%	27.39%
$H_{heuristic}$	58.13%	12.71%	58.13%	31.81%

Table 6.9: **Extracted Decision Trees for MRS of Breast Tissue - Fidelity Results:** The table shows average and standard deviation for the training and testing set fidelity of the extracted decision trees, respectively, as a function of the inductive bias H for the standard choice $H = 4$, the average over values of H ranging from 0 to 7 in increments of 0.1, and our heuristic $H_{heuristic}$ for choosing H.

Inductive Bias H	training set accuracy		testing set accuracy	
	μ	σ	μ	σ
$H = 4$	58.54%	16.92%	33.13%	19.68%
average	57.47%	14.82%	37.05%	16.35%
$H_{heuristic}$	55.40%	11.26%	38.75%	15.76%

Table 6.10: **Extracted Decision Trees for MRS of Breast Tissue - Accuracy Results:** The table shows average and standard deviation for the training and testing set accuracy of the extracted decision trees, respectively, as a function of the inductive bias H for the standard choice $H = 4$, the average over values of H ranging from 0 to 7 in increments of 0.1, and our heuristic $H_{heuristic}$ for choosing H.

in the previous sections delivered a decision tree through the TREPAN knowledge extraction algorithm. For each of these decision trees, the fidelity and accuracy was measured on the training and test set, as determined by the cross-validation, as well as the comprehensibility of the trees. *Fidelity* is defined as the percentage of examples on which the classification made by a tree corresponds with its neural network counterpart and *accuracy* is defined as the percentage of examples that are correctly classified by a tree. A decision tree's *comprehensibility* was measured by counting the number of internal nodes (after the simplification of the tree), the number of leaves, and the number of features used in the logical tests in the nodes.

We compared the performance of the decision trees extracted from neural networks encoded through the use of our heuristic to determine the strength of the inductive bias H, the standard inductive bias $H = 4$, and the average choice of the inductive bias. Figure 6.6 show typical examples of the extracted decision trees delivered by the

Inductive Bias H	# internal nodes		# leaves		# feature references	
	μ	σ	μ	σ	μ	σ
$H = 4$	3.23	1.01	4.23	1.01	8.1	2.68
average	3.22	0.81	4.22	0.81	7.23	2.92
$H_{heuristic}$	2.9	0.92	3.9	0.92	6.53	2.44

Table 6.11: **Extracted Decision Trees for MRS of Breast Tissue - Comprehensibility Results:** The table shows average and standard deviation for the comprehensibility of the extracted decision trees, respectively, as a function of the inductive bias H for the standard choice $H = 4$, the average over values of H ranging from 0 to 7 in increments of 0.1, and our heuristic $H_{heuristic}$ for choosing H.

TREPAN algorithm. It can be seen that decision trees extracted from the neural networks encoded using our heuristic (Figure 6.6b) are more comprehensible than decision tree extracted from networks encoded with the standard inductive bias (Figure 6.6a). The full results are shown in Tables 6.9, 6.10, and 6.11.

Not only are the trees more comprehensible (see Table 6.11) than the trees extracted using the standard inductive bias or trees extracted using the average choice of the inductive bias, they achieve 16.96% and 4.59% higher accuracy on the test set, respectively. Although the extracted trees using the standard inductive bias or the average choice of the inductive bias perform better on the accuracy of the training set, our results can be seen as more consistent because of the lower standard deviation. Using our heuristic for determining the strength of the inductive bias produces decision trees that have 10.22% and 9.94% less internal nodes than trees extracted using the standard inductive bias and trees extracted using the average choice of the inductive bias, respectively; leaves are 7.80% and 7.58% less, respectively; and feature references are reduced by 19.38% and 9.68%, respectively.

The refined domain theory classifies 38.75% of unseen data, whereas the initial domain theory only explained 20% of all the data. This improvement of 93.75% of the domain theory proves that combining neural and symbolic methods are of great importance, especially in the domain medical diagnoses. Medical experts can use this refined domain theory to better classify unknown occurrences. The low actual accuracy of the domain theory can be attributed to the sparse data for this difficult domain.

6.3 Summary

We have applied our heuristic for determining the strength of the *explicit inductive bias* H to problems from the domain of molecular biology and medical diagnoses. The results show that our heuristic performs well on these two complex real-world problems; it outperforms the suggested standard inductive bias $H = 4$ and has on average a much better performance than the average choice of the inductive bias. Not only does our heuristic produce better results in the neurocomputing paradigm, it delivers more concise and comprehensible refined domain theories.

Thus, these results suggest that our heuristic can be applied to various problems and it provides a means for assessing the quality of a initial domain theory as well as the applicability of the available data to a specific problem and the proposed theory.

Chapter 7

Conclusions and Directions for Future Research

In conclusion, we give an overview of the main aspects of this thesis and its contributions. We mention the accomplishments and discuss future areas of research which follow from the described work.

Combining symbolic and neural learning was shown to be important. Above the traditional method, we proposed and evaluated a method for biasing this combination for increased performance. This heuristic method took into account the prior symbolic knowledge, the training data, the training method, and the network architecture. We showed that using this heuristic to determine the inductive bias, we achieved superior results above the standard method of having a fixed bias for combining symbolic and neural methods. We not only achieved better performance for the trained neural networks, but the extracted refined domain knowledge was superior, especially based on its comprehensibility. We applied our heuristic method to well-known synthetic problems as well as difficult published real-world problems.

Thus, we are now capable of better using domain knowledge to our advantage and have gleaned some insights into the importance of having an *explicit* inductive bias that can be adjusted according to some criteria (e.g. in our case, the minimisation of error).

Applying our heuristic to more real-world problems is needed to verify its usability on a broad range of applications. Our heuristic is not dependent on the architecture, domain knowledge, and data for a specific problem, but dependent on the specific learning algorithm used, i.e. learning methods that use gradients to minimise the error. Through our heuristic we used information of the error surface to our advantage, but because of the high dimensionality of typical neural networks, gaining more insight into the relationships between the error surfaces and the vector spaces attributed by the weights parameters will lead to a better understanding of solutions found for a certain set of parameters.

We have empirically verified—for some difficult real-world problems—that our heuristic is effective, at least for feedforward neural networks. It would be useful to establish a mathematical foundation for our heuristic, i.e. an analysis which supports our observations. This may or may not be possible for general cases; however, it would be instructive to be able to make a mathematical argument even for special cases of feedforward networks.

Learning problems exist for which there are either no initial domain theories or for which it is difficult to elicit such prior knowledge. In these cases, we cannot prestructure networks; moreover, we have no guidelines at all for choosing the network architecture. It would be interesting to investigate whether the network architecture can be determined from knowledge extracted from a feedforward network *during* training. Knowledge is then repeatedly extracted and used to initialise a new network; the architecture of this new network is presumably better suited for the learning task than the previous network. A similar knowledge-driven incremental learning method for recurrent networks has not only removed the need for guessing a network architecture, it has also shown excellent training and generalisation performance compared to standard training methods.

Clearly, combining learning methods and gaining insight into these hybrid systems are important open research questions.

Bibliography

[1] Y.S. Abu-Mostafa. Learning from hints in neural networks. *Journal of Complexity*, 6:192–198, 1990.

[2] Y.S. Abu-Mostafa. Hints and the VC dimension. *Neural Computation*, 5:278–288, 1993.

[3] L. Akers, D. Ferry, and R. Grondin. Synthetic neural systems in VLSI. In *An Introduction to Neural and Electronic Systems*, pages 317–336. Academic Press, San Diego, CA, 1990.

[4] R. Andrews and J. Diederich, editors. *Proceedings of the NIPS'96 Rule Extraction from Trained Artificial Neural Network Workshop.* 1996. Snowmass, Colorado.

[5] R. Andrews, J. Diederich, and A. Tickle. A survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*, 8(6):373–389, 1995.

[6] P. Baldi and S. Brunak. *Bioinformatics, the Machine Learning Approach.* MIT Press, Cambridge, MA, 1998.

[7] E. Barnard and D. Casasent. Invariance and neural nets. *IEEE Transactions on Neural Networks*, 2:498–508, 1991.

[8] W.G. Baxt. Application of artificial neural networks to clinical medicine. *Lancet*, 346:1135–1138, 1995.

[9] Y. Bengio. Markovian models for sequential data. Technical Report TR 1049, Department IRO, Univercity of Montreal, Montreal, Canada, 1996.

[10] H.R. Berenji. Refinement of approximate reasoning-based controllers by reinforcement learning. In L.A. Birnbaum and G.C. Collins, editors, *Machine Learning,*

Proceedings of the Eighth International International Workshop, pages 475–479, San Mateo, CA, 1991. Morgan Kaufmann Publishers.

[11] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal ACM*, 36:929–965, 1989.

[12] P.P. Bonissone, Y-T. Chen, K. Goebel, and P.S. Khedkar. Hybrid soft computing systems: Industrial and commercial applications. *Proceedings of the IEEE*, 87(9):1641–1667, 1999.

[13] H. Bourlard and N. Morgan. Hybrid HMM/ANN systems for speech recognition: Overview and new research directions. In *Summer School on Neural Networks*, pages 389–417, 1997.

[14] C.E. Brodley. Recursive automatic bias selection for classifier construction. *Machine Learning Journal*, 20:63–94, 1995.

[15] C. Cardie. Using cognitive biases to guide feature set selection. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, pages 469–471. Lawrence Erlbaum Associated, 1993.

[16] A. Cleeremans, D. Servan-Schreiber, and J. McClelland. Finite state automata and simple recurrent recurrent networks. *Neural Computation*, 1(3):372–381, 1989.

[17] I. Cloete. VL$_1$ANN: Transformation of rules to artificial neural networks. In I. Cloete and J.M. Zurada, editors, *Knowledge-Based Neurocomputing*, chapter 6. MIT Press, Cambridge, MA, 1999.

[18] I. Cloete and J.M. Zurada, editors. *Knowledge-Based Neurocomputing.* MIT Press, Cambridge, MA, 1999.

[19] H. Cobb. Inductive biases in a reinforcement learner. In *Proceedings of the ML92 Workshop on Biases in Inductive Learning*, 1992.

[20] J. Connan and C.W. Omlin. Bibliography extraction with hidden Markov models. Technical Report US-CS-TR-00-06, University of Stellenbosch, Stellenbosch, 2000.

[21] M.W. Craven. *Extracting Comprehensible Models from Trained Neural Networks.* PhD thesis, University of Wisconsin, Madison, WI, 1996.

[22] M.W. Craven and J.W. Shavlik. Extracting tree-structured representations of trained networks. *Advances in Neural Information Processing Systems*, 8, 1996.

[23] M.W. Craven and J.W. Shavlik. Understanding time-series networks: A case study in rule extraction. *International Journal of Neural Systems*, 8(4):373–384, 1997. Special Issue on Noisy Time Series.

[24] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2:303–314, 1989.

[25] M. desJardins and D.F. Gordon. Evaluation and selection of biases in machine learning. *Machine Learning Journal*, 20:1–17, 1995.

[26] T.G. Dietterich, editor. *Machine Learning*, volume 20. Kluwer Academic Publishers, July/August 1995. Special Issue on Bias Evaluation and Selection.

[27] R. Drossu and Z. Obradović. Datamining techniques for designing neural network time series predictors. In I. Cloete and J.M. Zurada, editors, *Knowledge-Based Neurocomputing*, chapter 10, pages 325–367. MIT Press, Cambridge, MA, 1999.

[28] D.H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139–172, 1987.

[29] P. Frasconi, M. Gori, M. Maggini, and G. Soda. Unified integration of explicit rules and learning by example in recurrent networks. *IEEE Transactions on Knowledge and Data Engineering*, 7(2):340–346, 1995.

[30] L. Fu. Learning capacity and sample complexity on expert networks. *IEEE Transactions on Neural Networks*, 7(6):1517–1520, 1996.

[31] L.M. Fu. Integration of neural heuristics into knowledge-based inference. *Connection Science*, 1:325–240, 1989.

[32] L.M. Fu. Rule learning by searching on adapted nets. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 590–595, Anaheim, CA, 1991. AAAI Press.

[33] L.M. Fu. Rule generation from neural networks. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(8):1114–1124, 1994.

[34] L.M. Fu and L.C. Fu. Mapping rule-based systems into neural architecture. *Knowledge-Based Systems*, 3(1):48–56, 1990.

[35] S.I. Gallant. Connectionist expert systems. *Communications of the ACM*, 31(2):152–169, 1988.

[36] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–58, 1992.

[37] C.L. Giles and C.B. Miller. Extracting and learning an unknown grammar with recurrent neural networks. *Advances in Neural Information Processing Systems*, 4, 1992.

[38] M. Golea. On the complexity of rule-extraction from neural networks and network-querying. Technical report, Department of Systems Engineering, Australian National University, Canberra, Australia, 1996.

[39] D. Gordon and D. Perlis. Explicitly biased generalization. *Computational Intelligence*, 5(2):67–81, 1989.

[40] M. Hassoun. *Fundamentals of Artificial Neural Networks.* MIT Press, Cambridge, MA, 1995.

[41] Y. Hayashi. A neural expert system with automated extraction of fuzzy if-then rules and its application to medical diagnosis. In R. Lippmann, J. Moody, and D.S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, San Mateo, CA, 1991. Morgan Kaufmann Publishers.

[42] Y. Hayashi and A. Imura. Fuzzy neural expert system with automated extraction of fuzzy if-then rules from a trained neural network. In *Proceedings of the First IEEE Conference on Fuzzy Systems*, pages 489–494, 1990.

[43] G.E. Hinton and S.J. Nowlan. How learning can guide evolution. *Complex Systems*, 1(1):495–502, 1987.

[44] J. Holland, editor. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, 1975.

[45] K. Karplus, C. Barrett, and R. Hughey. Hidden Markov models for detecting remote protein homologies. *Bioinformatics*, 14(10):846–856, 1998.

[46] T. Kohonen. An introduction to neural computing. *Neural Networks*, 1:3–16, 1989.

[47] A. Krogh, M. Brown, I.S. Mian, K. Sjolander, and D. Haussler. Hidden Markov models in computational biology: Applications to protein modeling. Technical Report UCSC-CRL-93-32, Computer and Information Sciences, Sinsheimer Laboratories, University of California, Santa Cruz, CA, 1993.

[48] R.C. Lacher, S.I. Hruska, and D.C. Kuncicky. Backpropagation learning in expert networks. *IEEE Transactions on Neural Networks*, 3(1):62–72, 1992.

[49] N. Lavrac. Selected methods for data mining in medicine. *Artificial Intelligence in Medicine*, 16(3):3–23, 1999.

[50] R. Maclin and J.W. Shavlik. Refining algorithms with knowledge-based neural networks: Improving the Chou-Fasman algorithm for protein folding. In S. Hanson, G. Drastal, and R. Rivest, editors, *Computational Learning Theory and Natural Learning Systems.* MIT Press, 1992.

[51] J.J. Mahoney and R.J. Moore. Combining neural and symbolic learning to revise probabilistic rules bases. In S. Hanson, J. Cowans, and C.L. Giles, editors, *Advances in Neural Information Processing Systems 5*, San Mateo, CA, 1993. Morgan Kaufmann Publishers.

[52] C. McMillan, M.C. Mozer, and P. Smolensky. Rule induction through integrated symbolic and subsymbolic processing. In J. Moody, S. Hanson, and R. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, San Mateo, CA, 1992. Morgan Kaufmann Publishers.

[53] C.B. Miller and C.L. Giles. Experimental comparison of the effect of order in recurrent neural networks. *International Journal of Pattern Recognition and Artificial Intelligence, Special Issue on Applications of Neural Networks to Pattern Recognition*, 7(4):849–872, 1993.

[54] M. Minsky. *Computation: Finite and Infinite Machines*, chapter 3, pages 32–66. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1967.

[55] T.M. Mitchell. *Version Space: An Approach to Concept Learning.* PhD thesis, Stanford, CA, 1986.

[56] T.M. Mitchell and S.B. Thrun. Explanation-based neural network learning for robot control. In S. Hanson, J. Cowan, and C.L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 287–294. Morgan-Kaufmann Press, 1993.

[57] P.M. Murphy and D.W. Aha. UCI repository of machine learning databases. Department of Information and Computer Science. Irvine, CA: University of California, 1994. [http://www.ics.uci.edu/m̃learn/MLRepository.html].

[58] C.W. Omlin. Constructing deterministic finite-state automata in recurrent neural networks. *Journal of the ACM*, 43(6):937–972, 1996.

[59] C.W. Omlin and C.L. Giles. Training second-order recurrent neural networks using hints. In *Machine Learning: Proceedings of the Ninth International Conference (ML92)*. Morgan Kauffman, San Mateo, CA, 1992.

[60] C.W. Omlin and C.L. Giles. Extraction and insertion of symbolic information in recurrent neural networks. In V. Honavar and L. Uhr, editors, *Artificial Intelligence and Neural Networks: Steps toward Principled Integration*, pages 271–299. Academic Press, San Diego, CA, 1994.

[61] C.W. Omlin and C.L. Giles. Extraction of rules from discrete-time recurrent neural networks. *Neural Networks*, 9(1):41–52, 1996.

[62] C.W. Omlin and C.L. Giles. Rule revision with recurrent neural networks. *IEEE Transactions on Knowledge and Data Engineering*, 8(1):183–188, 1996.

[63] C.W. Omlin and C.L. Giles. Symbolic knowledge representation in recurrent neural networks: Insights from theoretical models of computation. In I. Cloete and J.M. Zurada, editors, *Knowledge-Based Neurocomputing*, chapter 3. MIT Press, Cambridge, MA, 1999.

[64] M.C. O'Neill. Escherichia coli promoters: Consensus as it relates to spacing class specificity, repeat substructure, and three-dimensional organization. *Journal of Biological Chemistry*, (264):5522–5530, 1989.

[65] D. Opitz and J.W. Shavlik. Dynamically adding symbolically meaningful nodes to knowledge-based neural networks. *Knowledge-Based Systems*, pages 301–311, 1996.

[66] S. K. Pal and S. Mitra. Multilayer perceptron, fuzzy sets, and classification. *IEEE Transactions on Neural Networks*, 3(5):683–697, 1992.

[67] S.K. Pal and S. Mitra. *Neuro-Fuzzy Pattern Recognition: Methods in Softcomputing.* John Wiley & Sons, Inc, New York, 1999.

[68] D.A. Pomerleau, J. Gowdy, and C.E. Thorpe. Combining artificial neural networks and symbolic processing for autonomous robot guidance. *Engineering Applications of Artificial Intelligence*, (4):279–286, 1991.

[69] F.J. Provost and B.G. Buchanan. Inductive policy: The pragmatics of bias selection. *Machine Learning Journal*, 20:35–61, 1995.

[70] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[71] J. Reggia. Neural computation in medicine. *Artificial Intelligence in Medicine*, 5(2):143–157, 1993.

[72] R. Rojas. *Neural Networks: A Systematic Introduction.* Springer Verlag, Berlin, 1996.

[73] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing*, chapter 8. MIT Press, Cambridge, MA, 1986.

[74] D.E. Rumelhart and J.L. McClelland. *Parallel Distributed Processing: Volume 1, Foundations.* MIT Press, Cambridge, MA, 1986.

[75] K. Saito and R. Nakano. Medical diagnostic expert system based on PDP model. In *Proceedings of the IEEE International Conference on Neural Networks (ICNN'88)*, volume 1, pages 255–262, 1988.

[76] M.S. Sanchez. *A Neurosymbolic Approach to the Classification of Scarce and Complex Data.* PhD thesis, School of Cognitive and Computing Sciences, University of Sussex, March 1999.

[77] R. Scott. Artificial intelligence: Its use in medical diagnosis. *Journal of Nuclear Medicine*, 34(3):510–514, 1993.

[78] T. Sejnowski and C. Rosenberg. Parallel networks that learn to pronounce english text. *Complex Systems*, 1:145–168, 1987.

[79] L. Shastri. Structured connectionist models. *In Arbib*, pages 949–952, 1995.

[80] J.W. Shavlik. Combining symbolic and neural learning. *Machine Learning*, 14:321–331, 1994.

[81] P.S. Simard, B. Victorri, Y. LeCun, and J. Denker. TangentProp—a formalism for specifying selected invariances in an adaptive network. In J. Moody et al., editor, *Advances in Neural Information Processing Systems 4*. Morgan Kaufmann, 1992.

[82] S. Snyders and C.W. Omlin. What inductive bias gives good neural network training performance? In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks*, volume 3, pages 445–450, 2000.

[83] S. Snyders and C.W. Omlin. Inductive bias in recurrent neural networks. In *Proceedings of the 6th International Work-Conference on Artificial and Natural Neural Networks (LCNS 2084)*, volume I, pages 339–346, 2001.

[84] S. Snyders and C.W. Omlin. Rule extraction from knowledge-based neural networks with adaptive inductive bias. In *Proceedings of the 8th International Conference on Neural Information Processing*, volume 1, pages 143–148, 2001.

[85] S. Snyders and C.W. Omlin. Inductive bias strength in knowledge-based neural networks: Application to magnetic resonance spectroscopy of breast tissues. *Artificial Intelligence in Medicine*, expected to appear 2003. To be published in the Special Issue on Knowledge-Based Neurocomputing.

[86] S. Suddarth and A. Holden. Symbolic neural systems and the use of hints for developing complex systems. *International Journal of Man-Machine Studies*, 34:291–311, 1991.

[87] R. Sutton. Two problems with backpropagation and other steepest-descent learning procedures for networks. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 823–831, 1986.

[88] G.G. Towell. *Symbolic Knowledge and Neural Networks: Insertion, Refinement and Extraction.* PhD thesis, University of Wisconsin, Madison, WI, 1991.

[89] G.G. Towell, M.W. Craven, and J.W. Shavlik. Constructive induction using knowledge-based neural networks. In L.A. Birnbaum and G.C. Collins, editors, *Eighth International Machine Learning Workshop*, page 213, San Mateo, CA, 1990. Morgan Kaufmann Publishers.

[90] G.G. Towell and J.W. Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13(1):71–101, 1993.

[91] G.G. Towell and J.W. Shavlik. Knowledge-based artificial neural networks,. *Artificial Intelligence*, 70:119–165, 1993.

[92] G.G. Towell, J.W. Shavlik, and M.O. Noordewier. Refinement of approximate domain theories by knowledge-based neural networks. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, volume 2, pages 861–866, San Mateo, CA, 1990. Morgan Kaufmann Publishers.

[93] V. Tresp, J. Hollatz, and S. Ahmad. Network structuring and training using rule-based knowledge. In C.L. Giles, S.J. Hanson, and J.D. Cowan, editors, *Advances in Neural Information Processing Systems 4*, San Mateo, CA, 1993. Morgan Kaufmann Publishers.

[94] P. Turney. How to shift bias: Lessons from the Baldwin effect. *Evolutionary Computation*, 4(3):271–295, 1997.

[95] University of Florida and American Association for Artificial Intelligence. *Proceedings of the International Symposium on Integrating Knowledge and Neural Heuristics*, Pensacola, Florida, May 9-10 1994.

[96] P.E. Utgoff. Shift of bias for inductive concept learning. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume II, pages 107–148, Los Altos, CA, 1986. Morgan Kaufmann.

[97] A. Vahed and C.W. Omlin. Knowledge-driven incremental training of recurrent neural networks. Technical report, Department of Computer Science, University of the Western Cape, 2002. Submitted to Neural Information Processing Systems.

[98] V.N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

[99] R.L. Watrous and G.M. Kuhn. Induction of finite-state languages using second-order recurrent networks. *Neural Computation*, 4(3):406–414, 1992.

[100] T. Wessels and C.W. Omlin. Refining hidden markov models with recurrent neural networks. Technical Report US-CS-TR-00-10, University of Stellenbosch, Stellenbosch, 2000.

[101] R.J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270–280, 1989.

[102] X. Yao. Evolutionary artificial neural networks. *International Journal of Neural Systems*, 4(3):203–222, 1993.

[103] Z. Zeng, R.M. Goodman, and P. Smyth. Learning finite state machines with self-clustering recurrent networks. *Neural Computation*, 5(6):976–990, 1993.

[104] J.M. Zurada. *Introduction to Artificial Neural Systems.* PWS, Boston, 1992.

[105] J. van Zyl and C.W. Omlin. Knowledge-based neural networks for modelling time series. In *Proceedings of the 6th International Work-Conference on Artificial and Natural Neural Networks (LCNS 2085)*, volume II, pages 579–586, 2001.