Neural Network Design
and the Complexity of Learning

J. Stephen Judd

Computer Science Department
California Institute of Technology

Caltech-CS-TR-88-20

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **SEP 1988**	2. REPORT TYPE	3. DATES COVERED **00-09-1988 to 00-09-1988**

4. TITLE AND SUBTITLE **Neural network Design and the Complexity of Learning**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Air Force Office of Scientific Research,875 North Randolph Street Suite 325,Arlington,VA,22203-1768**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR’S ACRONYM(S)
	11. SPONSOR/MONITOR’S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
see report

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES **133**	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**			

NEURAL NETWORK DESIGN AND THE COMPLEXITY OF LEARNING

A Dissertation Presented by

J. STEPHEN JUDD

Submitted to the Graduate School of the
University of Massachusetts in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 1988

Computer and Information Science Department

ACKNOWLEDGEMENTS

My schooling in connectionist research proceeded under and was inspired by A. G. Barto. His years of patience with my determined excursions into blind alleys were remarkably supportive. The complexity-theoretic content was encouraged, guided, corrected, extended, and made presentable by D. A. Mix Barrington. Many useful comments and refinements were made by R. L. Rivest.

I thank all past and present members of the Adaptive Networks research group, and especially Richard Sutton, for constant intellectual challenges, encouragement and resistance; S. Porat for proofreading my first (tedious) proof; Richard Yee for a conversation about bandwidth and for loaning me a paper on the topic; and Ian Parberry for pointing out the work by Muroga and for determined discussions to ferret out the difference between it and Hong's work.

This research was supported by the Air Force Office of Scientific Research through contract AFOSR F33615-83-C-1078 and grant AFOSR-87-0030. The author was also supported by an NSERC Canada Post-Graduate Scholarship.

ABSTRACT

NEURAL NETWORK DESIGN AND THE COMPLEXITY OF LEARNING

SEPTEMBER 1988

J. STEPHEN JUDD, B.SC., UNIVERSITY OF MANITOBA
M.SC., UNIVERSITY OF MANITOBA
PH.D., UNIVERSITY OF MASSACHUSETTS

Directed by: Professor Andrew G. Barto

We formalize a notion of learning that characterizes the training of feed-forward networks. In the field of learning theory, it stands as a new model specialized for the type of learning problems that arise in connectionist networks. The formulation is similar to Valiant's [Val84] in that we ask what can be feasibly learned from examples and stored in a particular data structure.

One can view the data structure resulting from Valiant-type learning as a 'sentence' in a language described by grammatical syntax rules. Neither the words nor their interrelationships are known a priori. Our learned data structure is more particular than Valiant's in that it must be a *particular* 'sentence'. The position and relationships of each 'word' are fully specified in advance, and the learning system need only discover what the missing words are. This corresponds to the problem of finding retrieval functions for each node in a given network.

We prove this problem *NP*-complete and thus demonstrate that learning in networks has no efficient general solution. Corollaries to the main theorem demonstrate the *NP*-completeness of several sub-cases. While the intractability of the problem precludes its solution in all these cases, we sketch some alternative definitions of the problem in a search for tractable sub-cases.

One broad class of subcases is formed by placing constraints on the network architecture; we study one type in particular. The focus of these constraints is on families of 'shallow' architectures which are defined to have bounded depth and unbounded width. We introduce a perspective on shallow networks, called the Support Cone Interaction (SCI) graph, which is helpful in distinguishing tractable from intractable subcases: When the SCI graph has tree-width $O(\log n)$, learning can be accomplished in polynomial time; when its tree-width is $n^{\Omega(1)}$ we find the problem *NP*-complete even if the SCI graph is a simple 2-dimensional planar grid.

TABLE OF CONTENTS

CHAPTER

APPENDICES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Drawing inspiration from neuroanatomy and spurred on by successes in modelling cognitive phenomena, the *connectionist* model of computation has recently drawn much attention (see for example the landmark volumes [RM86,MR86,AR88]). Connectionist networks are also called *neural networks.* This model is used in the study of how knowledge might be captured, represented, and processed by circuits that are similar in an abstract sense to biological computers. A neural network is characterized by its emphasis on using many richly interconnected processors that perform relatively slow and simple calculations in parallel. The connectionist approach shows promise of eventually providing a new language for designing and building computational devices, and possibly may yield clues to the centuries-old puzzle of brain function. Many aspects of connectionist networks, including structural design, I/O protocol, and behavioural phenomena have been compared to biological brains.

The model is loosely defined around three aspects: computing units, communication links, and message types. The computing units are small, homogeneous, plentiful, simple and can accept many input connections. These units are connected into networks by dedicated low-bandwidth links. These communication links are also considered to be cheap and plentiful—an attitude that is a response to neuroanatomical observations that individual neurons may have extremely high fan-in.

As yet, there seem to be few principles or methodologies for designing the specific connectivity patterns in these networks. To the best of our knowledge, all network designs in the literature have been rather ad hoc constructions for specific experiments. In our view, this is a major inadequacy of the discipline. The discovery of well-grounded and universal design principles would not only assist the development of artificial neural networks but would also strengthen links to neuroanatomy: hopefully neuroanatomists could confirm or repudiate the ideas by examining biological brain structure.

Some sources of design constraints arise from consideration of

- learning speed
- signal integrity (error correction)
- processing integrity (fault tolerance)
- retrieval speed
- memory capacity
- signalling capacity (bandwidth)
- 3-dimensional geometry
- power supplies and heat dissipation

A thorough theoretical understanding of these areas would advance the field of artificial neural networks.

We propose to focus on the first item in this list.

1.1 Learning

We think of *learning* as the capacity of a network to absorb information from its environment without requiring some external intelligent agent to 'program' it. Learning

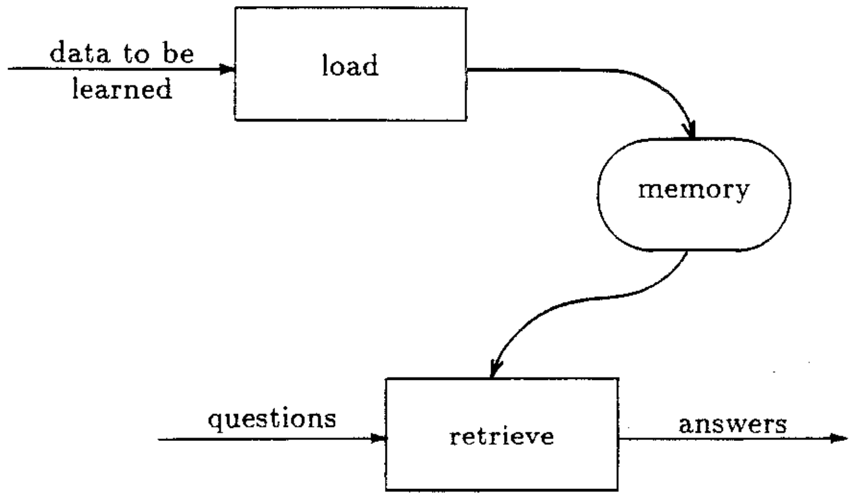


Figure 1.1: A Simple Model of Learning. Note the conceptual separation of the system into two processes (shown in rectangles). This separation does not correspond to any physical separation in a network. Typically, each node serves as a repository for a piece of the memory and participates in both processes that interact with memory.

is a quintessential ability of brains, and it is a major focus of much connectionist research. Unfortunately, the learning algorithms reported in the literature so far are all unacceptably slow in large networks. Although it is clear that we need to be able to scale up our applications to much bigger networks, it is not at all clear how to achieve this. Many researchers view this as the most pressing challenge for current connectionist research.

The networks have two modes—the so-called 'learning' or loading mode wherein data are loaded into the permanent memory base, and the 'retrieval' mode wherein those associative data are recalled from memory. Figure 1.1 depicts the general paradigm. During retrieval, each computing unit calculates an output value by some simple rule such as a threshold function on the linear weighted sum of its current inputs. The computation is performed repetitively at approximately the same cycle rate as all other units. Hence the typical signal transmitted via a connection is a single logical value or perhaps a scalar value. However, the network as a whole is expected to do such things as associate pairs of bit patterns or find completions of partial patterns.

These comments apply to all types of connectionist models and they also seem to describe standard circuit models of computation. However, connectionist devices are often elaborated with various features like bi-directional connections, learning capability, stochasticity, linear sum functions, or cyclic dynamics. For simplicity, this thesis discusses only those networks that retrieve data in the manner of a strictly unidirectional feed-forward deterministic circuit. We assume that the networks have some means of changing their behaviour but that this change does not involve altering their connectivity structure.

The implicit goal of connectionist learning research has been to find a single 'learning rule' that each network unit can follow in order to adjust the weights used in its linear threshold functions in such a way that the retrieval behaviour of the whole network is eventually correct. It was hoped that a learning rule would work for any network design. Many researchers have developed candidates for such a learning algorithm; some notable approaches are the Perceptron [Ros61,MP72], back-propagation [RHW86,Par85,lC85], Boltzmann [AHS85,HS86], and associative reward-penalty (A_{R-P}) [BA85,Bar85] schemes.

There is a theorem proving the effectiveness of the Perceptron Learning Rule for linearly separable tasks in a single layer of trainable nodes. In their book, Minsky and Papert studied this learning rule and also investigated several computing properties of 1- and 2-layer networks. But one of the tantalizing gaps that Minsky and Papert left regards the learning problem in *multi*-layered networks. They considered it an important research problem to extend results on learning algorithms for single-layer nets to the case of multi-layer nets:

> "Perhaps some powerful convergence theorem will be discovered, or some profound reason for the failure to produce an interesting learning theorem for the multi-layered machine will be found."

Descriptions of the back-propagation, Boltzmann, and A_{R-P} methods have each been published along with demonstrations of their ability on *selected* associative learning problems and their required learning time has been studied empirically (see Chapter 3). However, no proof of their effectiveness has been offered and no analytical treatment of their scale-up properties has appeared. The published successes in connectionist learning have been empirical results for very small networks, typically much less than 100 nodes. To fully exploit the expressive power of networks they need to be scaled up to much bigger sizes, but it is widely acknowledged that as the networks get larger and deeper the amount of time required for them to load the training data grows prohibitively [HV86,TJ88,Bar82,Omo87]. It is important to find out how to avoid this phenomenon.

The connectionist learning problem is treated here first of all as simple memorization of some given data by a given feed-forward network. This problem is described and discussed in Chapter 2. We ask if there exists an efficient algorithm for solving this learning problem. 'Efficient' is taken to mean that the worst-case learning time for a network of size n should be bounded above by a polynomial in n, something which can easily be proved by exhibiting an algorithm for it. An excellent theoretical test is available which indicates intractability in a problem, and that is to prove the problem to be *NP*-complete. We will explain and use this tool in Chapter 4.

Are there efficient algorithms for learning in large connectionist networks? Or is there some deep reason why there cannot be? Does network design affect learning ability? How does learning time scale up with network size? Can scale-up properties be manipulated through design techniques? This thesis addresses such questions.

1.2 Approach

We seek design principles by appealing to constraints of learnability. As is often the case in theoretical pursuits, it is easiest to investigate extreme cases first, in order to find the boundary conditions where the problem is certifiably easy or certifiably infeasible, and later to refine the middle ground.

This thesis begins by identifying and formalizing a model of the computational problem involved in getting a network to memorize data. The particular formulation we use is closely related to the types of experiments being reported in the connectionist literature. It then uses the model to make two important points:

1. The learning problem in its general form is too difficult to solve. By proving it to be *NP*-complete, we can claim that large instances of the problem would be wildly impractical to solve. (See Chapter 4). There is no reliable method to configure a given arbitrary network to remember a given arbitrary body of data in a reasonable amount of time.

This result shows that the simple problem of remembering a list of data items (something that is trivial in a classical random-access machine) is extremely difficult to perform in some fixed networks.

Of course, Connectionists would not be satisfied if all they got out of their systems was rote memory. Much of the fascination of Neural Networks comes from the possibility of their having generalization properties which could be employed to extend data, smooth over the domain, and induce the structure of the underlying data. Only so would they achieve compact representations, fast calculations, strong prediction, and intelligent learning. We will argue that success in generalizing presupposes the ability to memorize simple associative data faithfully and efficiently.

The intractability of memorization suggests that the connectionist model, even though it has demonstrated many attractive qualities, may have a crucial flaw. This might well be a disturbing theorem were it not for other insights that accompany it:

2. There are many ways to circumvent this negative result, and each one corresponds to a particular constraint on the learning problem. There are fast learning algorithms for cases where the network is of a very restricted design, or where the data to be loaded are very simple.

These two observations (the full problem is too hard; some sub-cases are easy), provide a foundation for theoretical inquiries into the design of connectionist networks. There are various ways to constrain the loading problem to find subcases that are solvable in polynomial time: by restricting the task to be learned, by restricting the architecture of the net, by relaxing the criterion of success, etc., or by combinations of these. The very general hard cases and the very restricted easy cases establish extrema within which a more complete theory can be constructed. This thesis promotes the usefulness of elaborating such a theory, and will consider a few special cases within the great variety of imaginable subcases.

1.3 Subcases

In Chapter 5 we discuss various ways of formulating sub-cases or simply different cases that might be feasibly solvable. Even in several of these restricted sub-cases the intractability remains, thus revealing a labyrinth of open and closed avenues for discovering what it is that large connectionist networks can or cannot learn. What follows here is a description of the major theaters in which constraining conditions can be posed.

1.3.1 Data to be Learned

By putting strong constraints on what the network is required to learn, some trivial (and uninteresting) learning problems arise. It is our desire to know if there are some interesting classes of learnable tasks. We prove it intractable for networks to learn even very small numbers of associated pairs, or to learn sets of pairs that are drawn from a monotonic function.

1.3.2 Network Design

By putting strong constraints on the type of network used for learning, some trivial learning problems arise. These networks may all be next to useless, but our main theorem shows that if we allow the network to be of *arbitrary* design then the learning problem is too hard. The challenge is to see if there are any intermediate network designs that are useful and can learn easily. Unknown cases include very deep nets and highly connected nets. For various reasons, we pursue studying the loading problem in one particular broad architectural family which we call shallow architectures. This family has a technical definition that effectively limits the depth of each network but does not limit the width. This family is interesting because it allows us to study the load-time scale-up issue without having to deal with issues that arise in deep networks. The connectionist literature uniformly reports great difficulties in loading deep nets so we have taken the strategic decision to avoid the issue altogether and concentrate on shallow nets. *NP*-completeness appears even in networks of depth 2 so there is still a considerable domain of issues to explore even in the shallow case. Furthermore, the shallow architectures are interesting because they might be a useful model of some brain structures.

For the discussion of shallow architectures, we introduce the notion of a support cone, which is the set of all nodes that can affect the behaviour of an output node. Then we define a Support Cone Interaction (SCI) graph, which captures how the support cones overlap with each other. When this SCI graph is a planar, 2-dimensional grid the loading problem is still NP-complete, but if the SCI graph has limited tree-width then the architecture can be loaded in polynomial time. Tree-width is a metric on graphs that is a generalization of the more widely known graph-theoretic notion of bandwidth.

1.3.3 Node Functionality

A third way of constraining the learning model is to imbue the network nodes with different amounts of functionality. The standard node type used in connectionist research is the linear sum type—capable of performing any linearly separable binary function. The NP-completeness found in our main theorem applies to this case, but we go further to prove that even when the nodes are capable of performing much more complex functions (e.g. arbitrary Boolean functions), or when the nodes are capable of performing only extremely simple functions, the computational problem is much the same. We had hoped that the theory might guide us in selecting appropriate types of nodes (e.g. by somehow demonstrating that the linearly separable functions are a logical or optimal choice). But the results are quite equivocal on this matter. Subsequent work by Blum and Rivest [BR88] suggests that the linear sum functions actually introduce special computational problems that could be avoided with simpler functions or with more complex functions.

Our complexity results are almost entirely independent of the type of node functions used in the networks. This is a strength in itself. But it offers a further conclusion: that the whole issue of node functionality is of secondary importance to learning complexity, even though significant research effort is now being spent on analyzing the particularities of one or two particular favourite types.

1.4 Philosophical Base

This study is based on the belief that the scale-up aspect of the learning issue is a rich source of imperatives for network design and that the development of a theory of learning is therefore well warranted. We posit that a thorough delineation of the polynomial-time solvable cases from the *NP*-complete cases will illuminate design constraints that all networks must adhere to in order to be capable of learning. Specifically, we posit that an understanding of the roots of *NP*-completeness in connectionist learning will yield techniques for building architectures that are easy to load.

We think that the general computational question framed here is a basis that could ultimately lead to a collection of definitions of tasks, architectural designs, and loading criteria, and to a theory of how these various aspects interact to create feasible or infeasible learning. This thesis is a beginning toward such a goal.

1.5 Outline

The next chapter formulates the general learning problem and sets up the formal question regarding its complexity.

Chapter 3 reviews some of the current theory on learning in general, and relates our model of learning to other models outside of the connectionist paradigm that also deal with learning from examples. It also examines what is currently known about scale-up issues in connectionist learning.

Chapter 4 proves our main theorem, which deals with the intractability of the general case. Section 4.2 reports that the complexity is invariant for almost any kind of node functionality. The following chapter (5) elaborates some of the implications of the main theorem and points out several corollaries applying to various special subcases.

Chapter 6 proves the tractability and intractability of various families of shallow networks.

Chapter 7 responds to some concerns about how well neural networks would be able to generalize from what they have learned to other parts of their domains that they have not had access to. Many people have high hopes for the abilities of neural networks to perform such generalization. Using Valiant's technical definition of induction, we show how the intractability of memorization implies the intractability of generalization as well.

Finally, Chapter 8 summarizes our results and discusses some implications and extensions to the work.

Chapter 2

OUR MODEL OF NETWORK LEARNING

2.1 The Learning Protocol

The type of learning investigated here is known as supervised learning. In this paradigm input patterns (called *stimuli*) are presented to a machine paired with their desired output patterns (called *responses*). The object of the learning machine is to remember all the associations presented during a training phase so that in future tests the machine will be able to emit the associated response for any given stimulus. This interaction is diagrammed in Figure 2.1.

The exact form of presentation of these data is not of concern here. Many connectionist experiments involve a long series of training samples wherein a single associative pair is presented to the network at a time, and where any particular pair may have to be presented many times over. But none of these details are relevant here, and our results are strengthened by abstracting away from them. We require only that the associative data are available in some reasonable encoding.

In what follows, every stimulus σ is a fixed-length string of s bits, and every response ρ is a string of r bits with "don't cares", that is, $\sigma \in \{0,1\}^s$ and $\rho \in \{0,1,*\}^r$. The output from a net is an element of $\{0,1\}^r$. The purpose of a response string is to specify constraints on what a particular output can be: We say that

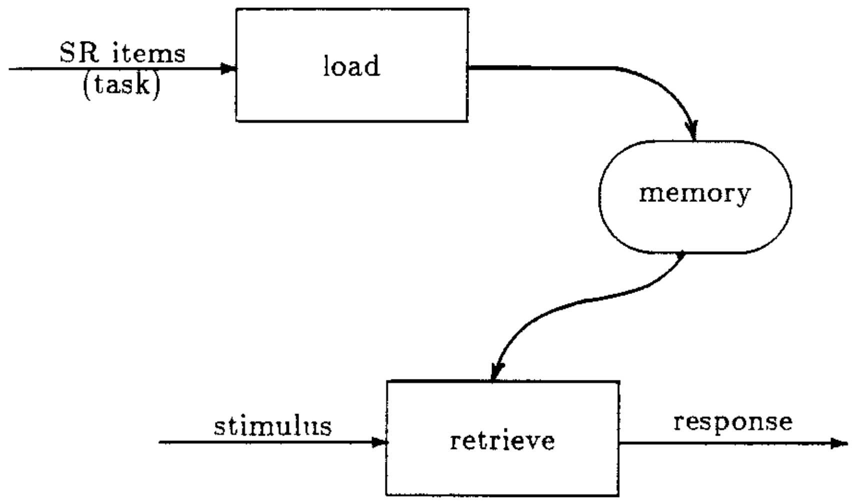


Figure 2.1: A Model of Supervised Learning. The loading process examines the SR items and alters memory to store that data. Later, the retrieval process accepts a stimulus and examines memory to find and emit the associated response.

an output string, θ, agrees with a response string, ρ, if each bit, θ_i, of the output equals the corresponding bit, ρ_i, of the response whenever $\rho_i \in \{0,1\}$. The notation for such agreement is $\theta \models \rho$. Each stimulus/response pair, (σ, ρ), is called an SR *item*. A *task* is a set of SR items that the machine is required to learn. To be reasonable, each distinct stimulus in a task should be associated with no more than one distinct response. Equivalently, a task T should be extendible to some function $f : \{0,1\}^s \rightarrow \{0,1\}^r$. We view functions as sets of ordered pairs and use the notation $T \subseteq f$ to mean $T \subseteq \{(\sigma, \rho) : f(\sigma) \models \rho\}$.

2.2 Network Architecture

The particular style of connectionist machines considered here is that of non-recurrent, or feed-forward, networks of computing elements. This is a generalized combinational circuit; the connections between nodes form a directed acyclic graph, and the nodes perform some function of their inputs as calculated by previous nodes in the graph.

We define an *architecture* as a 5-tuple $A = (P, V, S, R, E)$ where

P is a set of *posts*,

V is a set of n *nodes:* $V = \{v_1, v_2, \ldots, v_n\} \subseteq P$,

S is a set of s *input posts:* $S = P - V$,

R is a set of r *output posts:* $R \subseteq P$, and

E is a set of directed *edges:* $E \subseteq \{(v_i, v_j) : v_i \in P,\ v_j \in V,\ i < j\}$

The constraints on the edges ensure that no cycles occur in the graph. Denote the *set of input posts* to node v_k as $pre(v_k) = \{v_j : (v_j, v_k) \in E\}$. The size of this set (denoted $|pre(v_k)|$) is called the *fan-in.*

An architecture specifies everything about a circuit except what kind of functions the nodes perform (i.e. what kind of gates they are).

2.3 Node Functions

Each node in a network contributes to the overall retrieval computation by taking signals from its input edges and computing an output signal. Although some of our results apply to real-valued outputs, this paper considers only binary-valued functions:

$$f_i : \{0,1\}^{|pre(v_i)|} \rightarrow \{0,1\}$$

The function f_i is a member of a given set, $\mathcal{F}$, of functions, called a *node function set.* Typically, connectionists have used the set of linearly separable functions (LSFns) for $\mathcal{F}$. These functions are characterized by a real-valued threshold and a real-valued weight associated with each input to a node. An activation level is calculated from the weighted sum of the a inputs, and the output is one of two values depending on whether the activation is above or below the threshold.

$$LSFns = \{f : \{0,1\}^a \rightarrow \{0,1\} \mid \exists W, \Theta \in \Re \ \sum_{i=1}^{a} W_i X_i > \Theta \Leftrightarrow f(X) = 1\}$$

We consider LSFns as well as a variety of other node function sets. Two variants that are considered are called AOFns and LUFns. AOFns is the 2-element set {AND,OR} (AOFns is from And-Or Functions). LUFns is the set of all Boolean functions (LU is from Look-Up table). Note the inclusion hierarchy LUFns $\supseteq$ LSFns $\supseteq$ AOFns for any given fan-in.

We also consider two sets of node functions that have real values. Quasi-linear functions (QLFns) are functions composed of any bounded, monotonic function, E, applied to a linear combination of the inputs. (This definition is essentially the same as that used in [RHM86] and [Wil86a].) A special case of QLFns is the logistic-linear functions (LLFns), for which $E(x) = 1/(1+e^{-x})$. The back-propagation algorithm of [RHW86] is designed to work with LLFns.

A *configuration* of a network is a set of n functions $F = \{f_1, f_2, \ldots, f_n\}$ corresponding to the set of nodes, V, meaning that f_i is the function that node i computes.

2.4 The Computational Problem

In a configured network, every node performs a particular function and therefore the network as a whole performs a particular function which is a composition of the node functions. An architecture, A, and a configuration, F, together define a mapping from the space of stimuli to the space of responses:

$$\mathcal{M}_F^A : \{0,1\}^s \rightarrow \{0,1\}^r.$$

The A and F fully define a circuit and thus fully define how the network will behave during retrieval.

A task, as defined above, can be viewed as a collection of constraints on the mapping that a network is allowed to perform. Recall that an SR item in a task is a pair of strings (σ, ρ). When the posts in S are assigned the values of respective elements of σ, the network mapping defines values for each post in R. It is required that these values agree with respective elements of ρ. For stimuli not in the task, any output is acceptable—that is, $\mathcal{M}_F^A$ may be any consistent extension (generalization) of the task.

The process of ***loading*** can now be defined. In the learning problem we are considering, an architecture and a task are given, and loading is the process of assigning an appropriate response function to every node in the architecture, $load(A,T) \mapsto F$, so that the derived mapping includes the task. It is a procedure that accepts a pair (A,T) and returns a *solution*, which is a configuration F such that $T \subseteq \mathcal{M}_F^A$. If no such configuration exists, the procedure announces that fact.

The loading problem is a search problem, but it is usual to frame complexity questions in terms of decision problems. In the space of all possible (A,T) pairs, some pairs will have solution configurations and some will not; that is, for some pairs the architecture can *perform* the task, and for some it cannot. The *performability* decision problem is simply: "Can the architecture perform the task?" In the style of [GJ79], this is phrased as follows:

Instance: An architecture A and a task T.

Question: Is there a configuration F for A
such that $T \subseteq \{(\sigma, \rho) : \mathcal{M}_F^A(\sigma) \text{ agrees with } \rho\}$?

For purposes of our ensuing complexity questions, the decision problem embodies the crux of the loading problem.

Note that the above statements are technically incomplete because they hold no direct reference to the node function set being used. Our next (and last) re-phrasing of the loading problem redresses this oversight, and uses classical terminology for expressing decision problems: The performability problem is the problem of recognizing the following (parameterized) language:

$$Perf_{\mathcal{F}} = \{(A, T) : \exists F \in \mathcal{F}^n \ni T \subseteq \mathcal{M}_F^A\}.$$

The subscripted parameter indicates the node function set. We will be asking questions about a variety of such sets, and each time we change the subscript we are referring to a slightly different decision problem.

2.5 Classical Connectionist Learning

The dominant paradigm in current connectionist supervised learning research follows a style established by the Perceptron many years ago. The following algorithm illustrates the style using one version of the Perceptron Learning Rule. Let W be a set of weights and X be an input vector. Let a be one greater than the fan-in to a node. Then using a simple trick of notation, the threshold is treated as one of the weights so it does not explicitly appear in the expression of the linear threshold function.

```
start:
        choose any arbitrary set of weights $W \in \Re^a$
test:
        accept an input $X \in \Re^a$
        and a classification $c \in \{0, 1\}$
        if $W \cdot X \geq 0$ and $c = 1$
        or $W \cdot X < 0$ and $c = 0$ go to test
adjust:
        if $W \cdot X \geq 0$ and $c = 0$   set $W \leftarrow W - X$
        if $W \cdot X < 0$ and $c = 1$   set $W \leftarrow W + X$
        go to test
```

Whenever the weights are adjusted, the retrieval function can change. The process of adjusting the weights is therefore conceptually the same as choosing a node function $f \in LSFns$, as we phrase it.

As exemplified above, the learning process in the classical paradigm is a cyclic repetition of the following steps:

1. A stimulus is received from the environment.

2. An output is calculated by the retrieval process.

3. Some form of information about the correctness of the output is given.

4. A determination is made about how a change in each weight would individually affect the overall performance of the network.

5. All the weights are changed according to what step (4) would determine to be an improvement.

The determination in step (5) is made based on information available locally at the node in question, which in the case of the Perceptron was only the value of the input relevant to the weight in question and whether or not the most recent output was correct. Note that as a consequence, each determination regarding a weight is independent of every other determination, and the amount of information they mutually have access to is minimal.

In step (3) above, some form of information about the correctness of the output is given. This can come in various forms, but the most direct form is simply to be given the correct answer. When such is the case, the protocol is called *supervised learning.* Variants on this protocol include *reinforcement learning* in which the only information given is a scalar evaluation of how 'good' the output from step (2) was. Both these schemes can be complicated by introducing noise into the data. In such a case, the information supplied in step (3) has only a certain probability of being correct, so the system is faced with a problem in stochastic optimization. The literature on Learning Automata has studied this problem [NT74,NL77,TR81] but we will avoid it.

Most connectionist research has attempted to comply with some present-day notions of neurological plausibility. This tradition is very much attached to linear sum node functions (primarily LSFns and LLFns) but the major characteristic of the style has to do with the way in which the algorithm interacts with the environment and with its internal state variables. A so-called 'neural' algorithm has a style substantially akin to the Perceptron's in the following senses: the loading component operates with minimal information beyond what the retrieval component uses; each node acts independently and somewhat simultaneously in adjusting its weights; and every node relies on information locally available only, where locality is defined as per connectivity in the net. As is true for the Perceptron, a connectionist system often has no state variables except its weights, and the meaning of these weights is fully defined by the retrieval algorithm. There is no significance to these variables beyond what the retrieval algorithm gives them.

At this point in time it is difficult to formulate exactly what would be acceptable as 'neural', but a loose formal model of this idea would at least specify a constant number of bits of memory associated with each edge of the architecture and a constant number of bits associated with each node (independent of network size).

Any scheme adhering to the general 5-point procedure outline above can be treated as an 'on-line' algorithm. An on-line system is one that guesses a response to each given stimulus before it is told the required response and is held to account for these guesses. One could view this as a model of an adaptive system having to make tactical decisions in an ongoing environment. Of course, any such system will sometimes make mistakes in its guesses; it is interesting to find upper and lower bounds on the number of mistakes an on-line system will make.

2.6 Discussion

There are three ways in which this formulation seems to stray from the connectionist paradigm. First, there is nothing in our model of learning that reflects any of the neural desiderata. However, this means that any intractability result will therefore be conservative. We have chosen not to adhere to these neural constraints in order to strengthen our results.

Second, connectionists are apt to find the performability problem a strange formulation because from their point of view the architecture is not an *input* to the problem but rather a specification of the machine that is to *solve* the problem. The reason for this formal rearrangement lies in the research strategy of the connectionist community. The prevailing goal has been to find a 'learning rule' that can be employed in each node of a network to pass information back and forth, to witness the various task inputs and errors made during early operation, and to eventually settle on a node function (i.e. a set of weights) that will eliminate output errors. The point to note is that this search for learning rules has historically been a search for a *universal* rule—one that could afford to be oblivious to the type of architecture into which the rule will be deposited to do its work. It has been implicitly hoped that the architecture has little to do with the difficulty of loading. Of course it was recognized that the architecture has a great influence over what mappings can be performed, but after assuming that the network was adequate to perform a given task, it was hoped that a general purpose (i.e. a non-architecture-specific) learning rule would be able to configure the weights correctly. Therefore we can freely vary the architecture when formulating the computational problem faced by such a learning rule; i.e. we can make it an input parameter.

Third, Connectionists might again object to this formulation because of the lack of any mention of the architecture in the model of computation that will be used to actually find the configuration. By omitting any reference to the machine, the phraseology above implicitly poses the problem in terms of a Turing machine, or at least in terms of some standard *serial* model of computation. The connectionist approach is to run the learning rule in all nodes of the network simultaneously, so one wonders if this parallel model might not be more powerful. Perhaps so, but note that the number of nodes is at most linear in the problem size (since the input includes a specification of the architecture), so the speed-up due to parallelism can be no more than linear. In the face of the *NP*-completeness result to follow in Chapter 4, this is inconsequential.

A word about our use of the word "loading". The connectionist literature rather uniformly uses the word "learning"; why should we use another word? Firstly, the word "learning" is used in AI and other fields to refer to a great variety of different things and it is useful to distinguish some of these uses from others. Secondly, although the connectionist literature is fairly consistent about what their learning problem is, our loading problem is not exactly the same as theirs either, so it behooves us to be precise by using a different name for a different problem.

Specifically, the loading problem involves:

1. a given, (previously unknown) network,

2. total, easy, ongoing access to the network structure,

3. a given, (previously unknown) task, and

4. total, easy, ongoing access to all items in the task,

where by 'total' we mean freedom from locality constraints; by 'easy' we mean $O(|A|)$ cost to read the whole data; and by 'ongoing' we mean there is no limit to the number of accesses allowed.

Of the four aspects listed here, (3) is certainly true for the classical connectionist learning, and (1) is often true although implicit. Both (2) and (4) are usually not part of classical connectionist learning. Strangely, although (3) is seemingly paramount, it may be the least important aspect of the model in the sense that knowing the task in advance might not make any difference to the computational complexity of the problem.

Note that (4) implies noise-free supervised learning—the input data are always dependable in that items are always consistent. It also implies knowledge of the exact number of items, something that classical models do not have access to. Aspect (2) indirectly implies that the architecture is fixed and cannot be altered during loading. Both (2) and (4) more or less imply that a Turing machine will be employed to perform the loading function; the algorithm is not required to run on a distributed machine.

The loading problem, then, is our formalization of a particular computational problem which is closely akin to classical connectionist learning but is altered slightly to be on the easy side of three major issues:

- the type of machine used to solve it,

- the style of processing required, and

- the type of information available.

It should be noted that when NP-completeness is found with this model, it is especially germane because we have focussed on the easiest and least restrictive conditions for all three of these issues. By applying automatically to many of the more difficult cases, the theory is particularly relevant.

When we find loading to be difficult, we will know that the classical connectionist learning problem must be at least as difficult; When we find loading to be easy, we will only have suggestive evidence that the classical connectionist learning problem is easy.

Chapter 3

REVIEW OF RELATED WORK

Some background in other formal learning theories will help put our work in perspective. It will also help explain why our theory will help connectionist design problems, whereas the others will not.

There is a long tradition of research on the problem of inferring a general rule to describe a set of specific examples. Philosophers [Bac42,Car50], then cyberneticists, cognitive psychologists [HMS66], engineers, and more recently AI researchers [Mit77,DM81] have all considered the problem. In a distilled form, the quest is to find a procedure that can take objects representing positive and negative examples of a concept and find an expression (in some form to be discussed) that expresses whether a given object is a positive or negative example of the underlying concept. This type of process is a major component of what is colloquially called 'learning'. Such inference procedures could be used for classifying unseen examples, for predicting future events, for storing data in a compressed format, or just for storing data in a convenient format. The last two purposes might be valid even when all instances of the concept have already been witnessed. Our primary motivation is similar to the last one—to store data in a particular format.

Some mathematical models of learning from examples have recently been developed. We will review the relevant aspects of the learning formalisms defined by Gold and by Valiant, and then contrast our formalism with theirs.

φ is a learning procedure.

t is a text. A text is an enumeration of an r.e. set, which is equivalent to strings in a language. Every r.e. set is equal to the domain of some function φ_i.

W_i is the domain of φ_i.

t_j is the first j elements of t.

L is a language.

$\mathcal{L}$ is a class of languages.

$$\varphi \text{ converges on } t \text{ to } i \iff \begin{cases} \text{(a) } \varphi \text{ is defined on } t. \\ \text{(b) } \exists n \ni \varphi(t_j) = \varphi(t_n) \forall j \geq n. \end{cases}$$

$$\varphi \text{ identifies } t \iff \begin{cases} \text{(a) } \varphi \text{ converges on } t \text{ to some } i. \\ \text{(b) } \mathrm{rng}(t) = W_i. \end{cases}$$

φ identifies $L \iff \varphi$ identifies all texts for L.

φ identifies $\mathcal{L} \iff \varphi$ identifies every $L \in \mathcal{L}$.

$\mathcal{L}$ is identifiable $\iff$ some φ identifies $\mathcal{L}$.

Figure 3.1: Gold's definition of learnable (identifiable)

3.1 Gold

Gold [Gol67] established a field of learning theory in 1967 which he labelled 'identification in the limit'. He asked whether there is a procedure which could read in an endless sequence of example strings in a language and eventually find a grammar for the language. See Figure 3.1 for a formal definition.

Many other researchers [BB75,Cho80,OSW86,WC80,AS83,Sha81] have built up the theory. The questions concern infinite languages, and therefore they involve infinite 'texts', meaning that the system must see unbounded amounts of data. In the main, the questions asked place no bounds on time or space required for learning. The flavour is very much like computability theory as opposed to complexity theory.

F is a class of programs (concepts).
p is a polynomial.
A is an algorithm.
ϵ, δ are probabilities.
n is a positive integer.
f and g are programs.
D^+ is a probability distribution of positive examples.
D^- is a probability distribution of negative examples.

$$\text{“}F\text{ is learnable”} \iff \left\{ \begin{array}{l} (\exists p, A) \text{ such that} \\ (\forall n)(\forall f \in F_n)(\forall D^+, D^-)(\forall \epsilon, \delta > 0) \\ A \text{ halts in time } p(n, size(f), 1/\epsilon, 1/\delta) \\ \text{with output } g \in F_n \text{ that} \\ \text{with probability} \geq 1 - \delta \\ \text{has property } \sum_{g(\vec{x})=0} D^+(\vec{x}) < \epsilon \\ \text{and property } \sum_{g(\vec{x})=1} D^-(\vec{x}) < \epsilon \end{array} \right.$$

Figure 3.2: Valiant's definition of learnable

3.2 Valiant

Valiant [Val84] established a lower-level field of learning which has subsequently been elaborated by himself and others [Val85,PV86,KLPV87]. His paradigm is concerned not just with what is learnable but with what is *feasibly* learnable (see Figure 3.2). The definition of feasibility relies on the well-honoured distinction between 'polynomial' problems and 'super-polynomial' (or *NP*-hard) problems.

Valiant's definition concerns data represented by a fixed (finite) number of variables which typically are all binary-valued. The learning system must discover the underlying rule that describes whether such a given bit string is or is not an example of a 'concept'. The learner views example after example and tries to deduce a description of the concept, so it is similar to Gold's paradigm in this regard, but it is different from Gold's in at least three other regards:

1. fixed-length bit strings, ergo finite bodies of data to purview;

2. bounded time to accomplish the learning, specifically time bounded by a polynomial in various parameters of the problem;

3. specific guidelines as to the form of the concept description.

The third difference is the most fundamental to the formulation and is the most germane to our discussion. Basically, Valiant's theory is intended to determine whether concepts *of a certain class* are easy to learn. For instance, if a concept can be expressed in conjunctive normal form with at most 4 variables per disjunct, is it possible to deduce that expression from seeing examples alone? If a concept can be expressed as a disjunct of two conjuncts, is it possible to deduce that expression from seeing examples alone?

His definition of learnability has some important other subtleties which capture probabilistic aspects of generalization. These are discussed in Section 7, but are not relevant to the present purposes.

3.3 Our Model

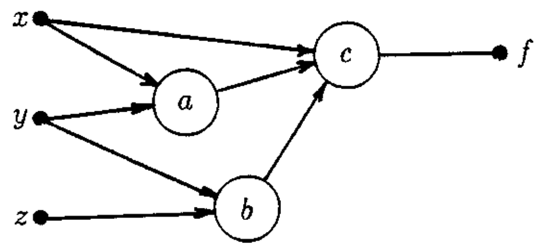
The present work describes a third field of learning theory which might be viewed as the lowest level of the three. It is inspired by the computational problem underlying the connectionist approach to learning. Whereas Valiant differed from Gold primarily on the issue of time, Valiant and this work differ primarily on the concern for the circuit involved in representing the data. Our paradigm is concerned not just with what is feasibly learnable, but with what is feasibly learnable in a machine *with a certain fixed structure.* It shares the same similarities and differences with Gold as Valiant, but its "specific guidelines as to the form" of the representation

$\mathcal{A}$ is a design class of architectures (networks).
A is an architecture (network).
p is a polynomial.
B is an algorithm.
F, G are configurations for A (i.e. settings for all the adjustable variables in all nodes of A).
$\mathcal{M}_F^A$ is the behaviour of A when configured with F.
$T \subseteq \{(\sigma, \rho) | \mathcal{M}_F^A(\sigma) = \rho\}$ is a task.

$$\text{“}\mathcal{A}\text{ is loadable”} \iff \left\{ \begin{array}{l} (\exists p, B) \text{ such that} \\ (\forall A \in \mathcal{A})(\forall F \text{ for } A)(\forall T \subseteq \mathcal{M}_F^A) \\ B \text{ halts in time } p(|A| + |T|) \text{ with} \\ \text{output } G \text{ such that } T \subseteq \mathcal{M}_G^A \end{array}\right.$$

Figure 3.3: Our definition of learnable (loadable)

(item 3 above) are even more strict than are Valiant's. See Figure 3.3. It requires that a representation for the learned data be found that can be embodied in a specific network structure. To achieve it, details of the function at each point in the net are alterable, but no alterations to the connectivity of the network are allowed. For example, if the given network were this:



and the data to be learned were values for f, x, y, and z such that f were some function of x, y, and z, then the objective of the learning system would be not only to discover the function $f(x, y, z)$ but also to find 3 more functions a, b, and c

such that $f(x, y, z) = c(x, a(x, y), b(y, z))$. This is more constrictive than Valiant's formulation because Valiant places only general grammatical guidelines on the form of f where we have an exact expression, minus only the specifications of a, b, and c. This prior knowledge of the form does not make the general learning paradigm any easier or harder, but merely different. It asks a question about whether a *particular* network can be made to represent some data, not whether it is possible to to find *some* network to represent those data.

3.4 Comparison Summary

This section summarizes the similarities and differences between the three learning paradigms being considered.

3.4.1 Requirements

In broad terms, each formulation is phrased as "For each member of problem class X, and for each of many different ways of presenting the learning data, the class is said to be learnable if there is a dependable way to remember the data." The first two clauses in this sentence correspond to different formalisms in each paradigm:

paradigm	For each member of the class	For each presentation of the data
Gold	$(\forall L \in \mathcal{L})$	$(\forall$ texts for $L)$
Valiant	$(\forall f \in F)$	$(\forall D^+, D^-)$
this work	$(\forall A \in \mathcal{A})$ $(\forall$ configurations F for $A)$ $(\forall T \subseteq \mathcal{M}_F^A)$	$(\forall$ permutations of $T)$

3.4.2 Motivation

Gold's study of Comparative Grammars is an attempt to characterize the class of natural languages through formal specification of their grammars. His original motivation for setting up a formal learning theory was not to understand learning for its own sake but to develop a tool to understand natural language. In attempting to define what a natural language is, he looked for constraints on its form provided by the observable fact that 2-year-olds can learn it. Furthermore, they learn it mostly by listening to others speak it. Thus formal learning theory was originally conceived to assist the comparative study of grammars but it ultimately might contribute to theories of psychology or neural architecture.

Valiant's motivation can be viewed as an attempt to develop a foundation for learning in AI. He wants to discover good models relevant to building devices that can learn, and to find the limits to what can be feasibly learned. True to traditions of AI, he uses an abstract Turing machine as the model of computation. Hence his paradigm is not concerned with any structural or functional constraints on the algorithms; it merely asks that an algorithm complete its task within a certain amount of time. Valiant's formulation is exactly relevant to AI, which shares these same freedoms and constraints.

Our motivation arises from the search to understand a very particular computational model. Like Gold, we use it not directly for its own sake but as a tool to constrain an ulterior theory. In seeking to understand connectionist computation, we seeks constraints on network design provided by the computational feasibility of learning.

3.4.3 Quantitative comparison

The various quantities involved in the three formulations are compared in this table:

	variables	input size	output size	time	data structure
Gold	infinite	infinite	finite	finite	any
Valiant	fixed	bounded	bounded	bounded	constrained
this work	fixed	bounded	fixed	bounded	fixed

3.4.4 'Grammatical Focus'

Gold studies the relation between finite evidence and infinite languages. This necessarily involves a grammar, but the form of the grammar is not explicitly mentioned. For example, he asks whether it is possible to find a description (i.e. a grammar) for any given recursively enumerable set.

We can cast the other 2 models in terms of grammars and languages as well. The data structure resulting from Valiant-type learning can be seen as a 'sentence' in a language described by grammatical syntax rules, where neither the words nor their interrelationships are known a priori but where the grammar serves as a validity test for the sentence. Valiant studies the relation between such a grammar for representation and the complexity of finding an appropriate sentence complying with that grammar. For example, he asks how hard it is to find a 3-CNF expression for a given body of data. The grammar involved is '3-CNF-ness', and the sentence sought would have to be a 3-CNF expression.

We study the relation between a grammar and the complexity of finding words to fill a specific sentence structure from that grammar. Our learned data structure is more particular than Valiant's in that it is not *any* sentence from some grammar, but is a *particular* sentence from it. For example, we ask how hard it is to load a network drawn from the family of two-layered networks. Two-layered-ness would be the 'grammar'. The specific 'sentence' involved could be the example network used on page 28, and using that example the 'words' sought would be specifications of the functions a, b, and c. The position and relationships of each 'word' are fully specified in advance, and the learning system need only discover what the missing words are.

3.4.5 Environment

One of Gold's original definitions is of an *informant*, which is a particular kind of 'environment', or protocol for interaction between a learning system and a source of data. An informant is an environment in which strings are presented serially to a machine paired with an indication of whether that string is in the target language or not.

Like Gold, Valiant explores a variety of environments, but one environment is quite similar to an informant. His terminology for it is 'positive and negative examples' of a concept.

Our protocol for gathering information is also quite similar to Gold's informant. The learner is presented with pairs of strings called *stimulus* and *response*. The object is to remember what response is appropriate for each stimulus. If the response string were only one bit long, it would be equivalent to saying IN/OUT (à la Gold) or POS/NEG (à la Valiant). The response string is a useful generalization of the one-bit notion but is not a conceptual deviation from the basic idea.

Hence we consider the three paradigms as having nearly equivalent learning environments. In fact, it is this commonality of supervised learning that makes the comparisons meaningful.

3.5 Studies in Connectionist Learning

Many researchers have developed algorithms for supervised learning in connectionist networks. A good review is given by Hinton [Hin87]. Some of the approaches most relevant to our study are the Perceptron [Ros61,MP72], linear associators [And72,Koh77,Koh84] back-propagation [RHW86,Par85,lC85], and the associative reward-penalty ($A_{R'-P}$) scheme [BA85,Bar85]. All of these are 'neural' algorithms for feed-forward networks.

A neural algorithm has been given for Boltzmann machines [AHS85,HS86], but the Boltzmann machine is a recurrent network. Hopfield [Hop82] gives a non-neural method, also for training recurrent (and thus dynamic) networks. Our work does not address this style of retrieval mechanism. For unsupervised learning paradigms, research has been done including [RZ85, and references therein]. The present work does not speak directly to this paradigm either so none of it will be reviewed here.

Analyses of the feed-forward models have been mostly for a single linear threshold unit or for a 2-layered machine where only one layer is trainable (Perceptron). A *layered* machine is one where the nodes are divided into disjoint sets called layers, network inputs are connected only to the first layer, and subsequent layers get their input signals only from a previous layer. There have also been some investigations of more general structures, which we will review after considering the work on simple networks.

3.5.1 Simple Networks

In the 'one-layer' case, learnability results span a great range. Some problems are impossible to solve; some can be solved 'in the limit', i.e. by using infinite time; some have time bounds that are known only to be finite; some have exponential time; some polynomial; and some logarithmic. The scaling arguments are with respect to s, the number of bits in the input vector/string. They are considered here in this same order.

Impossible: There are not nearly as many linearly-separable functions as there are general Boolean functions on $\{0,1\}^s$, so most Boolean functions on a large number of variables can not be performed (or perforce, learned) by a single-node linear threshold unit. In their book *Perceptrons: An Introduction to Computational Geometry* [MP72], Minsky and Papert answered questions regarding the functional powers of the 2-layer model and characterized classes of functions that could not be performed when both layers have bounded fan-in. Of course, any function can be performed with an exponential fan-in, but this is clearly impractical.

Infinite: Several asymptotic results have been given for stochastic approximation methods [SW81,DH73], for stochastic Learning Automata [NT74], and for a combination of these [BA85]. For instance, when placed in a stochastic setting, and modified by gradually reducing the adjustment constant, the classic Widrow-Hoff rule [WH60] has been shown to converge asymptotically to the solution of least squared error with probability 1. Another convergence theorem was given by Barto and Anandan [BA85] for a difficult reinforcement training protocol that involves noisy data and an impoverished form of feedback. They prove in a restricted case that the stochastic A_{R-P} procedure in one node will almost surely converge to correct responses. But these convergence theorems are only for *asymptotic* performance, which means the time upper bound is infinite.

Finite: Rosenblatt [Ros61] and others proved a theorem stating that the various Perceptron learning rules will eventually converge to correct weights if such weights do exist. See Nilsson [Nil65] for notes on the history of its various proofs. This development demonstrated that the Perceptron would learn in finite time, even though it was a very simple and 'neural' device.

Exponential: Muroga [Mur65] showed that there are linearly separable functions whose weights are approximately as large as 2^s. Thus even when the function is performable, it will take the various Perceptron learning rules $\Omega(2^s)$ adjustments before getting acceptable weights. Hampson and Volper [HV86] extended the argument to the average case (as opposed to the worst case) and derive a bound of $\Omega(1.4^s)$.

Tesauro [Tes87] measured learning time as a function of the size of the task. He used 3 networks of a particular style, one particular algorithm (back-propagation), and one particular function from which he draws t random items to make up a task. He then plotted learning time as a function of t, and found it to be the sum of a polynomial and an exponential. The polynomial dominated in the low ranges but after a certain point, the exponential dominated.

Polynomial: Hampson and Volper [HV86,VH86,HV87] explore several algorithms and learning situations for the single Perceptron to see how they behave as the number of input dimensions, s, is scaled up. They report exponential times for all but a few simple cases. When the additional dimensions are irrelevant or redundant, or when the task being learned is an OR or AND, then low polynomials in s are found.

Logarithmic: Littlestone [Lit87] found polynomial on-line mistake-bounds for a variety of classes of functions. He considers a node function set with the same form as linear threshold functions but he demands a minimum amount of separability between the different classes. (This restriction is a very appealing refinement to the model of a 'neural' node function set, since it allows the separating plane to be placed anywhere within a range and thereby relaxes the unrealistic requirement for arbitrary precision in the weights.) For the case where the target function is a simple disjunction of some subset of the input bits, he gives an algorithm that makes $O(k \log s)$ mistakes, k being the size of the relevant subset. When learning k-DNF expressions (for some fixed k), his algorithm has an upper bound of $O(kl \log s)$ mistakes. (l is the length of the expression learned, and s essentially measures the number of irrelevant input bits.) This is remarkable both for being linear in k and for being logarithmic in s.

Peled and Simeone [PS85] proved that it is *NP*-complete to decide if a function given in disjunctive normal form is linearly separable. This problem is more difficult than ours in that it has a very short input and must capture the whole function in a set of weights. Our problem has a much longer (extensional) representation of the desired function (which by the definitions of complexity affords an algorithm more time to run), and only requires the net to remember those items that are explicitly given. So with less to do and more time to do it, our loading problem is computationally easier; therefore their result is not tight enough for our purposes.

All these learning results are for single nodes (possibly preceded or followed by a layer of other non-learning nodes). They shed little light on our question about large, arbitrarily-shaped networks.

3.5.2 Complex Networks

Some attempts have been made to analyze the behaviour of learning algorithms in the context of composite networks. Rumelhart, Hinton, and Williams [RHW86] have shown that when the generalized delta rule is used in an arbitrary feed-forward network for making weight updates, the net has a gradient-descent behaviour. This is a pleasing result but there are at least two deficiencies: 1) No time bounds are available yet, and 2) Because the surface in weight-space is multimodal, the algorithm may descend into a local minimum and thereby never discover fully correct responses.

Tesauro and Janssens [TJ88] report empirical results studying the relationship between learning time and the predicate order, q, of a task. They measure a series of (network, task) pairs parameterized only by q. The net has q inputs, $2q$ nodes in the first layer (fully connected to each input) and a single output node (fully connected to each node in the first layer). The task is a complete listing of the $t = 2^q$ items for the parity function on q bits. When trained using back-propagation, they observe learning times of approximately 4^q. Since the task has size 2^q, this means the training time is $4^q/2^q = 2^q$ times the amount of data to be learned.

This result might also be re-interpreted as evidence that the learning time scaled exponentially in the size of the network.

In summary, there is good evidence that in general training neural networks is extremely time-consuming for large applications. Overall these results give one the impression that some very simple learning problems are easy, but when the problems are only slightly more difficult they become intractable. However, none of the results are really conclusive for networks of arbitrary shape. Networks of certain designs might find it easy to learn functions that are difficult to learn in the particular one-node or two-layer designs explored in the literature; or perhaps they will find easy ones hard.

Even beyond the explicit studies reviewed here, it is widely acknowledged that as networks get larger and deeper, their learning time grows prohibitively. The scale-up issue is therefore an important research problem for current connectionist research.

Chapter 4

THE INTRACTABILITY OF LOADING

Our major question is about the intrinsic nature of the learning problem we have posed: How difficult is it to load a given task into a given architecture? As discussed in the previous chapter, this amounts to asking how much time is required for a Turing machine to recognize the following language:

$$Perf_{\mathcal{F}} = \{(A,T) : \exists F \in \mathcal{F}^n \ni T \subseteq \mathcal{M}_F^A\}.$$

(Terminology used here and the related complexity-theoretical concepts of *NP*-completeness are explained thoroughly in Garey and Johnson [GJ79].)

The measure of how difficult a decision problem is must be relative to the size of a particular instance of the problem. The size of an instance of the performability problem is taken to be the number of bits that it takes to represent the instance, i.e. the architecture and the task. This number is roughly proportional to $|A| + |T|$. As the architecture gets bigger or as the task gets bigger, one would expect any algorithm to take longer to solve it, but the question we would like to answer is "How much longer?" What is the asymptotically minimum function $g(x)$ for the worst-case amount of time required to solve an instance of size x?

We prove below that $Perf_{AOFns}$ is *NP*-complete. This means that it belongs to a class of computational problems for which no polynomial time algorithms have ever been found. All *NP*-complete problems can be transformed into any other *NP*-complete problem in polynomial time, so the development of a poly-time algorithm would automatically give a poly-time solution to all of them. In fact it would imply that a deterministic machine could solve all the same problems that could be solved by a non-deterministic machine (i.e. a machine with a psychic ability to guess solutions) with no more than a polynomial degradation in running time. Technically, this development would be expressed as "$P = NP$", but it is believed to be exceedingly unlikely. Indeed, decades of experience have shown that the scale-up function for any *NP*-complete problem is an exponential expression that becomes unmanageably large even for small instances of the problem [GJ79, Chapter 1].

The fact that $Perf_{AOFns}$ is *NP*-complete is not a statement about the running time for one particular learning algorithm—it is a result about the intrinsic difficulty of the *problem.* Hence it is not practical to try to decide large instances of the performability question. (The instance of a loading problem is large when the network itself is large, even though there might only be a small amount of data to be loaded.) *No* learning rule can always solve this problem in polynomial time.

Furthermore, because this decision problem is no harder than the search problem from which it is distilled, the loading problem per se is also intractable. Assuming $P{\neq}NP$, no general-purpose algorithm can be developed for use in arbitrary architectures that is guaranteed to load any given performable task in polynomial time. (This is true whether the algorithm is conceived as a nodal entity working in a distributed fashion with other nodes, or as a global entity working in a centralized fashion on the network as a whole.)

The parallelism inherent in most neural network systems does not avoid this intractability. An exponential expression (c^n) cannot be contained by dividing it by a linear expression (cn). In many connectionist approaches to learning, there is a strong reason why large numbers of computing elements will not accomplish the loading problem in feasible time: By doubling the number of nodes available, you are doubling the computational resources but you may also be doubling (or squaring!) the amount of computing that has to be done. Naive attempts to exploit parallelism can actually be counterproductive.

Hence it might appear that we cannot hope to build large connectionist networks that will reliably learn simple supervised learning tasks.

The following section states and proves the fundamental theorem for one node function set and Section 4.2 shows how the result also applies to most other node function sets.

4.1 Proof of General Case using AOFns

To prove a problem, P_1, to be *NP*-complete, one must take another problem, P_2, that is known to be *NP*-complete and transform it into P_1. That is, one must give a polynomial time algorithm that can translate any arbitrary instance of P_2 into an instance of P_1 that is true if and only if the instance of P_2 is true. This algorithm is then called a 'reduction from P_2 to P_1'. See [GJ79] for an explanation of this technique. Of course if there is a poly-time solution to P_1 there will automatically be a poly-time solution to P_2 by first applying the reduction algorithm and then the solution algorithm, but such a composed procedure is presumed not to exist, so the solution algorithm is presumed not to exist either.

The particular *NP*-complete problem that we will use in the role of P_2 is called 3SAT. An instance of 3SAT is a expression in Boolean variables given in conjunctive normal form (i.e. a conjunction of disjunctions) in which all the disjunctions have exactly 3 literals. A literal is a logical variable or its negation. The instance is said to be true (or satisfiable) if the variables can all be given values such that the whole logical expression is true. For example the expression $(x_1, x_3, \overline{x_4})(x_2, \overline{x_3}, x_4)(\bar{x}_1, x_2, x_3)$ is satisfied by the assignments $x_1 = 0$, $x_2 = 1$, $x_3 = 1$, $x_4 = 0$.

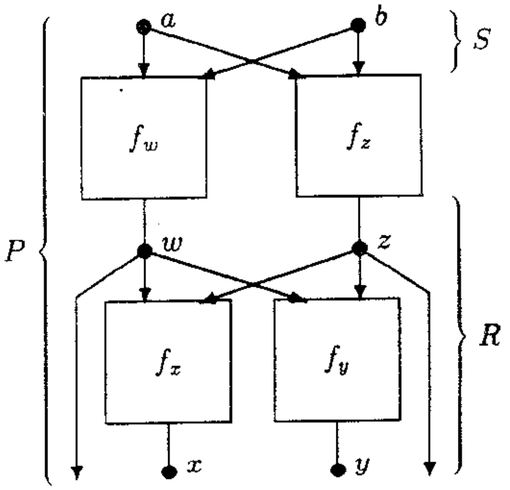
Theorem 1 *$Perf_{AOFns}$ is NP-complete.*

Proof: by reduction from 3SAT. Let the 3SAT problem be (Z, C) where Z is a set of variables $\{\varsigma_1, \varsigma_2, \varsigma_3, \ldots\}$ and C is a set of disjunctive clauses over them. Each clause has 3 literals. For (Z, C) to be satisfiable, there must be an assignment $\Pi : Z \rightarrow \{0, 1\}$ such that at least one literal in each clause has value 1.

A formal construction is given here for the architecture and task, followed by an exposé. Let $w = |Z|$ be the number of variables and $m = |C|$ the number of clauses. The 3SAT instance (Z, C) is reduced to (A, T), an instance of the loading problem, where

$$\begin{aligned}
A &= (P, V, S, R, E) \\
S &= \{a, b\} \\
R = V &= \{w_i, x_i, y_i, z_i : \varsigma_i \in Z\} \cup \{c_j : C_j \in C\} \\
P &= S \cup V \\
E &= \{(a, w_i), (a, z_i), (b, w_i), (b, z_i), \\
&\quad (w_i, x_i), (w_i, y_i), (z_i, x_i), (z_i, y_i) : \varsigma_i \in Z\} \\
&\quad \cup \{(w_i, c_j) : \varsigma_i \in C_j\} \cup \{(z_i, c_j) : \overline{\varsigma_i} \in C_j\} \\
T &= \{I_1, I_2, I_3\} \\
I_1 &= (0\,0,\ (0\,0\,0\,0)^w\, 0^m) \\
I_2 &= (1\,1,\ (1\,1\,1\,1)^w\, *^m) \\
I_3 &= (0\,1,\ (*\,0\,1\,*)^w\, 1^m)
\end{aligned}$$

This arcane piece of notation is explained in a 2-stage reader-friendly example. Stage 1: For every variable $\varsigma_j \in Z$ construct the partial architecture and partial task shown in Figure 4.1. From item 1 we know that $f_w(0,0) = f_z(0,0) = 0$; hence $f_x(0,0) = 0$ and $f_y(0,0) = 0$. From item 2 we know that $f_w(1,1) = f_z(1,1) = 1$;



	a b,	w x y z
item 1 :	(0 0,	0 0 0 0)
item 2 :	(1 1,	1 1 1 1)
item 3 :	(0 1,	$*$ 0 1 $*$)

Figure 4.1: The construction for each variable $\varsigma \in Z$.

hence $f_x(1,1) = 1$ and $f_y(1,1) = 1$. By comparing item 2 and item 3 we know

$$f_x(f_w(1,1), f_z(1,1)) = 1 \neq 0 = f_x(f_w(0,1), f_z(0,1))$$

$$f_w(1,1) \neq f_w(0,1) \text{ or } f_z(1,1) \neq f_z(0,1)$$

$$1 \neq f_w(0,1) \text{ or } 1 \neq f_z(0,1). \tag{4.1}$$

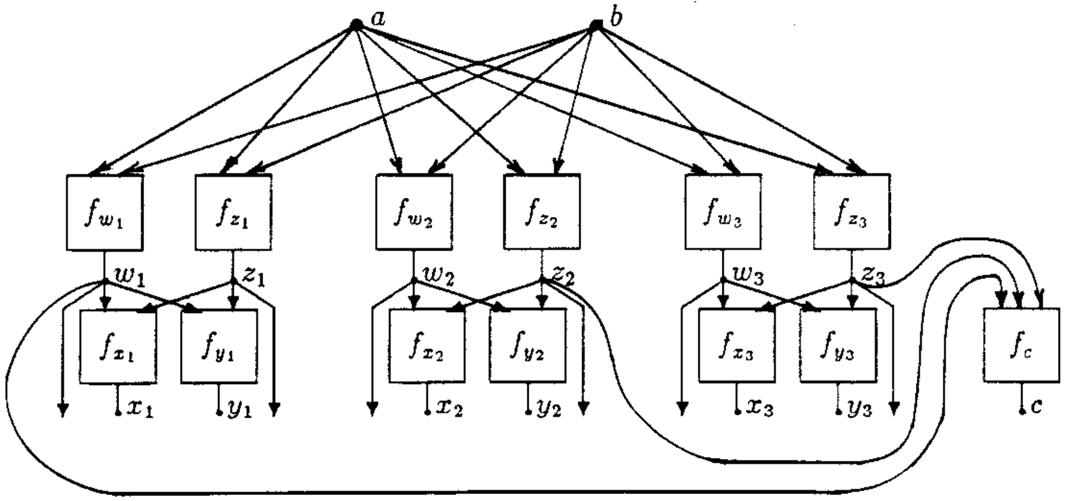
By comparing item 1 and item 3 we know

$$f_y(f_w(0,0), f_z(0,0)) = 0 \neq 1 = f_y(f_w(0,1), f_z(0,1))$$

$$f_w(0,0) \neq f_w(0,1) \text{ or } f_z(0,0) \neq f_z(0,1)$$

$$0 \neq f_w(0,1) \text{ or } 0 \neq f_z(0,1). \tag{4.2}$$

And from (4.1) and (4.2) we conclude $f_w(0,1) \neq f_z(0,1)$. We will associate some SAT variable ς_j with the group of nodes in this construction. For mnemonic value and brevity, let $\langle \varsigma_j \rangle$ stand for "the value computed by the w-node in the block of nodes associated with ς_j when given the input 0 1". And let $\langle \overline{\varsigma_j} \rangle$ stand for its negation—i.e. the output from the z-node for input 0 1.



	a	b,	w_1	x_1	y_1	z_1	w_2	x_2	y_2	z_2	w_3	x_3	y_3	z_3	c
item 1 :	(0	0,	0	0	0	0	0	0	0	0	0	0	0	0	0)
item 2 :	(1	1,	1	1	1	1	1	1	1	1	1	1	1	1	1)
item 3 :	(0	1,	*	0	1	*	*	0	1	*	*	0	1	*	1)

Figure 4.2: The composed construction for Theorem 1. This example is for the single clause $(\varsigma_1, \overline{\varsigma_2}, \overline{\varsigma_3})$.

Stage 2: For each clause in the SAT system construct a single node in the second layer of the architecture with inputs from all nodes associated with its participating literals. Putting variables' nodes and the clause node together, we get what is shown in Figure 4.2. It shows the construction for an example SAT system consisting of only one clause $(\varsigma_1, \overline{\varsigma_2}, \overline{\varsigma_3})$. Observe that each item consists of the stimulus from an item from Figure 4.1, three replications of its response (one per variable), and another response bit for the clause node (node c).

Claim: The constructed architecture can perform the task iff the SAT instance is satisfiable.

Proof: Remember that $f_w(0,0) = f_z(0,0) = 0$ in each variable construct. By inspecting item 1 and item 3,

$$f_c(0,0,0) = 0 \neq 1 = f_c(\langle\varsigma_1\rangle, \langle\overline{\varsigma_2}\rangle, \langle\overline{\varsigma_3}\rangle).$$

Hence

$$\langle\varsigma_1\rangle \neq 0 \text{ or } \langle\overline{\varsigma_2}\rangle \neq 0 \text{ or } \langle\overline{\varsigma_3}\rangle \neq 0,$$

which is exactly the semantics of a disjunctive clause. If Π exists then let $\langle\varsigma_j\rangle = \Pi(\varsigma_j)$, that is

$$f^j_w = \begin{cases} \text{OR} & \text{if } \Pi(\varsigma_j) = 1 \\ \text{AND} & \text{if } \Pi(\varsigma_j) = 0 \end{cases} \quad \text{and} \quad f^j_z = \begin{cases} \text{AND} & \text{if } \Pi(\varsigma_j) = 1 \\ \text{OR} & \text{if } \Pi(\varsigma_j) = 0 \end{cases} .$$

For all variables ς_j let $f^j_x = $ AND and $f^j_y = $ OR, and for the clause node let $f_c = $ OR. The reader is welcome to check that this configuration performs the task.

Conversely, if a configuration exists let $\Pi(\varsigma_j) = \langle\varsigma_j\rangle$, and observe $\varsigma_1 = 1$ or $\overline{\varsigma_2} = 1$ or $\overline{\varsigma_3} = 1$ as required. This proves the claim. □

The extension to multi-clause systems should be clear.

Thus we have SAT $\propto$ $Perf_{AOFns}$ and it is easy to see that the algorithm for the transformation runs in polynomial time (in fact linear time and log space).

Finally, it must be demonstrated that there is a non-deterministic machine that can decide $Perf_{AOFns}$ in time polynomial in the length of (A, T). Writing down a complete configuration of AOFns takes one bit for each node in A. That the configuration is correct can be checked by evaluating each node function once for each item in T. This takes time $O(|V| \times |T|)$ under the assumption that it takes constant time to evaluate any single f_i.

This, and SAT $\propto$ $Perf_{AOFns}$ implies $Perf_{AOFns}$ is NP-complete. □

This proof is intended to be applicable to $Perf_{\mathcal{F}}$ for $\mathcal{F}$ being more than just AOFns. Hence it begins by forcing $f_w(0,0) = 0$ and $f_w(1,1) = 1$ (amongst other things). Such could have been assumed from the outset since $\text{OR}(0,0) = 0$ and $\text{AND}(1,1) = 1$ but we chose not to exploit these peculiarities of AOFns in the proof. Regardless of the value for $f_w(0,1)$, one of $\{\text{AND},\text{OR}\}$ will satisfy all the requirements, so the proof is strong enough to apply to AOFns while not being specific to it.

This proof uses the "don't-care" symbol, but such is not always a part of the learning protocol used in connectionist studies. In appendix C there is another version of the proof that avoids the "don't-care" by using some extra signals and nodes. Hence this detail does not strongly alter the nature of the problem.

4.2 Other Node Function Sets

The intent of this section is to demonstrate that the intractability of the performability problem does not depend much on the particular node function set being used—its difficulty remains for essentially all non-trivial cases.

Theorem 1 deals only with AOFns, but connectionist studies typically use LSFns, the linearly separable functions. LSFns includes all of AOFns and, when the number of inputs to a node is large, it is considerably more powerful. It might seem, therefore, that this extra power would make loading easier. Unfortunately, this case (and even LUFns) is just as hard.

Corollary 2 *For any node function set $\mathcal{F}$ such that all members of F are binary-valued functions, and $\mathcal{F} \supseteq \{\text{AND}, \text{OR}\}$, $Perf_{\mathcal{F}}$ is NP-hard.*

Proof: Both directions of the proof of the claim in Theorem 1 require nodes able, at least, to perform functions from AOFns. The reduction thus follows for any node function set that includes them. □

Corollary 3 *$Perf_{LSFns}$ is NP-complete.*

Proof: *NP*-hardness follows from Corollary 2, so we need only to show that $Perf_{LSFns}$ is in *NP*. For this to be true, there must exist some poly-time way of guessing a function from LSFns and being sure that indeed it *is* from LSFns. If fan-in were bounded in our model, then this would be easy since we could get the non-deterministic selection to be from a fixed table of all LSFns up to that input size. Without bounds on fan-in, this technique will not work. One might attempt to achieve a selection from LSFns by simply writing down the weights that are used in the linear sum, but since the weights are assumed to be *real* (i.e. of a potentially infinite number of decimal places), this technique is also inadequate. However, Hong [Hon87] has recently proved that approximations to the weights are sufficient to encode any and all members of LSFns. Specifically, only a polynomial number of digits are required (polynomial in the fan-in), and hence $Perf_{LSFns}$ is *NP*-complete. □

Muroga [Mur71, thm 9.3.2.1] implicitly proves the same result about polynomial bounds on the weights in LSFns. It is tighter but less direct.

Corollary 4 *$Perf_{LUFns}$ is NP-complete.*

Proof: Again, *NP*-hardness follows from Corollary 2, but to prove $Perf_{LUFns}$ to be in *NP*, we must give some format for guessing members of LUFns. It must have some poly-time way of writing down an arbitrary function and checking that it is in LUFns.

To fully specify an arbitrary member of LUFns requires $2^{|pre(v_i)|}$ bits and hence it takes exponential time to write it down. (The statement of the theorem implies no bound on the fan-in to a node.) However, each node function will be invoked exactly $t = |T|$ times in the performance of the task; hence we can specify a function $F \in$ LUFns by asserting a default value (1, say) to cover most inputs, and then listing the exceptional inputs, α, for which $F(\alpha) = 0$ (of which there are at most t). Since T has a unary encoding of t, there is a representation of F that is polynomial in the length of (A, T), and this means that a function can always be written down in poly time.

Making sure that such a function is a member of LUFns is trivial since *all* binary-valued functions are members. Hence $Perf_{\mathcal{F}} \in NP$ even when $\mathcal{F}$ = LUFns, and $Perf_{LUFns}$ is NP-complete. □

LSFns is a special case of the quasi-linear functions (QLFns). Theorem 3 pertains only to discrete, binary-valued signals and does not apply to real-valued quasi-linear functions. However, another theorem pertains specifically to the popular logistic-linear functions (LLFns) used in back-propagation:

Theorem 5 *$Perf_{LLFns}$ is NP-complete.* □

Proof in appendix B. As a corollary, performability with the more general class of quasi-linear functions, $Perf_{QLFns}$ is also NP-hard.

These theorems indicate that the difficulty in the loading problem has very little to do with the choice of node function sets. This observation is strengthened below in Theorem 12 below which states that some tasks which are performable using very restricted node function sets, are difficult to load even when that node function set is greatly expanded. This argues that the difficulties of loading will not be overcome by searching for ever more powerful node types.

We end this chapter with a more convenient statement of our main result:

Corollary 6 *Loading is NP-complete.*

Proof: The decision problem is NP-complete, and since being able to solve the search problem would allow one to answer the decision problem, the search problem must be at least as hard. □

Note that no node function set is explicitly mentioned in this corollary. There are two ways in which this makes the corollary technically loose. First is that for an absurdly simple node function set (e.g. where the set has only one member), the problem is not NP-hard. Second is that deciding membership in an absurdly complicated node function set (e.g. where the truth-table representation of each function must name a halting program), might not be NP-easy. However, in the common cases, and in other reasonable cases we explored the result is robustly true. Because it holds for any node function set of interest, we will hereafter omit the specification of the node function set in order to imply generality.

We note that a different proof of the NP-completeness of $Perf_{LSFns}$ has recently been found by Blum and Rivest [BR88]. Their proof differs from ours in that different parameters are scaled up. Our proof scales up the size of the architecture and the number of bits in the response strings while keeping the number of items and the number of bits in the stimulus strings constant. Their proof keeps the size of the architecture and the number of bits in the response strings constant while scaling up the number of items and the number of bits in each stimulus string.

It might also be noted that there are node function sets for which performability can be proved NP-complete without scaling up the size of the task or the length of the strings at all—using such a node function set, machines of arbitrary size would be unable to load even a *fixed* amount of data!

Chapter 5

SUBCASES

Our results preclude only the broadest, most ambitious interpretation of the goal of connectionist learning. Essentially, the goal we have formulated is to find an algorithm that is *guaranteed* to load *any* performable task in *any* conceivable net. One can imagine several ways to constrain the problem in such a way that the new loading problem would have some special regularity might facilitate its solution. Such constraints would involve

- restrictions on architectural design,
- restrictions on tasks restrictions, and/or
- different criteria of success.

For most such sub-cases, our theorem says nothing.

This section discusses several ways to define sub-problems and/or different problems that may be easier to solve than the general loading problem formulated above. Interspersed amongst these comments are several corollaries to the above proof that state further negative results.

5.1 Architectural Constraints

First, Theorem 1 is a statement about networks and tasks *in general*, but there may be large useful classes of networks (defined by some design restrictions) where loading a task would always be achievable in polynomial time. It has been an empirical observation that although some algorithms (notably back-propagation) work well in nets that have only a few levels intervening between input posts and output posts, they work much slower in deep nets. One might be tempted to infer that shallow nets would be intrinsically easier to load. By examining the construction in the above proof we see this is not so. The construction uses only 2 layers and yet an algorithm for loading it was shown to be equivalent to an algorithm for solving 3SAT. Hence:

Corollary 7 *Loading is NP-complete even when the architectures are restricted to be of depth* ≤ 2 *and of fan-in* ≤ 3. □

Rather than limit the *maximum* depth or fan-in, what is more likely to help is a restriction that sets a *minimum* depth (say as a function of the width of the net), or a *minimum* fan-in, because this forces a minimum number of degrees of freedom everywhere. Since experimental evidence seems to contradict both these suggestions, it would be important to resolve the issue.

Other architectural design constraints have been explored. As a first piece of analysis, we have some examined issues in shallow networks that have gross structure extending through their width. The results are interesting and substantial enough to warrant a separate chapter. See Chapter 6.

One avenue of freedom usually not exploited by connectionist learning schemes is to alter the architecture as learning proceeds. When carried to extremes, this would amount to an exercise in arbitrary circuit design, rather than in connectionist learning, but adhering rigidly to the starting architecture may be just too constrictive; somewhere between these two extremes there may be a balance that combines the best of both worlds. Valiant and others [Val84,KLPV87] have initiated the study of what can be feasibly learned using total freedom of connectivity within a certain class of architectures. For example, their μ-expressions are the same as tree-shaped architectures that use AOFns.

It is conceivable that the difficulties in loading stem specifically from the non-recurrence of the nets and the fact that all their 'knowledge' about a stimulus must be elicited in one single evaluation of each node function. If so, then a more reasonable model of network memory might involve storing data as cycles in state-space where the power of attractor dynamics could be exploited to make loading easier (albeit at the cost of more expensive retrieval). Such would be a large departure from our model but there are plenty of pitfalls there too; Porat [Por87] proves that in such a system the problem of deciding just if a configured network stabilizes or cycles is NP-hard. See also [God87,Lip87].

5.2 Task Constraints

Next, our formulation of the learning problem may be inappropriate in that it requires a network to be able to load too large a class of tasks. By using performability as the decision problem, we are in effect defining the task class in terms of the architecture itself and asking that any architecture A be able to load any task in the set $P^A = \{T : \exists F \ni \mathcal{M}^A_F \supseteq T\}$. But it is not necessary to expect an architecture to be able to load all of these tasks. From a practical point of view, all that is necessary is that it be able to perform and load some useful class, $\mathcal{T}$, of tasks. Obviously, it is necessary that $\mathcal{T} \subseteq P^A$, and the results herein show that it is too ambitious to have $\mathcal{T} = P^A$ for arbitrary A. However, there are many ways to define $\mathcal{T}$ so as to exclude some tasks in P^A, thus possibly leading to a loadable class. It would be useful to be able to characterize just what class of tasks a network could learn, or conversely, to be able to determine what types of architectures could learn a given class of tasks.

Our main theorem has implications for the restricted classes of monotonic tasks, small tasks, and tasks that are performable using very small and simple node function sets.

Define $\sigma \succeq \delta$ to mean that every element of the binary vector σ is a 1 if its corresponding element in binary vector δ is a 1. A *monotonic function* is a function g such that

$$\sigma \succeq \delta \Rightarrow g(\sigma) \succeq g(\delta).$$

A *monotonic task*, T, is a set of items such that for some monotonic function g, T agrees with g:

$$(\sigma, \rho) \in T \Rightarrow \rho \text{ agrees with } g(\rho).$$

Corollary 8 *Loading is NP-complete even when tasks are restricted to be monotonic.* □

Corollary 9 *Loading is NP-complete even when there are there are only two bits in the stimulus strings ($s = |S| = 2$).* □

Corollary 10 *Loading is NP-complete even when tasks are restricted to be of no more than 3 items.* □

A more promising avenue is to define the task restrictions in terms of what is performable by a network that is in some way less powerful than the network being loaded. One technique for doing this uses the notions of 'teacher' and 'learner'. A *teacher* is a network that is used to define the set of tasks that a *learner* network will be required to load. The word 'teacher' is to connote what has to be learned; it is not to be confused with some mechanism for facilitating the loading process. For example, suppose we have a network, A, that can perform a task, T, using only those node functions in the set $\mathcal{G}$. And suppose that another network of the same architecture but capable of using a (larger) node function set $\mathcal{F}$ is charged with loading T. Call the first network the *teacher* and the second the *learner*. If $\mathcal{G} \subset \mathcal{F}$ then the tasks performable by the teacher will be a subset of the tasks performable by the learner. Is it easier to decide performability of this smaller set of tasks?

To denote this new question, the parameters for describing the teacher are written to the left of *Perf* and those for the learner are written to the right; the current example is denoted by ${}_{\mathcal{G}}Perf_{\mathcal{F}}$. Formally, it requires for all architectures, A, and for all tasks, T, to be able to compute an output, d, such that

$$d = 1 \quad \Rightarrow \quad \exists F \in \mathcal{F}^n : T \subseteq \mathcal{M}_F^A$$

$$d = 0 \quad \Rightarrow \quad \nexists G \in \mathcal{G}^n : T \subseteq \mathcal{M}_G^A$$

Note that in some cases *either* answer would be correct, and hence we call this a *relaxed decision* problem.

The teacher/learner device parallels the technique used by Pitt and Valiant in [PV86, definition 1.2].

The question ${}_{\mathcal{F}}Perf_{\mathcal{F}}$ is exactly the original type of question $Perf_{\mathcal{F}}$.

The following theorem shows that no advantage can be made of extra node function power to load tasks:

Corollary 11 *${}_{\mathcal{G}}Perf_{\mathcal{F}}$ is NP-complete for $\mathcal{F}$ and $\mathcal{G}$ being any reasonable superset of AOFns.*

Proof: This follows by the same argument used for Corollary 2. Both directions of the proof of the claim on page 43 in Theorem 1 only require nodes able, at least, to perform functions from AOFns. As long as $\mathcal{F}$ includes AOFns, one direction of the proof holds, and as long as $\mathcal{G}$ includes AOFns, the other direction of the proof holds. □

Just to emphasize how a large difference in node functionality makes no difference in loading complexity, witness an extreme case of the above corollary:

Corollary 12 *Loading an architecture A using LUFns is NP-complete even when the tasks are restricted to be performable by A using AOFns.* □

This corollary deals with a type of task restriction, but it also provides further evidence that the *NP*-completeness of the loading problem does not derive from difficulties inherent in the node function set. Devising ever more powerful node functionality will not overcome the intractability here.

5.3 Relaxed Criteria

Finally, our mathematical question has a very exacting criterion of success in training: either the machine performs perfectly or it doesn't. If the criterion was more lenient then the problem might be much easier. Some probabilistic or approximate criterion of learning might be more appropriate. Here is one that won't help:

Corollary 13 *Loading is NP-complete even when only 67% of the items are required to be retrieved correctly.*

Proof: Loading slightly more than $\frac{2}{3}$ of 3 items is the same requirement as loading all 3 items. □

5.4 Summary

Note that all the restrictions mentioned in this section actually hold simultaneously.

Corollary 14 *Loading is NP-complete even when*

- *the architectures are restricted to be of depth* ≤ 2 *and of fan-in* ≤ 3,
- *tasks are restricted to be monotonic,*
- *there are there are only two bits in the stimulus strings* ($s = |S| = 2$),
- *tasks are restricted to be of no more than 3 items,*
- *only 67% of the items are required to be retrieved correctly, and*
- *tasks are restricted to be performable by AOFns, although a configuration may draw node functions from LUFns.*

Chapter 6

SHALLOW ARCHITECTURES

The loading problem is *NP*-complete even for networks of depth 2, so rather than attempting to deal with deep nets, we shall limit our attention to shallow nets and try to identify additional constraints that yield tractable loading problems. For further justification of this strategy, we quote Baldi and Venkatesh [BV87]:

> It is not unusual to hear discussions about the tradeoffs between the depth and width of a circuit. We believe that one of the main contributions of complexity analysis is to show that this tradeoff is in some sense minimal and that in fact there exists a very strong bias in favour of shallow (i.e. constant depth) circuits. There are multiple reasons for this. In general, for a fixed size, the number of different functions computable by a circuit of small depth exceeds the number of those computable by a deeper circuit. That is, if one had no prior knowledge regarding the function to be computed and was given m hidden units then the optimal strategy would be to choose a circuit of depth two with the m units in a single layer. In addition, if we view computations as propagating in a feedforward mode from the inputs to the output unit, then shallow circuits compute faster. And the deeper a circuit, the more difficult become the issues of time delays, synchronization, and precision on the computations. Finally, it should be noticed that given overall responses of a few hundred milliseconds and given the known time scales for synaptic integration, biological circuitry must be shallow, at least within a "module" and this is corroborated by anatomical data.

We introduce the notion of a support cone, which is the set of nodes that can affect the behaviour of an output node. On this is built the notion of the Support

Cone Interaction (SCI) graph of an architecture, which isolates computationally salient features of an architecture by explicitly denoting only the overlaps between support cones. Finally, by applying a limit to the size of the support cones, we create a type of formal constraint that is powerful enough to mask off the difficult issues involved in loading deep nets without interfering with our theoretical investigation into issues of width. We have used the term 'shallow networks' to mean a family of networks whose maximum support cone size is limited by some parameter but where there is no limit on the number of nodes. This has the effect of defining a family of bounded depth and unbounded width.

We show that limiting the size of the support cones is not enough in itself to make loading tractable. Indeed, even when attention is further restricted to architectures whose SCI graphs are regular planar grids the problem is *NP*-complete. Only when additional constraints are added that serve to prohibit the existence of large grids within the SCI graph are feasible problems identified: polynomial-time loadable architectures are found for the case where the SCI graph is of limited tree-width.

6.1 Definitions

Definition In an architecture $A = (P, V, S, R, E)$, each output node $x \in R$ has a *support cone*, $\mathrm{sc}(x)$, which is the set of all nodes in V that can potentially affect the output of that node; that is, it is the set of predecessor nodes:

$$\mathrm{sc}(x) = \{x\} \cup \{\mathrm{sc}(y) : y \in \mathit{pre}(x) \cap V\}.$$

The network retrieval behaviour at any particular output node is determined by (and only by) the functions assigned to each node in its support cone.

Definition A *support cone interaction graph* (SCI graph) for an architecture, is an accounting of the interactions between support cones. It is a graph with nodes

$\{z_1, z_2, \ldots, z_r\}$ corresponding one-to-one with the output nodes, R_i, and having edges $\{(z_i, z_j) : \mathrm{sc}(R_i) \cap \mathrm{sc}(R_j) \neq \emptyset\}$.

Definition A *partial configuration* for node x is an assignment of functions to each node in its support cone:

$$F_x : sc(x) \rightarrow \mathcal{F}.$$

A *partial configuration for a group of nodes*, X, is an assignment of functions to all nodes in all of its support cones:

$$F_X : \bigcup_{x \in X} sc(x) \rightarrow \mathcal{F}.$$

Definition The *support cone configuration space* (sccs) for output node x is the set of all partial configurations for the support cone of x.

Since we are considering only binary functions of binary values for each node in a finite graph, the size of a sccs is always finite.

Definition A family of architectures is *shallow* if the size of the largest sccs in each architecture is bounded. (At first, assume it is bounded by a constant; this will be loosened later.)

Note that this limitation has the effect of bounding the depth of a network, the maximum fan-in to any node, and the number of different functions in the node function set, although it does not dictate how these things are traded off against each other.

The complete sccs for any node in any architecture in a shallow family can be exhaustively searched in constant time.

6.2 Grids and Planar Cases

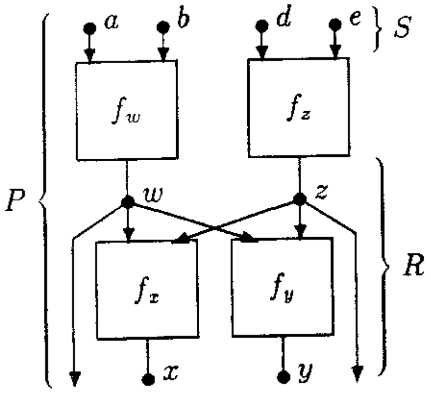
This section starts from our previous NP-completeness result on shallow architectures and tightens it to apply to two progressively more constrained families of shallow architectures.

The proofs are extensions of the one used for Theorem 1 so the first thing we do in this section is import the construction used there and make a minor change. Note the construction in Figure 6.1a is almost identical to the one given earlier in Figure 4.1 (page 42) except that $S = \{a, b, d, e\}$ instead of just $\{a, b\}$. The tasks remain functionally the same, however, because input a is identical to input d, and b is identical to input e.

To make the next proofs easy to read, a pictorial notation for architectures and tasks is used which eliminates excessive formality. In Figure 6.1a the network has been depicted on the page so that information flowed across the plane of the page, as is customary in the connectionist literature. Figure 6.1b shows an alternate view of this same architecture, the *plan view*, which is a view "from above". If a network is drawn in such a way that during retrieval the Stimulus originates above the page, information flows into the page, and the Response arrives below the page then the network is drawn in plan view. The items shown in Figures 6.1a and 6.1b are also different representations of the same task.

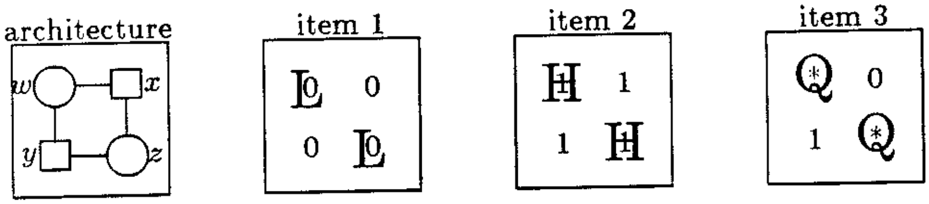
As in the proof for Theorem 1, each clause in the 3SAT system corresponds to a single node in the second layer of the constructed architecture with inputs from all nodes associated with its participating literals. Putting all the variables' nodes together with the clause node yields something like what is shown in Figure 6.2. It is a plan-view re-representation of Figure 4.2.

As before, the largest support cone in this construction has only 4 nodes in it and the largest fan-in is only 3, so the largest sccs is of limited size. Hence this



	a	b	d	e,	w	x	y	z
item 1 :	(0	0	0	0,	0	0	0	0)
item 2 :	(1	1	1	1,	1	1	1	1)
item 3 :	(0	1	0	1,	$*$	0	1	$*$)

(a) The construction for each variable in the SAT system. The architecture shown on the left is drawn in the classic side view. The 3 items in the task as shown on the right. Zeroes and ones are desired responses; the asterisks are 'don't cares'. This construction is nearly identical to the one used in Figure 4.1, except that $s = |S| = 4$ rather than 2.



(b) The plan view of the construction for each variable in the SAT system. This is a different representation of what is shown in part (a) above. On the left is the plan view of the architecture. Round nodes are first-layer nodes and each has 2 external input connections (which are not shown). Square nodes are second-layer nodes and have input connections from the round nodes. All nodes have external output connections (which are not shown). The 3 diagrams on the right are pictorial representations for the same 3 items as appear in (a) above. The letter L stands for the 2-bit input 0 0; H stands for 1 1; and Q stands for 0 1. The zeroes and ones are desired responses; the asterisks are 'don't cares'. Each character is positioned to correspond to a node as drawn in the left diagram. First layer nodes have stimulus bits and required responses as well.

Figure 6.1: Plan view notation

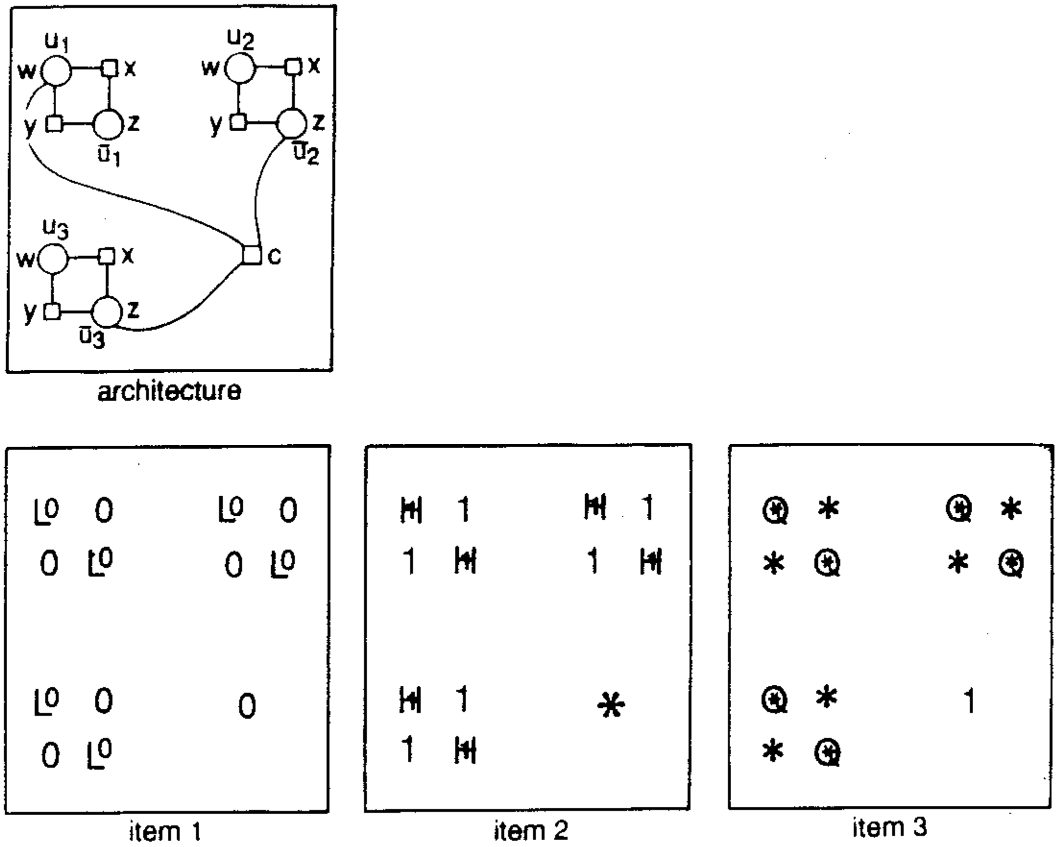


Figure 6.2: Plan view of the composed construction. It uses notation established in Figure 6.1. This example is for the single clause $(u_1, \overline{u_2}, \overline{u_3})$. At top left is the plan view of the architecture. Node c is a second-layer node that is used to enforce the disjunctive semantics of the clause. Below are the 3 items.

family of constructions fits the definition of shallow networks, and this construction is therefore sufficient to prove what is actually a looser version of Corollary 7:

Corollary 15 *Loading shallow architectures is NP-complete.* □

Our first intuition after realizing this was that the problem was difficult because the architecture lacked any regular structure—constraints in one part of the network could immediately impact options in any other part of the network. Connections in the architecture could reach and thereby propagate constraints from anywhere to anywhere. To prevent this, we sought reasonable restrictions to place on the SCI graph so that constraints generated in one part of the architecture would stay somewhat local to the area in which they originated. One such device was to require the SCI graph to be planar. Unfortunately,

Theorem 16 *Loading shallow planar-SCI architectures is NP-complete.*

Proof[1]: Note one incidental fact about the reduction used in the proof for Theorem 15—that the SCI graph for an architecture in that family of constructions is identical to the plan view of the architecture (minus directions on the edges). We will use a similar construction in this theorem; the architecture used will have a planar plan view and a planar SCI graph simultaneously.

The proof of Corollary 15 can be re-employed for the present theorem here so long as we can arrange for no arcs to cross in the drawing of the SCI graph. This is done in the usual way (see [Lic82])—we show how to eliminate all crossing arcs without altering the relevant aspects of the graph. See the 'crossover construct' in Figure 6.3.

[1] This proof employs a node function set which is not linearly separable, and therefore is not directly applicable to the conventional connectionist devices. However, there is a more elaborate construction based on an invention by Lichtenstein [Lic82] that holds for the standard linear threshold functions. See appendix D.

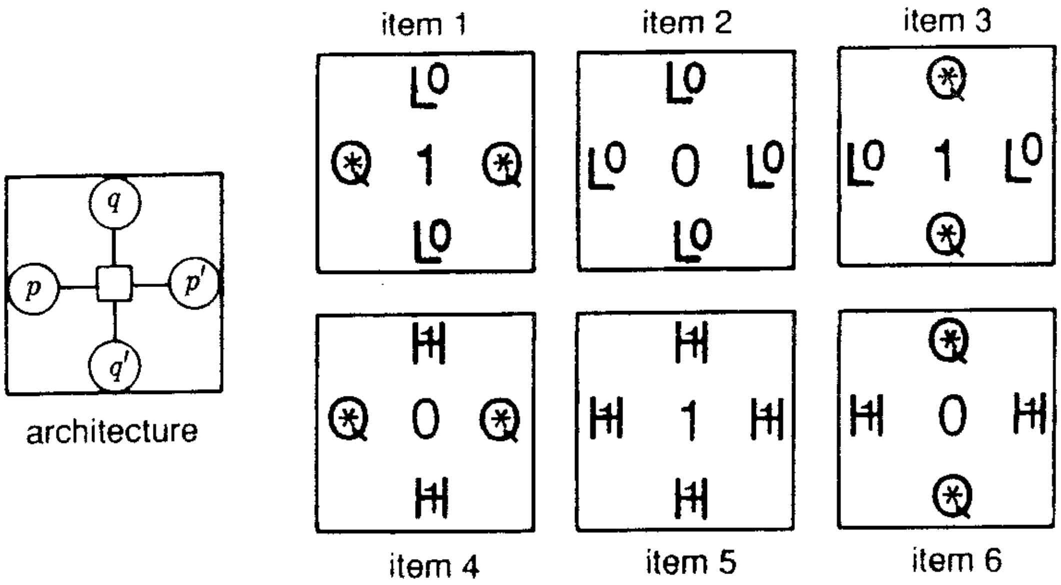


Figure 6.3: The construction for crossovers. The architecture is shown in plan view at left. The 6 items shown at right force $f_p(0,1) = \neg f_{p'}(0,1)$ and $f_q(0,1) = \neg f_{q'}(0,1)$.

Let the label in a node in this diagram also denote the value emitted by that node for input 0 1 (input 0 1 is abbreviated as a Q in the item diagram). By comparing item 1 with item 2 deduce that $p \neq 0$ or $p' \neq 0$. By comparing item 4 with item 5 deduce that $p \neq 1$ or $p' \neq 1$. From these it follows that $p \neq p'$. Similarly, by comparing item 2 with item 3, and item 5 with item 6, it follows that $q \neq q'$. Thus p' is a copy (albeit a negative copy) of p, and q' is a (negative) copy of q. The copies can be re-inverted using the construction in Figure 6.1b. Thus the information about p and q 'pass through each other' in the plane and the techniques for proving Theorem 15 can be used for the present theorem as well.

Since there are only a polynomial number of crossing points in a graph, each one can be replaced by the (fixed) amount of extra construction given here and we still have a polynomial reduction from 3SAT. □

SCI planarity is not a tight enough constraint to escape *NP*-completeness. In fact, no kind of local topology constraint on the SCI graph that is still open to 2-dimensional expansion seems to hold much promise. Define a *grid* as a checkerboard graph on nodes $x_{i,j}$ and edges are either $(x_{i,j}, x_{i+1,j})$ or $(x_{i,j}, x_{i,j+1})$. Witness:

Theorem 17 *Loading shallow grid-SCI architectures is NP-complete.*

Proof: All the individual constructs in Figure 6.1b and Figure 6.3 can fit easily into a grid topology. It remains to show how they can all be connected. For this we need only show how to transform one of the arbitrary-shaped and arbitrary-lengthed arcs of Figure 6.2 into an equivalent implication while following grid lines; i.e. how a variable can be propagated from one point on the grid to most any other point. Using the construction from Figure 6.1b we can make a negated copy of a variable in a diagonally adjacent node. Using the construction from Figure 6.3 we can make a negated copy of a variable in a node 2 places away horizontally or vertically. Using combinations of these, we can copy a variable either positively or negatively to any other node in the grid. See Figure 6.4 for examples. Thus any construction for Theorem 16 can be padded with extra nodes until it becomes a grid structure. □

These grid SCI graphs have node degree 4. Loading is also NP-complete when the SCI graph is a hexagonal array (node degree 3). Proof omitted. When node degree is limited further to just 2, the SCI graph becomes a chain and the problem is easy. Proof in the next section.

6.3 Definitions Again

Definition Let $\mathrm{DOM}(X)$ denote the domain of the function X. Two configurations F and G are said to be ***compatible***, written $F \cong G$, if they have a common extension:

$$F \cong G \iff \forall v \in \mathrm{DOM}(F) \cap \mathrm{DOM}(G) \quad F(v) = G(v)$$

Note that a partial configuration for node a is trivially compatible with a partial configuration for node b if $\mathrm{sc}(a) \cap \mathrm{sc}(b) = \emptyset$.

The union of two configurations F and G is defined when $G \cong H$:

$$F = G \cup H \iff \mathrm{DOM}(F) = \mathrm{DOM}(G) \cup \mathrm{DOM}(H), F \cong G, F \cong H$$

The usual notion of restrictions on functions is also useful:

$$F = G|_A \iff \mathrm{DOM}(F) = A,\ F \cong G,\ \mathrm{DOM}(G) \supseteq A$$

Definition A *correct partial configuration*, $\tilde{F}$, for node x is a partial configuration with the property that for any extension of $\tilde{F}$ to a complete configuration F, $\mathcal{M}_F^A$ at node x agrees with the corresponding response bit over all items in the task. A correct partial configuration for a group of nodes contains a correct partial configuration for each node in the group.

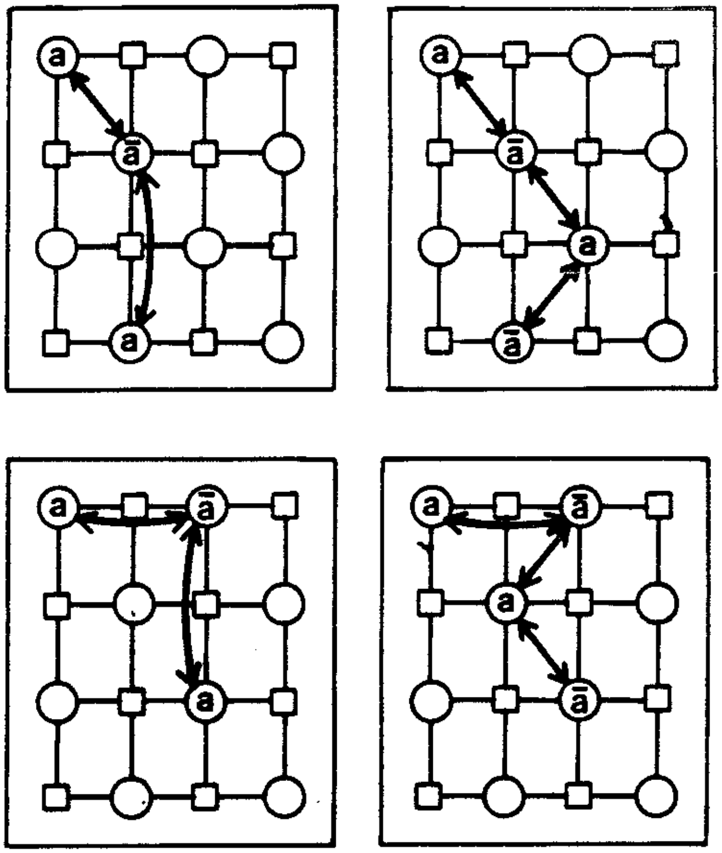


Figure 6.4: Example task designs for propagating variables. Each diagram shows the plan view of a 2-layer architecture. The horizontal and vertical arrows indicate the effect of the task construct in Figure 6.3; the diagonal arrows indicate the effect of the task construct in Figure 6.1b. These four diagrams illustrate that a variable or its negation can be propagated throughout a grid architecture from one first-layer node to any other first-layer node.

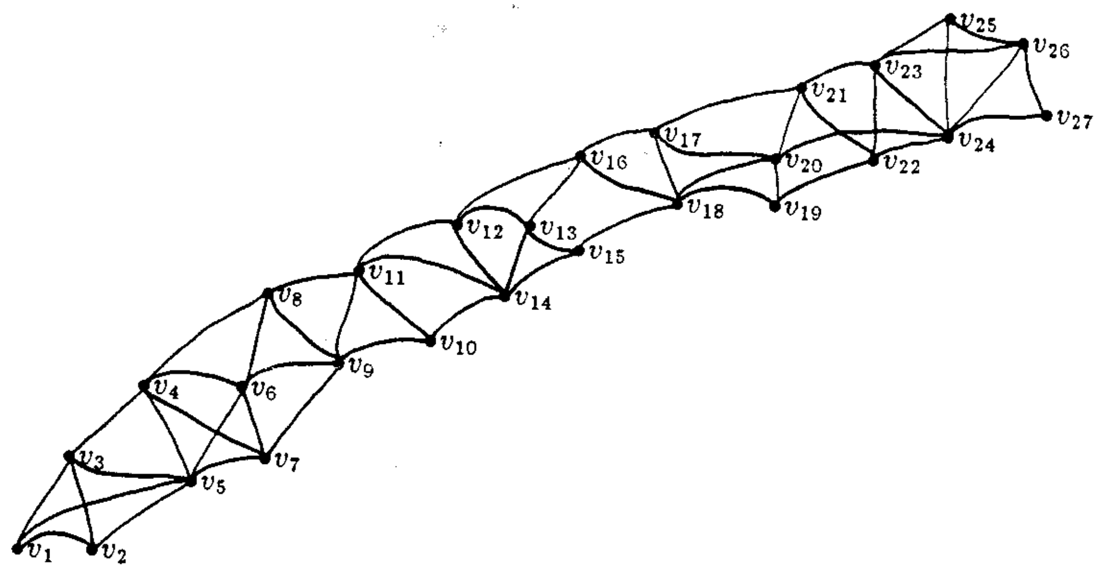


Figure 6.5: An example graph with bandwidth 4. Note that the gross structure is lineal, which could be extended indefinitively without increasing the bandwidth. The layout for the graph is given by the subscripts to the node labels.

Definition The *bandwidth* of a graph measures the greatest distance that any two adjacent vertices in a graph must be separated when the nodes are strung out in a straight line. Let G be a graph with nodes $V(G)$ and edges $E(G)$. Let a one-to-one function $\ell : V \rightarrow \{1, 2, \ldots, |V(G)|\}$ be called a *layout* of G. Then G has bandwidth b if there exists some layout, ℓ, such that for all $(x, y) \in E, |\ell(x) - \ell(y)| \leq b$. An example graph and its layout are given in Figure 6.5.

Definition The *tree-width*[2] of a graph is defined by [RS86] in the following way: Let G be a graph. A *tree-decomposition* of G is a family $\{X_i : i \in I\}$ of subsets of $V(G)$, together with a tree T with $V(T) = I$, which have the following properties:

- $\bigcup\{X_i : i \in I\} = V(G)$
- Every edge of G has both its ends in X_i for some $i \in I$.
- For $i, j, k \in I$, if j lies on the path in T from i to k then $X_i \cap X_k \subseteq X_j$.

The *width* of a tree-decomposition is $\max\{|X_i| - 1 : i \in I\}$. The *tree-width* of G is the minimum width over all possible tree-decompositions.

As examples of this concept, trees and forests have tree-width ≤ 1, and series-parallel graphs have tree-width ≤ 2. For $n \geq 1$, the complete graph K_n has tree-width $n - 1$, and the $n \times n$ rectangular grid (as in Theorem 17) has tree-width n. The bandwidth of a graph is never smaller than its tree-width, but it is known that trees (tree-width 1) have unbounded bandwidth even when their fan-in is limited to 3 [GGJK78]. Figure 6.6 shows an example graph that has tree-width 4.

6.4 Tree-Width Constraints

The theorems above deal with constrained families of architectures and assert that the loading problem is intractable for those families. This section examines a different type of constraint and reports polynomial-time algorithms for them, which we

[2] The *armwidth* of a graph is a generalization of bandwidth which we developed and defined in terms of a pebbling game or a vertex-elimination procedure. During preparation of this document we discovered that the notion has been independently developed by others [ACP87,AP88,WHL85,CK87]. The treatment given by Robertson and Seymour [RS86] is more appealing than our definition for the purposes of the proof below so we use their notation and name for it. Our definition can be found in a technical report [Jud88a] proving its equivalence to "embeddings in partial k-trees" and "tree-width".

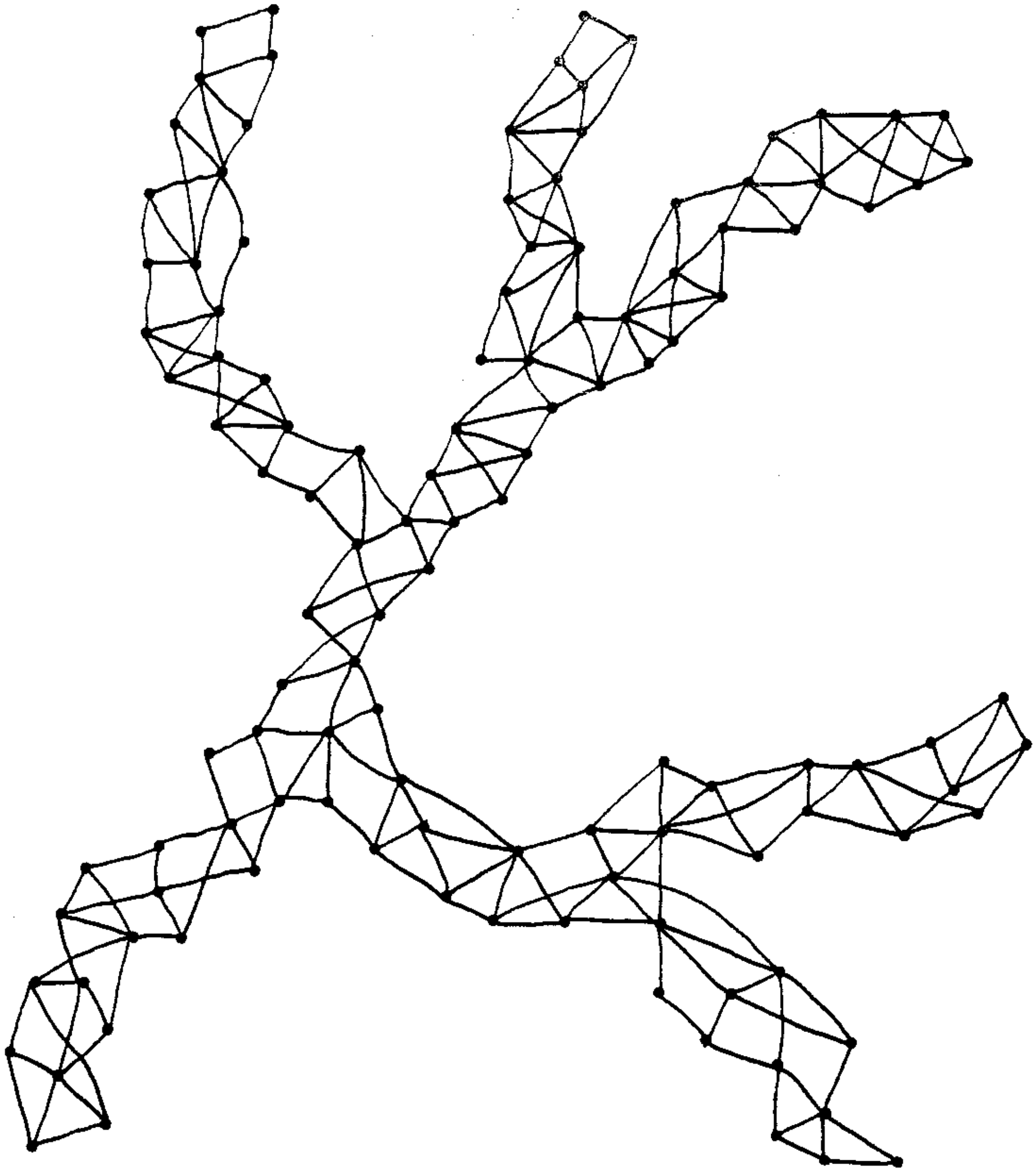


Figure 6.6: An example graph with tree-width 4. Its bandwidth is ≈ 16. Note that the gross structure is a tree, but each arm in this tree is not a simple path graph as a true tree would have, but is a 'fatter' structure. Each of these fat arms, taken independently, is a graph with bandwidth 4.

interpret as tractable. We begin with an example family of networks called columnar lines. These architectures are described graphically in Figure 6.7a. They are of some fixed depth (4 in the example shown) and of unbounded width, so they qualify as a shallow family. Their fish-net pattern of connectivity gives rise to the family of SCI graphs depicted in Figure 6.7b. Regardless of the width of the architecture, its SCI graph has a bandwidth (and tree-width) of 3 (one less than the depth of the net).

Observation: Columnar line architectures can be loaded in polynomial time.

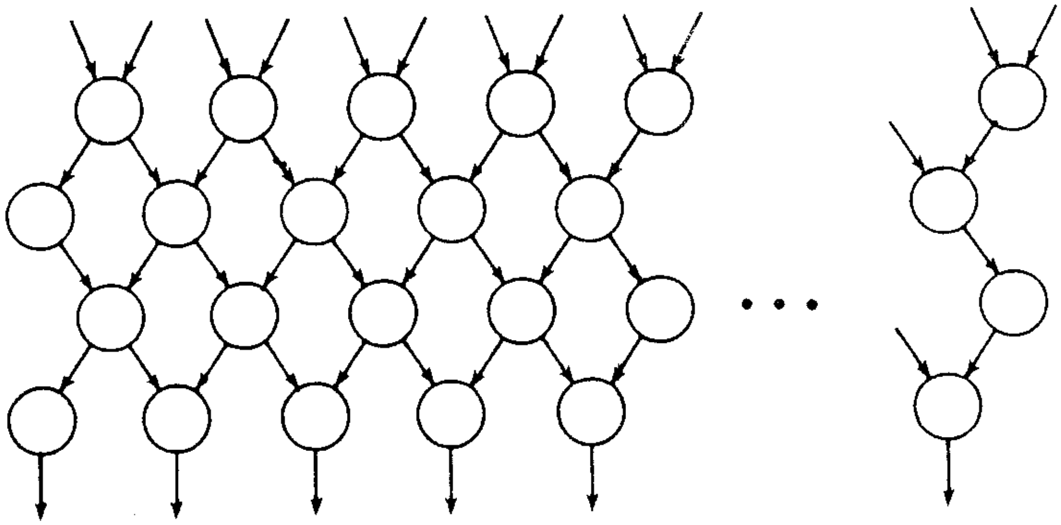
Proof sketch: Create a graph with a collection $h_1, h_2, h_3, \ldots$ of sets of nodes, where a node h_k^i stands for the i^{th} correct partial configuration for the support cone of the k^{th} output node. Then add edges (h_k^i, h_{k+1}^j) whenever $h_k^i \cong h_{k+1}^j$. A solution to the loading problem corresponds to a connected path from some member of h_1 to some member of h_2 to some member of h_3 and so on to the end. Finding such a path requires only polynomial time. □

The next theorem generalizes the previous observation.

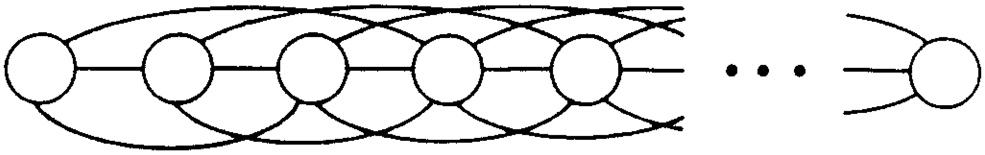
Theorem 18 *Loading shallow architectures whose SCI graphs are of limited tree-width can be accomplished in polynomial time, provided that a tree-decomposition is given that exhibits the required width.*

Proof: Let T and $\{X_i : i \in V(T)\}$ be the tree-decomposition of the SCI graph. Let $\tau, \tau_1, \tau_2, \ldots$ stand for subtrees of T. Let region$(\tau) = \bigcup\{X_i : i \in V(\tau)\}$. Let the set of all correct partial configurations for the group of architectural output nodes corresponding to a group, X, of nodes in the SCI graph be denoted scpc$[X]$. Any member of scpc[region(T)] is a solution configuration. Let the root node of a subtree τ be denoted *root*(τ).

The following recursive dynamic programming subroutine has access to an ar-



(a) The family of columnar line architectures (of depth 4) shown in classic side view. At right is a single 'column' and on the left is a sample of 5 columns joined together. An architecture in this family is composed of any number of such columns joined in the manner shown.



(b) The SCI graph for the columnar line architecture of (a) above. Each node corresponds to an output node of the architecture. Arcs occur wherever their associated support cones overlap. Regardless of the length of this graph, it has bandwidth 3.

Figure 6.7: Columnar line architectures and their SCI graphs.

chitecture, its SCI graph, and the tree-decomposition for the graph, and it takes some subtree of T as an argument:

```
SOLVE($\tau$):
    for every immediate subtree $\tau_j$ of $\tau$
        calculate $S_j \leftarrow$ SOLVE($\tau_j$)
    calculate $\hat{S} \leftarrow \text{scpc}[X_{root(\tau)}]$
    calculate $S \leftarrow \{\hat{F} : \hat{F} \in \hat{S}, \forall j \exists F_j \in S_j\ \hat{F} \cong F_j\}$
    return $S$
```

We claim for any given subtree, τ, that every member of the returned set $S \leftarrow \text{SOLVE}(\tau)$ has an extension that is correct for all of τ; and that all correct configurations for τ must be extensions of some member of S.

Claim: $\exists \hat{F} \in \text{SOLVE}(\tau) \iff \exists F \in \text{scpc}[\text{region}(\tau)]\ \ \hat{F} = F|_{X_{root(\tau)}}$.

Proof, by induction on the height of τ. For the basis case where τ is a single leaf node, ℓ, SOLVE returns $S = \hat{S} = \text{scpc}[X_\ell]$ so the claim is true. For the inductive step, assume the claim true for any subtrees $\tau_1, \tau_2, \tau_3, \ldots$ and consider a deeper subtree, τ^+, consisting of a root node, h, and subtrees $\tau_1, \tau_2, \tau_3, \ldots$ immediately below it. Say $\exists \hat{F} \in \text{SOLVE}(\tau^+)$. Then $\hat{F} \in \text{scpc}[h]$, and $\forall j \exists H_j \in \text{SOLVE}(\tau_j)$ such that $\hat{F} \cong H_j$, by the calculation of S. So $\exists F_j \in \text{scpc}[\text{region}(\tau_j)]$ and $H_j = F_j|_{X_{root(\tau_j)}}$ by the inductive assumption.

Now by definition of the tree-decomposition, $\text{DOM}(H_j) \supseteq \text{DOM}(F_j) \cap \text{DOM}(\hat{F})$. So $\exists F_j^+ = F_j \cup \hat{F} \in \text{scpc}[\text{region}(\tau_j) \cup X_h]$. It remains to show that all the F_j^+ are mutually compatible; this must be so because a path from one subtree to any other must pass through the root h. Hence $\text{DOM}(F_i^+) \cap \text{DOM}(F_j^+) \subseteq \text{DOM}(\hat{f})$ for any i, j, and $\exists F^+ = \hat{F} \cup \bigcup_j \{F_j\} \in \text{scpc}[\text{region}(\tau^+)]$. This proves the $\Rightarrow$ direction.

Conversely, if $\exists F^+ \in \text{scpc}[\text{region}(\tau^+)]$ then the (exhaustive) algorithm must find $\hat{F} = F^+|_{X_h}$. This completes the $\Leftarrow$ direction and proves the claim.

To determine if a solution configuration exsits for the whole network, run this algorithm:

```
Pick any node in $T$ to be the root
Calculate $\mathcal{S} \leftarrow \text{SOLVE}(T)$
If $\mathcal{S} = \emptyset$ then reject else accept
```

Any member of $\mathcal{S}$ indicates the presence of a solution configuration so this algorithm accepts if and only if the task is performable.

Finally, we must show that $g(n)$, the running time of this algorithm, is polynomial. Consider first the running time, $g_1(n)$, of the non-recursive parts of the subroutine SOLVE. This is $O(|\text{scpc}[X]|)$, which is exponential in the size of X, but since the size of X is limited, execution time is also limited. So $g_1 = O(1)$. There are a polynomial number of nodes in T, bounded by (n Choose k) $= O(n^k)$, if not by something linear. The algorithm invokes SOLVE once per node in T, so total time is $g(n) = O(n^k) \times O(1)$. □

Note that this theorem holds even if we loosen the definition of shallow architectures so that the largest sccs size is polynomial in n (as opposed to being a constant). In such a case, $g_1(n)$ and $g(n) = O(n \times g_1(n))$ are still polynomial.

Theorem 18 refers to 'limited' tree-width and was worded to imply "limited by a constant", but this is over-strong. Consider a family of architectures characterized only by a growth function $G(n)$ for the tree-width of its SCI graph. The theorem is worded for the case $G(n) = O(1)$, but it holds true for the case $G(n) = O(\log n)$ because g_1 is only exponential in $\log n$, which means that it is polynomial. So g is still polynomial as well.

Now remember that when $G(n) = O(n)$ the loading problem is *NP*-complete (since this is a non-constraint—the tree-width of *any* graph of n nodes is at most n). These bounds leave a gap between $O(\log n)$ and $O(n)$ which can be narrowed somewhat:

Theorem 19 *For shallow architecture families with a growth function for the tree-width of their SCI graph $G(n) = n^{\Omega(1)} = n^{\epsilon}$, loading is NP-complete.*

Proof: Take an arbitrary instance of 3SAT and perform the reduction as in Theorem 15. Consider the graph defined on the 3SAT instance which has a node for every variable, a node for every clause, and edges connecting variable nodes to all the clause nodes they participate in. If this graph is of size n and tree-width w (and $w \leq n$ always), then the constructed instance of loading will have size $O(n)$ and tree-width w. Now pad the construction with enough isolated nodes to bring it up to size $n' = G^{-1}(n)$. This will not change the tree-width of the loading instance but it will ensure that $w \leq G(n')$, thus satisfying the criterion for membership in the family. Since G is polynomial, G^{-1} is also polynomial. No matter how small ϵ is, as long as it is greater than 0 there is a polynomial-sized reduction from SAT to loading. □

This narrowed window of bounds hangs on the tree-width constraint alone and is therefore common to many combinatorial search problems, not just the loading problem.

Theorem 18 stipulates that the tree-decomposition of the SCI graph must be given as input to the problem because in general determining minimum tree-width is an *NP*-complete problem in itself [ACP87]. This is probably not a problem for connectionists because the network design methodologies we hope to find would presumably be amenable to easy a priori structural analysis. (Assume the network does not change its connectivity during use.) However, if this theorem were to be exploited in a direct implementation it does imply that the nodal learning rules would have to be aware of the structure of the SCI graph, i.e. knowledge of the tree-decomposition would have to be 'wired in' to the network somehow.

6.5 Additional Comments

The algorithms given above are for purposes of demonstrating the polynoı ıial-time complexity of the problems. They are not intended to have any neural plausibility. The running time constants could be markedly improved in the algorithm given, but note that the running time is *linear* in the size of the architecture and in task size. This problem can therefore be added to the list [MS81,CES81] of *NP*-complete problems that become easier with diminishing bandwidth. That characterization may now obsolete, though, because it seems all of those results can be re-cast in terms of the weaker notion of tree-width.

By limiting the size of the sccs in all theorems above, we have finessed the whole issue of how the loading problem gets more difficult with depth. This trick has allowed us to focus on the issues arising from expansion of the width of an architecture. But putting individual limits on the sccs size and on the tree-width is unnecessarily strong. The real constraint required by the proof of Theorem 18 is only that the scpc for any tree-decomposition set, X_i, be calculable in a polynomial amount of time.

We have ignored the possibility of an efficient search for correct partial configurations and have chosen here to enumerate all possibilities. We have dismissed this particular inquiry as being a "depth issue".

We have studied fixed-depth architectures partly because they are easy enough to analyze, but they are of interest because of a possible correspondence with cortical structures. Certain parts of the brain (e.g. visual cortex [HW79]) are quite shallow compared to their great width, and the direction of information flow is predominantly unidirectional along the shallow axis. Connections are more or less localized in 3D space surrounding a neuron. Of course real cortical structures are complicated by many connections and other specifics not modelled here, but we feel that the process of developing a theory of how such structures work could benefit by analyzing a few judicious constraints at a time. The constraints chosen here are an approximation to what seem to be the major computatior al aspects of some cortical structures.

Chapter 7

MEMORIZATION AND GENERALIZATION

One would hesitate to use neural networks just to memorize and store data because it is probably not economical at all—there are many other engineering techniques that are strong competitors for that honour. But one common motivation for studying neural networks is that they can generalize, and thus perform something of great value beyond mere storage.

The issue of generalization does not seem to be the primary concern of this thesis. However, we state in this chapter that before the issue of generalization can be addressed, the memorization problem must first be solved; hence our results about memorization have a direct bearing on the other issue.

Following, we give several statements of the same idea; the reader who accepts any one of the arguments might skip the others.

Statement 1 We have shown that a network cannot always remember all the items that is has seen. One should therefore not expect it to always be able to extend its knowledge to things it has not seen.

Statement 2 When specifying what is meant by 'generalization', one could require that the chosen function agree in all places with the given data, or one might allow some degree of deviation from the given data. In the case where the allowed generalizations must all be consistent with the given task, our results are directly applicable, showing that consistency is, in general, too hard to reliably achieve. The business of finding regularities in data and generalizing from them depends totally on the embedded problem of simply remembering data.

Statement 3 In the case where the application could tolerate 'generalizations' that need not be completely consistent with the given data, our results are sometimes less directly relevant. But Corollary 13 is strong enough to apply to some such situations: Even if you allow a loading system to alter the responses on anything less than 1/3 of the items (allowing the system to select which items and what to change them to), it is still *NP*-complete to achieve consistency with the rest.

Statement 4 When one is given a small sampling of items and asked to find a configuration that is consistent with those items, there are typically a vast number of candidate configurations. The notion of "good" generalization corresponds to making an "appropriate" selection from amongst this field of options. The definition of "appropriate" is of course going begging here. But our *NP*-completeness theorems indicate that it is too difficult to identify even a *single* configuration from this field of candidates. Hence the definition of "appropriate" is of little concern. Regardless of how one might prefer to define generalization, consistency is the nub of the problem.

Statement 5 A system that learns and generalizes from what it learns is often treated in a two-phase experimental paradigm. The first phase is called the training phase, and in it some subset of items is selected (by the experimenter) from a task and presented to the system. The second phase is called the testing phase. In it some subset of items (presumably disjoint from the training set) is selected from the task and the system is asked to induce what the responses should be. Of course the performance of the system will be sensitive to how representative the training set is of the overall task and how complete it is. Also, it will be sensitive to how representative the testing set is of the overall task. Amongst the community using this paradigm there is a widely-held meta-theorem which says that the better a system does on the training set, the better it will do on the test set. And this observation would have us concentrate on solving the memorization problem; poor performance in memorization bodes for poor performance in generalization.

Statement 6 The representativeness of the training set and the representativeness of the testing set are very subjective quantities. Hence the two-phase experimental paradigm can give erratic and non-rigorous results. Valiant's definition of learnability has an ingenious mechanism for handling all of these quantities in a standard mathematical way that utilizes a probabilistic criterion of success in learning and generalization. See Figure 3.2, page 26. Rather than arbitrarily choosing a training set in advance (which is open to many vagaries and biases), he selects a training set by randomly choosing items according to some unknown a priori distribution over them. Hence the make-up of the training set is objective, albeit probabilistic, and it is biased only by the distribution. He also selects a testing set in the same way, for the same reason. And since the same distribution is used in both cases, the training set is an unbiased sampling of the testing set (and vice versa). Furthermore, the size of the training set is not determined by the experimenter either—it becomes a decision of the algorithm how many items to sample. The testing set is in effect the whole task, but each item is weighted by its relative probability.

The definition then requires that the system usually be correct for most items in the test set. 'Usúally' is defined by a confidence probability parameter, δ, and 'most' is defined by an accuracy probability parameter, ϵ. The criterion of success

$\mathcal{A}$ is a design class of architectures.
A is an architecture.
p is a polynomial.
B is an algorithm.
ϵ, δ are probabilities.
F, G are configurations for A.
$\mathcal{M}^A_F$ is the behaviour of A when configured with F.
$T \subseteq \{(\sigma, \rho) | \mathcal{M}^A_F(\sigma) = \rho\}$ is a task.
D is a probability distribution over task items.

$$\text{“}\mathcal{A}\text{ can generalize”} \iff \left\{ \begin{array}{l} (\exists p, B) \text{ such that } (\forall A \in \mathcal{A})(\forall F \text{ for } A) \\ (\forall T \subseteq \mathcal{M}^A_F)(\forall D \text{ over } T)(\forall \epsilon, \delta > 0) \\ B \text{ halts in time } p(|A|, |T|, 1/\epsilon, 1/\delta) \text{ with} \\ \text{output } G \text{ that with probability } \geq 1 - \delta \\ \text{has property } \sum_{\{(\sigma,\rho) \in T : \mathcal{M}^A_G(\sigma) \neq \rho\}} D(\sigma) < \epsilon \end{array} \right.$$

Figure 7.1: A definition of generalization in networks

requires that the learning algorithm terminate in time that is polynomial in $1/\delta$ and $1/\epsilon$ for any given $\delta, \epsilon > 0$. Obviously if the algorithm terminates in polynomial time, then it can afford to sample only a polynomial number of items, but the system is granted a bigger budget of time whenever more confidence or more accuracy is desired.

This definition has been examined by Blumer et al [BEHW87] and they prove that the probability that all consistent hypotheses have error at most ϵ is larger than $1 - (1 - \epsilon)^m r$, where m is the number of samples and r is the number of hypotheses in the space of all hypotheses. This again is an endorsement of consistency—if you can be true to the training set, you will be true to the testing set.

Statement 7 We have utilized Valiant's ϵ, δ ploy in our definition of generalization for networks as given in Figure 7.1. Using this definition, we ask if the class, $\mathcal{A}$, of all networks can 'generalize':

Corollary 20 *Networks cannot generalize.*

Proof: Use the same construction as in Theorem 1. Set $\epsilon < 1/3$, and let D be uniform over the three items. Then with probability $\geq 1 - \delta$ the algorithm, B, must find a configuration that is consistent with all three of the items. This implies that B will be a probabilistic polynomial-time algorithm for 3SAT, which implies 3SAT $\in RP$.[1] Assuming $RP \neq NP$,[2] this is impossible. □

Statement 8 We are not claiming that useful generalization can never be performed by connectionist networks. But what we do claim is that the consistency problem is a *prior* consideration. If simple consistency cannot be achieved when required (at least for the target family of tasks), then it is premature to worry about making predictions for unseen stimuli.

[1] The complexity class RP is defined as the set of decision problems that have algorithms that run in time polynomial in n and $1/\delta$ and which will always return 0 if the answer is NO, and if the answer is YES will return 1 with probability $\geq 1 - \delta$.

[2] Assuming $RP \neq NP$ is good for your theorebellum.

Chapter 8

CONCLUSIONS

8.1 Lessons Drawn from Current Results

Loading is hard: The job of simply remembering associated pairs of strings requires only linear time in a von Neumann machine, but we have shown that a large-scale version of this trivial problem can become very difficult if it must be achieved in a given non-recurrent network. Hence there is reason for connectionist research to find out why this phenomenon occurs and how to avoid it. The scale-up problem will not be solved without a deeper understanding of the issues involved in learning, and without a narrower definition of what kinds of learning we want to achieve.

Neural networks have been touted as having more natural and more powerful learning abilities than traditional AI learning systems. Certainly, there is some appeal and basis for the argument. It is more comfortable to believe that a small adjustment to a few weights in a net will (a) create a new behaviour that is substantially like the old behaviour, and (b) quite possibly improve the behaviour. In contrast, a small adjustment to a few bits in the program of a Turing Machine will (a) usually produce,radically different behaviour, and (b) often produce a totally useless behaviour. Whether this argument is fair or not fair, this thesis has demonstrated that before we can harness this quality of gentle adaptations, we still need to know a lot more about the network model, how to design it, how to program it, and what applications to put it to.

Issues of Node Function Sets: A significant set of side questions arose during our research regarding the justification and appropriateness of the type of node functions typically used in the connectionist literature[1]:

- Is there any support for the choice of node function sets that use linear summing techniques? i.e. Why use LSFns? LLFns?
- Can learning theory speak to the issue?
- Are some node function sets easier to learn with than others?

We have good evidence that the difficulty of the loading problem is independent of the choice of type of functions that each node can perform. For all reasonable sets, our results are completely independent of the choice of node function set; hence we conclude that nothing in our work either supports or detracts from the use of LSFns or LLFns.

As mentioned near the end of Chapter 4, Blum and Rivest [BR88] have found a different proof of the *NP*-completeness of $Perf_{LSFns}$ and their argument depends directly on having to linearly separate many points in *s*-space. If the three nodes in their construction were using AOFns, LUFns, some other functions instead of LSFns, the proof would not hold. Hence the only evidence from learning complexity uncovered so far speaks somewhat *against* linear summing! However this is quite a weak argument as it stands—there is no need to seriously question linear sums yet.

Basically, what we can conclude from our results being independent of the node function set is that the complexity of loading does not derive from the node function set.

[1] We thank I. Aleksander [Ale84] for resolutely avoiding the dominant viewpoint on what constitutes a good node function set, thus prompting our questions on the topic.

What it does derive from is the connectivity patterns of the network. This much is clear. Notwithstanding this, there is a great deal of research effort being put into understanding linear threshold functions and also into studies of linear threshold networks. For instance Minsky and Papert [MP72] treat linear threshold devices as a primary issue. This is reasonable strategy for investigating the power of small networks; indeed in the case of tiny networks (i.e. one node or one layer), the role of the node function set is ascendant because it has an overwhelming effect on what can be performed by the net. In large or deep networks, the role of the node function set in determining computational power fades quickly and is replaced by issues like the size, depth, and connectivity of the net. We suggest, therefore, that studies of linear threshold devices, if not totally irrelevant to learnability, are at least guilty of placing undue importance on an issue that will only help settle minor issues. See also [MP72, footnote page 165].

Generalization: Although generalization properties are exciting possibilities for neural networks, we have argued in several ways that the simple issue of consistency is a central and prior consideration. Good generalization requires good memorization.

Design Constraints: We have shown that loading can be hard, that it can be easy, and that one of the things this depends on is the family of architectures being loaded. The theorems serve as warnings and as guideposts to better designs. When the SCI of a shallow architecture has limited tree-width, loading is tractable, but this constraint may not yield useful families of networks. Less constrained families that we looked at (e.g. grid SCI graphs) have *NP*-complete loading problems. These results are some evidence that architectural constraints alone will not serve as a useful exit from *NP*-completeness. Other aspects of the problem will need to be changed, possibly in conjunction with architectural constraints.

Methodology: We have outlined a wide range of questions regarding narrowed or altered models of the connectionist learning goal. The particular subcases considered here are merely a few of the myriad avenues open for research. The tool of NP-completeness can direct the search for good learning rules and/or easily-loaded architectures and/or easily-loaded tasks without requiring extensive simulations. By carefully refining definitions and searching for a more complete description of the boundary between solvable and infeasible problems a more useful theory will develop that will have applications to the design of many kinds of network machines.

8.2 Contributions of this Thesis

We have focussed on the scale-up problem in supervised learning as an area requiring major effort and applied standard tools of complexity theory to try to understand it.

The first major contribution made in this research program is to have identified and formalized the basic computational problem underlying the connectionist learning problem. There are four little parts that went into its construction. (1) The 5-step cycle of classical connectionist learning (see Section 2.5) which took a stimulus and response and produced a weight change, was condensed into taking a task into a configuration of weights. (2) The notion of a node function set was generalized so that we stopped referring to a configuration of weights and simply referred to a configuration. (3) The distributed nature of the classical algorithms was removed and supplanted by serial computation. (4) The architecture was made into an explicit input. Altogether, this gave us the form of the loading problem as a function from (architecture, task) pairs to configurations.

The computational question has been demonstrated here to be of broad general value in finding design constraints for neural networks. There is a very large class of related questions that follow the basic formulation but particularize it by stating restrictive definitions on the various aspects of the problem. Thus we feel that the development of the formulation itself represents a major component of this thesis.

We have recognized the relationship between our model of loading and other models of learning given by Valiant and Gold.

In their book *Perceptrons* [MP72], Minsky and Papert lament the lack of an effective procedure for loading networks and express a hope that "some profound reason for the failure to produce an interesting learning theorem for the multilayered machine will be found." Our results supply such a reason, and the proofs of our theorems stand as opening insights into the reasons why the loading problem is so difficult. The fact that network learning is NP-complete may not be surprising in itself, but its proof is still a valuable contribution on its own.

In penetrating the issues surrounding expansion of network width, we have developed the notions of shallowness and SCI graphs and demonstrated their usefulness by identifying some polynomial time problems and some closely related problems that are NP-complete.

We have also raised the question of how we might justify linear sum functions in networks or find another node function set that might be more appropriate for learning.

In attempting to answer that question, we have found good evidence that the difficulty of the loading problem does not derive from features of the node function set. In fact our theorems find no evidence in support of any node function set over any other and we argue that good research strategy ignores the particularities of any one node function set and concentrates instead on higher-level issues.

We have illustrated why the development of a theory of learning in networks would directly contribute to the otherwise black art of network design. We have found numerous avenues to follow in order to study the effect of scale-up on the learning issue, and to thereby derive principles that contribute to a methodology of network design.

Lastly, we have developed the notion of armwidth (aka tree-width and partial k-trees) to characterize an important constraint on graphs that yields polynomial subcases for otherwise NP-complete problems. This idea is tangential to the present document but deserves wide exposure to graph theorists and algorithm designers. Indeed its importance is underscored by the fact that other researchers have independently discovered the same notion and papers on the topic are now appearing in the literature.

8.3 Future Work

The obvious extensions to this work include refining the classes of architectures considered and the classes of tasks considered, so as to more closely understand the relationship between networks and what they can learn. Some specific directions are outlined in the following subsections.

8.3.1 Task Constraints

Although this study has focussed on what it expressed as *architectural* design issues, these results could just as easily have been expressed as *task* design issues, and in fact they are both. Whenever we found poly-time loadable architectures we were also implicitly identifying poly-time loadable tasks, since the class of tasks that such architectures were capable of loading was given as the set of all tasks performable by that architecture. Hence the investigation has had dual purpose throughout. However, it is a limitation of this work that we have generally inquired only about the ability of a network to load *all* of its performable tasks instead of asking about its ability to load some useful subset of its performable tasks. This should be explored further.

To pursue such questions, one needs to identify interesting classes of tasks and find useful formal definitions for them. In Section 5.2 we used a teacher/learner formalism to constrain the class of tasks a network might be asked to load. This technique was used only in conjunction with differing node function sets but it is also useful in other contexts. For example, it might be useful to be able to describe exactly what tasks an architecture is capable of learning by referring to a teacher network whose tasks it can easily load. As before, we denote this by writing the parameters for describing the teacher to the left of *Perf* and those for the learner to the right. For example the question as to whether network A' could learn all of what network A could perform would be ${}^{A}Perf^{A'}$. We ask if there is a reasonable ϕ function from architectures to architectures such that for all A, ${}^{A}Perf^{\phi(A)}$ is tractable.

This question can be answered positively. When given a network, A, and any task of t items, one can construct a network approximately t times as big as A that can easily load that task. Although this is too loose a ϕ function to be termed a 'result', it does gives us an upper bound on the size of learner network required. Because the factor is t rather that some power of t or some exponential in t, we believe that tighter answers to the ϕ question might indeed be interesting.

The purpose of the teacher/learner formalism is to unbundle the architecture class from the task class and to deal with them explicitly and independently.

8.3.2 Relaxed Criteria

The basic loading problem asked for a *guarantee* that the algorithm would complete its job of finding a configuration. It might be that some probabilistic criterion of success would be easier to comply with. Perhaps for some class of architectures we will be able to find a randomized procedure that will run in polynomial time and report a solution configuration with a certain minimum probability. Repeated invocations of the procedure would give asymptotic certainty regarding performability.

Such an algorithm could be used in applications where it was possible to judge how much loading time each situation warranted.

8.3.3 Mutating the Network

Another avenue of freedom usually not exploited by connectionist learning schemes is to alter the architecture as learning proceeds. When carried to extremes, this would amount to an exercise in circuit design, for which Valiant's formulation of the learning problem is the most relevant. This is a far cry from current approaches to connectionist learning, but adhering rigidly to the starting architecture may be too constrictive; somewhere between these two extremes we may find a scheme that combines the best of both approaches.

8.3.4 Returning to Classical Form

As discussed in Section 2.6, the loading problem is on the easy side of three issues, and therefore whenever a tractable loading problem is identified we do not have complete evidence that the problem will be easy in the classical connectionist setting. For such a case we would still have three aspects to adapt:

- The type of machine used: The serial algorithm would have to be broken up and distributed throughout the network.
- The style of processing required: The process would have to be re-implemented in a 'neural' style.
- The type of information available: The system would have to be altered to accept information in an on-line fashion.

All of these transformations would require a special research effort since none of them are well understood.

8.3.5 Recurrent Networks

We have specifically focussed on feed-forward networks. Recurrent networks have a fundamentally different retrieval process in that they start at some point in state space and under the influence of the input they travel through state space, possibly reaching a stable point or a limit cycle. The definition of what constitutes its 'output' may therefore be problematical, but this sort of machine is very interesting and the problem of loading them should be studied.

We have suggested that our shallow feed-forward models might be relevant to (long-term) storage of information in the brain. Hypotheses about short-term memory in the brain are often based on cyclic electrical mechanisms which require recurrent networks.

8.3.6 Other Learning Paradigms

We have limited our inquiries to the supervised learning paradigm. Many other types of protocols (e.g. unsupervised learning, or the use of queries) are useful models of learning environments but have not been formally explored in the context of learning in networks.

8.4 Philosophical Summary

A theory is developed by progressing from one hard, clear definition of a problem to another. Clearly, at this point in time, it is still ill-defined what connectionists require of a learning system. There are many formulations of it other than ours that might be appropriate for different situations. It may be reasonable just to ask for the 'best' configuration for a network, rather than the 'correct' configuration. It may be reasonable just to ask for the configuration that yields performance better than a simple regression procedure would. It may be reasonable just to ask for a configuration that makes maximal use of the hardware (i.e. supports the greatest number of items in the given network). The contribution of such research may be to help define connectionist learning by showing which formulations are achievable. We have formulated a basic question. Other formulations based on more refined definitions could lead to successively more useful models of practical connectionist concerns. Because no exact definitions of connectionist learning are yet widely accepted, we think that an analysis of various definitions leading to tractable loading problems would help establish and focus the research in this area.

The successful development of a theory of an intensely complicated system like the brain depends on a judicious sequence of selections of constraints. To begin, one must select one or two appropriate constraints; then study them to understand how they interact; choose another constraint; then add it to the others and elaborate further. At each choice point, one must be carefully conscious of what level of detail the system is being modelled at and choose constraints that act at that same level. We think there has been too much emphasis placed on modelling brains at the level of neurons using constraints like spike train frequencies, linear threshold functions or the sodium pump. This is like trying to discover the principles of flight by studying the microbiology of birds. The useful level of study is much coarser than that. Similarly here, just by taking a view from the next larger scale of detail, our investigations have discovered a universe of issues that are almost oblivious to the functionality of individual nodes. We suggest therefore that these coarser levels of detail would be more productive levels of modelling for computer scientists to pursue. Our hunch is that after a theory of learnability is fleshed out, tuning it for a specific node function set will change things only slightly. In general the coarser levels are the more important levels.

What we have attempted here is to look at the level of mid-size neuroanatomical structures (e.g. cortical slabs), and we hope that our choice of simple constraints will prove propitious. We have pursued the study of feed-forward networks and especially the architectural family of shallow networks because of their potential for modelling structures in natural brain cortex. Our model will be relevant if we have been lucky in choosing constraints and if the neural structures they model happen also to be engaged in the kind of information loading and retrieval that we are exploring. We might have the wrong model of the salient aspects of these slabs of cortical columns; we might have the wrong model of how these slabs actually retrieve their stored information; or we might just be asking the wrong analytical question. (The performability question used here requires total, exact, dependable recognition of the set of performable tasks. This seems unduly demanding and of the three suspicions listed here, the last one seems to deserve the first examination.)

Whatever the case, our underlying assumption is that complexity analysis (and specifically the P vs NP distinction) provides a means to narrow down the things that biological machines do and how they do it. Our strategy is to take the general NP-complete problem and add architectural constraints, task constraints, or other types of constraints, and search for polynomial-time loading problems. We feel very safe in assuming that the brain cannot be solving any NP-hard problem, and we feel secure in assuming further that evolution would have found efficient ways to utilize the available hardware. Ergo brain mechanisms are likely to be described by decision problems found 'just below' the level of NP-completeness. Hence the general outline and thrust of our research program.

By providing guidelines for ensuring that a network can learn efficiently, we hope to contribute to an urgently needed general methodology of how connectionist networks should be constructed. And by distinguishing between those forms of learning that are achievable and those forms that are not, we will be helping to identify the applications to which neural networks can be profitably applied. This thesis has provided only the first steps toward such a theory.

Appendix A

ALTERNATE PROOF OF GENERAL THEOREM

This appendix proves a slightly weaker version of Theorem 1 in that it uses a node function set called SAFns, which is larger than AOFns. SAFns is the set of node functions that can be constructed with a single AND gate augmented with optional inverters at the inputs and output. The construction in the following proof is somewhat different from those used in earlier chapters. The next appendix extends this theorem to real-valued node function sets using the same construction used here.

First, we introduce some general purpose notation for manipulating strings. If α and β are strings then $\alpha \cdot \beta$ is the concatenation of α and β, and α^n is the concatenation of n copies of α. We use $\bigodot_{i=1}^{n} \alpha_i$ to denote $\alpha_1 \cdot \alpha_2 \cdot \alpha_3 \cdot \ldots \cdot \alpha_n$.

If α is a string, A and B are sets (with distinct elements), $B \subseteq A$, and the length of α is $|A|$, then the notation $\alpha[{}^{A}_{B}]$ denotes the string of length $|B|$ that is formed by associating successive elements of α with successive members of A (which has an implicit ordering), and then selecting from α only those elements that are associated with members of B. For example,

$$\begin{array}{rl}
\text{if} & \alpha = \ 2 \cdot 7 \cdot 4 \cdot 1 \cdot 9 \cdot 8 \\
\text{and} & A=\{d_{10}, d_{14}, d_{15}, d_{16}, d_{17}, d_{19}\} \\
\text{and} & B=\{d_{10}, d_{17}, d_{19}\} \\
\hline
\text{then} & \alpha[^{A}_{B}] = \ 2 \cdot 9 \cdot 8
\end{array}$$

Another notational device is used to select single elements from a string; $\alpha\langle k\rangle$ represents the k^{th} element of α. Formally, $\alpha\langle k\rangle = \alpha[^{\{1,2,3,\ldots,a\}}_{\{k\}}]$ where a is the length of α.

For precision, we define the semantics of computation in a network as the unique string that satisfies the inductive expression

$$Comp^{A}_{F}(\sigma) = \sigma \cdot \bigodot_{i=1}^{n} f_i(Comp^{A}_{F}(\sigma)[^{P}_{p(v_i)}])$$

Such a string is unique because A is acyclic and the output of each node is dependent only on the output of previous nodes. The network mapping can now be stated as

$$\mathcal{M}^{A}_{F}(\sigma) = Comp^{A}_{F}(\sigma)[^{P}_{R}]$$

Theorem 21 *$Perf_{SAFns}$ is NP-complete.*

Proof: We reduce the classic satisfiability problem (SAT) to $Perf_{SAFns}$. (See [GJ79] for an explanation of this process.) Let (U, Γ) be an arbitrary instance of SAT, where U is a set of variables and Γ is a set of clauses; $U = \{u_1, u_2, \ldots u_w\}$, $\Gamma = \{(\gamma_i, G_i) : 1 \leq i \leq m\}$. We use a novel representation of Γ, the set of clauses: for each $i \leq m$, $\gamma_i \in \{0,1\}^w$, and $G_i \subseteq U$. A string Π is said to *satisfy* the instance (U, Γ) iff $\Pi[^{U}_{G_i}] \neq \gamma_i[^{U}_{G_i}]$ for all $i \leq m$. (This representation of a clause can be obtained from the traditional disjunctive form by applying de Morgan's Law once and padding for variables that are not in the clause.)

We must construct an architecture A and a task T such that T is performable by A iff (U, Γ) is satisfiable. The set of nodes V will be composed of a set V_1 of "first-layer nodes" and a set V_2 of "second-layer nodes".

$$S = \{v_{0,i} : 0 \leq i \leq w\}$$

$$V_1 = \{v_{1,j} : 1 \leq j \leq w\}$$

$$V_2 = \{v_{2,i} : 1 \leq i \leq m\}$$

$$P = S \cup V_1 \cup V_2$$

$$R = V = V_1 \cup V_2$$

$$E = \{(v_{0,0}, v_{1,j}), (v_{0,j}, v_{1,j}) : 1 \leq j \leq w\} \cup \{(v_{1,j}, v_{2,i}) : u_j \in G_i\}$$

$$A = (P, V, S, R, E)$$

The task is composed of 3 kinds of items. The first kind is called the "truth-value items" and associates a binary value with 'true' and 'false':

$$T_1 = \{(0 \cdot 0^w, 0^w \cdot *^m), (0 \cdot 1^w, 1^w \cdot *^m)\}$$

The second kind of item is called the "disjunct semantics items":

$$T_2 = \{(0 \cdot \gamma_i, *^w \cdot *^{i-1} \cdot 0 \cdot *^{m-i}) : (\gamma_i, G_i) \in \Gamma\}$$

The third kind of item is called the "conjunct semantics item":

$$T_3 = \{(1 \cdot 0^w, *^w \cdot 1^m)\}$$

$$T = T_1 \cup T_2 \cup T_3$$

Figure A.1 gives a construction for an example instance of SAT.

Claim: A solution configuration, F, for (A, T) exists iff a satisfying assignment Π, exists for (U, Γ).

proof ($\exists F \Leftarrow \exists \Pi$): Assume $(U, \Gamma) \in$ SAT by virtue of the satisfying assignment

Take as an example the following SAT problem expressed in traditional CNF form: $(\overline{u_1} \vee u_2 \vee u_3)(u_2 \vee \overline{u_3} \vee \overline{u_4})$. In the required form, this is equivalent to

$$\begin{array}{ll} \gamma_1 = 1 \cdot 0 \cdot 0 \cdot 0 & G_1 = \{u_1, u_2, u_3\} \\ \gamma_2 = 0 \cdot 0 \cdot 1 \cdot 1 & G_2 = \{u_2, u_3, u_4\} \end{array}$$

The task for this problem is

$$\begin{array}{rll} T_1 = & (0 \cdot 0 \cdot 0 \cdot 0 \cdot 0, & 0 \cdot 0 \cdot 0 \cdot 0 \cdot * \cdot *) \\ & (0 \cdot 1 \cdot 1 \cdot 1 \cdot 1, & 1 \cdot 1 \cdot 1 \cdot 1 \cdot * \cdot *) \\ T_2 = & (0 \cdot 1 \cdot 0 \cdot 0 \cdot 0, & * \cdot * \cdot * \cdot * \cdot 0 \cdot *) \\ & (0 \cdot 0 \cdot 0 \cdot 1 \cdot 1, & * \cdot * \cdot * \cdot * \cdot * \cdot 0) \\ T_3 = & (1 \cdot 0 \cdot 0 \cdot 0 \cdot 0, & * \cdot * \cdot * \cdot * \cdot 1 \cdot 1) \end{array}$$

The architecture is as follows:

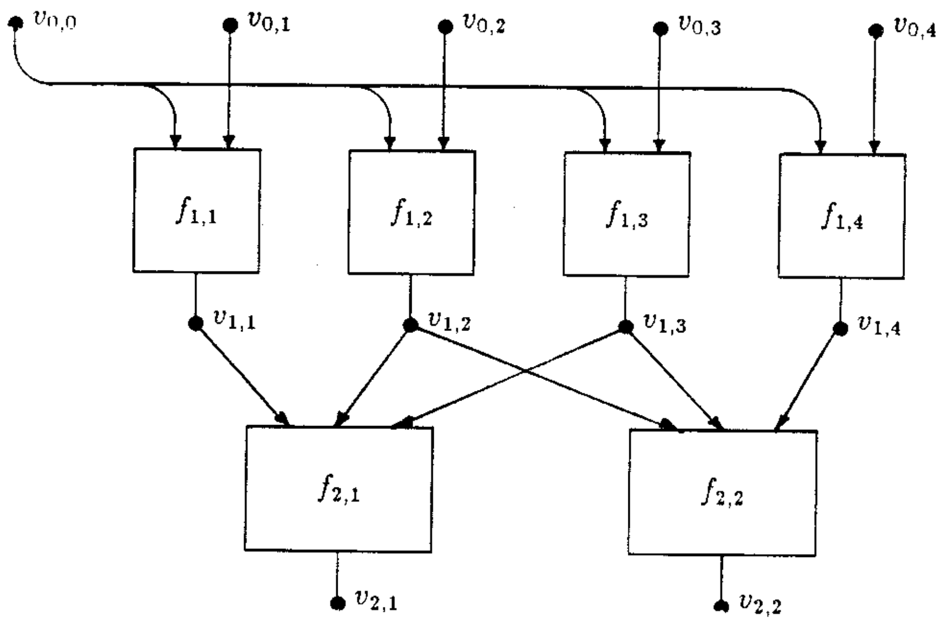


Figure A.1: Example construction for proof of theorem using SAFns.

string Π. Associate the node function $f_{k,i}$ with each node $v_{k,i} \in V$, and then let $F = \{f_{1,1}, f_{1,2}, \ldots, f_{1,w}, f_{2,1}, f_{2,2}, \ldots, f_{2,m}\}$ where

$$f_{1,j}(a \cdot b) = \begin{cases} b & \text{if } a = 0 \\ \Pi\langle j \rangle & \text{if } a = 1 \end{cases}$$

$$f_{2,i}(\alpha) = \begin{cases} 0 & \text{if } \alpha = \gamma_i[{}^{U}_{G_i}] \\ 1 & \text{otherwise} \end{cases}$$

We must show that $\mathcal{M}_F^A \supseteq T$, which we do by showing $\mathcal{M}_F^A \supseteq T_1$, $\mathcal{M}_F^A \supseteq T_2$, and $\mathcal{M}_F^A \supseteq T_3$ individually. First, note that since $f_{1,j}(0 \cdot b) = b$ for all $j \leq w$, we have for any α,

$$Comp_F^A(0 \cdot \alpha)[{}^{P}_{V_1}] = \bigodot_{j=1}^{w} f_{1,j}((0 \cdot \alpha)[{}^{S}_{p(v_{1,j})}]) = \bigodot_{j=1}^{w} f_{1,j}(0 \cdot \alpha\langle j \rangle) = \bigodot_{j=1}^{w} \alpha\langle j \rangle = \alpha \quad \text{(A.1)}$$

Equation A.1 proves $Comp_F^A(\sigma)[{}^{P}_{V_1}] = \rho[{}^{R}_{V_1}]$ for both items $(\sigma, \rho) \in T_1$. Since responses for V_2 are undefined, $\mathcal{M}_F^A \supseteq T_1$.

For each $v_{2,i} \in V_2$ there is only one item in T_2 which is defined, and to agree with that response, we must show that $\mathcal{M}_F^A(0 \cdot \gamma_i)[{}^{R}_{V_2}]\langle i \rangle = 0$.

$$\begin{aligned}
\mathcal{M}_F^A(0 \cdot \gamma_i)[{}^{R}_{V_2}]\langle i \rangle &= f_{2,i}(Comp_F^A(0 \cdot \gamma_i)[{}^{P}_{p(v_{2,i})}]) \\
&= f_{2,i}(Comp_F^A(0 \cdot \gamma_i)[{}^{P}_{V_1}][{}^{V_1}_{p(v_{2,i})}]) \qquad \text{since } p(v_{2,i}) \subseteq V_1 \\
&= f_{2,i}(\gamma_i[{}^{V_1}_{p(v_{2,i})}]) \qquad \text{by (A.1) above} \\
&= f_{2,i}(\gamma_i[{}^{U}_{G_i}]) \qquad \text{by definition of } E \\
&= 0 \qquad \text{by definition of } f_{2,i}\text{, as required.}
\end{aligned}$$

Since this argument holds for every node in V_2, and responses for V_1 are not defined, $\mathcal{M}_F^A \supseteq T_2$.

The only stimulus in T_3 is $1 \cdot 0^w$.

$$Comp_F^A(1 \cdot 0^w)[{}^{P}_{V_1}] = \bigodot_{j=1}^{w} f_{1,j}((1 \cdot 0^w)[{}^{S}_{p(v_{1,j})}]) = \bigodot_{j=1}^{w} f_{1,j}(1 \cdot 0) = \bigodot_{j=1}^{w} \Pi\langle j \rangle = \Pi$$

$$\begin{aligned} Comp_F^A(1 \cdot 0^w)[^{P}_{V_2}] &= \bigodot_{i=1}^{m} f_{2,i}((Comp_F^A(1 \cdot 0^w))[^{P}_{V_1}][^{V_1}_{p(v_{2,i})}]) \qquad \text{since} \quad \bigcup_{v \in V_2} p(v) \subseteq V_1 \\ &= \bigodot_{i=1}^{m} f_{2,i}(\Pi[^{V_1}_{p(v_{2,i})}]) \qquad \text{by the previous equation} \\ &= \bigodot_{i=1}^{m} f_{2,i}(\Pi[^{U}_{G_i}]) \qquad \text{by definition of } E \\ &= 1^m \qquad \text{by definition of } f_{2,i} \text{ and } \Pi[^{U}_{G_i}] \neq \gamma_i[^{U}_{G_i}] \end{aligned}$$

So $\mathcal{M}_F^A(1 \cdot 0^w) = \Pi \cdot 1^m \models *^w \cdot 1^m$ which is to say $\mathcal{M}_F^A \supseteq T_3$. This completes the first half of the claim.

proof ($\exists F \Rightarrow \exists \Pi$): Assume $F = \{f_{1,1}, f_{1,2}, \ldots, f_{1,w}, f_{2,1}, f_{2,2}, \ldots, f_{2,m}\}$ is a configuration such that $\mathcal{M}_F^A \supseteq T$. What do we know about F? By inspecting T_1, we know

$$Comp_F^A(0 \cdot 0^w)[^{P}_{V_1}] = \bigodot_{j=1}^{w} f_{1,j}((0 \cdot 0^w)[^{S}_{p(v_{1,j})}]) = \bigodot_{j=1}^{w} f_{1,j}(0 \cdot 0) = 0^w$$

by the first item. Hence $f_{1,j}(0 \cdot 0) = 0$. By the second item, we can similarly show $f_{1,j}(0 \cdot 1) = 1$, which leads us to conclude what was shown in equation (A.1).

By inspecting T_2 and T_3, we have for every i, $1 \leq i \leq m$

$$f_{2,i}(Comp_F^A(0 \cdot \gamma_i)[^{P}_{p(v_{2,i})}]) = 0 \neq 1 = f_{2,i}(Comp_F^A(1 \cdot 0^w)[^{P}_{p(v_{2,i})}])$$

$$Comp_F^A(0 \cdot \gamma_i)[^{P}_{p(v_{2,i})}] \neq Comp_F^A(1 \cdot 0^w)[^{P}_{p(v_{2,i})}]$$

Applying equation (A.1) and the definition of E on the l.h.s.,

$$\text{l.h.s.} \ = Comp_F^A(0 \cdot \gamma_i)[^{P}_{V_1}][^{V_1}_{p(v_{2,i})}] = \gamma_i[^{V_1}_{p(v_{2,i})}] = \gamma_i[^{U}_{G_i}]$$

Simplifying the r.h.s. by letting $\Pi = Comp_F^A(1 \cdot 0^w)[^{P}_{V_1}]$,

$$\text{r.h.s.} \ = Comp_F^A(1 \cdot 0^w)[^{P}_{V_1}][^{V_1}_{p(v_{2,i})}] = \Pi[^{V_1}_{p(v_{2,i})}] = \Pi[^{U}_{G_i}]$$

Reassembling, we have $\gamma_i[^{U}_{G_i}] \neq \Pi[^{U}_{G_i}]$ for all $i, 1 \leq i \leq m$ which is to say that Π satisfies (U, Γ) and the claim is proved.

Thus we have SAT $\propto$ $Perf_{SAFns}$ and it is easy to see that the algorithm for the transformation runs in polynomial time (in fact linear time and log space).

Finally, it must be demonstrated that there is a non-deterministic machine that can decide $Perf_{SAFns}$ in time polynomial in the length of (A, T). That is, there must be a poly-time method of writing down a valid SAFns configuration and checking that it is correct. Writing down a function from SAFns requires one bit for every nodal input (to specify whether it should be inverted before entering the AND gate), and one bit for the output (to specify whether the whole function should be inverted). For the complete configuration, this takes one bit for each edge in A and one bit for each node in A. That the configuration is correct can be checked by evaluating each node function once for each item in T. This takes time $O(|V| \times |T|)$ under the assumption that it takes constant time to evaluate any single f_i.

This, and SAT $\propto$ $Perf_{SAFns}$ implies $Perf_{SAFns}$ is NP-complete. □

Appendix B

PROOF FOR LOGISTIC LINEAR NODE FUNCTIONS

The next theorem extends the previous one to the case of certain real-valued node functions. We consider a function set used in [RHW86] wherein every member of the set is a function composed of two parts. The first part is the logistic function and the second is a linear weighted sum of its inputs.

$$f(\alpha) = E(e(\alpha))$$

$$\text{where} \quad e(\alpha) = w_0 + \sum w_i \times \alpha\langle i\rangle,$$

$$\text{and} \quad E(x) = \frac{1}{1+e^{-x}}.$$

We call these functions LLFns (for Logistic Linear Functions). The E function is fixed for all nodes, so to specify a member of LLFns it is enough to specify the weights $w_0, w_1, \ldots$ used in e.

Following [RHW86] again, we say that a value agrees with 1 if it is no smaller than 0.9, and it agrees with 0 if it is no larger than 0.1. Note that $E(x)$ asymptotically approaches 1 as x approaches $+\infty$, and that $E(x)$ asymptotically approaches 0 as x approaches $-\infty$. Let d be some scalar value. We say that α *agrees for high* *d with* β (written $\alpha \overset{d}{\models} \beta$) if there is some value for d beyond which α always agrees with β. This implies that the value of α or β is a function of d.

$$\alpha(d) \overset{d}{\models} \beta(d) \Longleftrightarrow \exists d_0 \text{ such that } \alpha(d) \models \beta(a) \quad \text{for all } d \geq d_0$$

Such agreement is easy to prove if α is monotonic in d and β is constant.

Note that if two such agreement statements hold for the same high parameter then they hold simultaneously for that parameter.

$$(\alpha \overset{d}{\models} \beta \text{ and } \delta \overset{d}{\models} \xi) \Longleftrightarrow \alpha \cdot \delta \overset{d}{\models} \beta \cdot \xi$$

A new notational device is used to select single elements from a string in the case where the element's position in a string is not known except through its relative position in one of the clause sets, G_i, in Γ. For that situation, we use $\frac{i}{k}$ to mean the index in U of the k^{th} element of clause i. Formally, $\frac{i}{k} = (\bigodot_{j=1}^{w} j)\genfrac{[}{]}{0pt}{}{U}{G_i}\langle k \rangle$. Consequently, this identity holds: $\alpha\langle \frac{i}{k} \rangle = \alpha\genfrac{[}{]}{0pt}{}{U}{G_i}\langle k \rangle$.

Theorem 22 *$Perf_{LLFns}$ is NP-complete.*

Proof: We construct a performability problem (A, T) where the architecture, A, is the same as it was in the proof of Theorem 21 except that $R = V_2$ instead of $R = V$, and the task, T, is as follows:

$$T = T_1 \cup T_2 \cup T_3$$

$$\begin{aligned}
T_1 = &\quad \{(0 \cdot \gamma_i, \quad *^{i-1} \cdot 0 \cdot *^{m-i}) : 1 \leq i \leq m\} \\
T_2 = &\ \{\{(0 \cdot \gamma_i^{(k)}, \ *^{i-1} \cdot 1 \cdot *^{m-i}) : 1 \leq k \leq |G_i|\} : 1 \leq i \leq m\} \\
T_3 = &\quad \{(1 \cdot \gamma_i, \quad *^{i-1} \cdot 1 \cdot *^{m-i}) : 1 \leq i \leq m\}
\end{aligned}$$

where $\gamma_i^{(k)}$ is γ_i with the k^{th} relevant bit inverted:

$$\gamma_i^{(k)}\langle j \rangle = \begin{cases} 1 - \gamma_i\langle j \rangle & \text{if } j = \frac{i}{k} \\ \gamma_i\langle j \rangle & \text{otherwise} \end{cases}$$

Claim: there exists a solution configuration F to (A,T) iff there exists a solution assignment Π to (U,Γ). For both directions of the proof we shall use the following definitions (they each stand for the computation performed by the first layer of nodes when the net is given some stimulus in the task):

$$\varsigma_i = Comp_F^A(0\cdot\gamma_i)[{}^{P}_{V_1}] \quad \text{(from } T_1\text{)}$$

$$\beta_i^{(j)} = Comp_F^A(0\cdot\gamma_i^{(j)})[{}^{P}_{V_1}] \quad \text{(from } T_2\text{)}$$

$$\eta_i = Comp_F^A(1\cdot\gamma_i)[{}^{P}_{V_1}] \quad \text{(from } T_3\text{)}$$

proof ($\exists F \Leftarrow \exists\Pi$): Specify the node functions as follows:

$$f_{1,j}(a\cdot b) = \begin{cases} E(-d+2da+2db) & \text{if } \Pi\langle j\rangle = 1 \\ E(-d-2da+2db) & \text{if } \Pi\langle j\rangle = 0 \end{cases}$$

$$f_{2,i}(\alpha) = E(e_{2,i}(\alpha))$$

where

$$e_{2,i}(\alpha) = -d + 2d\sum_{k=1}^{|G_i|} W_{i,k}\times(\varsigma_i\langle\tfrac{i}{k}\rangle - \alpha\langle k\rangle)$$

$$W_{i,k} = \begin{cases} +1 & \text{if } \gamma_i\langle\tfrac{i}{k}\rangle = 1 \\ -1 & \text{if } \gamma_i\langle\tfrac{i}{k}\rangle = 0 \end{cases}$$

The above expression for $e_{2,i}$ is not in standard form but it is straightforward to rearrange it so that it is.

We shall check that each subtask is performed correctly by this configuration. Observe

$$f_{1,j}(0\cdot 0) = E(-d+0+0) \overset{d}{\models} 0$$

$$f_{1,j}(0\cdot 1) = E(-d+0+2d) \overset{d}{\models} 1$$

Hence $Comp_F^A(0\cdot\alpha)'[{}^{P}_{V_1}] \overset{d}{\models} \alpha$. Consequently $\varsigma_i \overset{d}{\models} \gamma_i$. Also

$$f_{2,i}(\varsigma_i[{}^{V_1}_{p(v_{2.i})}]) = E(-d+2d\sum_k W_{i,k}(\varsigma_i\langle\tfrac{i}{k}\rangle - \varsigma_i\langle\tfrac{i}{k}\rangle))) \overset{d}{\models} 0$$

The agreement holds because the total value of the summation is 0. This argument applies to each value of i, and hence for high d, $\mathcal{M}_F^A \supseteq T_1$.

Consider a typical item in T_2. Note that $\beta_i^{(k)} \overset{d}{\models} \gamma_i^{(k)}$, and that $\beta_i^{(k)}\langle\frac{i}{k}\rangle$ therefore differs from $\gamma_i\langle\frac{i}{k}\rangle$ as d increases. The absolute difference converges monotonically to 1, so we have

$$f_{2,i}(\beta_i^{(k)}[^{V_1}_{p(v_{2,i})}]) = E(-d + 2d\sum W_{i,k}(\varsigma_i\langle\tfrac{i}{k}\rangle - \beta_i^{(k)}\langle\tfrac{i}{k}\rangle)) \overset{d}{\models} 1$$

Here we know the agreement for high d holds because the total value of the summation tends to 1 as d increases. Since the equation is valid for all $1 \leq k \leq |G_i|$ for each γ_i, $\mathcal{M}_F^A \supseteq T_2$ for high d.

Next we consider a typical item in T_3. For all nodes in layer 1, observe

$$\text{if } \Pi\langle j\rangle = 1, \quad f_{1,j}(1 \cdot x) = E(-d + 2d + 2dx) \overset{d}{\models} 1 \text{ for } x \in \{0,1\}$$

$$\text{if } \Pi\langle j\rangle = 0, \quad f_{1,j}(1 \cdot x) = E(-d - 2d + 2dx) \overset{d}{\models} 0 \text{ for } x \in \{0,1\}$$

Hence $f_{1,j}(1 \cdot x) \overset{d}{\models} \Pi\langle j\rangle$ and consequently $\eta_i \overset{d}{\models} \Pi$ for all i. Examining the second layer, we know

$$f_{2,i}(\eta_i[^{V_1}_{p(v_{2,i})}]) = E(-d + 2d\sum W_{i,k}(\varsigma_i\langle\tfrac{i}{k}\rangle - \eta_i\langle\tfrac{i}{k}\rangle)) \overset{d}{\models} 1$$

because as d increases the summation converges to some integer representing the number of places where $\varsigma_i[^U_{G_i}]$ is not equal to $\eta_i[^U_{G_i}]$, that is, the number of places where $\gamma_i[^U_{G_i}]$ is not equal to $\Pi[^U_{G_i}]$. By the initial assumption about Π, this integer is at least 1, so the agreement holds (for high d). This demonstrates that $\mathcal{M}_F^A \supseteq T_3$ for high d.

By selecting some value for d which satisfies all the above agreements, $\mathcal{M}_F^A \supseteq T$ and this completes the proof of one direction of the claim.

proof ($\exists F \Rightarrow \exists \Pi$): Let $y_{j,k}$ and $z_{i,k}$ be the weights employed in the node functions as follows: for all i, j, $1 \leq i \leq m$, $1 \leq j \leq w$, let

$$f_{1,j}(a \cdot b) = E(y_{j,0} + y_{j,1}a + y_{j,2}b)$$

$$f_{2,i}(\alpha) = E(z_{i,0} + \sum_{k=1}^{|G_i|} z_{i,k}\alpha\langle k\rangle)$$

Define the satisfying assignment:

$$\Pi\langle j\rangle = \begin{cases} 1 & \text{if } y_{j,1}y_{j,2} > 0 \\ 0 & \text{otherwise} \end{cases}$$

We must show Π satisfies (U, Γ).

By assumption, the configuration F performs T_1 and T_2, so we know for each i, $1 \leq i \leq m$ and for any k, $1 \leq k \leq |G_i|$

$$f_{2,i}(\varsigma_i\left[{V_1 \atop p(v_{2,i})}\right]) \models 0$$

$$f_{2,i}(\beta_i^{(k)}\left[{V_1 \atop p(v_{2,i})}\right]) \models 1$$

so

$$f_{2,i}(\varsigma_i\left[{V_1 \atop p(v_{2,i})}\right]) < f_{2,i}(\beta_i^{(k)}\left[{V_1 \atop p(v_{2,i})}\right])$$

$$E(z_{i,0} + \sum_c z_{i,c}\, \varsigma_i\langle \tfrac{i}{c}\rangle) < E(z_{i,0} + \sum_c z_{i,c}\, \beta_i^{(k)}\langle \tfrac{i}{c}\rangle)$$

but $\varsigma_i\langle j\rangle = \beta_i^{(k)}\langle j\rangle$ for all $j \neq \frac{i}{k}$, or more specifically $\varsigma_i\langle \frac{i}{c}\rangle \neq \beta_i^{(k)}\langle \frac{i}{c}\rangle$ only when $c = k$. Therefore

$$z_{i,k}\, \varsigma_i\langle \tfrac{i}{k}\rangle < z_{i,k}\, \beta_i^{(k)}\langle \tfrac{i}{k}\rangle$$

Let $j = \frac{i}{k}$ and expand both sides in terms of $f_{1,j}$.

$$z_{i,k}E(y_{j,0} + y_{j,2}\, \gamma_i\langle j\rangle) < z_{i,k}E(y_{j,0} + y_{j,2}(1 - \gamma_i\langle j\rangle))$$

$$z_{i,k}\, y_{j,2}\, \gamma_i\langle \tfrac{i}{k}\rangle < z_{i,k}\, y_{j,2}\, (1 - \gamma_i\langle \tfrac{i}{k}\rangle)$$

$$\text{if } \gamma_i\langle j\rangle = 0 \text{ then } 0 < z_{i,k}\, y_{j,2} \tag{B.1}$$

$$\text{if } \gamma_i\langle j\rangle = 1 \text{ then } z_{i,k}\, y_{j,2} < 0 \tag{B.2}$$

Again, by assumption that F configures the net for T_1 and T_2,

$$f_{2,i}(\varsigma_i\big[{V_1 \atop p(v_{2,i})}\big] \models 0$$

$$f_{2,i}(\eta_i\big[{V_1 \atop p(v_{2,i})}\big] \models 1$$

$$E\big(z_{i,0} + \sum_k z_{i,k}\, \varsigma_i\langle {i \atop k}\rangle\big) < E\big(z_{i,0} + \sum_k z_{i,k}\, \eta_i\langle {i \atop k}\rangle\big)$$

$$\sum_k z_{i,k}\, \varsigma_i\langle {i \atop k}\rangle < \sum_k z_{i,k}\, \eta_i\langle {i \atop k}\rangle$$

For this to be true for a given i, there must be at least one k such that

$$z_{i,k}\, \varsigma_i\langle {i \atop k}\rangle < z_{i,k}\, \eta_i\langle {i \atop k}\rangle$$

Letting $j = {i \atop k}$ and expanding both sides as $f_{1,j}$,

$$z_{i,k}\, E\big(y_{j,0} + y_{j,2}\, \gamma_i\langle j\rangle\big) < z_{i,k}\, E\big(y_{j,0} + y_{j,1} + y_{j,2}\, \gamma_i\langle j\rangle\big)$$

$$0 < z_{i,k}\, y_{j,1}$$

From this and (B.1), we find that

$$\gamma_i\langle j\rangle = 0 \Rightarrow 0 < z_{i,k}\, z_{i,k}\, y_{j,1}\, y_{j,2} \Rightarrow 0 < y_{j,1}\, y_{j,2} \Rightarrow \Pi\langle j\rangle = 1$$

Similarly, $\gamma_i\langle j\rangle = 1 \Rightarrow \Pi\langle j\rangle = 0$. Summarizing, for all γ_i there exists a k such that $\gamma_i\langle {i \atop k}\rangle \neq \Pi\langle {i \atop k}\rangle$, or rather $\gamma_i[{U \atop G_i}] \neq \Pi[{U \atop G_i}]$. That is, Π satisfies (U, Γ) and the claim is proved.

The claim establishes that the reduction from SAT is valid. Since the transformation can be performed in polynomial time, $Perf_{LLFns}$ is NP-hard.

$Perf_{LLFns}$ is in NP if there is a polynomial-time procedure to write down values for all the weights. For the case where the weights are truly real-valued (meaning that a weight would have a potentially infinite number of digits), it has not yet been proven that there is a finite approximation that is effectively equivalent to the real numbers (as Hong has done for LSFns). However, for the more realistic case of fixed resolution in each 'real' weight, specifying the configuration is easily performed in polynomial time. With that minor caveat, we have proved $Perf_{LLFns}$ is NP-complete. □

Three aspects of LLFns are crucial to the preceding proof: E is monotonic, E is bounded, and e is linear. Other aspects were convenient but not necessary; for example, every node had a fixed E function, every node had the *same* E function, and that E was onto the unit interval $[0,1]$. We proved the theorem for LLFns only in order to avoid excessive abstraction, but the theorem is extendible to other node function sets.

If we define the quasi-linear functions (QLFns) as all those functions of the form $E(e(\alpha))$ where e is linear and E is a bounded and monotonic, then for some appropriate definition of agreement we have

Corollary 23 *$Perf_{QLFns}$ is NP-complete.*

The theorem is probably extendible to different manifestations of non-linearity, but we note that something about E should be non-linear, for if E (as well as e) is linear, then the net as a whole can implement only linear mappings. From the point of view of connectionists, this is uninteresting.

Appendix C

PROOF FOR CASE WITHOUT DON'T CARES

The proof of Theorem 1 uses the * symbol to denote 'don't cares' in the response strings. This is often not a feature of connectionist experiments so the following proof avoids the * in order to demonstrate that it is not an important change to the model.

Theorem 24 *$Perf_{AOFns}$ is NP-complete even when responses have no *'s.*

Proof: by reduction from 3SAT. The proof is modelled on the one for Theorem 1. Let the 3SAT problem be (Z, C) where Z is a set of variables $\{\varsigma_1, \varsigma_2, \varsigma_3, \ldots\}$ and C is a set of disjunctive clauses over them. Let $w = |Z|$ be the number of variables and $m = |C|$ the number of clauses. For (Z, C) to be satisfiable, there must be an assignment $\Pi : Z \to \{0,1\}$ such that at least one literal in each clause has value 1.

Formally, the 3SAT instance (Z, C) is reduced to (A, T), where

$$\begin{aligned}
A &= (P, V, S, R, E) \\
S &= \{a, b, d, e\} \\
V &= \{u_i, v_i, w_i, x_i, y_i, z_i : \varsigma_i \in Z\} \cup \{c_j : C_j \in C\} \\
R &= \{u_i, x_i, y_i, v_i : \varsigma_i \in Z\} \cup \{c_j : C_j \in C\} \\
P &= S \cup V
\end{aligned}$$

$$\begin{aligned}
E &= \{(a, w_i), (a, z_i), (b, w_i), (b, z_i), \\
&\quad (w_i, u_i), (w_i, x_i), (w_i, y_i), (z_i, x_i), (z_i, y_i), (z_i, v_i), \\
&\quad (d, u_i), (d, v_i), : \varsigma_i \in Z\} \\
&\quad \cup \{(w_i, c_j) : \varsigma_i \in C_j\} \cup \{(z_i, c_j) : \overline{\varsigma_i} \in C_j\} \cup \{(e, c_j) : C_j \in C\} \\
T &= \{I_1, I_2, I_3\} \\
I_1 &= (0\,0\,1\,1,\ (0\,0\,0\,0)^w\, 0^m) \\
I_2 &= (1\,1\,1\,0,\ (1\,1\,1\,1)^w\, 0^m) \\
I_3 &= (0\,1\,0\,1,\ (0\,0\,1\,0)^w\, 1^m)
\end{aligned}$$

This construction is explained in a 2-stage example. Stage 1: For every variable $\varsigma_j \in Z$ construct the partial architecture and partial task shown in Figure C.1. This is very similar to Figure 4.1 on page 42. The differences are that w and z are no longer network outputs; instead they go to new nodes u and v which are network outputs. Also there is a new input, d, which goes only to these new nodes. In the task, note that all response bits for u and v are the same as they were for w and z except that the *'s have been replaced by 0's (arbitrarily). In the items where w and z had been defined, d is a 1; in the items where w and z had been 'don't cares', d is a 0.

From items 1 and 2 we know

$$f_u(f_w(0,0),1) = 0 \neq 1 = f_u(f_w(1,1),1). \tag{C.1}$$

Hence

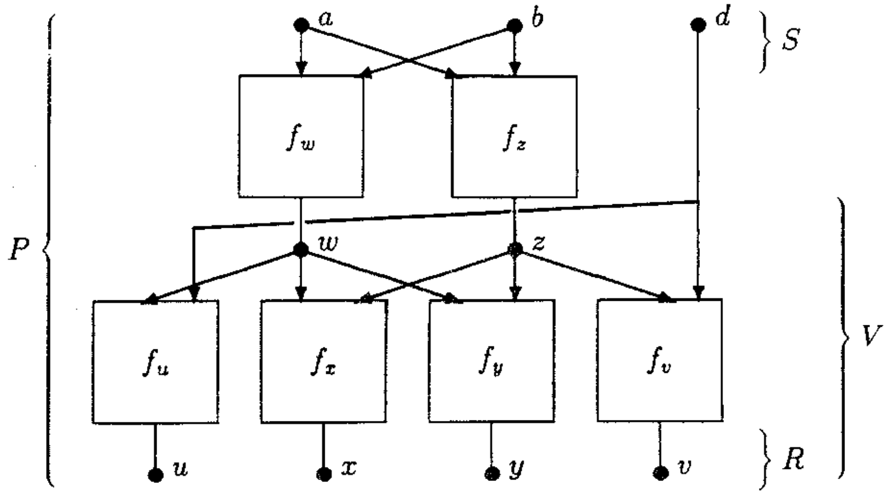
$$f_w(0,0) \neq f_w(1,1). \tag{C.2}$$

Similarly

$$f_z(0,0) \neq f_z(1,1). \tag{C.3}$$

By comparing item 2 and item 3 we know

$$f_x(f_w(1,1), f_z(1,1)) = 1 \neq 0 = f_x(f_w(0,1), f_z(0,1))$$



$$\begin{array}{rl} & a\ b\ d, \quad u\ x\ y\ v \\ \text{item 1}: & (0\ 0\ 1, \quad 0\ 0\ 0\ 0) \\ \text{item 2}: & (1\ 1\ 1, \quad 1\ 1\ 1\ 1) \\ \text{item 3}: & (0\ 1\ 0, \quad 0\ 0\ 1\ 0) \end{array}$$

Figure C.1: The construction for each variable $\varsigma \in Z$.

$$f_w(1,1) \neq f_w(0,1) \text{ or } f_z(1,1) \neq f_z(0,1),$$

and by using (C.2) and (C.3)

$$f_w(0,0) = f_w(0,1) \text{ or } f_z(0,0) = f_z(0,1). \tag{C.4}$$

By comparing item 1 and item 3 we know

$$f_y(f_w(0,0), f_z(0,0)) = 0 \neq 1 = f_y(f_w(0,1), f_z(0,1))$$

$$f_w(0,0) \neq f_w(0,1) \text{ or } f_z(0,0) \neq f_z(0,1) \tag{C.5}$$

We will associate some SAT variable ς_j with the group of nodes in this construction. For mnemonic value and brevity, let $\langle\varsigma\rangle$ stand for the truth of the inequality $f_w(0,0) \neq f_w(0,1)$. And let $\langle\bar{\varsigma}\rangle$ stand for the truth of the inequality

$f_z(0,1)$, $f_z(0,1)$. Translating (C.4) and (C.5) we have

$$(\text{not}\langle\varsigma\rangle \text{ or } \text{not}\langle\overline{\varsigma}\rangle) \text{ and } ((\langle\varsigma\rangle \text{ or } \langle\overline{\varsigma}\rangle)$$

which implies $\langle\varsigma\rangle = \text{not}\langle\overline{\varsigma}\rangle$.

Stage 2: For each clause in the SAT system construct a single node in the second layer of the architecture with inputs from all nodes associated with its participating literals and an input from post e. Putting variables' nodes and the clause node together, we get what is shown in Figure C.2. It shows the construction for an example SAT system consisting of only one clause $(\varsigma_1, \overline{\varsigma_2}, \overline{\varsigma_3})$. Observe that each item consists of the stimulus from an item from Figure C.1, a new stimulus bit for e, three replications of the associated response (one per variable), and another response bit for the clause node.

Claim: The constructed architecture can perform the task iff the SAT instance is satisfiable.

Proof: By inspecting item 1 and item 3,

$$f_c(f_w^1(0,0), f_z^2(0,0), f_z^3(0,0), 1) = 0$$

$$f_c(f_w^1(0,1), f_z^2(0,1), f_z^3(0,1), 1) = 1$$

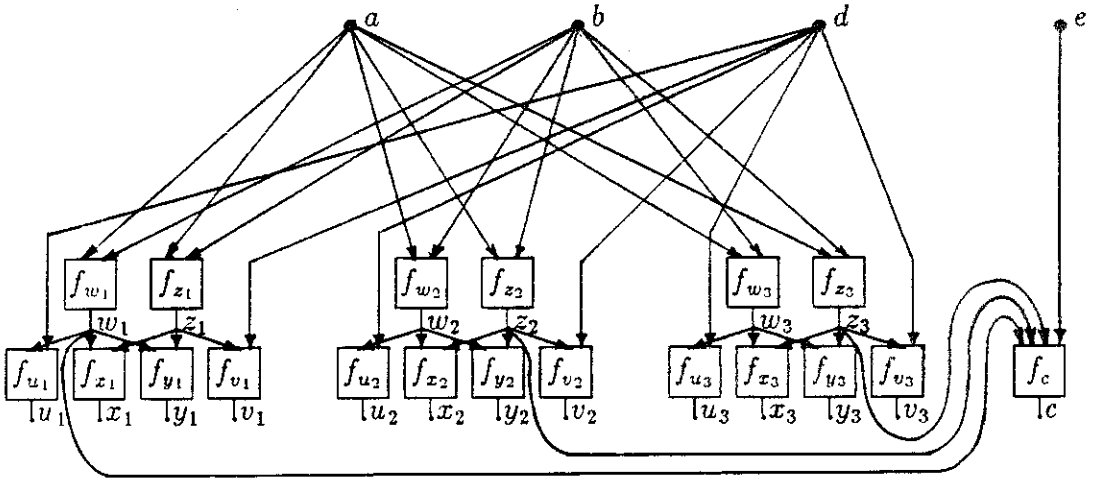
Since not all of the arguments can be the same, conclude

$$\langle\varsigma_1\rangle \text{ or } \langle\overline{\varsigma_2}\rangle \text{ or } \langle\overline{\varsigma_3}\rangle.$$

Now if Π exists then let $\langle\varsigma_j\rangle = \Pi(\varsigma_j)$, that is, let $f_w^j(0,0) = 0$ and $f_w^j(0,1) = \Pi(\varsigma_j)$ and $f_w^j(1,1) = 1$ for all j, or more definitively, let

$$f_w^j = \begin{cases} \text{OR} & \text{if } \Pi(\varsigma_j) = 1 \\ \text{AND} & \text{if } \Pi(\varsigma_j) = 0 \end{cases} \quad \text{and} \quad f_z^j = \begin{cases} \text{AND} & \text{if } \Pi(\varsigma_j) = 1 \\ \text{OR} & \text{if } \Pi(\varsigma_j) = 0 \end{cases}.$$

For all variables ς_j let $f_u^j = f_v^j = f_x^j = \text{AND}$ and $f_y^j = \text{OR}$, and for the clause node let $f_c = \text{OR}$. The items are all performed correctly.



	a	b	d	e,	u_1	x_1	y_1	v_1	u_2	x_2	y_2	v_2	u_3	x_3	y_3	v_3	c
item 1 :	(0	0	1	1,	0	0	0	0	0	0	0	0	0	0	0	0	0)
item 2 :	(1	1	1	0,	1	1	1	1	1	1	1	1	1	1	1	1	1)
item 3 :	(0	1	0	1,	0	0	1	0	0	0	1	0	0	0	1	0	1)

Figure C.2: The composed construction for Theorem 24. This example is for the single clause $(\varsigma_1, \overline{\varsigma_2}, \overline{\varsigma_3})$.

Conversely, if a configuration exists let $\Pi(\varsigma_j) = \langle\varsigma_j\rangle = f_w^j(0,1) \oplus f_w^j(0,0)$, and observe $\langle\varsigma_1\rangle$ or $\langle\overline{\varsigma_2}\rangle$ or $\langle\overline{\varsigma_3}\rangle$ implies $\varsigma_1 = 1$ or $\overline{\varsigma_2} = 1$ or $\overline{\varsigma_3} = 1$ as required. This proves the claim. □

The extension to multi-clause systems should be clear.

Thus we have SAT $\propto$ $Perf_{AOFns}$ and it is easy to see that the algorithm for the transformation runs in polynomial time.

Finally, as argued for Theorem 1, $Perf_{SAFns} \in NP$. Hence $Perf_{SAFns}$ is NP-complete. □

Recall that items 1 and 2 produced equation C.1 which forced a relationship between $f_w(0,0)$ and $f_w(1,1)$ given by equation C.2. From items 2 and 3 we know

$$f_u(f_w(1,1),1) = 1 \neq 0 = f_u(f_w(0,1),0),$$

but this does not force any particular relationship between $f_w(1,1)$ and $f_w(0,1)$ (nor do items 1 and 3 force any relationship between $f_w(0,0)$ and $f_w(0,1)$). Hence $f_w(0,1)$ might just as well have been specified as a 'don't-care' as it was in the proof for Theorem 1. Thus input d and node u have been employed here as a switch to simulate the do-care/don't-care distinction for the output from node w. Similarly, d and v have been used for z. A roughly similar technique using input e simulated the do-care/don't-care distinction for the output from node c.

We believe these techniques could be applied generally. They make this theorem stronger at the expense of extra complications in the proof. We prefer to make use of the * in our other theorems in order to simplify their proofs.

Appendix D

PROOF FOR PLANAR CASE WITH LSFNS

The proof for Theorem 16 used LUFns as its node function set, and hence does not cover the specific (and conventional) case of node function sets that are linearly separable. This appendix gives a proof that is strong enough to cover LSFns. In particular, we give a construction for a crossover using SAFns, which is a node function set described in Appendix A. Because SAFns $\subseteq$ LSFns $\subseteq$ LUFns, this theorem is sufficient to cover the linearly separable case whereas the proof for Theorem 16 was not.

Figure D.1 is the construction used in [Lic82, Fig.4] as a crossover box in his proof of *NP*-completeness for planar SAT. For that purpose the circles were interpreted as variables and the squares as clauses. The diagram is a demonstration that the following SAT system has a planar layout:

clauses 1–3:	$(\overline{a_2} \vee \overline{b_2} \vee \alpha)(a_2 \vee \overline{\alpha})(b_2 \vee \overline{\alpha})$	i.e. $a_2 b_2 \Leftrightarrow \alpha$;
clauses 4–6:	$(\overline{a_2} \vee b_1 \vee \beta)(a_2 \vee \overline{\beta})(\overline{b_1} \vee \overline{\beta})$	i.e. $a_2 \overline{b_1} \Leftrightarrow \beta$;
clauses 7–9:	$(a_1 \vee b_1 \vee \gamma)(\overline{a_1} \vee \overline{\gamma})(\overline{b_1} \vee \overline{\gamma})$	i.e. $\overline{a_1}\overline{b_1} \Leftrightarrow \gamma$;
clauses 10–12:	$(a_1 \vee \overline{b_2} \vee \delta)(\overline{a_1} \vee \overline{\delta})(b_2 \vee \overline{\delta})$	i.e. $\overline{a_1} b_2 \Leftrightarrow \delta$;
clause 13:	$(\alpha \vee \beta \vee \gamma \vee \delta)$	
clauses 14–17:	$(\overline{\alpha} \vee \overline{\beta})(\overline{\beta} \vee \overline{\gamma})(\overline{\gamma} \vee \overline{\delta})(\overline{\delta} \vee \overline{\alpha})$	
clauses 18–19:	$(\overline{a} \vee a_2)(a \vee \overline{a_2})$	i.e. $a \Leftrightarrow a_2$;
clauses 20–21:	$(\overline{b} \vee b_2)(b \vee \overline{b_2})$	i.e. $b \Leftrightarrow b_2$;

We will use this construction in the proof of the following theorem:

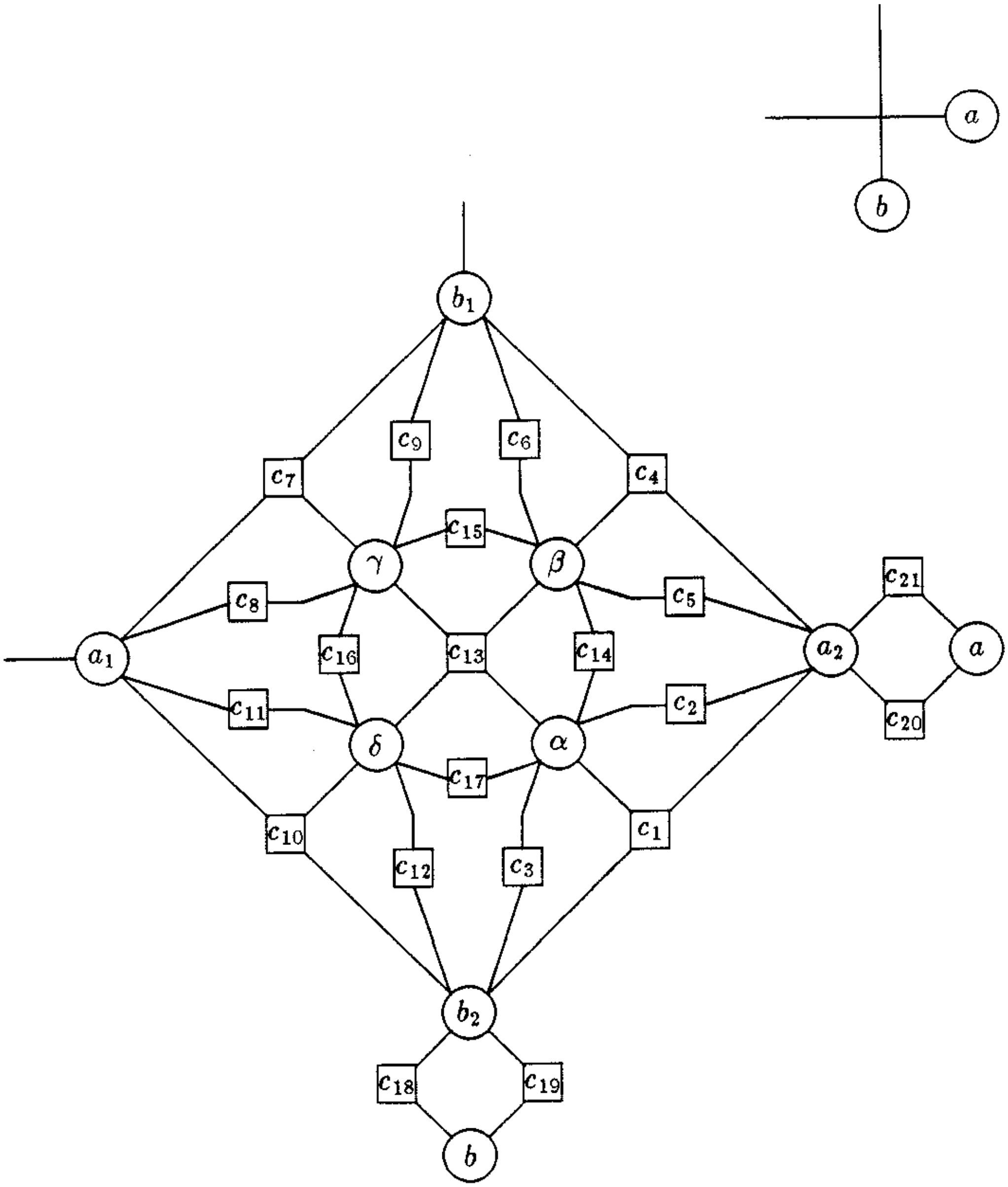


Figure D.1: Construction from Lichtenstein for Planar SAT. The prototypical crossover of two lines shown in the upper right is replaced by the much larger construction, which provides the same constraints as the smaller one would have.

Theorem 25 *For any node function set including SAFns, loading is NP-complete even for 2-layered architectures with planar SCI graphs.*

Proof: We give only a construction for a 'crossover box' which can be used to eliminate one crossing of connections as they might occur in the proof of Theorem 15.

For our purpose Figure D.1 is re-interpreted as the plan view of an architecture—the circles being first-layer nodes and the squares being second-layer nodes. To accompany this architecture, a task is constructed to mimic the effect of each clause. For this we use the techniques of constructing items that are used in the proof of Theorem 21. First, two items ensure that $f_i(0,0) = 0$ and $f_i(1,1) = 1$ for all variables $i \in \{a, a_1, a_2, b, b_1, b_2, \alpha, \beta, \gamma, \delta\}$:

a	a_1	a_2	b	b_1	b_2	α	β	γ	δ		a	a_1	a_2	b	b_1	b_2	α	β	γ	δ	c_1	c_2	c_3	$\ldots$	c_{21}
00	00	00	00	00	00	00	00	00	00	$\mapsto$	0	0	0	0	0	0	0	0	0	0	*	*	*		*
11	11	11	11	11	11	11	11	11	11	$\mapsto$	1	1	1	1	1	1	1	1	1	1	*	*	*		*

Each of Lichtenstein's clauses produces 2 items as in the following example:

a	a_1	a_2	b	b_1	b_2	α	β	γ	δ		a	a_1	a_2	b	b_1	b_2	α	β	γ	δ	c_1	c_2	c_3	$\ldots$	c_{21}
**	**	11	**	**	11	00	**	**	**	$\mapsto$	*	*	*	*	*	*	*	*	*	*	0	*	*		*
**	**	01	**	**	01	01	**	**	**	$\mapsto$	*	*	*	*	*	*	*	*	*	*	1	*	*		*

This corresponds to clause 1, which was $(\overline{a_2} \vee \overline{b_2} \vee \alpha)$. These two items insure that

$$f_{c_1}(f_{a_2}(1,1), f_{b_2}(1,1), f_\alpha(0,0)) = 0 \neq 1 = f_{c_1}(f_{a_2}(0,1), f_{b_2}(0,1), f_\alpha(0,1))$$

$$f_{c_1}(1,1,0) = 0 \neq 1 = f_{c_1}(f_{a_2}(0,1), f_{b_2}(0,1), f_\alpha(0,1))$$

$$f_{a_2}(0,1) \neq 1 \text{ or } f_{b_2}(0,1) \neq 1 \text{ or } f_\alpha(0,1) \neq 0$$

The direct correspondence to $(\overline{a_2} \vee \overline{b_2} \vee \alpha)$ should be clear. □

REFERENCES

[ACP87] Arnborg, S., D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a *k*-tree. *SIAM J. Algebraic and Discrete Methods*, 8(2), April 1987.

[AHS85] Ackley, D. H., G. E. Hinton, and T. J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9:147–169, 1985.

[Ale84] Aleksander, Igor. *Artificial Vision for Robots.* Chapman & Hall, New York, 1984.

[And72] Anderson, James A. A simple neural network generating an interactive memory. *Mathematical Biosciences*, 14:197–220, 1972.

[AP88] Arnborg, Stefan and Andrzej Proskurowski. Linear time algorithms for *NP*-hard problems restricted to partial *k*-trees. *Discrete Applied Mathematics*, 1988. To appear.

[AR88] Anderson, James A. and Edward Rosenfeld, editors. *Neurocomputing—Foundations of Research.* MIT Press, Cambridge, Massachusetts, 1988.

[AS83] Angluin, D. and C. Smith. Inductive inference: theory and methods. *Computing Surveys*, 15(3):237–269, September 1983.

[BA85] Barto, A. G. and P. Anandan. Pattern recognizing stochastic learning automata. *IEEE Transactions on Systems, Man, and Cybernetics*, 15:360–375, 1985.

[Bac42] Bacon, R. *Opus maius.* Pressus Gutenbergus, 1442. Manuscript received 1274.

[Bar82] Barahona, Fransisco. On the computational complexity of Ising spin glass models. *Journal of Physics A: Math. Gen.*, 15:3241–3253, 1982.

[Bar85] Barto, Andrew G. Learning by statistical cooperation of self-interested neuron-like computing elements. *Human Neurobiology*, 4:229–256, 1985.

[BB75] Blum, L. and M. Blum. Toward a mathematical theory of inductive inference. *Information and Control*, 28:125–155, 1975.

[BEHW87] Blumer, Anselm, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Occam's razor. *Information Processing Letters*, 24:377–380, 1987.

[BR88] Blum, A. and R. Rivest. Training a 3-neuron neural net is *NP*-complete. Typescript, M.I.T. Lab for Comp. Sci., 1988. Submitted to the 2nd Conference on Neural Information Processing Systems.

[BV87] Baldi, Pierre and Santosh S. Venkatesh. On properties of networks of neuron-like elements. Moore School of Electrical Engineering, U. Penn., December 1987. Typescript.

[Car50] Carnap, R. *Logical Foundations of Probability.* University of Chicago Press, Chicago, Illinois, 1950.

[CES81] Chung, M. J., W. M. Evangelist, and I. H. Sudborough. Some additional examples of bandwidth constrained *NP*-complete problems. In *Proceedings of 1981 Conference on Information Science and Systems*, Dept. of E.E., Johns Hopkins University, 1981.

[Cho80] Chomsky, N. Initial states and steady states. In M. Piatelli-Palmarini, editor, *Language and Learning*, pages 107–130, Harvard University Press, Cambridge, Mass., U.S.A., 1980.

[CK87] Corneil, D. G. and J. M. Keil. A dynamic programming approach to the dominating set problem on *k*-trees. *SIAM J. Algebraic and Discrete Methods*, 8(4), October 1987.

[DH73] R Duda, Richard O. and Peter E. Hart. *Pattern Classification and Scene Analysis.* Wiley, New York, 1973.

[DM81] Dietterich, T.G. and R.S. Michalski. Inductive learning of structural descriptions: Evaluation criteria and comparative review of selected methods. *Artificial Intelligence*, 16:601–617, 1981.

[GGJK78] Garey, M. R., R. L. Graham, D. S. Johnson, and D. E. Knuth. Complexity results for bandwidth minimization. *SIAM Journal of Applied Mathematics*, 34(3):477–495, 1978.

[GJ79] Garey, M. R. and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, San Francisco, 1979.

[God87] Godbeer, Gail H. *The Computational Complexity of the Stable Configuration Problem for Connectionist Models.* Master's thesis, Dept. Computer Science, University of Toronto, Toronto, Ontario, Canada, September 1987.

[Gol67] Gold, E. Language identification in the limit. *Information and Control*. 10:447–474, 1967.

[Hin87] Hinton, Geoffrey E. *Connectionist Learning Procedures*. Technical Report CMU-CS-87-115, Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA 15213, 1987.

[HMS66] Hunt, E. B., J. Marin, and P.J. Stone. *Experiments in Induction*. Academic Press, New York, 1966.

[Hon87] Hong, Jai-wei. *On Connectionist Models*. Technical Report, Dept. Computer Science, University of Chicago, Chicago, Ill., U.S.A., May 1987.

[Hop82] Hopfield, J. J. Neural networks and physical systems with emergent collective computational capabilities. In *Proceedings of the National Academy of Sciences*, pages 2554–2558, 1982.

[HS86] Hinton, Geoffrey E. and Terrence J. Sejnowski. Learning and relearning in Boltzmann machines. In David E. Rumelhart and Jay L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol.1: Foundations*, chapter 7, Bradford Books/MIT Press, Cambridge, MA., 1986.

[HV86] Hampson, S. E. and D. J. Volper. Linear function neurons: Structure and training. *Biological Cybernetics*, 53:203–217, 1986.

[HV87] Hampson, S. E. and D. J. Volper. Disjunctive models of boolean category representation. *Biological Cybernetics*, 56:121–137, 1987.

[HW79] Hubel, David H. and Torsten N. Weisel. Brain mechanisms of vision. In *The Brain*, chapter VII, Freeman, 1979. Also appeared in September 1979 issue of Scientific American.

[Jud87a] Judd, J. S. *Complexity of Connectionist Learning with Various Node Functions*. Technical Report 87-60, University of Massachusetts, Amherst, MA, 1987.

[Jud87b] Judd, J. S. Learning in networks is hard. In *Proceedings of the First International Conference on Neural Networks*, pages 685–692, I.E.E.E., San Diego, California, June 1987.

[Jud88a] Judd, J. S. *A Generalization of Bandwidth*. Technical Report, University of Massachusetts, Amherst, MA 01003, 1988. In preparation.

[Jud88b] Judd, J. S. The intractability of learning in connectionist networks. 1988. Submitted for publication.

[Jud88c] Judd, J. S. *Neural Network Design and the Complexity of Learning.* PhD thesis, Computer and Information Science dept., University of Massachusetts, Amherst, Mass. U.S.A., 1988. In hand.

[Jud88d] Judd, J. S. On the complexity of loading shallow neural networks. *Journal of Complexity*, September 1988. Special issue on Neural Computation, in press.

[KLPV87] Kearns, M., Ming Li, Leonard Pitt, and Leslie Valiant. On the learnability of boolean formulae. In *Proc. 19th Symposium on Theory of Computing*, pages 285–295, ACM, New York, 1987.

[Koh77] Kohonen, T. *Associative Memory—A System Theoretic Approach.* Springer-Verlag, Berlin, 1977.

[Koh84] Kohonen, T. *Self Organization and Associative Memory.* Springer-Verlag, Berlin, 1984.

[lC85] le Cun, Yann. Une procedure d'apprentissage pour reseau a sequil assymetrique. *Proceedings of Cognitiva*, 85:599–604, 1985.

[Lic82] Lichtenstein, David. Planar formulae and their uses. *SIAM Journal of Computing*, 11(2):329–343, 1982.

[Lip87] Lipscomb, John. *On the Computational Complexity of Finding a Connectionist Model's Stable State Vector.* Master's thesis, Dept. Computer Science, University of Toronto, Toronto, Ontario, Canada, October 1987.

[Lit87] Littlestone, Nick. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. In *28th Symposium on Foundations of Computer Science*, pages 68–77, I.E.E.E., 1987.

[Mit77] Mitchell, T. M. Version spaces: a candidate elimination approach to rule learning. In *Proceedings of IJCAI 5*, pages 305–310, 1977.

[MP72] Minsky, Marvin and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry.* MIT Press, Cambridge, Mass., 1972.

[MR86] McClelland, Jay L. and David E. Rumelhart, editors. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol.2: Psychological and Biological Models.* Bradford Books/MIT Press, Cambridge, MA., 1986.

[MS81] Monien, Burkhard and Ivan Hal Sudborough. Bandwidth constrained *NP*-complete problems. In *13th Symposium on Theory of Computing*, pages 207–217, 1981.

[Mur65] Muroga, S. Lower bounds of the number of threshold functions and a maximum weight. *Transactions on Electronic Computers*, 14:136–148, 1965.

[Mur71] Muroga, S. *Threshold Logic and its Applications.* Wiley-Interscience, 1971.

[Nil65] Nilsson, Nils J. *Learning Machines: Foundations of Trainable Pattern-Classification Machines.* McGraw-Hill, 1965.

[NL77] Narendra, Kumpati S. and S. Lakshmivarahan. Learning automata—a critique. *Journal of Cybernetics and Information Science*, 1, fall 1977.

[NT74] Narendra, Kumpati S. and M. A. L. Thathatchar. Learning automata—a survey. *I.E.E.E. Trans on Systems, Man, and Cybernetics*, SMC-4(4):323–334, July 1974.

[Omo87] Omohundro, Stephen M. *Efficient Algorithms with Neural Network Behaviour.* Technical Report UIUCDCS-R-87-1331, Dept. Computer Science, University of Illinois at Urbana-Champaign, 1304 W. Springfield Ave., Urbana, Il 61801, U.S.A., April 1987.

[OSW86] Osherson, Daniel N, Michael Stob, and Scott Weinstein. *Systems that Learn.* MIT Press, Cambridge, Massachusetts, 1986.

[Par85] Parker, D. B. *Learning Logic.* Technical Report TR-47, Massachusetts Institute of Technology, Cambridge, MA, USA 02195, 1985.

[Por87] Porat, Sara. *Stability and Looping in Connectionist Models with Asymmetric Weights.* Technical Report TR 210, Computer Science Dept., University of Rochester, Rochester, N.Y. 14627, March 1987.

[PS85] Peled, Uri N. and Bruno Simeone. Poynomial-time algorithms for regular set-covering and threshold synthesis. *Discrete Applied Mathematics*, 12:57–69, 1985.

[PV86] Pitt, L. and L. G. Valiant. *Computational Limitations on Learning From Examples.* Technical Report TR-05-86, Aiken Computing Lab, Harvard University, 1986. To appear in JACM.

[RHM86] Rumelhart, D. E., G. E. Hinton, and J. L. McClelland. A general framework for parallel distributed processing. In David E. Rumelhart and Jay L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol.1: Foundations*, chapter 2, Bradford Books/MIT Press, Cambridge, MA., 1986.

[RHW86] Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In David E. Rumelhart and Jay L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol.1: Foundations*, Bradford Books/MIT Press, Cambridge, MA., 1986.

[RM86] Rumelhart, David E. and Jay L. McClelland, editors. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol.1: Foundations.* Bradford Books/MIT Press, Cambridge, MA., 1986.

[Ros61] Rosenblatt, Frank. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Spartan Books, 6411 Chillum Place N.W., Washington, D.C., 1961.

[RS86] Robertson, Neil and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7:309–322, 1986.

[RZ85] Rumelhart, D. E. and D. Zipser. Feature discovery by competitive learning. *Cognitive Science*, 9:75–112, 1985.

[Sha81] Shapiro, Ehud Y. *Inductive Inference of Theories From Facts.* Technical Report Research Report 192, Yale University, department of Computer Science, New Haven, Connecticut, USA, February 1981.

[SW81] Sklansky, Jack and Gustav N. Wassel. *Pattern Classification and Trainable Machines.* Springer-Verlag, 1981.

[TDC86] Toulouse, Gèrard, Stanislas Dehaene, and Jean-Pierre Changeux. Spin glass model of learning by selection. *Proc. National Academy of Science USA*, 1695–1698, March 1986. Biophysics.

[Tes87] Tesauro, Gerald. Scaling relationships in back-propogation learning: Dependence on training set size. *Complex Systems*, 1:367–372, 1987.

[TJ88] Tesauro, Gerald and Robert Janssens. Scaling relationships in back-propogation learning: Dependence on predicate order. *Complex Systems*, 2:39–44, 1988.

[TR81] Thathachar, M. A. L. and K. R. Ramakrishnan. An automaton model of a hierarchical learning system. In *Proceedings of 8th Biannual Congress, Kyoto Japan*, pages 1065–1070, Control Science and Technology, 1981.

[Val84] Valiant, L. G. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

[Val85] Valiant, L. G. Learning disjunctions of conjunctions. In *Proceedings of the 9th IJCAI*, pages 560–566, Los Angeles, California, August 1985.

[VH86] Volper, D. J. and S. E. Hampson. Connectionist models of boolean category representation. *Biological Cybernetics*, 54:393–406, 1986.

[WC80] Wexler, Kenneth and Peter W. Culicover. *Formal Principles of Language Acquisition.* MIT Press, Cambridge, Massachusetts, 1980.

[WH60] Widrow, Bernard and Marcian E. Hoff. Adaptive switching circuits. In *1960 IRE WESCON Convention Record*, pages 96–104, IRE, New York, 1960.

[WHL85] Wimer, T. V., S. T. Hedetniemi, and R. Laskar. A methodology for constructing linear graph algorithms. In *Congressium Numerantium*, pages 43–60, 1985.

[Wil86a] Williams, Ronald J. The logic of activation functions. In David E. Rumelhart and Jay L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol.1: Foundations*, Bradford Books/MIT Press, Cambridge, MA., 1986.