

May 1993

LIDS-TH-2177

Sponsor Acknowledgments

Army Research Office
ARO DAAL03-86-K-0171

Army Research Office
ARO DAAL03-92-G-0115

Complexity Issues dealing with Networks that Compute Boolean Functions

Upendra Vasant Chaudhari

This report is based on the unaltered thesis of Upendra Vasant Chaudhari submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of Master of Science at the Massachusetts Institute of Technology in May 1993.

This research was conducted at the M.I.T. Laboratory for Information and Decision Systems with research support gratefully acknowledged by the above mentioned sponsors.

Laboratory for Information and Decision Systems
Massachusetts Institute of Technology
Cambridge, MA 02139, USA

**Complexity Issues dealing with Networks that
Compute Boolean Functions**

by

Upendra Vasant Chaudhari


Submitted to the Department of Electrical Engineering and
Computer Science
in partial fulfillment of the requirements for the degree of
Master of Science in Electrical Engineering and Computer Science
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY


May 1993

© Upendra Vasant Chaudhari, MCMXCIII. All rights reserved.

The author hereby grants to MIT permission to reproduce and to
distribute copies
of this thesis document in whole or in part, and to grant others the
right to do so.

Author 
Department of Electrical Engineering and Computer Science

May 11, 1993

Certified by 

Sanjoy K. Mitter
Professor of Electrical Engineering
Thesis Supervisor

Accepted by
Campbell L. Searle
Chairman, Departmental Committee on Graduate Students

Complexity Issues dealing with Networks that Compute Boolean Functions

by

Upendra Vasant Chaudhari

Submitted to the Department of Electrical Engineering and Computer Science
on May 11, 1993, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

This thesis will analyze Linear Threshold Functions as well as the complexity of constructing minimal network structures composed of those functions for the task of classification. The main contributions of this Thesis fall into two categories. The first deals with Linear Threshold Functions. Here a novel methodology is presented to determine the linear separability of Boolean functions which makes use of a matrix formulation of the problem and reduces it to a vector formulation. The second part of the thesis will show that the problem of finding a network with the fewest functional elements, of the *SAF* class, which computes a specified Boolean function is intractable in that this procedure implicitly solves an NP-complete problem.

Thesis Supervisor: Sanjoy K. Mitter
Title: Professor of Electrical Engineering

Acknowledgments

First and foremost I would like to thank Professor Sanjoy Mitter for his considerable and significant contributions toward the direction, scope, and content of this thesis. I have learned a great deal in this endeavor.

I very much appreciate the help that the LIDS staff has given me over the course of my work. I have also found talking with the students in LIDS to be very valuable.

And of course I would like to thank my family for their support.

This work was supported by the Army Research Office under contracts DAAL03-86-K-0171 and DAAL03-92-G-0115 (Center for Intelligent Control Systems).

Contents

1	Introduction	7
1.1	Contributions of the Thesis	12
2	Transformation Between an MLP and a Decision Tree	13
2.1	MLPs to Decision Trees	13
2.1.1	Decision Trees to MLPs	16
2.2	Relevance	16
3	Some Results for Perceptrons (of order 1)	17
3.1	A Formalism for Discussion	17
3.2	A Different Approach	18
3.3	A Linear Algebraic Formulation	19
3.3.1	An Example of the use of Theorem 1	22
4	Some Results for Multi-Layered Perceptrons (of order 1)	23
4.1	Changing the Perceptron Function	23
4.2	Novel Complexity Measures	26
4.3	Finding A Minimal Representation	28
4.4	A Second Method to Find A Minimal Representation	29
4.5	More Than Two Layers	29
4.6	Complexity of Reduction	30
4.6.1	Judd's Definitions and Results	30
4.7	Extensions	31
4.7.1	Further Extensions	39

List of Figures

1-1	A Decision Tree	8
1-2	A Multi-Layered Perceptron	10
4-1	A Directed Graph Representation of a Boolean Function	24
4-2	An MLP Layer	25
4-3	Operation of One MLP Layer	26
4-4	A minimal specification	37
4-5	Judd's Architecture (computes the constructed task)	38

Chapter 1

Introduction

The area of classification is heavily studied. In the literature, two prominent classification schemes, Decision Trees and Neural Networks, get a significant amount of the attention. The basic problem is simple: Given some data, there are various ways that they can be arranged in an abstract data structure. For example, if A is a set of n cars, then A can be partitioned into two sets A_1 and A_2 such that A_1 contains all the cars in A will less than 50,000 miles on their odometers, and A_2 has the rest. The mileage on the odometer is a property of the elements of A , which may have many more properties of interest. As more properties (color, age, etc.) are used, the partition of A can be made finer and finer grained, i.e. more and more subsets are created. Each element of any given subset that constitutes the partition is associated with a label corresponding to that subset. Once this abstract partition is specified in terms of properties, the relevant question is, how can one determine the appropriate classification label of an observed object if its properties are known?

To answer this question, again abstractly, one can think pictorially in terms of trees and networks. A Decision Tree is one such visualization and it can be thought of as a succession of dependent decisions. That is, every decision made can be, though it need not be, dependent on all the prior decisions made. A graphical way to view such a partition is to think of a binary tree each node of which represents a decision testing for the presence or absence of some given property. See figure 1-1. An object is classified by traversing the tree starting at the root. Since the test at each node

is binary, either the left branch or the right branch of the tree is followed after the decision at any given node. When a leaf is reached, the object is classified with the label at the leaf.

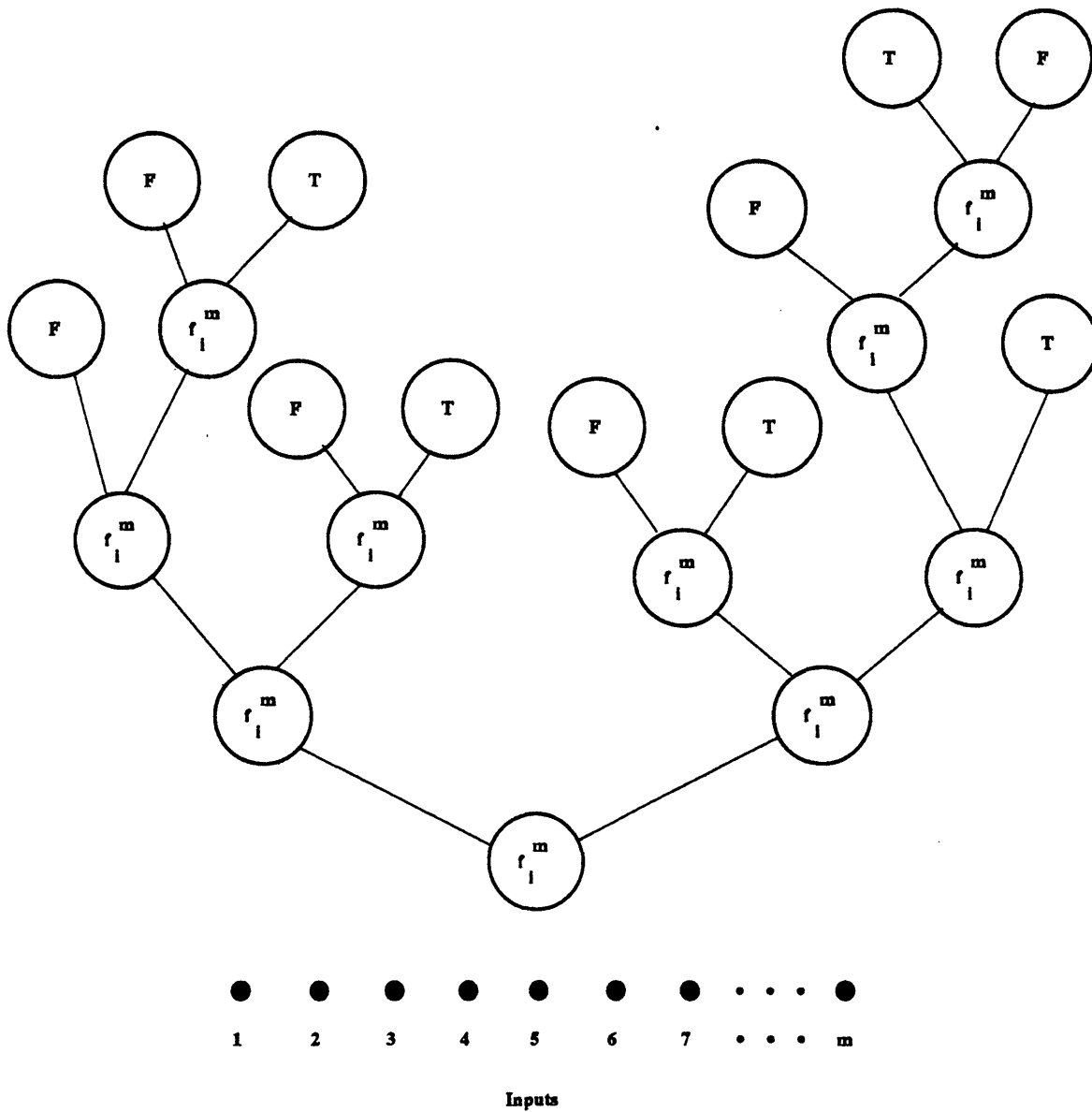


Figure 1-1: A Decision Tree

But a decision tree inherently specifies an order of importance on the properties, i.e. some properties are checked before others, which can make the classification independent of certain properties. Another structure, the feedforward network in

figure 1-2, can be used as a classifier where many decisions can be made in parallel. This structure is best viewed as a succession of layers, with each layer consisting of a set of nodes which make binary decisions. Here, in general, only nodes in the first layer make decisions similar "in meaning" to those in the Decision Tree whereas the rest make decisions about the previous layer's decisions. i.e. only the first layer nodes test for properties of the observation. The output of the first, which is the input to the next layer, is a binary vector representing decisions. Thus the layers after the first make decisions about these initial property decisions.

Given these different structures, it is natural to ask if an optimal classifier can be constructed. This involves, first of all, defining what "optimal" means. Then, the question of whether it is practical to construct or "find" this structure becomes important.

Breiman et al. give a procedure, the CART method (described in [1]), for growing and pruning a Decision Tree based on certain classification error rates. In effect an overly large tree is grown and then, using what is referred to as an honest estimate of the error rate, the most favorable pruned tree is selected out of a given parameterized set of possible trees. In [1] an analysis of a "hybrid" classifier is presented. The structure studied can be thought of as a composition of Decision Trees with Multi-Layered Perceptrons.

However, there are other ways to define and choose an optimal tree. Quinlan and Rivest [2] use the Minimum Description Length Principle (MDLP) in order to pick, or infer, their optimal, "best," tree. The use of the MDLP is elucidated when considering a communication game [2]. Given a set of data and the properties and classification label of each element in the set, the game is to transmit the classification information using the fewest number of bits. This is of course dependent on the encoding scheme, but given this, the MDLP can be thought of as choosing the tree that allows the most compact representation of all of the classification information. A particular tree has within its structure, information about the relationship of property values to classification. The MDLP requires choosing a tree which captures the most useful information.

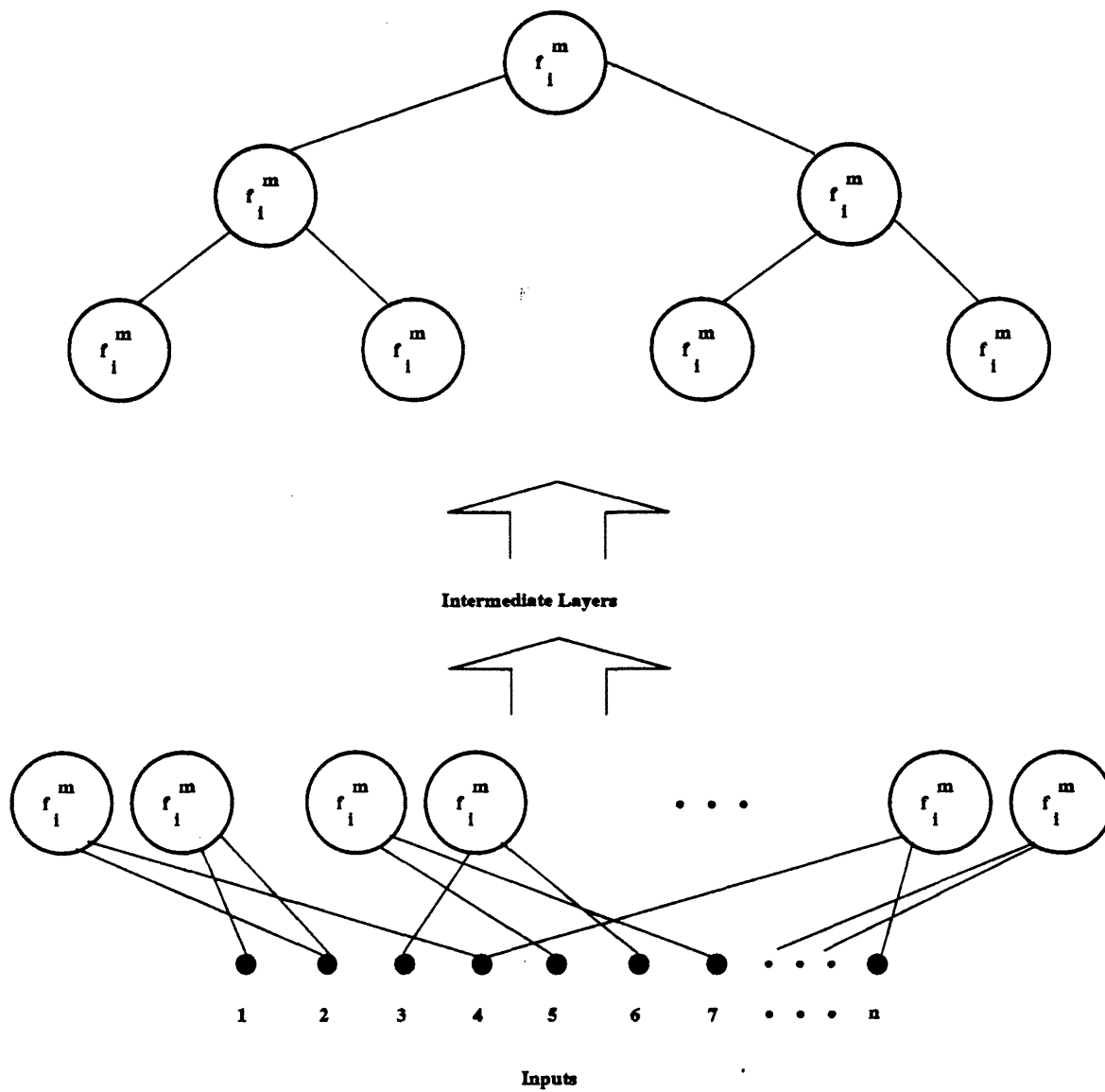


Figure 1-2: A Multi-Layered Perceptron

However, it is quite clear from the current research that Multi-Layered Perceptrons (MLPs), a class of feedforward networks, are important classifiers which have a significant relationship to Decision Trees. Sethi [3] has given a transformation from Decision Trees to MLPs. In this thesis, a proof that a transformation in the reverse direction exists is given. Hyafil and Rivest [4] have shown that the problem of finding an optimal, based on external path length, Decision Tree is NP-complete. This suggests that it is worthwhile studying the properties of MLPs to verify the educated guess that an analogous problem for MLPs is intractable as well.

The questions that this thesis will address have to do with determining how hard, in terms of computation, it is to find a minimal or in some sense "optimal" MLP classification structure. This thesis will approach the problem of defining and finding a minimal classifier by starting with Linear Threshold Functions. The capabilities of the non-linear version of such functions have been examined by Minsky and Papert [5]. (It must be noted that throughout most of this thesis, Perceptron will mean Linear Threshold Function, and not the generally more powerful structure presented in [5]) This thesis will present a different framework in which to study these functions that will allow the proof of some lower bounds on their capabilities.

Then, the linear threshold functions will be used as the building blocks of a classification structure called a Multi-Layered Perceptron. Various properties of the MLP will be defined. In particular minimality will be defined as the number of functions, or graph nodes, required to classify a given set of objects. For purposes of clarity, a boolean classification problem will be considered. Judd [6] has worked on the problem of learnability in networks and has shown that the general problem of deciding whether or not a given network architecture is consistent with a particular input-output task is NP-complete. One goal of this thesis is to extend this work to considerably reduce the architecture specification. In this way the thesis will deal with arbitrary architectures of a given size rather than any specific architecture.

1.1 Contributions of the Thesis

The main contributions of this Thesis fall into two categories.

Linear Separability The first category deals with Linear Threshold Functions. Here, a novel methodology is presented for determining the linear separability of Boolean functions. Dertouzos has, quite a while back, looked at this problem. The method presented here represents a different approach in that it makes use of a matrix formulation of the original problem and reduces it to a vector formulation.

Minimization of Networks The second part of the thesis will address the problem of finding a network with the fewest functional elements, of the *SAF* class, which computes a specified Boolean function. In particular it will be shown that finding a structure with this characteristic is intractable in that this procedure implicitly solves an NP-complete problem.

Chapter 2

Transformation Between an MLP and a Decision Tree

In studying to the issue of network classifiers it is informative to make a connection between the MLP and Decision Tree structures. It is shown that, under circumstances to be stated, any MLP structure can be Transformed into a Decision Tree. These results in conjunction with those of Sethi [3], who has given a construction in the reverse direction, establish a constructive equivalence between Decision Trees and MLPs. This is important because any results that apply to one structure may be applicable to the other.

2.1 MLPs to Decision Trees

Let the MLP have k layers.

Let layer i have m_i perceptrons.

Let v denote a vertex of a hypercube.

Let $b(v)$ be the binary coordinates of v .

Furthermore, let all Perceptrons have unlimited fanin.

A Decision Tree is a structure of the form given in Figure 1-1. The inputs can go to all of the perceptrons, but the process starts at the root node, and the output of the Perceptron at each node determines which of the two paths to take in traversing

the tree. Computation ends at a leaf, which classifies the input. In the computation of a boolean function, this classification is an element of $\{0, 1\}$.

Any MLP computing a Boolean function can be transformed into one with a partitioning layer, an AND layer, and an OR layer as follows:

The output of layer i , O_i , is an m component binary vector which takes on the values of the coordinates for the vertices of a unit hypercube, i.e. one that has coordinates given by a binary vector, in R^m : $O_i \in b(v_{m_i})$.

The input to layer $i = 1$, the partition layer, is a vector in R^n . Each subsequent layer has an input that is the output of a Perceptron, described above.

The partition layer, P_1 , then can be viewed as giving information about where the input lies in relation to a partition of the input space, with each Perceptron in the layer providing a boolean answer indicating the placement of the input vector in relation to its defining hyperplane.

Every perceptron in layers after the first, P_i , with $i > 1$, describes a set S_i , of vertices, those on one side of a hyperplane, of the unit hypercube in $R^{m_{i-1}}$. If the inequality of the perceptron is satisfied then $O_{i-1} \in S_i$.

Let $SV_{i-1,i}(S_v)$ be a set of vertices in $R^{m_{i-1}}$ described by a set of vertices S_v in R^{m_i} . Each binary vector in S_v , also to be referred to as a vertex, represents the output of each Perceptron in the i^{th} layer. If the output of P_i is 1 then this j^{th} Perceptron allows the set of vertices in $R_{m_{i-1}}$ that satisfy its inequality. Similarly, a 0 output means that the allowed set consists of the vertices in $R_{m_{i-1}}$ that do not satisfy the inequality. This set will be referred to as the complement of the allowed set when the output was 1. If a particular vector is at the output of layer i , then the allowed set for that vector is the intersection of the allowed sets for each constituent Perceptron determined by whether its output is 1 or 0 in the vector. And $SV_{i-1,i}(S_v)$ is the union over all the vectors in S_v of each vectors allowed set. Thus $SV_{i-1,i}(S_v) = \bigcup_{q=1}^{|S_v|} \bigcap_{l=1}^{m_i} f(S_{i_l})$ where (letting $v_q \in S_v$ be treated as a binary number, bin_q , where $P_{i_{m_i}}$ specifies the MSB) $f(S_{i_l}) = S_{i_l}$ if the integer part of $2^{-l+1} * bin_q$ is odd, and $f(S_{i_l}) = -S_{i_l}$ if the integer part of $2^{-l+1} * bin_q$ is even.

Layer k can be considered the classification layer. Each P_{k_j} will describe a set of vertices in R^{m_1} , i.e. a set of outputs of the partitioning layer:

$$S_{k-1}^{Tot_j} = S_{k_j}$$

$$S_{k-2}^{Tot_j} = SV_{k-2,k-1}(S_{k-1}^{Tot_j})$$

$$S_{k-3}^{Tot_j} = SV_{k-3,k-2}(S_{k-2}^{Tot_j})$$

...

$$S_1^{Tot_j} = SV_{1,2}(S_2^{Tot_j})$$

$S_1^{Tot_j}$ is a set of outputs from the partitioning layer of the perceptron. Each of these outputs constitutes a certain ANDing of the decisions of the constituent perceptrons. P_{k_j} in effect makes a decision as to whether a given input vector satisfies one of the AND conditions in $S_1^{Tot_j}$. Since all that the output perceptron does is to test whether an output of the partitioning layer is in its specified $S_1^{Tot_j}$, the MLP can be reduced to a perceptron with only two layers above the partitioning layer. The layer immediately above the partitioning layer will perform each of the AND operations in $S_1^{Tot_j}$ and the output layer above it will OR these together, which is what P_{k_j} does \square

It is important to note that this structural formulation is only possible if the fanin of the Perceptrons is not a priori restricted. If a restriction were to be put on the fanin of the constituent Perceptrons of the MLP, then it will not be possible for every MLP to have this structure. For example, consider the case where the satisfying vertices are in the intersection of 3 hyperplanes. If the restriction on the fanin is 2, then the function cannot be represented in this way because 3 hyperplanes must be ANDed together in the layer following the partitioning layer.

Once the MLP is in this structure, each of the ANDing layer nodes can be thought of as a leaf of a degenerate Decision Tree constructed as a long chain: Arrange the ANDing layer nodes in some way. Then start with the first node. If it is satisfied, then output TRUE or 1. If not, then check the next node in the order, and repeat. If all are unsatisfied, then output FALSE or 0.

2.1.1 Decision Trees to MLPs

As established previously a Decision Tree of the form in Figure 1-1 and a Multi-Layered Perceptron of the form in Figure 1-2 are equivalent in the sense that one can be transformed into the other in a straightforward manner. In these two figures n is the number of inputs and m represents the fanin of the computations. The illustrations take $m = 2$. Figure 1-2 has an arbitrary number of layers, but every node in each layer has fanin m . In general, as Sethi points out, when a Decision Tree is transformed into an MLP, the resulting structure will have 3 layers. The partitioning, ANDing, and ORing layers. In this case, all the nodes in the partitioning layer have a fanin of m , but the ANDing and ORing layers can have greater fanin.

In Sethi's mapping, the nodes in the partitioning layer have a special relationship to the nodes in the ANDing layer. In particular, each one of the partitioning Perceptrons has to be an input to every ANDing Perceptron corresponding to the leaves in the subtree when it is taken as the root.

2.2 Relevance

Much work [1] [2] has been devoted to the study of Decision Trees. And the results of this section together with [3] suggest a concrete relationship between Decision Trees and MLPs. This thesis then will be concerned with studying MLPs in the sequel.

Chapter 3

Some Results for Perceptrons (of order 1)

3.1 A Formalism for Discussion

A Perceptron [5] can be defined as a function

$$f_i^n(\varphi_1, \varphi_2, \varphi_3, \dots, \varphi_n) = g\left(\sum_{j=1}^n \alpha_j \varphi_j\right), R^n \mapsto \{0, 1\}$$

where the function $g(x) = 1$ if $x > \theta$ otherwise $g(x) = 0$. The action of a Perceptron is to determine on which side of a given hyperplane its input, which can be viewed as a vertex of a hypercube in n dimensions when the φ_i are boolean, resides. The Perceptron partitions the vertices of the hypercube into two linearly separable sets. Call the number of such partitions p_n . The equation of the hyperplane is $\sum_i \alpha_i \varphi_i = \theta$. Since there are a finite number of boolean functions of n variables, there is an infinite number of hyperplanes which yield a single function. In the definition of the Perceptron, $i > 0$ and $i \leq p_n$, thus it is an index to a particular linear separation in n dimensions and not to a particular hyperplane.

Let F be a boolean function of n variables, $\{0, 1\}^n \mapsto \{0, 1\}$, and F_{norm} be the "Disjunctive Normal Form," or the unique reduced OR of ANDs representation of F ,

i.e.

$$F_{norm} = OR(mask_1, mask_2, mask_3, \dots, mask_m)$$

where a group of variables ANDed together is referred to as a mask. Let

$$f_order = \max_{me\{set\ of\ masks\ in\ F_{norm}\}} (cardinality\ of\ the\ support\ of\ mask_m)$$

A measure of the complexity of the function F is the concept of order developed by Minsky and Papert [5]. It is a property of the function F which is elucidated by a Perceptron representation. F can be represented in terms of Perceptrons, e.g.

$$F_k = f_i^m(mask_1, mask_2, mask_3, \dots, mask_m)$$

, where the subscript k is used to indicate that the set of masks and function f_i^m are not unique. The *order* of a function F is

$$\min_k \left(\max_{me\{set\ of\ masks\ in\ F_i\}} (cardinality\ of\ the\ support\ of\ mask_m) \right)$$

3.2 A Different Approach

Presented here is a Linear Algebraic formulation of the problem of determining the separation capability of a single Linear Threshold Function (LTF). As mentioned previously, these are special cases of the Perceptrons, those of order 1. The result of this section will be a precise methodology for determining whether a single LTF can compute a given boolean function.

3.3 A Linear Algebraic Formulation

Consider a Boolean function of n variables and let:

$$A = \left[\begin{array}{c} \vdots \\ u_i \\ \vdots \end{array} \right] \left. \vphantom{\begin{array}{c} \vdots \\ u_i \\ \vdots \end{array}} \right\} 2^n, u_i \in \{0, 1\}^n$$

$\underbrace{\hspace{10em}}_n$

and $\cup_{i=1}^{2^n} u_i = \{\text{all sequences } \{0, 1\}^n\}$, further, let the first n rows of $A = I_n$

$$Y = \left[\begin{array}{c} \dots \\ \dots \quad y_i \quad \dots \\ \dots \end{array} \right] \left. \vphantom{\begin{array}{c} \dots \\ \dots \quad y_i \quad \dots \\ \dots \end{array}} \right\} 2^n, \text{span}\{y_i\} = \eta(A^T G^T)$$

$\underbrace{\hspace{10em}}_{2^n - n}$

and Y is chosen such that all the elements are either -1 , 0 , or 1 .

$$G = \left[\begin{array}{cccccc} \pm 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & \pm 1 & 0 & 0 & \dots & 0 \\ \cdot & & \cdot & & & \\ \cdot & & & \cdot & & \\ \cdot & & & & \cdot & \\ 0 & \dots & 0 & \pm 1 & & \end{array} \right] \left. \vphantom{\begin{array}{cccccc} \pm 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & \pm 1 & 0 & 0 & \dots & 0 \\ \cdot & & \cdot & & & \\ \cdot & & & \cdot & & \\ \cdot & & & & \cdot & \\ 0 & \dots & 0 & \pm 1 & & \end{array}} \right\} 2^n$$

G is a premultiplier for A , where a -1 in the i^{th} row indicates that the vertex given by the binary vector u_i is "chosen." Thus G describes the group of vertices, or assignment of boolean variables, that should satisfy a given boolean equation.

If there are two groups G_1 and G_2 , possibly choosing some of the same vertices, then the premultiplier becomes:

$$G = \left[\frac{1}{2}(G_1 - G_2) \right]^2 + \frac{1}{2}(G_1 + G_2)$$

This process can be iterated for more groups.

If, on the other hand, the G_i are restricted to choose different vertices, then:

$$G = \prod G_i$$

$$\text{Let } \bar{1} = \underbrace{[11 \cdots 1]}_{2^n}^T$$

Definition A Boolean function of n bits is *linearly separable* if the vertices of the binary hypercube where it evaluates to TRUE are separated by an $n - 1$ dimensional plane from those that evaluate to FALSE. i.e. if it is computable by an LTF.

Lemma 1 *The Boolean function described by G , $n \times n$, is linearly separable iff: $\exists \bar{w}$, and θ such that:*

$$GA\bar{w} > \text{sgn}(\theta) |\theta| G\bar{1}$$

Proof: Each row of the above inequality is an LTF constraint, one for each binary vertex in n dimensions. A -1 in the i^{th} row of G simply reverses the inequality in the LTF constraint.

(\implies) If the matrix inequality can be satisfied with \bar{w} and θ , then the hyperplane that separates the vertices is given by $\bar{x} \cdot \bar{w} = \theta$.

(\impliedby) Let $\bar{x} \cdot \bar{w}' = \theta'$ be the hyperplane that separates the Boolean function. Then \bar{w}' and θ' will satisfy the matrix inequality \square

Theorem 1 *The Boolean function given by G , $n \times n$, is linearly separable iff:*

$\exists \bar{w}$, and θ such that:

$$-\text{sgn}(\theta)Y^T G\bar{1} > Y'^T \bar{v}_1$$

for some $\bar{v}_1 > 0$. Y' and \bar{v}_1 are the top n rows of Y and \bar{v} respectively.

Proof:

From Lemma 1 the Boolean function given by G , $n \times n$, is linearly separable iff:

$$GA\bar{w}' = \text{sgn}(\theta)G\bar{1} + \bar{v}$$

for some $\bar{v} > 0$, which by the Theorem of the Alternative, has a solution iff:

$$Y^T(\text{sgn}(\theta)G\bar{1} + \bar{v}) = 0, \text{ for some } \bar{v} > 0 \quad (3.1)$$

Since A has the special structure described previously, $Y = [Y'^T | I_{2^n-n}]^T$ where all the elements are either -1 , 0 , or 1 . Then equation 3.1 becomes:

$$Y^T(\text{sgn}(\theta)G\bar{1} + \bar{v}) = 0$$

$$Y^T\bar{v} = -\text{sgn}(\theta)Y^TG\bar{1}$$

$$[Y'^T | I_{2^n-n}]\bar{v} = -\text{sgn}(\theta)Y^TG\bar{1} \quad (3.2)$$

Let $\bar{v} = [\bar{v}_1^T \bar{v}_2^T]^T$ where $\bar{v}_1^T = [v_1, \dots, v_n]$, $\bar{v}_2^T = [v_{N+1}, \dots, v_{2^n}]$, and $\bar{v} = [v_1, \dots, v_{2^n}]$.

Then from equation 3.2,

$$Y'^T\bar{v}_1 + I_{2^n-n}\bar{v}_2 = -\text{sgn}(\theta)Y^TG\bar{1}$$

$$I_{2^n-n}\bar{v}_2 = -(\text{sgn}(\theta)Y^TG\bar{1} + Y'^T\bar{v}_1)$$

$$\bar{v}_2 = -(\text{sgn}(\theta)Y^TG\bar{1} + Y'^T\bar{v}_1)$$

choose $\bar{v}_1 =$ to some positive vector, then the solution is appropriate if $\bar{v}_2 > 0$, or if

$$-\text{sgn}(\theta)Y^TG\bar{1} > Y'^T\bar{v}_1$$

□

3.3.1 An Example of the use of Theorem 1

Consider the vector $\bar{0}$ and define $H(1) = \{\text{vectors of Hamming distance 1 from } \bar{0}\}$.

Claim Any Boolean Function that evaluates to TRUE for $\bar{0}$ and every $v_i \in h(1)$, where $h(1) \subset H(1)$, and to FALSE otherwise is computable by an LTF.

Proof:

Condition from Theorem 1 :

$$-sgn(\theta)Y^T G \bar{1} > Y'^T \bar{v}_1$$

for some $\bar{v}_1 > 0$.

With G restricted as above, $Y^T G =$ a constant matrix: $(GA)^T$ can have negative components only in the first n columns, which pick out only values of magnitude 1 from Y . These multiply Y' in the null space expression. Since the rest of Y only chooses a value of 1 from GA all that is necessary is that every row in Y corresponding to a chosen vertex should have the opposite sign of that row in $G \implies GY =$ a constant matrix $\implies Y^T G =$ a constant matrix.

In particular, $Y^T G = [G'^T | I_{2^n - n}]$, where G'^T has only 0's and -1's in all rows. So it can be deduced that $Y^T G \bar{1}$ is a vector with all negative components. So in the equation

$$-sgn(\theta)Y^T G \bar{1} > Y'^T \bar{v}_1$$

$sgn(\theta)$ and \bar{v}_1 can always be chosen so that the equation is satisfied \implies all G' 's restricted as above are separable.

Chapter 4

Some Results for Multi-Layered Perceptrons (of order 1)

A Multi-Layer Perceptron (MLP) will be defined as a structure that has an arbitrary number of layers, each layer being composed of an arbitrary number of Perceptrons. To expand the previous notation, the expression

$$F = f_3^2(f_1^m(\alpha_1, \dots, \alpha_m), f_5^2(f_8^m(\alpha_1, \dots, \alpha_m), f_7^m(\alpha_1, \dots, \alpha_m)))$$

describes an MLP.

Let G_F be directed graph representation of F where the operations are the nodes, and where each of the node operation's operands specify an edge from that operand, which may be another node or a variable, to the node. e.g. if

$$F = f_3^2(f_1^m(\alpha_1, \dots, \alpha_m), f_5^2(f_8^m(\alpha_1, \dots, \alpha_m), f_7^m(\alpha_1, \dots, \alpha_m)))$$

as before, then G is the directed graph of Figure 4-1.

4.1 Changing the Perceptron Function

The threshold function for each Perceptron need not always be the step function. When sigmoid type thresholds are used instead of the step function, the propagation

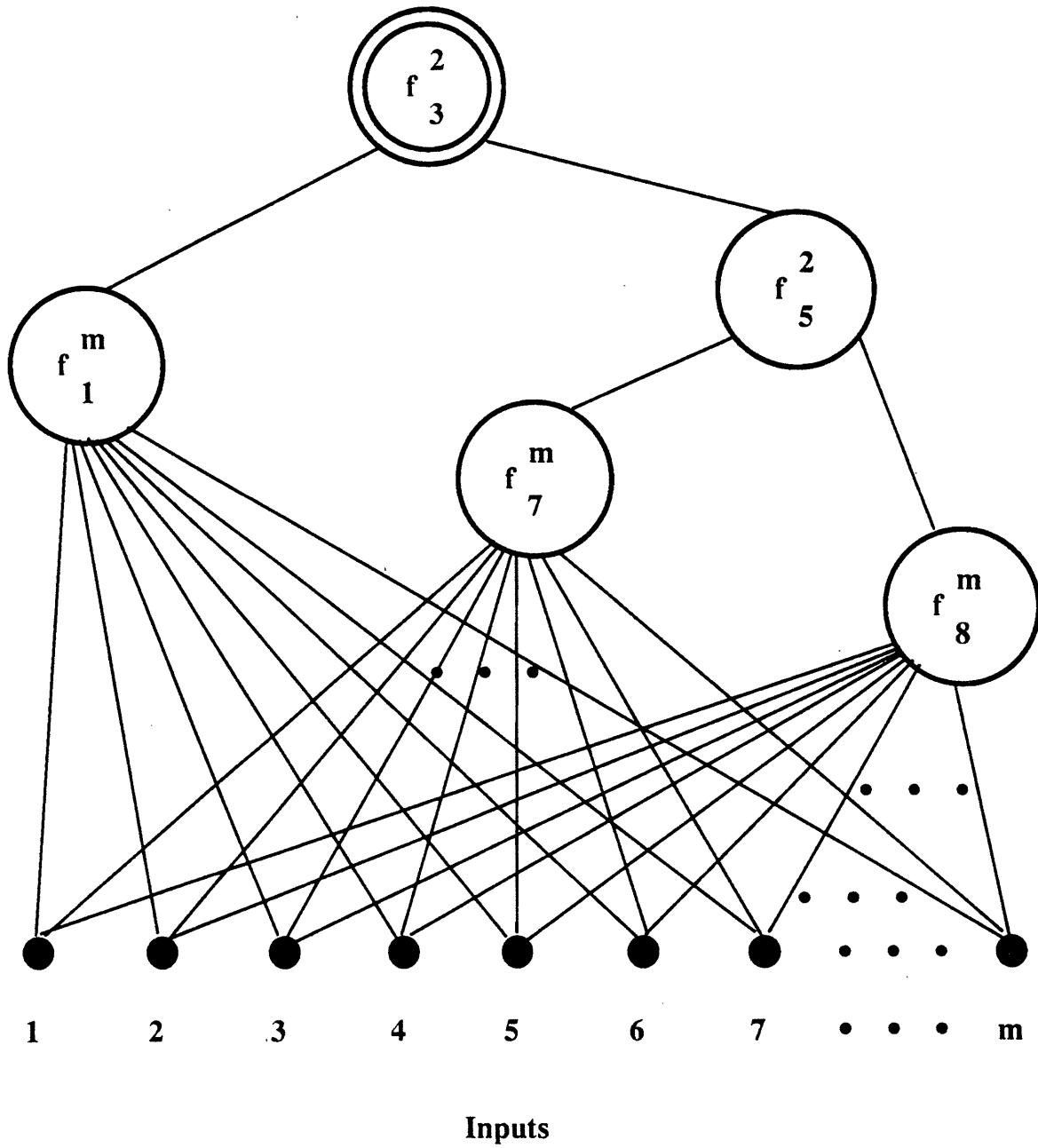


Figure 4-1: A Directed Graph Representation of a Boolean Function

of output values through a network created with these functions will result in different output values even when those structures were equivalent with the step threshold.

One way to think about the threshold function is to view it as a convolution of the step discontinuity with a smoothing function, e.g. a sigmoid is the convolution of a step with a Gaussian [7]. Denote the step function by $u(x)$ and the smoothing function by $g(x)$. Then the threshold, centered around zero, is given by $t(x) = u(x) * g(x)$.

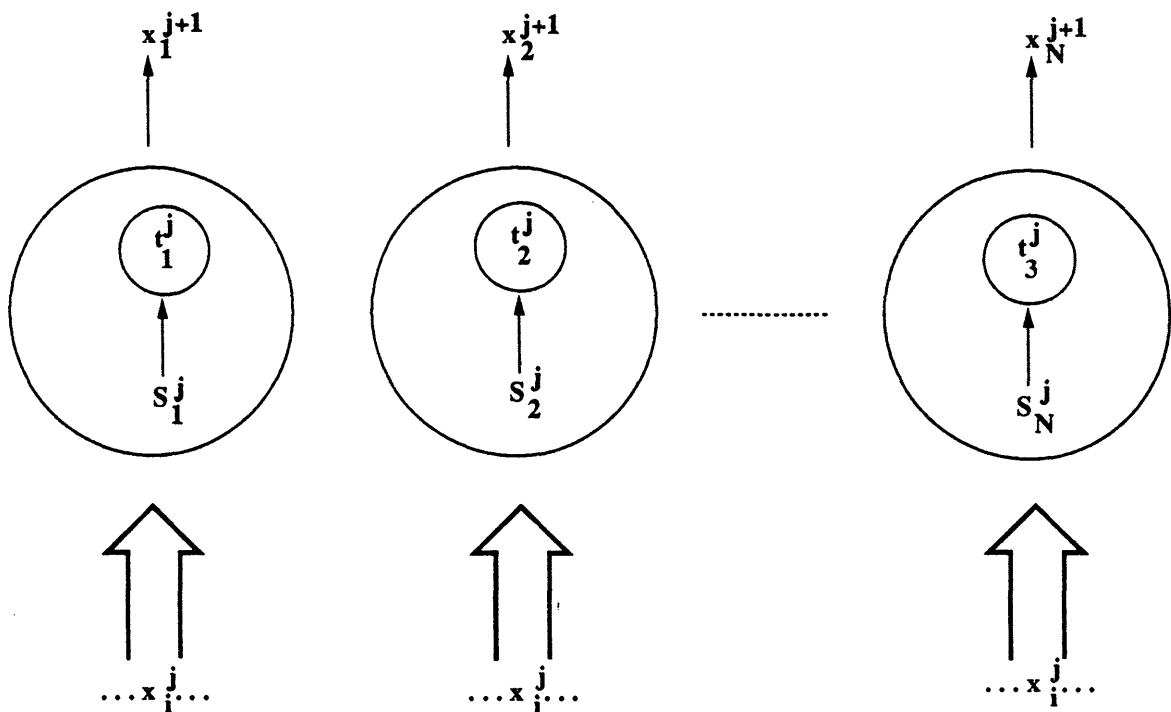


Figure 4-2: An MLP Layer

The output of a perceptron can be described as a sample of the above convolution sum taken at $x = \sum_i \alpha_i \varphi_i - \theta$. Figure 4-2 is a schematic representation of layer j in a given MLP. The x_i^j are the inputs to the j^{th} layer and the x_i^{j+1} are its outputs. $S_i^j = \sum_k \alpha_{k,i}^j x_k^j$ is the weighted sum of the perceptron inputs. $S_i^j - t_i^j$ is the value where the convolution sum is sampled to get the output of the i^{th} perceptron in the j^{th} layer.

Figure 4-3 is a description of the propagation of inputs to one layer, the j^{th} , through a perceptron in the next layer, the $j + 1^{\text{st}}$. The i^{th} impulse in $i(x)$ is located

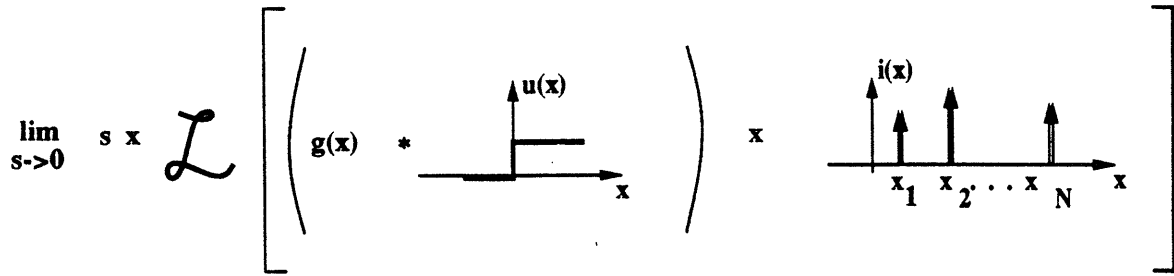


Figure 4-3: Operation of One MLP Layer

at the combination of inputs and threshold for the i^{th} perceptron in the j^{th} layer, and its height is equal to the coefficient multiplier for the i^{th} input in the perceptron in the next layer. Figure 4-3 makes use of the Final Value Theorem for Laplace Transforms: the final value of the x domain operations shown is the output of the perceptron in the $j + 1^{st}$ layer.

4.2 Novel Complexity Measures

To gain a better understanding of the computation processes, it is necessary to look at the complexity of the structure, or the directed graph G_F , as well as that of the boolean function itself.

As stated earlier, the representation of F in terms of nested Perceptrons is not unique. And in fact, one measure of the complexity of the boolean function F is the number of ways that it can be represented as an MLP. This idea can be connected to that of weight space volume described in [8]. One way to view weight space volume is to think of it as the size of the set of weight specifications that compute a particular function on a particular MLP architecture. The idea presented here measures the *number of architectures* rather than the particular weight specifications. In this way it is hoped that a more informative notion will emerge.

But perhaps a more informative measure of this structural complexity, which will be called the *path complexity*, is given by analyzing the number of paths over which a particular input node has influence. The idea is more clearly seen from G_F . In

Figure 4-1, every input takes three paths to the output node. This information is also present in the representation of F as an expression of nested Perceptrons.

$$C_{path} = \min_{MLP_i(F)} (\max_{input\ nodes} (\# \text{ of paths to output}))$$

This is a measure of the distributedness of the computation process in that it gives an idea of the flow of information in a computation. It is hoped that in this formalism will be captured the salient features of local computation.

The complexity measure of interest in this thesis however, is *size*. The *size* of an MLP will be taken to be the number of nodes in its directed graph representation. This choice is motivated by [4], where the *number of decisions* was an important measure. In the MLP context, the number of nodes is a fundamental quantity because it is equal to the number of partition tests (hyperplane tests) plus the number of decisions that need to be made on the results of those tests to compute a given function.

Here the relevant question will be: "How easy is it to determine the minimum size MLP for a boolean classification task?" And, the structure of interest is the MLP with LTFs in the nodes.

4.3 Finding A Minimal Representation

Having established the results on linear separability, it is now possible to provide a method, albeit brute force, to enumerate the possible MLP structures for any given boolean function. Also if the number of layers ≥ 2 is specified in advance, it is possible to find the minimal MLP for a given boolean function. An MLP will be said to be minimal if it has the fewest number of linear separation operations.

Given a boolean function F of n variables, the first step is to find the linearly separable (l.s.) sets of vertices for an n dimensional hypercube. $LS(n) =$ the number of linearly separable sets of vertices of an n dimensional hypercube. This is the critical step that is made possible by the approach outlined earlier. Call the set of l.s. sets for n dimensions S_1 (the subscript signifies the layer) and let $|S_1| = n_{S_1}$. Then $s_1 \in S_1$ can be separated in the partition layer.

Define S_2 to be a set of sets such that $s_2 \in S_2$ is a set of vertices separated by performing an l.s. operation on an arbitrary number of outputs of the l.s. functions described by S_1 . Then

$$|S_2| = \sum_{i=1}^{n_{S_1}} \binom{n_{S_1}}{i} LS(i)$$

This counts all of the possibilities, possibly with repetition due to degeneracies. Note that for this step the l.s. sets of a $|S_1|$ dimensional hypercube are required, but these can be obtained as stated earlier. Then S_3, S_4 , etc. can be defined in the same way:

$$|S_k| = \sum_{i=1}^{n_{S_{k-1}}} \binom{n_{S_{k-1}}}{i} LS(i)$$

Each function in any of the sets, e.g. $s_k \in S_k$, describes a boolean function by giving a set of hypercube vertices, in the input space, that are "accepted." The minimal MLP for a given boolean function is the first function found that accepts the right set where the search is started with the function that has the smallest number of units in S_1 . When S_1 is used up, the search is continued in S_2 starting with the function that uses the fewest elements, or linear separations, total (counting the elements in

all the previous layers). This process is continued until the result is obtained. If a restriction is placed on the number of layers ahead of time, then the search can be terminated after all of the structures in that many layers are tested.

4.4 A Second Method to Find A Minimal Representation

If the minimal representation is desired, without a restriction on the number of layers, then one can simply consider in succession all MLP's with 1 unit, then 2 units, then 3 ... until a structure is found that represents the Boolean function of interest. A minimal representation is thus guaranteed.

4.5 More Than Two Layers

Since every function can be computed in two layers, given the structures described here, adding more layers does not broaden the class of boolean functions that are computable. Each addition of a layer though, does increase the number of structures that are possible. So the density of structures for computing any particular boolean function is increasing with the number of layers beyond two. This is relevant in the case where boolean functions are being learned by a net. It suggests that using a greater number of layers will increase the probability that the right function will be learned on the correct architecture. Also robustness and generalization may be increased.

4.6 Complexity of Reduction

In this section the tractability, or computational complexity, of finding minimal MLPs will be studied. In [6] Judd has looked at the problem of determining whether or not a specific architecture can compute a given task. Here, his results are extended to reduce the specification of the architecture.

4.6.1 Judd's Definitions and Results

$$PERF(SAF_{n,s}) = \{ \langle A, T \rangle : \exists F \in SAF_{n,s}^n : T \text{ is consistent with } M_F^A \}$$

A is an *architecture* which is a 5-tuple $A = (P, V, S, R, E)$ where

P is a set of *posts*,

V is a set of n *nodes*, into which the functions will be loaded,

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

R is a set of r *output posts*: a subset of P , and

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

Also, let the set of *input posts to node* v_k be denoted $p(v_k) = v_j : (v_j, v_k) \in E$.

A *task* T is a set of pairs of binary inputs and responses, where some bits in the response may not be significant.

F is a function configuration of $SAF_{n,s}$ that is loaded in the architecture.

$SAF_{n,s}$ are single-AND functions where inverters can be placed on any number of the inputs or output.

M_F^A is a mapping defined when the architecture A is loaded with the functions in configuration F .

T is consistent with $f = M_F^A$ if it is a subset of $\{(\sigma, \rho) : f(\sigma) \text{ agrees with } \rho \text{ on the significant bits}\}$.

Theorem 2 $PERF(SAF_{n,s})$ is NP-complete.

A proof can be found in [6].

4.7 Extensions

In this section, by extending Judd's work, it is shown that the more general language $COMP(SAF_{ns})$ is NP-complete, which suggests that the problem of finding minimal architectures is intractable.

$COMP(SAF_{ns}) = \{ \langle S, \{l_i\}, T \rangle : \exists \text{ an architecture } A \text{ and } F \in SAF_{ns}^n \text{ where } A \text{ has } S \text{ function nodes with } l_i \text{ nodes in layer } i \text{ and where the inputs to layer } i = \text{ the outputs of layer } i - 1, \text{ such that } T \text{ is contained in } M_F^A \}$

Theorem 3 $COMP(SAF_{ns})$ is NP-complete.

Issues in Proof:

Judd shows that SAT can be "easily" (i.e. in polynomial time) transformed to $PERF(SAF_{ns})$. However, he has knowledge of the connections between nodes in the architecture. In transforming SAT to $COMP(SAF_{ns})$ no architecture connections can be specified. To account for this lack of connection specification, T is augmented with i/o pairs that enforce independence.

Proof

1. $COMP(SAF_{ns}) \in NP$:

Non-deterministically guess an architecture of size S (there are a finite number) and then resort to the fact that $PERF_{SAF_{ns}}$ is NP-complete.

2. $SAT \leq_p COMP(SAF_{ns})$ (\leq_p means polynomial time reducible to):

Let $\Phi =$ an instance of SAT with w variables and m clauses

Construct T and choose S , and $\{l_i\}$ in polynomial time as follows:

Let $\gamma = \bar{0}$ where the length of $\gamma = w$ and each bit in γ corresponds to a variable in Φ arranged in some order.

For each clause, i , in Φ construct:

$\gamma_i = \gamma + \sum_{i: i^{th} \text{ variable} \in \text{clause}} \text{vector with } i^{th} \text{ bit set if complemented in clause}$

$\gamma_{ijo} = \gamma_i \text{ with } j^{th} \text{ clause bit flipped}$

$\gamma_{ij1} = \gamma_i$ with j^{th} clause bit and all non - clause bits flipped

$\gamma_i^s = \gamma_i \cup \gamma_i$ with all non - clause bits flipped

$$\alpha_i^{00} = \{0^{i-1} \cdot 0 \cdot 0^{w-i}\}$$

$$\alpha_i^{01} = \{1^{i-1} \cdot 0 \cdot 1^{w-i}\}$$

$$\alpha_i^{10} = \{0^{i-1} \cdot 1 \cdot 0^{w-i}\}$$

$$\alpha_i^{11} = \{1^{i-1} \cdot 1 \cdot 1^{w-i}\}$$

From the construction, let $V = V_1 \cup V_2$ where V_1 are the w nodes that correspond to the first w bits of the output in T_1 and V_2 are the last m nodes in that output. V_1 and V_2 are layers 1 and 2 respectively. Then let the functions assigned to the nodes be referenced as $f_{i,j}$ for the j^{th} node in $V_i = V_{i,j}$.

$$T_1 = \{(0 \cdot \alpha_i^{jk}, *^{i-1} \cdot j \cdot *^{w+m-i}) : 1 \leq i \leq w, \text{ and } j, k \in \{0, 1\}\}$$

$$T_{2a} = \{(0 \cdot \gamma_i', *^w \cdot *^{i-1} \cdot 0 \cdot *^{m-i}) : \gamma_i' \in \gamma_i^s \text{ for each clause } i\}$$

$$T_{2b} = \{(0 \cdot \gamma_{ijk}, *^w \cdot *^{i-1} \cdot 1 \cdot *^{m-i}) : 1 \leq j \leq |\text{clause } i| \text{ and } k \in \{0, 1\} \text{ for each clause } i\}$$

$$T_2 = T_{2a} \cup T_{2b}$$

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

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

$$S = w + m$$

$$l_1 = w \text{ and } l_2 = m$$

The task given here is an augmented version of the task used by Judd to show NP-completeness of $PERF(SAF_{ns})$. The extension allows the exclusion of a specific architecture and permits the question: "Can this function be computed on an architecture with L layers and of size S ?" rather than "Can *this architecture* compute this function?"

It must now be shown that $\exists \Omega : \Phi \text{ is satisfied} \iff \langle S, \{l_1, l_2\}, T \rangle \in COMP(SAF_{ns})$

where Ω is the string of 0's and 1's corresponding to a satisfying assignment for Φ .

$$\exists \Omega : \Phi \text{ is satisfied} \implies \langle S, \{l_1, l_2\}, T \rangle \in COMP(SAF_{n_s})$$

This is evident from Judd's construction: The idea is to let the first layer nodes represent the variables of Φ and the second layer nodes, the clauses of Φ . Then load the i^{th} node in the first layer with the SAF

$$f(x \cdot y) = \begin{cases} y & \text{if } x = 0 \\ \Omega \langle i \rangle & \text{if } x = 1 \end{cases}$$

The i^{th} node in the second row should be loaded with the SAF

$$f(\bar{x}) = \begin{cases} 0 & \text{if } \bar{x} = \text{the concatenation of the bits of } \gamma_i \text{ that correspond} \\ & \text{to variables in clause } i, \\ 1 & \text{otherwise.} \end{cases}$$

The nodes in the first layer will have two inputs, a variable input and a solution input. When the solution input, x above, is set to 0 the output of the first layer nodes will be the variable value. So, T_1 will be consistent with the mapping induced by the above functions.

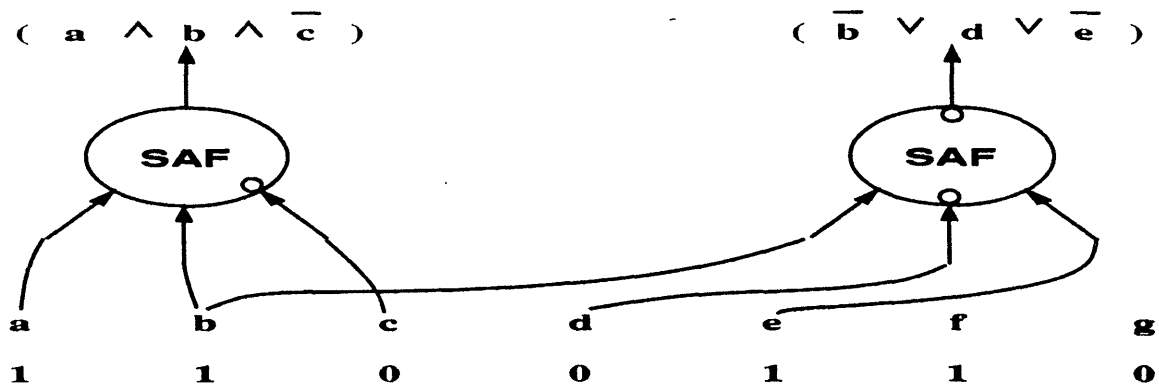
This being the case, by definition the nodes in the second layer, whose inputs will be taken to be the first layer outputs that correspond to the variables in the corresponding clause, will have output equal to 0 when the solution bit is set to 0 and the other inputs (variables) are set such that the corresponding clause is not satisfied. And so, T_2 is consistent with the induced mapping.

Since there exists an Ω that satisfies Φ , then there must exist outputs of the first layer such that the second layer nodes all have 1 as their output. Thus T_3 is consistent. Note that the satisfying assignment is in no way specified because the first layer nodes could be either in the AND or the OR configuration.

$$\langle S, \{l_1, l_2\}, T \rangle \in COMP(SAF_{n_s}) \implies \exists \Omega : \Phi \text{ is satisfied}$$

This is true because the basic idea behind the extension is that now T_1 and $T_2 = T_{2a} \cup T_{2b}$ actually check that the architecture is correct, thus eliminating the need for its

complete specification.



A 1 output should stay 1 if any noninput bits are flipped.

A 0 output should stay 0 if any noninput bits are flipped.

Figure 4.7 shows the two possible configurations for an *SAF*. In the AND configuration, if the output of the node is a 1, then flipping any combination of bits which are not inputs to the node will keep the output at 1. And flipping any combination of bits which include inputs to the node will change the output to 1. A similar argument can be made for the OR configuration with a 0 output initially.

So if it is enforced in the task that given the output of a certain node in layer i is a 1, it should remain a 1 when certain output bits in layer $i - 1$ are flipped and also that given the output is a 0 it should remain a 0 when the same bits are flipped, then the output of that node is independent of those bits and they are not inputs to the node. This is the fundamental idea that allows the reduction in specification of the architecture.

Using this idea T_1 makes sure that the first layer nodes are properly connected such that when a particular "solution" bit in the input string, not corresponding to a variable in Φ , is set to 0, then the output of the i^{th} node in the first layer is exactly the i^{th} input variable. When the solution bit is set to 1, the outputs of the first layer are unspecified.

Using this information then, T_2 completely specifies the behavior of each clause in Φ : T_{2a} checks that the inputs corresponding to variables in clause i are the only ones that affect the output of $V_{2,i}$. If a clause is not satisfied, flipping a variable value in it will make it so. So T_{2a} checks that flipping bits corresponding to variables not in the i^{th} clause keeps the output of $V_{2,i}$ 0 given that it was initially 0. T_{2b} checks that the inputs to node $V_{2,i}$ have the correct complementation according to clause i , i.e. given that the output of $V_{2,i}$ is 0, it checks that flipping a bit corresponding to a variable in clause i changes the output of $V_{2,i}$ to 1.

$T_3 \implies \exists$ assignment of inputs of the layer 2 nodes that set all the outputs to 1 $\implies \exists \Omega$ because from the definition of $COMP(SAF_{ns})$ only the first layer outputs affect the second layer \square

Note The fanin of the nodes in the MLP may be restricted, in the above language, to be equal to either 2 or the maximum number of variables in a clause, whichever is greater, without affecting the validity of the theorem. In fact, since 3SAT is NP-complete, the theorem will hold even if the fanin were restricted to 3.

The following example, adapted from Judd [6], illustrates the idea of the proof. Let $\Phi = (\overline{u_1} \vee u_2 \vee u_3) \wedge (u_2 \vee \overline{u_3} \vee \overline{u_4})$ be a candidate element of the SAT language. Then construct an instance of $COMP(SAF_{ns})$ as follows:

Let $\gamma = 0000$ where each bit in γ corresponds to $u_1 u_2 u_3$ and u_4 respectively.

$$\gamma_1 = 1000$$

$$\gamma_2 = 0011$$

$$\gamma_{110} = 0000$$

$$\gamma_{120} = 1100$$

$$\gamma_{130} = 1010$$

$$\gamma_{210} = 0111$$

$$\gamma_{220} = 0001$$

$$\gamma_{230} = 0010$$

$$\gamma_{111} = 0001$$

$$\gamma_{121} = 1101$$

$$\gamma_{131} = 1011$$

$$\gamma_{211} = 1111$$

$$\gamma_{221} = 1001$$

$$\gamma_{231} = 1010$$

$$\gamma_1^s = \{1000, 1001\}$$

$$\gamma_2^s = \{0011, 1011\}$$

$$\alpha_i^{00} = \{0^{i-1} \cdot 0 \cdot 0^{w-i}\}$$

$$\alpha_i^{01} = \{1^{i-1} \cdot 0 \cdot 1^{w-i}\}$$

$$\alpha_i^{10} = \{0^{i-1} \cdot 1 \cdot 0^{w-i}\}$$

$$\alpha_i^{11} = \{1^{i-1} \cdot 1 \cdot 1^{w-i}\}$$

$$T_1 = \{(00000, 0 * * * * *), (00111, 0 * * * * *), (01000, 1 * * * * *), \\ (01111, 1 * * * * *), (00000, * 0 * * * * *), (01011, * 0 * * * * *), \\ (00100, * 1 * * * * *), (01111, * 1 * * * * *), (00000, * * 0 * * * *), \\ (01101, * * 0 * * * *), (00010, * * 1 * * * *), (01111, * * 1 * * * *), \\ (00000, * * * 0 * * *), (01110, * * * 0 * * *), (00001, * * * 1 * * *), \\ (01111, * * * 1 * * *)\}$$

$$T_{2a} = \{(01000, * * * * 0 *), (01001, * * * * 0 *), (00011, * * * * * 0), \\ (01011, * * * * * 0)\}$$

$$T_{2b} = \{(00000, * * * * 1 *), (01100, * * * * 1 *), (01010, * * * * 1 *), \\ (00111, * * * * * 0), (00001, * * * * * 0), (00010, * * * * * 0), \\ (00001, * * * * 1 *), (01101, * * * * 1 *), (01011, * * * * 1 *), \\ (01111, * * * * * 0), (01001, * * * * * 0), (01010, * * * * * 0)\}$$

$$T_2 = T_{2a} \cup T_{2b}$$

$$T_3 = \{10000, * * * * 11\}$$

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

The instance of $COMP(SAF_{ns})$ is $\langle 6, 2, \{4, 2\}, T \rangle$ which only specifies what is shown in figure 4-4 for the architecture. No inter-layer connections are given, with the result that the significant specification is the size.

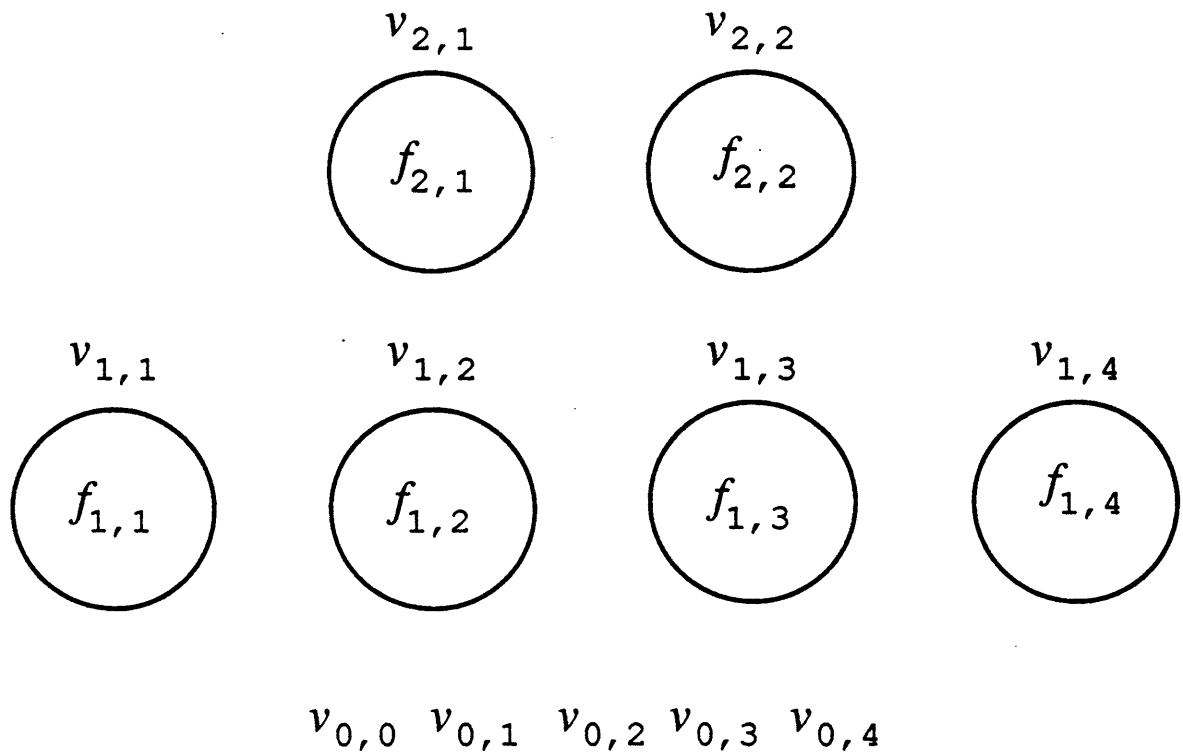


Figure 4-4: A minimal specification

$$\exists \Omega : \Phi \text{ is satisfied} \implies \langle 6, 2, \{4, 2\}, T \rangle \in COMP(SAF_{ns})$$

That this is true can be seen from figure 4-5, where the first and second layer functions are as defined previously in Judd's construction.

$$\langle S, L, \{l_i\}, T \rangle \in COMP(SAF_{ns}) \implies \exists \Omega : \Phi \text{ is satisfied}$$

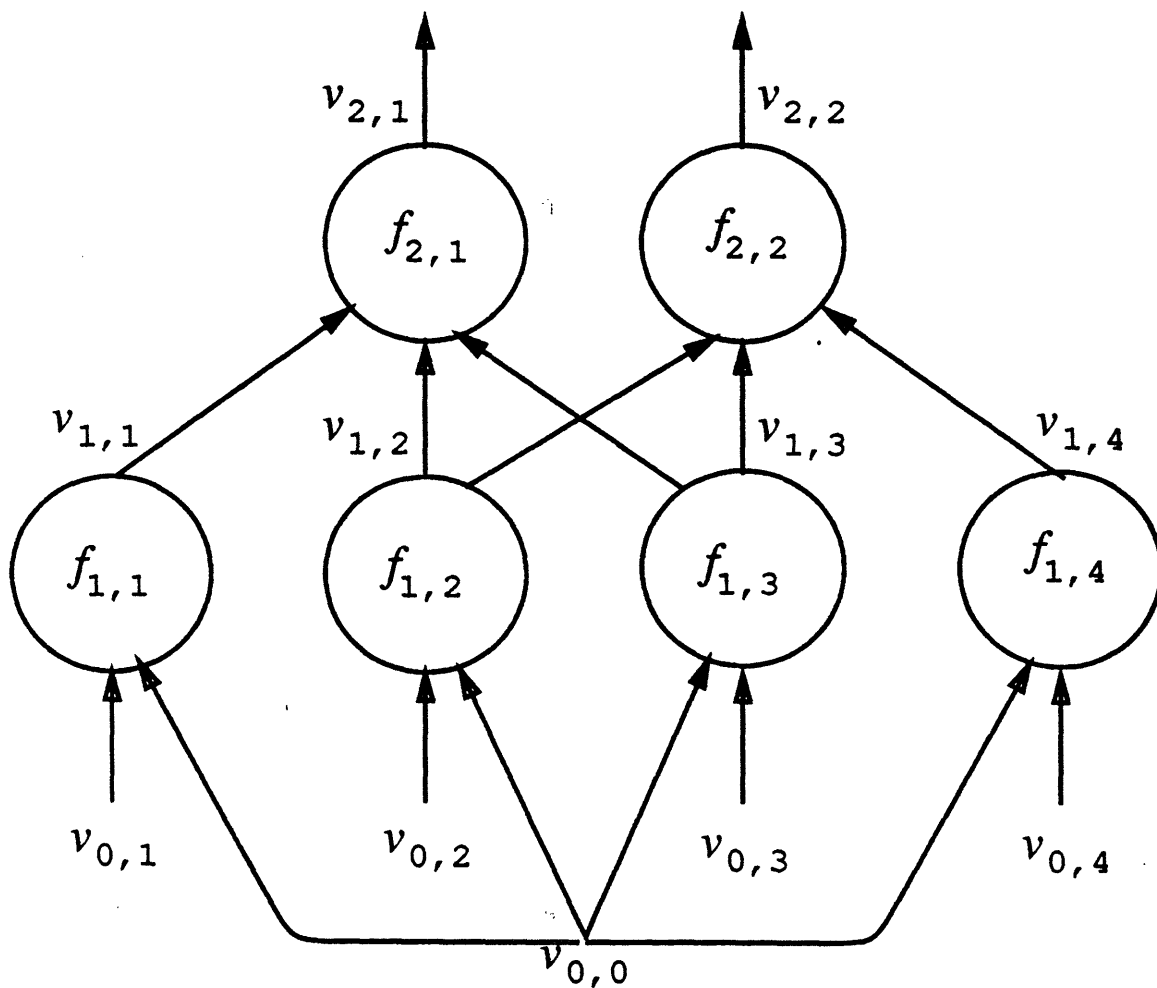


Figure 4-5: Judd's Architecture (computes the constructed task)

Studying the task reveals this to be true. $T_3 \implies \exists$ an assignment to the outputs of layer 1 such that all the layer 2 nodes, i.e. all the clauses, have value 1.

4.7.1 Further Extensions

$COMPS(SAF_{n,s}) = \{ \langle S, \{l_i\}, T \rangle : \exists \text{ an architecture } A \text{ and } F \in SAF_{n,s}^n \text{ where } A \text{ has } \leq S \text{ function nodes with } \leq l_i \text{ nodes in a layer and where the inputs to layer } i = \text{ the outputs of layer } i - 1, \text{ such that } T, \text{ possibly with some completely unspecified outputs removed, is contained in } M_F^A \}$

Theorem $COMPS(SAF_{n,s})$ is NP-complete.

Idea:

Given $\langle S, \{l_i\}, T \rangle$ construct $\langle S, \{l_i\}, T \rangle$, i.e the transformation is the identity function.

$\langle S, \{l_i\}, T \rangle \in COMP(SAF_{n,s}) \implies \langle S, \{l_i\}, T \rangle \in COMPS(SAF_{n,s})$

Directly.

$\langle S, \{l_i\}, T \rangle \in COMPS(SAF_{n,s}) \implies \langle S, \{l_i\}, T \rangle \in COMP(SAF_{n,s})$

Assume the first can be done with $s \leq S$ nodes then it can be done with S nodes by adding trivial nodes.

Consider the problem of determining the l -layer minimal architecture that computes a particular Boolean function (a Task). If this could be solved easily, i.e. in polynomial time, then this would imply that $COMPS(SAF_{n,s})$ could be tractably decided. However, $COMPS(SAF_{n,s})$ is NP-complete and is considered intractable which implies that the minimization problem is intractable as well.

Chapter 5

Conclusions

The results of this thesis fall into two categories. Those that deal with linear threshold functions and those that deal with Multi-Layered Perceptrons. First, A constructive relationship was demonstrated between Decision Trees and Multi-Layered Perceptrons as justification for the study of MLPs.

The main interest in this thesis however, was the minimization problem that dealt with finding the minimal, in terms of functional nodes, MLP architecture of a given number of layers that computes a specified Boolean input/output task. Extension of Judd's work gives justification to believe that finding minimal MLP structures where SAF_n are the functional units is intractable.

The Linear Algebraic framework developed reduces a problem using matrix inequalities to one that uses vector inequalities. If one is interested in a Boolean function of n bits then the computation involved in determining linear separability uses exponentially sized structures. However, the framework is useful conceptually, as was demonstrated in the example.

Further research needs to be done to extend the result to the general LTF case. In particular, it must be determined whether or not the Linear Algebraic framework will allow for the specification of an appropriate polynomial size *task* in the proof of the NP-completeness theorem.

Bibliography

- [1] Saul B. Gelfand, C. S. Ravishankar, and Edward J. Delp. An iterative growing and pruning algorithm for classification tree design. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 13(2):163–174, February 1991.
- [2] J. Ross Quinlan and Ronald L. Rivest. Inferring decision trees using the minimum description length principle. Technical report, September 1987.
- [3] Ishwar K. Sethi. Entropy nets: From decision trees to neural networks. In *Proc. IEEE*, volume 78, pages 1605–1613, October 1990.
- [4] Laurent Hyafil and Ronald L. Rivest. Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 5(1):15–17, May 1976.
- [5] Marvin L. Minsky and Seymour A. Papert. *Perceptrons*. The MIT Press, Cambridge, MA, 1969.
- [6] J. Stephen Judd. Complexity of connectionist learning with various node functions. Technical Report 87-60, COINS, July 1987.
- [7] Tom Richardson. *Private communication to Prof. Mitter*.
- [8] D. B. Schwartz, V. K. Samalam, Sara A. Solla, and J. S. Denker. Exhaustive learning. *Neural Computation*, (2):374–385, 1990.