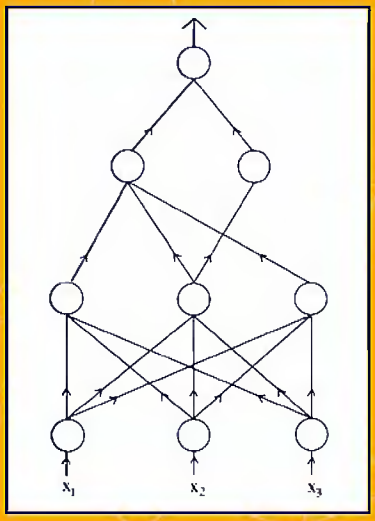
Discrete Mathematics of Neural Networks

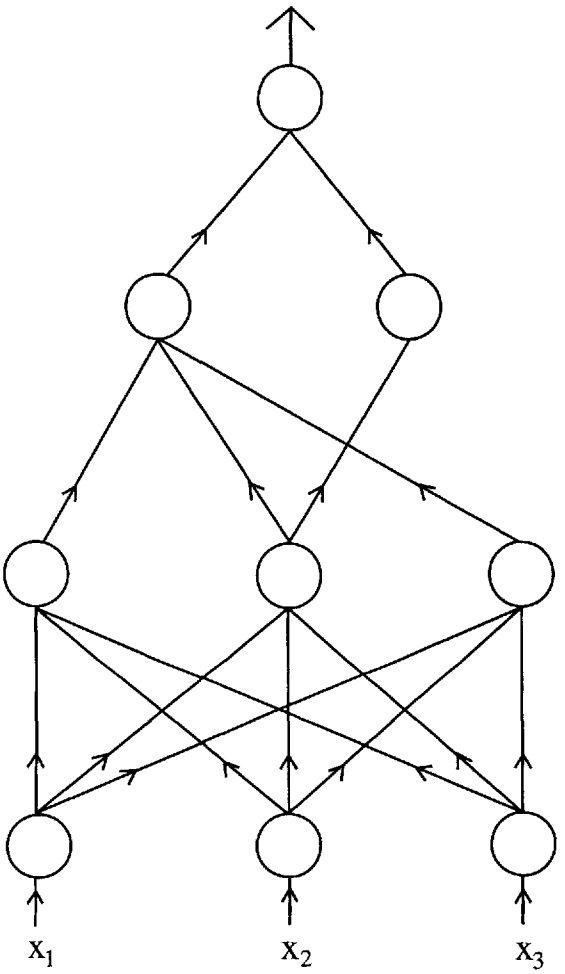
Selected Topics



Martin Anthony

SIAM Monographs on Discrete Mathematics and Applications

DISCRETE MATHEMATICS OF NEURAL NETWORKS



SIAM Monographs on Discrete Mathematics and Applications

The series includes advanced monographs reporting on the most recent theoretical, computational, or applied developments in the field; introductory volumes aimed at mathematicians and other mathematically motivated readers interested in understanding certain areas of pure or applied combinatorics; and graduate textbooks. The volumes are devoted to various areas of discrete mathematics and its applications.

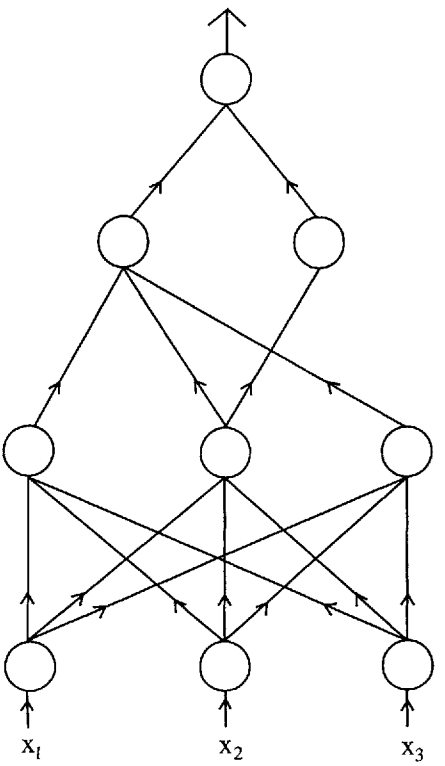
Mathematicians, computer scientists, operations researchers, computationally oriented natural and social scientists, engineers, medical researchers, and other practitioners will find the volumes of interest.

Series Volumes

Anthony, M., *Discrete Mathematics of Neural Networks: Selected Topics*
Creignou, N., Khanna, S., and Sudan, M., *Complexity Classifications of Boolean Constraint Satisfaction Problems*
Hubert, L., Arabie, P., and Meulman, J., *Combinatorial Data Analysis: Optimization by Dynamic Programming*
Peleg, D., *Distributed Computing: A Locality-Sensitive Approach*
Wegener, I., *Branching Programs and Binary Decision Diagrams: Theory and Applications*
Brandstädt, A., Le, V. B., and Spinrad, J. P., *Graph Classes: A Survey*
McKee, T. A. and McMorris, F. R., *Topics in Intersection Graph Theory*
Grilli di Cortona, P., Manzi, C., Pennisi, A., Ricca, F., and Simeone, B., *Evaluation and Optimization of Electoral Systems*

Discrete Mathematics of Neural Networks

Selected Topics



Martin Anthony

The London School of Economics and Political Science
London, UK

siam

Society for Industrial and Applied Mathematics
Philadelphia

10 9 8 7 6 5 4 3 2 1

Library of Congress Cataloging-in-Publication Data

Anthony, Martin.
Discrete mathematics of neural networks : selected topics / Martin Anthony.
p. cm. --(SIAM monographs on discrete mathematics and applications)
Includes bibliographical references and index.
ISBN 0-89871-480-X
1. Neural networks (Computer science)--Mathematics. I. Title. II. Series.

QA76.87 A58 2001
006.3'2'0151--dc21

00-067940

To Colleen and my parents.

This page intentionally left blank

Contents

Preface

This short book considers selected topics involving the interplay between certain areas of discrete mathematics and the simplest types of artificial neural networks. Graph theory, some partially ordered set theory, computational complexity, and discrete probability theory are among the mathematical topics involved.

The aim of this book is to give those interested in discrete mathematics a taste of the large, active, and expanding field of artificial neural network theory. The book might be best regarded as a series of extended essays on topics involving neural networks, discrete mathematics, and Boolean functions. A book of this length can only touch on some of the very many interesting issues involved, and those that are considered can all be explored much more deeply. Inevitably, therefore, the book focuses—and only briefly—on topics which have been of most interest to me, and I apologize if the selection appears idiosyncratic. Pointers are given in each chapter to further reading on the given topics and on related topics, and the reader is encouraged to pursue these leads.

I am happy to acknowledge the support of the European Union through the "Neurocolt" and "Neurocolt 2" grants. I am grateful to Graham Brightwell for providing useful comments on draft chapters of the book. I thank the many colleagues who have encouraged my interest in the topics covered, and who have influenced the way I think about them, particularly Peter Bartlett, Norman Biggs, Graham Brightwell, Peter Hammer, and John Shawe-Taylor. This book owes much to Peter Hammer's enthusiastic encouragement, for which I am very grateful. Deborah Poulson and her colleagues at SIAM have been very helpful throughout the preparation of this book, and I thank them for their patience.

MARTIN ANTHONY

This page intentionally left blank

Chapter 1

Artificial Neural Networks

1.1 Introduction

There has recently been intense and fast-growing interest in "artificial neural networks." These are machines (or models of computation) based loosely on the ways in which the brain is believed to work. Neurobiologists are interested in using these machines as a means of modeling biological brains, but much of the impetus comes from their applications. For example, engineers wish to create machines that can perform "cognitive" tasks, such as speech recognition, and economists are interested in financial time series prediction using such machines. Here, we take what may be called an "engineering approach" to artificial neural networks. In such an approach, we are not concerned with the issue of whether artificial neural networks are plausible models of real neural networks, but rather, we start from the fact that they exist and are being used extensively.

In this book we shall be interested mainly in *feed-forward* artificial neural networks[1] and, within this class, we shall be interested in *linear threshold networks* and *sigmoid networks*. We shall also, to a much lesser extent, be interested in a type of network known as a *Boltzmann machine*. However, since Boltzmann machines are very different from feed-forward networks, and since we consider them only very briefly, description of Boltzmann machines is deferred to a later chapter.

1.2 Feed-forward networks generally

It appears that one reason why the human brain is so powerful is the sheer complexity of connections between neurons. In computer science parlance, the brain exhibits huge parallelism, with each neuron connected to many other neurons. This has been reflected in the design of artificial neural networks. One advantage of such parallelism is that the resulting network is *robust*: in a serial computer, a single fault can make computation impossible, whereas in a system with a high degree of parallelism and many computation paths, a small number of faults may be tolerated with little or no upset to the computation.

[1] We shall usually omit the word "artificial," but it is to be understood that we discuss machines or models of computation which, although motivated by the way the brain has been believed to operate, need not be a realistic model of a biological neural network.

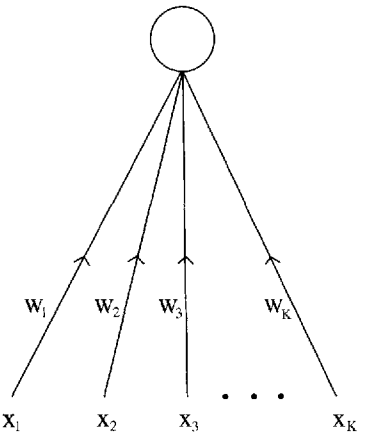


Figure 1.1: A pictorial representation of a linear threshold or sigmoid unit.

Generally speaking, we can say that an artificial neural network consists of a directed graph with *units* (or *neurons*) situated at the vertices. Some of these are *input units*, which receive signals from the outside world. The remaining are termed *computation units*. One or more of these computation units are specified as *output units*. These are the units with zero out-degree in the directed graph. We shall consider in this book networks in which there is only one output unit. Each unit produces an output, which is transmitted to other units along the arcs of the directed graph. The outputs of the input units are simply the input signals that have been applied to them. The computation units have *activation functions* determining their outputs. The degree to which the output of one computation unit influences those of its neighbors is determined by the weights assigned to the network. This description is quite abstract at this stage, but we shall concretize it shortly by focusing on particular types of networks.

1.3 Neurons

The building blocks of feed-forward networks are *computation units* (or *neurons*). In isolation, a computation unit has some number, k, of *inputs*, and is capable of taking on a number of *states*, each described by a vector $w = (w_0, w_1, \ldots, w_p) \in \mathbb{R}^p$ of p real numbers, known as *weights* or *parameters*. Here, p, the number of parameters of the unit, will depend on k. If the unit is a *linear threshold* unit or *sigmoid unit* (both of which are described below), then $p = k + 1$ and, in these cases, it is useful to think of the weights $w_1, w_2, \ldots, w_k$ as being assigned to each of the k inputs, as Figure 1.1 illustrates.

Generally, when in the state described by $w \in \mathbb{R}^p$, and on receiving input $x = (x_1, x_2, \ldots, x_k)$, the computation unit produces as output an *activation* $g(w, x)$, where $g : \mathbb{R}^p \times \mathbb{R}^k \to \mathbb{R}$ is a fixed function. We may regard the unit as a parameterized function class. That is, we may write $g(w, x) = g_w(x)$, where, for each state w, $g_w : \mathbb{R}^k \to \mathbb{R}$ is the function computed by the unit on the inputs x. For the special cases of linear threshold

and sigmoid units, where $p = k + 1$, the function g takes a particularly simple form:

$$g(w, x) = f(w_0 + w_1 x_1 + \cdots + w_k x_k),$$

where $f : \mathbb{R} \to \mathbb{R}$ is a *fixed* function, known as the *activation function* of the computation unit.

1.3.1 Linear threshold units

A computation unit is said to be a *linear threshold unit* (or simply a *threshold unit*, sometimes called a *perceptron*) if $p = k + 1$ and

$$g(w, x) = f(w_0 + w_1 x_1 + \cdots + w_k x_k),$$

where the activation function f is the sign function sgn, given by

$$\mathrm{sgn}(z) = \begin{cases} 1 & \text{if } z \geq 0, \\ 0 & \text{if } z < 0. \end{cases}$$

This function is a "step" function: it is piecewise constant, with a discontinuity at 0. Thus, when the state of the unit is given by $w = (w_0, w_1, \ldots, w_k)$, the output is either 1 or 0, and it is 1 precisely when

$$w_0 + w_1 x_1 + \cdots + w_k x_k \geq 0,$$

which may be written as

$$w_1 x_1 + \cdots + w_k x_k \geq \theta,$$

where $\theta = -w_0$ is known as the *threshold*. In other words, the computation unit gives output 1 (in biological parlance, it *fires*) if and only if the weighted sum of its inputs is at least the threshold θ.

Example 1.1 Suppose that a two-input linear threshold unit has weights $w_0 = -1$, $w_1 = w_2 = 1$. Then, on input $x = (x_1, x_2)$, the output of the unit is 1 if and only if

$$x_1 + x_2 \geq 1.$$

This means that the set of $x = (x_1, x_2) \in \mathbb{R}^2$ on which the output is 1 is the closed half-space lying above the line $x_1 + x_2 = 1$, and that for x belonging to the open half-space below that line, the output of the unit is 0. If we restrict attention to values of x_1, x_2 belonging to $\{0, 1\}$, then the output of the unit on $x \in \{0, 1\}^2$ is 1 if and only if at least one of x_1, x_2 equals 1. In other words, the unit computes the Boolean "or" function.

The computation of Boolean functions by linear threshold units will be an important concern in the book, and will be discussed at some length later. For the moment, we present the standard example showing that linear threshold units cannot compute all Boolean functions. From now on, we simplify notation somewhat by writing $x = (x_1, x_2, \ldots, x_n) \in \{0, 1\}^n$ simply as $x_1 x_2 \ldots x_n$. For example, instead of $(0, 0)$, we write 00.

Example 1.2 The *exclusive-or* Boolean function, XOR, of two variables is given by $\mathrm{XOR}(x_1, x_2) = 1$ if and only if *exactly one* of x_1, x_2 is 1. So, the function has

$$00 \mapsto 0,\ 01 \mapsto 1,\ 10 \mapsto 1,\ 11 \mapsto 0.$$

This function cannot be computed by a linear threshold unit. If it could, then, with the weights being w_1, w_2 and the threshold θ, we would have (examining each of the four function assignments in turn),

$$\begin{aligned} 0 &< \theta, \\ w_2 &\geq \theta, \\ w_1 &\geq \theta, \\ w_1 + w_2 &< \theta. \end{aligned}$$

But the last of these inequalities contradicts the fact (from the first three) that

$$w_1 + w_2 \geq 2\theta > \theta.$$

Sometimes, Boolean functions are regarded as mappings from $\{-1,1\}^n$ to $\{-1,1\}$, rather than from $\{0,1\}^n$ to $\{0,1\}$. Correspondingly, we would use a slightly different definition of the function sgn, in which $\text{sgn}(z)$ is defined to be -1 (rather than 0) if $z < 0$.

1.3.2 Sigmoid units

For a sigmoid unit (sometimes referred to as a *standard* sigmoid unit), $p = k + 1$ and

$$g(w, x) = \sigma\left(w_0 + w_1 x_1 + \cdots + w_k x_k\right),$$

where the activation function σ is the *sigmoid function*,

$$\sigma(z) = \frac{1}{1 + e^{-z}}.$$

Writing $\theta = -w_0$ as we did above for the linear threshold unit, we see that the output of the sigmoid unit is $\sigma(\sum_{i=1}^{k} w_i x_i - \theta)$. If the weighted sum $\sum_{i=1}^{k} w_i x_k$ is much larger than the threshold, then the output is close to 1; if it is much less than the threshold, the output is close to 0; and if it is very close to the threshold, then the output is close to 1/2. In fact, the sigmoid function can be thought of as a "smoothed" version of the sign function, sgn, since σ maps from $\mathbb{R}$ into the interval $(0,1)$, is differentiable, and satisfies

$$\lim_{z \to -\infty} \sigma(z) = 0, \quad \lim_{z \to \infty} \sigma(z) = 1.$$

Note that, whereas the linear threshold unit has output in $\{0,1\}$, the output of a sigmoid unit lies in the interval $(0,1)$ of real numbers.

1.3.3 Polynomial threshold units

The linear threshold and sigmoid units both work with $w_1 x_1 + \cdots + w_k x_k$, a linear combination of the inputs to the unit, but we can generalize from this and consider instead units which use a nonlinear combination of the x_i. For example, when $k = 3$, imagine a unit which computes the quadratic expression

$$w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_1^2 + w_5 x_2^2 + w_6 x_3^2 + w_7 x_1 x_2 + w_8 x_1 x_3 + w_9 x_2 x_3$$

for some constants w_i $(1 \leq i \leq 9)$ and then compares this with a threshold value θ. Such a unit is a *polynomial threshold unit* of degree 2. We now set up a description of this

generalization of linear threshold units. We shall denote by $[n]^m$ the set of all selections, in which repetition is allowed, of at most m objects from the set $[n] = \{1, 2, \ldots, n\}$. Thus, $[n]^m$ is a collection of "multisets." For example, $[3]^2$ consists of the multisets

$$\emptyset, \{1\}, \{1,1\}, \{2\}, \{2,2\}, \{3\}, \{3,3\}, \{1,2\}, \{1,3\}, \{2,3\}.$$

A *polynomial threshold unit of degree* m has $p = \binom{n+m}{m}$ parameters w_S, one for each multiset $S \in [n]^m$. For $S \in [n]^m$ and $x = x_1 x_2 \ldots, x_n \in \mathbb{R}^n$, let x_S denote the product of the x_i for $i \in S$ (with repetitions as required). For example, $x_{\{1,2,3\}} = x_1 x_2 x_3$ and $x_{\{1,1,2\}} = x_1^2 x_2$. When $S = \emptyset$, the empty set, we interpret x_S as the constant 1. The output of the unit is given by

$$g_w(x) = g(w, x) = \operatorname{sgn}\left(\sum_{S \in [n]^m} w_S x_S\right).$$

Of course, when $m = 1$ we obtain a linear threshold-unit. But for $m > 1$, a polynomial threshold unit can compute functions that a linear threshold unit is incapable of computing. Furthermore (and this will prove useful later), note that if we restrict the inputs x_i to belong to $\{0, 1\}$, then we do not need terms of the form $w_S x_S$ where the multiset S contains repeated members: this is simply because if $x_i \in \{0, 1\}$, then $x_i^r = x_i$ for all $r > 1$.

Example 1.3 Consider the case $n = m = 2$ and suppose we take

$$w_\emptyset = -\frac{1}{2},\ w_{\{1\}} = w_{\{2\}} = 1,\ w_{\{1,2\}} = -2,$$

with the remaining weights $w_{\{1,1\}}$ and $w_{\{2,2\}}$ equal to 0. Then

$$g_w(x) = \operatorname{sgn}\left(-\frac{1}{2} + x_1 + x_2 - 2x_1 x_2\right).$$

If we consider only $x \in \{0, 1\}^2$, then the Boolean function computed is exactly the exclusive-or function XOR, because

$$\begin{aligned} g_w(00) &= \operatorname{sgn}(-1/2) = 0, \\ g_w(10) &= \operatorname{sgn}(1/2) = 1, \\ g_w(01) &= \operatorname{sgn}(1/2) = 1, \\ g_w(11) &= \operatorname{sgn}(-1/2) = 0. \end{aligned}$$

As observed earlier (in Example 1.2), this function is not computable by a linear threshold unit.

1.4 Networks

As mentioned in the general description above, a neural network is formed when we place units at the vertices of a directed graph, with the arcs of the digraph representing the flows of signals between units. Some of the units are termed *input units*: these receive signals not from other units, but instead they take their signals from the outside environment. Units that do not transmit signals to other units are termed *output units*.

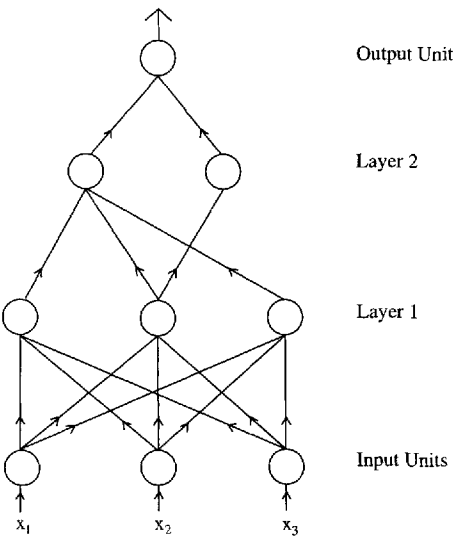


Figure 1.2: A three-layer feed-forward network.

The network is said to be a *feed-forward network* if the underlying directed graph is acyclic (that is, it has no directed cycles). This feed-forward condition means that the units can be labeled with integers in such a way that if there is a connection from the computation unit labeled i to the computation unit labeled j, then $i < j$. We will often be interested in *multilayer* networks. In such networks, the units may be grouped into *layers*, labeled $0, 1, 2, \ldots, \ell$, in such a way that the input units form layer 0, these feed into the computation units, and if there is a connection from a computation unit in layer r to a computation unit in layer s, then we must have $s > r$. Note, in particular, that there are no connections between any two units in a given layer. We call such a network an ℓ-layer network. (Strictly speaking, it has $\ell + 1$ layers, but one of these consists entirely of input units, and it is the number of layers of computation units that is usually important.) If there are connections *only* between adjacent layers, then we say the network is *stratified*. Any feed-forward network is a multilayer network (since we could just take the layers to consist of single computation units), but we shall often be interested in feed-forward networks with a small number of layers. It is easy to see that the smallest ℓ for which such a layering is possible is the *depth* of the network, defined as the length of the largest directed path in the underlying directed graph. Figure 1.2 provides an illustration of a three-layer feed-forward network with one output unit. Layer 0, comprising the input units, is usually called the *input layer*, and the last layer is called the *output layer*. All the layers of computation units in between are described as *hidden layers*.

We shall be interested in polynomial threshold units and in feed-forward networks having just one output unit, in which the computation units are linear threshold units or sigmoid units. Such a network with n input units is capable of computing a number of functions from $\mathbb{R}^n$ to $\mathbb{R}$, or (simply restricting the input signals to be $\{0,1\}$-valued) from $\{0,1\}^n$ to $\mathbb{R}$. The precise function computed depends on the state of each computation unit. Recall (since we are considering networks only of linear threshold or sigmoid units)

that if a unit has k inputs, then the state is a vector of $k+1$ real numbers: one of these numbers (w_0 or its negative, the threshold θ in the description above) can be thought of as being attached to the unit itself, and the other k can be thought of as describing the weight attached to each of the k arcs feeding into the unit. Suppose that the network has N computation units, labeled $1, 2, \ldots, N$, and that computation unit i has k_i inputs. Then the total number of weights in the network is

$$\sum_{i=1}^{N}(k_i + 1) = N + \sum_{i=1}^{N} k_i = N + E,$$

where E denotes the total number of arcs in the digraph. We may therefore say that the *state of the network* as a whole is described by a vector w of $W = N + E$ real numbers. When there are n input units and one output unit, the network computes, for each state w, a function $h_w : \mathbb{R}^n \to \mathbb{R}$. The set of functions *computable* by the network when the weight vector can be chosen from a subset Ω of $\mathbb{R}^W$ is $\{h_w : w \in \Omega\}$. (Often, Ω will simply be $\mathbb{R}^W$, but one may want, for example, to restrict the sizes of the allowable weights, in which case Ω will be a strict subset of $\mathbb{R}^W$.)

Note that a single neuron is a special, simple type of neural network. (We call this simplest of all networks the *perceptron* though that term is sometimes also used more generally.) We shall see that there are many interesting questions to be answered, even for the perceptron.

Example 1.4 Consider the feed-forward threshold network in Figure 1.3. The weights are shown on the arcs, and the thresholds are indicated on computation units. It is easily checked (simply by calculating the output on each of the four inputs in $\{0,1\}^2$) that the network computes the exclusive-or function on $\{0,1\}^2$.

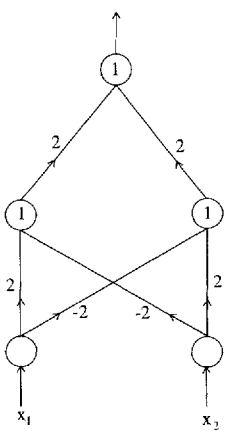


Figure 1.3: A feed-forward linear threshold network computing the exclusive-or function on $\{0,1\}^2$.

1.5 Additional remarks and bibliographical notes

There are many good general texts on neural networks, such as [16, 47, 87]. Linear threshold networks have long been studied, and were the subject of much work in

"threshold logic" in the 1960s; see the books by Muroga [75] and Hu [48] and the papers cited there.

Questions concerning the type of function computable by a polynomial threshold unit have been worked on by a number of researchers and were considered in [72, 29, 78]. For more recent results, see the survey article by Saks [90]: this provides an excellent overview of much of the theoretical work on functions computable by threshold and polynomial threshold units and related areas (some of which will be touched on later in this book). See also Wang and Williams [104].

Chapter 2

Boolean Functions

2.1 Introduction

One of the areas discussed in this book is the representation of Boolean functions by neural networks. Specifically, we investigate questions about the sort of Boolean functions computable by certain types of neural network, and the types of network required to compute certain classes of Boolean functions. In this chapter, therefore, we set up some notations and terminology concerning Boolean functions. Some of these will be familiar to most readers, but it is useful to recall the key ideas we shall most heavily draw upon in what follows.

2.2 Boolean functions and the hypercube

2.2.1 The Boolean hypercube

A Boolean function is simply a function mapping from $\{0,1\}^n$ to $\{0,1\}$. Clearly, there are 2^{2^n} Boolean functions, since for each of the 2^n members of $\{0,1\}^n$, the function can take one of two values. Sometimes, it is more convenient to think of a Boolean function as mapping from the set $\{-1,1\}^n$ to $\{-1,1\}$. This approach is occasionally more useful, but we shall normally use the standard convention.

The domain $\{0,1\}^n$ is often called the *Boolean hypercube*, and there are a number of useful ways of thinking about it.

2.2.2 Geometry of the Boolean hypercube

The geometry of the Boolean hypercube is sometimes important when we begin to seek answers to questions about Boolean functions computable by neural networks. A set of points in $\mathbb{R}^n$ is said to be in *general position* if no $n+1$ of them lie in an $(n-1)$-dimensional hyperplane (or "flat") of $\mathbb{R}^n$. The rigid geometrical arrangement of the points in $\{0,1\}^n$ means that they are not in general position. (Indeed, no subset of $2n+1$ of the points is in general position, since at least $n+1$ of them belong to the $(n-1)$-dimensional flat with equation $x_1 = 0$ or to the flat with equation $x_1 = 1$.) This lack of general position is, as we shall see, often a complication.

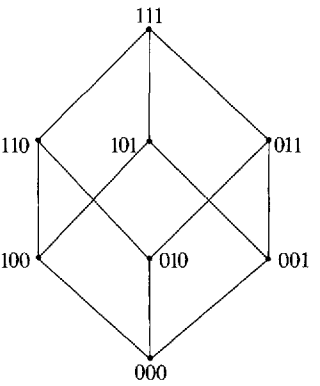


Figure 2.1: The three-dimensional hypercube $\{0,1\}^3$ represented as a graph.

2.2.3 The Boolean hypercube as a graph

The Boolean hypercube may be regarded as the set V of vertices of the hypercube graph $G = (V, E)$, where two vertices are joined by an edge if they differ in exactly one coordinate: that is, if their Hamming distance is 1. The hypercube of dimension three is illustrated as a graph in Figure 2.1.

2.2.4 The Boolean hypercube as a partially ordered set

There is a fairly natural partial ordering on the set $\{0,1\}^n$. We may regard the vertices of the hypercube as corresponding to all subsets of an n-set (so that the origin corresponds to the empty set and the examples with k entries equal to 1 correspond to the k-subsets). Then the examples in $\{0,1\}^n$ have a partial order $\preceq$ on them, corresponding directly to the inclusion partial order in the power set of the n-set. In this partial order, $x \preceq y$ if $x_i = 1$ implies $y_i = 1$. (Here, as always, for $z \in \mathbb{R}^n$, z_i is the ith entry of z.) There are other ways of defining a partial order on $\{0,1\}^n$, but this is perhaps the most natural, and it is one we shall make much use of later.

For example, $01010 \preceq 11011$, but 01010 and 11101 are incomparable.

2.3 Boolean formulas

2.3.1 Representing Boolean functions by formulas

There is a standard method of representing Boolean functions $\{0,1\}^n \to \{0,1\}$ by means of formulas. The symbols we need to do this are the literals $u_1, \bar{u}_1, \ldots, u_n, \bar{u}_n$, the logical connectives "$\vee$," meaning *or*, and "$\wedge$," meaning *and*, and the parentheses "(" and ")". Each one of the literals u_i and $\bar{u}_i$ itself defines a Boolean function as follows:

$$u_i(y) = \begin{cases} 1 & \text{if } y_i = 1, \\ 0 & \text{if } y_i = 0, \end{cases}$$

$$\bar{u}_i(y) = \begin{cases} 1 & \text{if } y_i = 0, \\ 0 & \text{if } y_i = 1. \end{cases}$$

We say that $\bar{u}_i$ is the *negation* of u_i. For the sake of simplicity, we identify the literal u_i with the Boolean function it represents.

We can use the logical connectives, together with parentheses, to construct other Boolean formulas. (We shall continue to identify formulas with the functions they represent.) If ϕ and ψ are Boolean formulas (for example, single literals), then we can form the new formula $\phi \vee \psi$ from them, known as the *disjunction* of ϕ and ψ. The Boolean function represented by $\phi \vee \psi$ acts as follows:

$$(\phi \vee \psi)(y) = \begin{cases} 1 & \text{if } \phi(y) = 1 \text{ or } \psi(y) = 1, \\ 0 & \text{otherwise.} \end{cases}$$

Similarly, we have the *conjunction*, denoted $\phi \wedge \psi$, which defines a Boolean function by

$$(\phi \wedge \psi)(y) = \begin{cases} 1 & \text{if } \phi(y) = 1 \text{ and } \psi(y) = 1, \\ 0 & \text{otherwise.} \end{cases}$$

Disjunction and conjunction, together with the usual conventions about the use of parentheses, allow the formation of formulas like

$$(\bar{u}_1 \wedge u_4) \vee (u_1 \wedge u_2 \wedge \bar{u}_3).$$

There is a convention that the symbol $\wedge$ for conjunction may be omitted (just as one customarily omits the multiplication sign "$\times$" in algebraic expressions), so that the function above is usually written as $\bar{u}_1 u_4 \vee u_1 u_2 \bar{u}_3$.

It is often useful to speak of a Boolean formula being "true" or "false" on a particular input (or argument) $y \in \{0,1\}^n$. The formula ϕ is true on y if the function f represented by ϕ satisfies $f(y) = 1$, and ϕ is said to be false on y otherwise. We say that y is a *true point* or *positive example* of f if $f(y) = 1$, and that y is a *false point* or *negative example* otherwise.

Example 2.1 Consider the Boolean function described by the formula $\bar{u}_1 u_4 \vee u_1 u_2 \bar{u}_3$ just given. It describes the function $f : \{0,1\}^4 \to \{0,1\}$ such that, for $y = y_1 y_2 y_3 y_4$ in $\{0,1\}^4$, $f(y) = 1$ if and only if *either* $y_1 = 0$ and $y_4 = 1$, *or* $y_1 = y_2 = 1$ and $y_3 = 0$. So, for instance, 0011 is a true point of f and 1001 is a false point of f. (The formula also describes the Boolean function from $\{0,1\}^5$ which acts as f does on the first four entries of any argument: indeed, generally, it describes such a function on $\{0,1\}^n$ for any $n \geq 4$. But in practice one knows what the domain of interest is.)

2.3.2 DNF and CNF representations

DNF formulas. A vital property of Boolean formulas is that every Boolean function $\{0,1\}^n \to \{0,1\}$ can be represented by a formula. In fact, there are different ways of representing a given function, but there are two types of formulas which are particularly convenient. A formula in *disjunctive normal form* (or a DNF formula) is one of the form

$$T_1 \vee T_2 \vee \cdots \vee T_k,$$

where each T_l is a *term* of the form

$$T_l = \left(\bigwedge_{i \in P} u_i \right) \bigwedge \left(\bigwedge_{j \in N} \bar{u}_j \right),$$

for some disjoint subsets P, N of $\{1, 2, \ldots, n\}$.

Example 2.2 The formula we considered above, $\bar{u}_1 u_4 \vee u_1 u_2 \bar{u}_3$, is a DNF formula. The function f it represents is also represented by the DNF formula

$$\bar{u}_1 u_4 \vee u_1 u_2 \bar{u}_3 \vee \bar{u}_1 u_3 u_4.$$

The first of these formulas is clearly simpler. The second can be seen to be equivalent to the first by noting that its third term $\bar{u}_1 u_3 u_4$ is "absorbed" by its first, $\bar{u}_1 u_4$, in the sense that if the third term is true then the first is also. This means that the third term can be deleted from the formula without changing the function represented by the formula. DNF formulas can often be simplified using such observations.

Example 2.3 Sometimes a DNF formula can be simplified not by deleting terms but by combining terms. For example, the formula $\bar{u}_1 u_2 u_4 \vee \bar{u}_1 \bar{u}_2 u_4$ is equivalent to the simpler formula $\bar{u}_1 u_4$.

The existence of a DNF formula for any Boolean function may easily be established as follows. Let f be a Boolean function of n variables. For each true point y of f, form the term

$$T_y = \left(\bigwedge_{\{i:y_i=1\}} u_i \right) \bigwedge \left(\bigwedge_{\{j:y_j=0\}} \bar{u}_j \right).$$

Then $T_y(x) = 1$ if and only if $x = y$. Thus, if $y_1, y_2, \ldots, y_k$ are *all* the positive examples of f, then f is represented by the DNF formula

$$T_{y_1} \vee T_{y_2} \vee \cdots \vee T_{y_k}.$$

This procedure produces a rather long DNF formula representing f, and so is of little practical use, but it certainly establishes that a DNF representation exists.

Example 2.4 The function f such that $f(y) = 1$ if and only if $y_1 = 1$ has DNF representation u_1, yet the procedure just described would produce a DNF with 2^{n-1} terms. On the other hand, consider the *parity* function PARITY_n on $\{0,1\}^n$. This is defined by $\text{PARITY}_n(y) = 1$ if and only if y has an odd number of entries equal to 1. (So, PARITY_2 is simply the exclusive-or function.) The function PARITY_n has 2^{n-1} true points, and the above procedure would produce a DNF representation involving 2^{n-1} terms. In fact, this is the shortest possible DNF representation for PARITY_n. To establish this, we first note that in any DNF representation of PARITY_n, each term must have degree n. To see why, suppose some term T contained fewer than n literals, and that neither u_i nor $\bar{u}_i$ were present in T. Then there are $x, y \in \{0,1\}^n$ which are true points of T, but which differ only in position i. Then, since T is a term in the DNF representation of PARITY_n, we would have $\text{PARITY}_n(x) = \text{PARITY}_n(y) = 1$. But this cannot be: one of x, y will have an odd number of entries equal to 1, and one will have an even number of such entries. It follows that each term must contain n literals, in which case each term has only one true point, and so we must have 2^{n-1} distinct terms, one for each true point.

Prime implicants. We saw in Example 2.2 that terms in a DNF formula can sometimes be deleted without changing the function represented by the formula. To explain this a little more formally, and to introduce the important idea of prime implicant, we present a useful notation. For two Boolean functions f and g, we write $f \leq g$ if $f(x) \leq g(x)$ for all x; that is, if $f(x) = 1$ implies $g(x) = 1$. Similarly, for two Boolean

formulas ϕ, ψ, we shall write $\phi \leq \psi$ if, when f and g are the functions represented by ϕ and ψ, then $f \leq g$. A term T of a DNF is said to *absorb* another term T' if $T' \leq T$. In Example 2.5 we observed that $T = \bar{u}_1 u_4$ absorbs the term $T' = \bar{u}_1 u_3 u_4$. That is, whenever T' is true, so is T. This means that the formula

$$\bar{u}_1 u_4 \vee u_1 u_2 \bar{u}_3 \vee \bar{u}_1 u_3 u_4$$

is equivalent to $\bar{u}_1 u_4 \vee u_1 u_2 \bar{u}_3$.

A term T is an *implicant* of f if $T \leq f$; in other words, if T true implies f true. The terms in any DNF representation of a function f are implicants of f. The most important type of implicants are the *prime implicants*. These are implicants with the additional property that there is no other implicant of f absorbing T. Thus, a term is a prime implicant of f if it is an implicant, and the deletion of any literal from T results in a nonimplicant T' of f (meaning that there is some x such that $T'(x) = 1$ but $f(x) = 0$). If we form the disjunction of all prime implicants of f, we have a DNF representation of f.

CNF formulas. Any Boolean function can also be represented by a formula in *conjunctive normal form* (or, by a *CNF formula*). Such a formula is of the type

$$C_1 \wedge C_2 \wedge \cdots \wedge C_k,$$

where each C_l is a *clause*,

$$C_l = \left(\bigvee_{i \in P'} u_i \right) \vee \left(\bigvee_{j \in N'} \bar{u}_j \right),$$

for some disjoint subsets P', N' of $\{1, 2, \ldots, n\}$.

Example 2.5 Consider again the function f which is represented by the DNF formula $\bar{u}_1 u_4 \vee u_1 u_2 \bar{u}_3$. We can find a CNF formula representing f by using some basic rules of Boolean algebra, and in particular the distributivity of the conjunction and disjunction operations over each other. We can proceed as follows, where in writing $\phi = \psi$ for two Boolean formulas, we mean that they represent the same function.

$$\begin{aligned} \bar{u}_1 u_4 \vee u_1 u_2 \bar{u}_3 &= (\bar{u}_1 \vee (u_1 u_2 \bar{u}_3))\,(u_4 \vee (u_1 u_2 \bar{u}_3)) \\ &= (\bar{u}_1 \vee u_1)(\bar{u}_1 \vee u_2)(\bar{u}_1 \vee \bar{u}_3)(u_4 \vee u_1)(u_4 \vee u_2)(u_4 \vee \bar{u}_3) \\ &= (\bar{u}_1 \vee u_2)(\bar{u}_1 \vee \bar{u}_3)(u_1 \vee u_4)(u_2 \vee u_4)(\bar{u}_3 \vee u_4). \end{aligned}$$

(Note that $(\bar{u}_1 \vee u_1)$ is always true and can therefore be omitted. According to our definition, it is not even a valid clause, so in fact it must be omitted.)

2.3.3 Dualization

For $x \in \{0,1\}^n$, the *negation* of x, denoted $\bar{x}$, is obtained from x by changing the entries equal to 1 to 0, and the entries equal to 0 to 1. For a Boolean function f, the *negation* (or *complement*) of f is the function $\bar{f}$ defined by $\bar{f}(x) = 1$ if and only if $f(x) = 0$. The *dual* of a Boolean function f is the function f^d such that

$$f^d(x) = \bar{f}(\bar{x}).$$

If we take a DNF formula representing f, then we can obtain a "dual" CNF formula for f^d by replacing every occurrence of $\vee$ with $\wedge$ and every occurrence of $\wedge$ with $\vee$ (bearing in mind that the $\wedge$ symbol will sometimes be implicit). For example, consider the formula $\bar{u}_1 u_4 \vee u_1 u_2 \bar{u}_3$, which, if we explicitly write in the $\wedge$ symbols, is

$$(\bar{u}_1 \wedge u_4) \vee (u_1 \wedge u_2 \wedge \bar{u}_3).$$

The dual of this is

$$(\bar{u}_1 \vee u_4) \wedge (u_1 \vee u_2 \vee \bar{u}_3).$$

The existence of a CNF representation of any Boolean function f can be established by noting that "expansion" of a DNF formula using the rules of Boolean algebra will result in an equivalent formula (as in Example 2.5). It can also be established by considering dualization as follows. Let us take a DNF representation of the complement $\bar{f}$ of f, replace each u_i by $\bar{u}_i$ and each $\bar{u}_i$ by u_i, and dualize. We obtain the dual of $\bar{f}(\bar{x})$, which is equal to f. Moreover, we have obtained a CNF expression. So, from a DNF representing the complement of f (the existence of which we established above), we have obtained a CNF formula representing f.

2.3.4 Classes of Boolean functions

By placing restrictions on the number of and types of terms in a disjunctive normal formula, we obtain special sets of Boolean functions. A Boolean function is said to be

- an l-DNF function if it has a DNF formula in which, for each term, the number of literals ($|P| + |N|$) is at most l;
- a k-term-DNF function if it has a DNF formula involving at most k terms;
- a k-term-l-DNF function if it has a DNF formula in which there are at most k terms, each involving at most l literals.

The 1-term-DNF functions (and the corresponding formulas) are usually referred to simply as *monomials*. (Thus, monomials are single terms.)

The *degree* of a DNF formula is the largest number of literals in any term; that is (using the earlier notation), it is the maximal value of $|P| + |N|$ over all k terms. The *degree* of a Boolean function f is defined to be the minimal degree of any DNF formula representing f. In particular, if a Boolean function has degree no more than d, then it has a DNF representation in which each term contains no more than d literals, and so it is a d-DNF. (Thus, the degree is the maximum length of a prime implicant of the function.)

Restricting the degree of the terms in a DNF usually results in a relatively small set of Boolean functions. For instance, if $l \in \mathbb{N}$ is fixed, then some very crude bounding shows that the number of l-DNF functions defined on $\{0,1\}^n$ is no more than $2^{(2n)^l}$, which, for large n, is vanishingly small compared with the total number, 2^{2^n}, of all Boolean functions. The effect of placing a bound on the number of terms in the DNF representation results is similarly restrictive. If $k \in \mathbb{N}$ is fixed, then the number of k-term-DNF functions defined on $\{0,1\}^n$ is easily seen to be no more than 2^{2nk}. These bounds are substantially smaller than 2^{2^n} even if k, l grow with n subexponentially.

2.4 Monotonicity and regularity

2.4.1 Increasing functions

A Boolean function f is said to be *increasing* (or *positive*, or *monotonic*) if $x \preceq y$ implies $f(x) \leq f(y)$. Given that f takes only values 0 or 1, this may be rephrased as follows: if $f(x) = 1$ and $x \preceq y$, then $f(y) = 1$. Recalling the definition of the ordering $\preceq$, this means that if we have a true point x and we replace some 0-entries in x by 1, then the resulting y is also a true point. It is clear that if a DNF formula has no negated literals in any of its terms, then the function represented by the formula is increasing. A type of converse holds: if a Boolean function f is increasing, then there is a DNF formula ϕ representing f such that no literal appears negated in ϕ. (Such a DNF formula is said to be *positive*.) In fact, the disjunction of all prime implicants is such a formula, and we call it the "irredundant DNF formula" for f.

2.4.2 Unate functions

A Boolean function is *unate* if it has a DNF representation in which all occurrences of any given literal are either negated or nonnegated. That is, $f : \{0,1\}^n \to \{0,1\}$ is unate if there is a DNF formula ϕ representing f such that, for each i between 1 and n, either u_i does not appear in any of the terms of ϕ, or $\bar{u}_i$ does not appear in any of the terms of ϕ. Such a formula is said to be *unate*, and it is easy to see that a formula is unate if it can be obtained from a positive DNF by replacing, for some of the i between 1 and n, all occurrences of u_i by $\bar{u}_i$. An equivalent characterization of unateness, in terms of the value of the function rather than the type of formula representing it, can be given. To describe this, for $i = 1, 2, \ldots, n$, we let e_i denote the binary vector with all entries equal to 0, except for entry i, which equals 1. We can see that a function is increasing if and only if for every $x \in \{0,1\}^n$ with $x_i = 0$, we have $f(x) \leq f(x + e_i)$. A similar description of unate functions can be given. A function f is unate if and only if for each i between 1 and n, *either*

$$\text{for all } x \text{ with } x_i = 0, f(x) \leq f(x + e_i)$$

or

$$\text{for all } x \text{ with } x_i = 0, f(x) \geq f(x + e_i).$$

The first case corresponds to the situation where there is a formula for the function in which each occurrence of u_i is not negated, and the second in which each occurrence is negated.

2.4.3 Regular and 2-monotonic functions

A *regular* function is an increasing function with some additional properties. Roughly speaking, the function f is regular if it is increasing and the presence of a 1 in position i carries more weight than the presence of a 1 in position j if $i < j$. Formally, f is regular if for every pair of numbers i, j in $\{1, 2, \ldots, n\}$, with $i < j$, if $x_i = 0$ and $x_j = 1$, then "swapping" entries i and j of x does not decrease f: that is, $f(x) \leq f(x + e_i - e_j)$. Given $x \in \{0,1\}^n$, a *left shift* of x is a $y \in \{0,1\}^n$ obtained by performing a sequence of "swaps" of the type just described: that is, by replacing, for some pairs i, j with $i < j$, a 0 in position i with a 1, and a 1 in position j with 0. Then, an increasing function is regular if and only if any left shift of a true point is a true point. An alternative

characterization may be given in terms of special types of shift, in which a 1 is shifted only one place to the left: it is quite easy to see that an increasing function is regular if and only if for every i between 1 and n, for all x such that $x_i = 0$ and $x_{i+1} = 1$, $f(x) \leq f(x + e_i - e_{i+1})$.

An increasing function is said to be 2-*monotonic* if the numbers $1, 2, \ldots, n$ can be arranged in such a way that if we then permute all the entries of every $x \in \{0,1\}^n$ accordingly, the function becomes regular. If f is 2-monotonic, then for each pair i and j between 1 and n, *either*

$$\text{for all } x \text{ with } x_i = 0 \text{ and } x_j = 1, f(x) \leq f(x + e_i - e_j)$$

or

$$\text{for all } x \text{ with } x_i = 0 \text{ and } x_j = 1, f(x) \geq f(x + e_i - e_j).$$

2.4.4 MFPs and MTPs of increasing functions

Suppose we know that a Boolean function f is increasing. Then the set of false points $F(f)$ forms a down-set, and the set of true points $T(f)$ an up-set, with respect to the ordering $\preceq$. (That is, if $x \in F(f)$ and $y \preceq x$ then $y \in F(f)$, and if $x \in T(f)$ and $x \preceq z$ then $z \in T(f)$.) Let MFP(f) be the set of maximal elements in $F(f)$, which we call *maximal false points*, and let MTP(f) be the set of minimal elements in $T(f)$, called *minimal true points*. Thus, x is a minimal true point (or an MTP) of f if x is a true point of f (that is, $f(x) = 1$) and there is no other true point y such that $y \preceq x$. Similarly, x is a maximal false point (or an MFP) if x is a false point ($f(x) = 0$) and there is no other false point y such that $x \preceq y$. Suppose now that z is given. If $x \preceq z$ for some MTP x then, because f is increasing, $f(z) = 1$. On the other hand, if there is no MTP x such that $x \preceq z$, then z cannot be a true point. (If it were, it would either itself be an MTP or would be above some MTP in the $\preceq$ ordering, and neither of these is the case.) Thus, we deduce $f(z) = 0$. We see, then, that any increasing function is determined precisely by the set MTP(f) of all its minimal true points. Such a function is equally well determined by the set MFP(f) of all its maximal false points, as similar reasoning will verify.

Example 2.6 Let $1 \leq k < n$, and consider the increasing function $f : \{0,1\}^n \to \{0,1\}$ which has as its minimal true points all $x \in \{0,1\}^n$ with exactly k entries equal to 1. Then $y \in \{0,1\}^n$ is a true point if and only if it has at least k entries equal to 1. Note that the maximal false points are all x having exactly $k - 1$ entries equal to 1.

The MTPs of an increasing function correspond directly with its prime implicants. To see this, note first that any prime implicant of f will contain no negated literal. (It is easy to see that if T is any implicant of f containing negated literals, then the term obtained from T by deleting the negated literals is also an implicant of f, because f is increasing.) Now suppose $\bigwedge_{i \in P} u_i$ is a prime implicant. Then the point x with $x_i = 1$ for $i \in P$ and all other entries equal to 0 is a minimal true point of f. Conversely, if x is a MTP of f, and $P = \{i : x_i = 1\}$, then the conjunction $\bigwedge_{i \in P} u_i$ is a prime implicant of f.

There is a useful relationship between the MTPs of the dual of an increasing function and the MFPs of the function itself. Explicitly, for an increasing Boolean function f, the MTPs of the dual f^d are the negations of the MFPs of f; that is, x is an MTP of f^d if and only if $\bar{x}$ is an MFP of f.

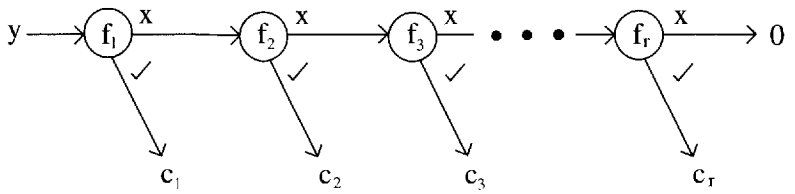


Figure 2.2: A diagrammatic representation of a decision list.

An alternative characterization of regularity, based on MTPs, can be given. This is that an increasing function f is regular if and only if for all pairs (v, i), where v is an MTP of f, $1 \leq i \leq n$, and $v_i = 0, v_{i+1} = 1$, we have $f(v + e_i - e_{i+1}) = 1$. It is quite straightforward to see that this condition on MTPs implies (and is, of course, implied by) the corresponding condition for all true points, and hence is equivalent to regularity.

2.5 Decision lists

2.5.1 Definition of decision lists

We will find it useful to think about representing Boolean functions by *decision lists* rather than by standard DNF or CNF formulas. The idea of a decision list is quite simple, and surprisingly versatile. Let K be any set of $\{0,1\}$-valued functions on $\mathbb{R}^n$, where n is fixed. A function $f : \mathbb{R}^n \to \{0,1\}$ is said to be a *decision list* based on K if it can be evaluated as follows, for some fixed functions $f_1, f_2, \ldots, f_r \in K$ and fixed $c_1, c_2, \ldots, c_r \in \{0,1\}$: given an example y, we first evaluate $f_1(y)$. If $f_1(y) = 1$, we assign value c_1 to $f(y)$; if not, we evaluate $f_2(y)$, and if $f_2(y) = 1$ we set $f(y) = c_2$. Otherwise we evaluate $f_3(y)$, and so on. If, for all i, $f_i(y) = 0$, we define the value $f(y)$ to be 0. The evaluation of a decision list f can be thought of as a sequence of `if then else` commands:

```
if f1(y) = 1 then set f(y) = c1
             else if f2(y) = 1 then set f(y) = c2
                                    ...
                                    ...
                                    else if fr(y) = 1 then set f(y) = cr
                                                      else set f(y) = 0.
```

More formally, a *decision list based on* K is defined by a sequence

$$f = (f_1, c_1),\ (f_2, c_2), \ldots,\ (f_r, c_r),$$

where $f_i \in K$ and $c_i \in \{0,1\}$ $(1 \leq i \leq r)$. The values of f are defined by

$$f(y) = \begin{cases} c_j & \text{if } j = \min\{i : f_i(y) = 1\} \text{ exists;} \\ 0 & \text{otherwise.} \end{cases}$$

It is useful to think about the evaluation procedure diagramatically, as Figure 2.2 illustrates. We denote by $DL(K)$ the set of all decision lists based on K.

Example 2.7 Suppose that K is the set of monomials on $\{0,1\}^3$ involving no more than 2 literals. (That is, K is the set of 1-term-2-DNF functions on $\{0,1\}^3$.) The

decision list

$$(u_2, 1),\ (u_1\bar{u}_3, 0),\ (\bar{u}_1, 1)$$

may be thought of as operating in the following way on $\{0,1\}^3$. First, those points for which u_2 is true are assigned the value 1: these are $010, 011, 110, 111$. Next the remaining points for which $u_1\bar{u}_3$ is satisfied are assigned the value 0: the only such point is 100. Finally, the remaining points for which $\bar{u}_1$ is true are assigned the value 1: this accounts for 000 and 001, leaving only 101, which is assigned the value 0.

We shall often be considering decision lists over the set of monomials of at most a fixed size, l, say. We refer to these decision lists as l-decision lists, and we denote the class of functions they represent by l-DL.

2.5.2 Decision lists and other Boolean function classes

There are some interesting relationships between the decision list representations of Boolean functions and the more standard DNF and CNF representations. We start with one important observation, which is that the disjunction of any number of functions in K can be expressed as a decision list over K. To see this, suppose $f_1, f_2, \ldots, f_r \in K$. Then the disjunction $f_1 \vee f_2 \vee \cdots \vee f_r$ is represented by the decision list

$$(f_1, 1), (f_2, 1), \ldots, (f_r, 1).$$

This means, for example, that any l-DNF function can be expressed as a decision list based on monomials of length no more than l (simply because it is a disjunction of such monomials). In other words, l-DNF $\subseteq$ l-DL. But, for $l < n$, such decision lists can also represent functions not in the class of l-DNF functions, as the following example shows.

Example 2.8 Consider the function $f : \{0,1\}^3 \to \{0,1\}$ with DNF formula $u_1u_2u_3$. This has only one true point, 111. This function is not a 2-DNF function, because any term involving at most two literals will have at least two true points, and hence any disjunction of such terms will have at least two true points and therefore cannot be equal to f. However, we can represent f as a decision list based on monomials of length 1, as follows:

$$(\bar{u}_1, 0), (\bar{u}_2, 0), (\bar{u}_3, 0), (1, 1),$$

where 1 denotes the monomial with no literals, which is the identically-1 function.

It follows also that, for a given n, *any* Boolean function of n variables can be represented by a decision list based on monomials of length no more than l, for l sufficiently large. (Indeed, taking l to be the degree of the function will do.)

2.6 Additional remarks and bibliographical notes

Much of the material in this chapter is "classical," and there are many elementary texts on Boolean functions and logic.

Decision lists were introduced by Rivest [88], who gave a simple "learning algorithm" for them.

We have commented that the irredundant DNF formula for an increasing function is often useful. For general (not necessarily increasing) functions, a number of techniques

exist for simplifying a DNF formula representing the function. A technique known as the *consensus method* can be used to reduce a DNF formula to the one which is the disjunction of prime implicants. (See [37], which cites Blake and Quine [17, 86].)

The alternative characterization of regularity (involving only shifts of one place) can be found in [75].

This page intentionally left blank

Chapter 3

Threshold Functions

3.1 Introduction

Here, we begin to look in detail at the linear and polynomial threshold functions, the functions computable by the linear threshold unit and polynomial threshold unit, the simplest types of neural networks. We consider the types of Boolean functions realizable, and not realizable, as threshold functions of a given degree, and we explain how it is often useful to think geometrically about these functions.

3.2 Threshold functions

3.2.1 Boolean threshold functions

A Boolean function t defined on $\{0,1\}^n$ is a *Boolean threshold function*, or simply a *threshold function* (sometimes known as a *linear threshold function*) if it is computable by a linear threshold unit. This means that there are $w = (w_1, w_2, \ldots, w_n) \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$ such that

$$t(x) = \operatorname{sgn}\left(\sum_{i=1}^{n} w_i x_i - \theta\right).$$

An alternative description of t (which will be useful when we think geometrically) is

$$t(x) = \begin{cases} 1 & \text{if } \langle w, x\rangle \geq \theta, \\ 0 & \text{if } \langle w, x\rangle < \theta, \end{cases}$$

where $\langle w, x\rangle$ is the standard inner product of w and x. Given such w and θ, we say that t is represented by $[w, \theta]$ and we write $t \leftarrow [w, \theta]$. The vector w is known as the *weight-vector*, and θ is known as the *threshold*. We shall denote by T_n the set of all threshold functions on $\{0,1\}^n$.

Of course, each $t \in T_n$ will satisfy $t \leftarrow [w, \theta]$ for ranges of w and θ, and we shall explore in later chapters some aspects of this representation. A technical point, which will prove useful in some of the following analysis, is that since $\{0,1\}^n$ is discrete, for any $t \in T_n$, there are $w \in \mathbb{R}^n$, $\theta \in \mathbb{R}$ and a positive constant c such that

$$t(x) = 1 \Longrightarrow \langle w, x\rangle \geq \theta + c, \quad t(x) = 0 \Longrightarrow \langle w, x\rangle \leq \theta - c.$$

A Boolean function is said to be a *homogeneous* threshold function if it is a threshold function and can be represented in such a way that the threshold θ is 0.

3.2.2 Real threshold functions

A function $f : \mathbb{R}^n \to \{0,1\}$ is a *real threshold function* if it is computable by a linear threshold unit; that is, if there are $w = (w_1, w_2, \ldots, w_n) \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$ such that

$$t(x) = \operatorname{sgn}\left(\sum_{i=1}^{n} w_i x_i - \theta\right).$$

The (Boolean) threshold functions are obtained by restricting the domain to $\{0,1\}^n$. Our main interest, at least for now, lies in Boolean threshold functions, but the notations we have developed for Boolean threshold functions extend in the obvious ways to real threshold functions.

In most cases, we will know whether we are considering a Boolean or a real threshold function, and so we shall often just use the terminology "threshold function."

3.3 Properties of threshold functions

3.3.1 Closure under projection

From any Boolean function $f : \{0,1\}^n \to \{0,1\}$, we can form two functions $f\uparrow, f\downarrow$ from $\{0,1\}^{n-1}$ to $\{0,1\}$ as follows:

$$f\uparrow(x_1, x_2, \ldots, x_{n-1}) = f(x_1, x_2, \ldots, x_{n-1}, 1),$$

$$f\downarrow(x_1, x_2, \ldots, x_{n-1}) = f(x_1, x_2, \ldots, x_{n-1}, 0).$$

Thus, $f\uparrow$ is the restriction of f to the hyperface $x_n = 1$ of the Boolean hypercube and $f\downarrow$ is its restriction to the hyperface $x_n = 0$. We call $f\uparrow$ the *up-projection* and $f\downarrow$ the *down-projection* of f. We shall use these concepts at various points in the book. For the moment, we establish the following useful result, which shows that the up- and down-projections of a threshold function are threshold functions.

Theorem 3.1 *If t is a threshold function of n variables, then $t\uparrow$ and $t\downarrow$ are threshold functions of $n-1$ variables. Furthermore, suppose $t \leftarrow [w, \theta]$, where $w = (w^*, w_n)$, for $w^* \in \mathbb{R}^{n-1}$ and $w_n \in \mathbb{R}$. Then $t\uparrow \leftarrow [w^*, \theta - w_n]$ and $t\downarrow \leftarrow [w^*, \theta]$.*

Proof. Suppose $w = (w_1, w_2, \ldots, w_n)$. Then the results follow directly from the observations that

$$\begin{aligned} t\uparrow(x_1, x_2, \ldots, x_{n-1}) = 1 &\iff t(x_1, \ldots, x_{n-1}, 1) = 1 \\ &\iff \sum_{i=1}^{n-1} w_i x_i + w_n \geq \theta \\ &\iff \sum_{i=1}^{n-1} w_i x_i \geq \theta - w_n \end{aligned}$$

and

$$t{\downarrow}\,(x_1, x_2, \ldots, x_{n-1}) = 1 \iff t(x_1, \ldots, x_{n-1}, 0) = 1$$
$$\iff \sum_{i=1}^{n-1} w_i x_i \geq \theta.$$

□

The notion of projection can be made considerably more general: we can obtain a function of r variables from a function f of n variables by fixing $n - r$ of the variables in f. Formally, suppose that $a \in \{0, 1, \star\}^n$ and that r entries of a are equal to $\star$. (The entries of a equal to 1 or 0 will specify which of the n variables in f are fixed, and what they are fixed at.) Then, define $f_a : \{0,1\}^r \to \{0,1\}$ by

$$f(x_1 x_2 \ldots x_r) = f(x_1^a x_2^a \ldots x_n^a),$$

where, for $1 \leq i \leq n$,

$$x_i^a = \begin{cases} 1 & \text{if } a_i = 1, \\ 0 & \text{if } a_i = 0, \\ x_k & \text{if } a_i = \star \text{ and } k = |\{j : j \leq i,\ a_j = \star\}|. \end{cases}$$

Example 3.2 If $a = (10 \star 001 \star \star)$, then f_a is a function of three variables, and $f_a(x_1 x_2 x_3) = f(10 x_1 001 x_2 x_3)$. So if, for instance, f is the function represented by the formula $u_1 u_3 \vee \bar{u}_5 \bar{u}_7 \vee u_8$, then f_a is the Boolean function on $\{0,1\}^3$ with formula $u_1 \vee \bar{u}_2 \vee u_3$.

Generally, a *projection* of a Boolean function f of n variables is a function of the form f_a for some $a \in \{0, 1, \star\}^n$. The following result holds. Its proof is an extension of that of Theorem 3.1.

Theorem 3.3 *If t is a threshold function, then all projections of t are threshold functions.*

3.3.2 2-monotonicity

A threshold function will not generally be increasing. However, any threshold function is unate. In fact, something stronger is true: any increasing threshold function is 2-monotonic, from which it follows that, on negation, if necessary, of some literals, any threshold function becomes 2-monotonic.

Theorem 3.4 *Any increasing threshold function is 2-monotonic.*

Proof. Let us suppose that the weights are arranged so that

$$w_1 \geq w_2 \geq \cdots \geq w_n.$$

This chain of inequalities can be obtained simply be permuting the variables. We show that, under this assumption, the function is regular. This will establish the 2-monotonicity, because a function is 2-monotonic if and only if its variables can be permuted to obtain a regular function. Now t is given by

$$t(x) = \operatorname{sgn}\left(\sum_{i=1}^{n} w_i x_i - \theta\right) = \operatorname{sgn}\left(\langle w, x\rangle - \theta\right).$$

Clearly, if $i < j$, and $x_i = 0, x_j = 1$, then

$$\langle w, x + e_i - e_j \rangle - \langle w, x \rangle = w_i - w_j \geq 0,$$

so

$$t(x + e_i - e_j) = \operatorname{sgn}(\langle w, x + e_i - e_j \rangle - \theta) \geq \operatorname{sgn}(\langle w, x \rangle - \theta) = t(x).$$

This shows that t is regular, and the result follows. □

Note, however, that not every 2-monotonic function is a threshold function, as the following example illustrates.

Example 3.5 Suppose that $f : \{0,1\}^{10}$ is the function defined as follows: $f(x) = 1$ if and only if

$$x_1 + x_2 + \cdots + x_{10} \geq 7$$

and

$$34x_1 + 29x_2 + 9x_3 + 7x_4 + 5x_5 + 5x_6 + 4x_7 + 3x_8 + 3x_9 + x_{10} \geq 50.$$

Thus, t is the conjunction of two threshold functions. Then f is regular (and, hence, 2-monotonic). This is quite easy to see. We simply note that if $i < j$ and x has $x_i = 0, x_j = 1$, then because the coefficients on the left-hand side of each linear inequality (that is, the weights) are nonincreasing, the left-hand sides cannot decrease if we replace x_i with 1 and x_j with 0. Since f is true if and only if each inequality is true, this observation implies that $f(x + e_i - e_j) \geq f(x)$, and so the function is regular. The function is not, however, a threshold function. To show this is not quite so easy. In theory, we would have to show that the two linear inequalities together are not equivalent to a single linear inequality. Perhaps the easiest way to do so is to prove it by contradiction, working with particular true and false points. Suppose then that f was a threshold function, represented by weight vector $w \in \mathbb{R}^{10}$ and threshold θ. Now, we note that if

$$a = 0111110101, \quad b = 1011111010, \quad c = 1111110000, \quad d = 0011111111,$$

then

$$f(a) = 1, f(b) = 1, f(c) = 0, f(d) = 0.$$

This means that

$$\langle w, a \rangle \geq \theta, \langle w, b \rangle \geq \theta, \langle w, c \rangle < \theta, \langle w, d \rangle < \theta.$$

But we should then have

$$\langle w, a + b \rangle = \langle w, a \rangle + \langle w, b \rangle \geq 2\theta, \quad \langle w, c + d \rangle = \langle w, c \rangle + \langle w, d \rangle < 2\theta.$$

However, this cannot be, since $a + b = c + d$. We therefore have a contradiction, and it follows that f is *not* a threshold function.

This example demonstrates some interesting features of threshold functions. First, the conjunction of threshold functions is not necessarily a threshold function. Second, there are regular functions that are not threshold functions. Third, it has demonstrated one possible technique for proving that a function is not a threshold function. (This is a technique we shall discuss further later in this chapter.)

3.4 Geometrical interpretation of threshold functions

3.4.1 Linear separability

Sometimes the threshold functions (real or Boolean) are referred to as *linearly separable functions*. This is because the true points (those for which $t(x) = 1$) and the false points (those with $t(x) = 0$) can be separated by a hyperplane. Explicitly, if t is represented by weight vector w and threshold θ then the hyperplane ℓ defined by the inequality

$$w_1x_1 + w_2x_2 + \cdots + w_nx_n = \theta$$

separates the true points from the false, because the true points lie in the closed half-space ℓ^+ defined by the inequality

$$w_1x_1 + w_2x_2 + \cdots + w_nx_n \geq \theta$$

and the false points lie on the other side of the hyperplane, in the open half-space ℓ^- with equation

$$w_1x_1 + w_2x_2 + \cdots + w_nx_n < \theta.$$

For a Boolean threshold function, it follows from an earlier observation that we can always choose w and θ in such a way that no point of $\{0,1\}^n$ lies on the hyperplane. For real threshold functions, this cannot, of course, be avoided.

Homogeneous threshold functions (Boolean or real) are those for which θ can be chosen to be 0, which means that the corresponding hyperplane passes through the origin.

3.4.2 Convexity and threshold functions

Suppose that f is a function from $\mathbb{R}^n$ to $\{0,1\}$. (This includes, as a restriction, the case of a Boolean function.) Then f is a (real) threshold function if and only if the set of true points $T(f) = \{x : f(x) = 1\}$ of f and the set $F(f) = \{x : f(x) = 0\}$ of false points of f are linearly separable. In other words, f is a threshold function if and only if there is some hyperplane ℓ such that $T(f) \subseteq \ell^+$ and $F(f) \subseteq \ell^-$. The following alternative characterization is quite useful. For this, we need the notion of convex hull. A subset C of $\mathbb{R}^n$ is *convex* if, given any two points x, y of C, the line segment between x and y lies entirely in C. More formally, C is convex if given any x, y in C and any real number λ with $0 \leq \lambda \leq 1$, the point $\lambda x + (1 - \lambda)y$ belongs to C. It is clear that the intersection of any collection of convex sets is again convex and therefore for any nonempty set S of points of $\mathbb{R}^n$, there is a smallest convex set containing S. This set, denoted by $\mathrm{conv}(S)$, is called the *convex hull* of S; $\mathrm{conv}(S)$ is the intersection of all convex sets containing S. For example, when S is a finite set of points in the plane $\mathbb{R}^2$, $\mathrm{conv}(S)$ is the smallest closed region which is bounded by a polygon and which contains S. The convex hull of a set S has an explicit description in terms of the members of S. A *convex combination* of $S \subseteq \mathbb{R}^n$ is an element of $\mathbb{R}^n$ of the form

$$\alpha_1x_1 + \alpha_2x_2 + \cdots + \alpha_nx_k$$

for some $k \in \mathbb{N}$ and $\alpha_i \in [0,1]$ (for $1 \leq i \leq k$), where $x_1, \ldots, x_k \in S$ and $\sum_{i=1}^{k} \alpha_k = 1$. The set of all convex combinations of S is a convex set containing S and any convex set containing S must contain all convex combinations of S, so it follows that $\mathrm{conv}(S)$ is precisely the set of all convex combinations of S.

Theorem 3.6 *Suppose that* $f : \mathbb{R}^n \to \mathbb{R}$ *and denote by* $T(f)$ *the set of* x *for which* $x = 1$ *and by* $F(f)$ *the set of* x *for which* $f(x) = 0$. *Then* f *is a threshold function if and only if*

$$\text{conv}\,(T(f)) \cap \text{conv}\,(F(f)) = \emptyset.$$

Proof. If $\text{conv}(T(f))$ and $\text{conv}(F(f))$ are disjoint, then it follows by the classical "Separating Hyperplanes Theorem" that there is some hyperplane separating $T(f)$ and $F(f)$, and hence that f is a threshold function. On the other hand, suppose that f is a threshold function. Then there is some hyperplane ℓ such that $T(f) \subseteq \ell^+$ and $F(f) \subseteq \ell^-$. Noting that ℓ^+ and ℓ^- are convex sets (and therefore are equal to their convex hulls), we have $\text{conv}(T(f)) \subseteq \ell^+$ and $\text{conv}(F(f)) \subseteq \ell^-$, and hence

$$\text{conv}(T(f)) \cap \text{conv}(F(f)) \subseteq \ell^+ \cap \ell^- = \emptyset,$$

establishing the necessity of the condition. □

3.5 Threshold functions and asummability

Part of the following result was suggested above in Example 3.5. The result provides a necessary and sufficient condition for a function to be a threshold function and, in particular, yields a way of proving that a function is not a threshold function (as we saw in Example 3.5).

Theorem 3.7 *The Boolean function* f *is a threshold function if and only if it is* asummable, *meaning that for any* $k \in \mathbb{N}$, *for any sequence* $x_1, x_2, \ldots, x_k$ *of (not necessarily distinct) true points of* f *and any sequence* $y_1, y_2, \ldots, y_k$ *of (not necessarily distinct) false points of* f,

$$\sum_{i=1}^{k} x_i \neq \sum_{i=1}^{k} y_i.$$

Proof. The proof of one half of Theorem 3.7 is easy (and the germ of it was presented in Example 3.5). If the x_i and y_i are as in the statement of the theorem, and if f were a threshold function, represented by weight vector w and threshold θ, then we should have $\langle w, x_i \rangle \geq \theta$ and $\langle w, y_i \rangle < \theta$ for $i = 1, \ldots, k$. But then

$$\left\langle w, \sum_{i=1}^{k} x_i \right\rangle \geq k\theta > \left\langle w, \sum_{i=1}^{k} y_i \right\rangle,$$

implying that $\sum_{i=1}^{k} x_i \neq \sum_{i=1}^{k} y_i$.

The proof of the other half of the theorem can be achieved using a standard "duality result" (or "theorem of the alternative"), namely that for a matrix A, precisely one of the following two systems of inequalities has a solution:

$$Az > \mathbf{0}, \qquad \begin{cases} A^T x = \mathbf{0}, \\ x \geq \mathbf{0}, x \neq \mathbf{0}. \end{cases}$$

Here, $\mathbf{0}$ denotes the all-0 vector of appropriate dimension, $b \geq a$ means $b_i \geq a_i$ for each i, and $b > a$ means $b_i > a_i$ for each i. Thus, if there is no solution z to $Az > \mathbf{0}$, then there is an x with nonnegative entries and with $x \neq \mathbf{0}$, such that $A^T x = \mathbf{0}$. Moreover,

if the matrix A^T has rational entries, then such an x can be found that has rational numbers as its entries. (Because A^T has rational entries, the Gauss–Jordan procedure for finding a basis of the solution set to $A^T x = \mathbf{0}$ will result in rational vectors.) Thus, there will in fact be a solution u with integer entries.

Now suppose that f is not a threshold function. Suppose that the true points of f are $x_1, x_2, \ldots, x_r$ and the false points are $y_1, y_2, \ldots, y_s$. Then, for each pair i, j with $1 \leq i \leq r$ and $1 \leq j \leq s$, let us form $x_i - y_j$, and denote the resulting rs elements of $\mathbb{R}^n$ by $c_1, c_2, \ldots, c_l$, where $l = rs$. Then the matrix inequality

$$\begin{pmatrix} c_1^T \\ c_2^T \\ \vdots \\ c_l^T \end{pmatrix} z > \mathbf{0}$$

has no solutions. For, if it did, we would have $\langle z, x_i \rangle > \langle z, y_j \rangle$ for $1 \leq i \leq r, 1 \leq j \leq s$. The true and false points could then be separated by the hyperplane with equation $\langle z, x \rangle = \theta$, where $\theta = \min_{1 \leq i \leq r} \langle z, x_i \rangle$. We know this cannot be, since f is not a threshold function. By the duality result, there therefore exists $u \in \mathbb{R}^l \setminus \{\mathbf{0}\}$ with integer entries such that

$$(c_1 \quad c_2 \quad \cdots \quad c_l)\, u = \mathbf{0},$$

so there are nonnegative integers n_{ij}, one for each pair i, j with i between 1 and r and j between 1 and s, and not all equal to 0, such that

$$\sum_{i,j} n_{ij}(x_i - y_j) = \mathbf{0}.$$

Rearranging this equation by taking all y_j to the right-hand side yields equal integer linear combinations of the x_i and the y_j, in which the sums of the coefficients on each side are the same. This shows that f is *not* asummable. (Note that, generally, these sums will involve repetitions of some true and false points.) □

Another view of the sufficiency of asummability arises by drawing some connection with the previous characterization, Theorem 3.6, of threshold functions. We first define a slightly different type of convex hull, which we call the *rational convex hull*. We have noted that the convex hull of a set S may be described as the set of all convex combinations of S; that is,

$$\operatorname{conv}(S) = \left\{ \sum_{i=1}^{k} \alpha_i x_i : k \in \mathbb{N},\ x_i \in S,\ 0 \leq \alpha_i \leq 1\ (i = 1, \ldots, k),\ \sum_{i=1}^{k} \alpha_i = 1 \right\}.$$

We define the rational convex hull $\operatorname{conv}^{\mathbb{Q}}(S)$ in a similar manner, but allowing only rational values of the α_i:

$$\operatorname{conv}^{\mathbb{Q}}(S) = \left\{ \sum_{i=1}^{k} \alpha_i x_i : k \in \mathbb{N},\ x_i \in S,\ \alpha_i \in [0,1] \cap \mathbb{Q}\ (i = 1, \ldots, k),\ \sum_{i=1}^{k} \alpha_i = 1 \right\}.$$

Asummability, namely the nonexistence of the sequences described in Theorem 3.7, can be seen to be equivalent to the assertion that $\operatorname{conv}^{\mathbb{Q}}(T(f))$ and $\operatorname{conv}^{\mathbb{Q}}(F(f))$ are disjoint. (In understanding this, it is important to realize that the sequences of positive and negative examples referred to in the theorem may contain repetititions.) It turns

out generally to be the case that if $S_1, S_2 \subseteq \mathbb{Q}^n$, then $\text{conv}(S_1) \cap \text{conv}(S_2) = \emptyset$ if and only if $\text{conv}^{\mathbb{Q}}(S_1) \cap \text{conv}^{\mathbb{Q}}(S_2) = \emptyset$. In particular, therefore, the asummability criterion given by Theorem 3.7 is quite directly equivalent, for the Boolean function case, to Theorem 3.6.

Example 3.8 Consider again Example 3.5, which showed that there were true points a, b and false points c, d such that $a + b = c + d$. By Theorem 3.7, the function is not linearly separable.

3.6 1-Decision lists are threshold functions

We turn attention again to decision lists. We have already seen that these provide a convenient, and reasonably powerful, way of representing Boolean functions. At this stage, we present the following result.

Theorem 3.9 *Any 1-decision list (that is, a decision list based over the set K of single literals) is a threshold function.*

Proof. We prove this by induction on the number of terms in the decision list. Suppose, for the base case of the induction, that a decision list has just one term and is of the form $(u_i, 1)$, or $(\bar{u}_i, 1)$, or $(\mathbf{1}, 1)$, where $\mathbf{1}$ is the identically-1 function (the monomial of length 0). (Note that if it were of the form $(u_i, 0)$, $(\bar{u}_i, 0)$, or $(\mathbf{1}, 0)$, then, since a decision list outputs 0 by default, the term would be redundant, and the decision list would compute the identically-0 function, which is certainly a threshold function.) In the first case, the function may be represented as a threshold function by taking the weight-vector to be $(0, \ldots, 0, 2, 0, \ldots, 0)$, where the nonzero entry is in position i, and by taking the threshold to be 1. In the second case, we may take weight-vector $(0, \ldots, 0, -2, 0, \ldots, 0)$ and threshold -1. In the third case, the function is the identically-1 function, and we may take as weight-vector the all-0 vector, and threshold 0. Assume, as the inductive hypothesis, that any decision list of length r is a threshold function, and suppose we have a decision list of length $r + 1$,

$$f = (\ell_1, c_1), (\ell_2, c_2), \ldots, (\ell_{r+1}, c_{r+1}),$$

where each ℓ_i is a literal, possibly negated. We shall assume, without any loss of generality (for one can simply rename the variables or, equivalently, permute the entries of the weight vector), that $\ell_1 = u_1$ or $\bar{u}_1$. By the induction hypothesis, the decision list

$$(\ell_2, c_2), \ldots, (\ell_{r+1}, c_{r+1})$$

is a threshold function. Suppose it is represented by weight-vector $w = (w_1, \ldots, w_n)$ and threshold θ, and let $\|w\|_1 = \sum_{i=1}^{n} |w_n|$ be the 1-norm of w. There are four possibilities for (ℓ_1, c_1), as follows:

1. $(u_1, 1)$,
2. $(u_1, 0)$,
3. $(\bar{u}_1, 1)$,
4. $(\bar{u}_1, 0)$.

Denoting by e_1 the vector $(1, 0, \ldots, 0)$, and letting $M = \|w\|_1 + |\theta| + 1$, we claim that the decision list f is a threshold function represented by the weight-vector w' and threshold θ', where, respectively,

1. $w' = w + Me_1, \quad \theta' = \theta,$
2. $w' = w - Me_1, \quad \theta' = \theta,$
3. $w' = w - Me_1, \quad \theta' = \theta - M,$
4. $w' = w + Me_1, \quad \theta' = \theta + M.$

This claim is easy to verify in each case. Consider, for example, the third case. For $x \in \{0,1\}^n$,

$$\langle w', x \rangle = \langle w - Me_1, x \rangle = \langle w, x \rangle - Mx_1,$$

and therefore $\langle w', x \rangle \geq \theta' = \theta - M$ if and only if

$$\langle w, x \rangle - Mx_1 \geq \theta - M.$$

If $x_1 = 0$ (in which case the decision list outputs 1), this inequality becomes $\langle w, x \rangle \geq \theta - M$. Now, for any $x \in \{0,1\}^n$, $-\|w\|_1 \leq \langle w, x \rangle \leq \|w\|_1$, and

$$\theta - M = \theta - (\|w\|_1 + |\theta| + 1) = -\|w\|_1 - 1 + (\theta - |\theta|) \leq -\|w\|_1 - 1 < -\|w\|_1,$$

so in this case the inequality is certainly satisfied, and the output of the threshold function is 1, equal to the output of the decision list. Now suppose that $x_1 = 1$. Then the inequality

$$\langle w, x \rangle - Mx_1 \geq \theta - M$$

becomes $\langle w, x \rangle - M \geq \theta - M$, which is $\langle w, x \rangle \geq \theta$. But, by the inductive assumption, the decision list

$$f' = (\ell_2, c_2), \ldots, (\ell_{r+1}, c_{r+1})$$

is a threshold function represented by the weight-vector w and threshold θ. So in this case, the output of the threshold function is 1 if and only if the output of decision list f' is 1, which is exactly how f calculates its output in this case. So we see that this representation is indeed correct. The other cases can be verified similarly. □

3.7 Polynomial threshold functions

We now turn our attention to polynomial threshold functions, a natural generalization of real and Boolean threshold functions.

3.7.1 Real polynomial threshold functions

We recall the definition of a polynomial threshold function defined on $\mathbb{R}^n$, which we shall refer to as a *real* polynomial threshold function. (It will be useful to distinguish between real polynomial threshold functions and those defined on $\{0,1\}^n$, which we shall term Boolean threshold functions.)

Recall that we denote by $[n]^m$ the set of all selections, in which repetition is allowed, of at most m objects from the set $[n] = \{1, 2, \ldots, n\}$, so that $[n]^m$ is a collection of "multisets." Then, a function $f : \mathbb{R}^n \to \{0,1\}$ is a polynomial threshold unit of degree

m if there are $p = \binom{n+m}{m}$ parameters w_S, one for each multiset $S \in [n]^m$, such that for all $x \in \mathbb{R}^n$,

$$f(x) = \operatorname{sgn}\left(\sum_{S \in [n]^m} w_S x_S\right).$$

Here, for $S \in [n]^m$ and $x \in \mathbb{R}^n$, x_S is the product of the entries x_i of x for $i \in S$ (with repetitions as required). (We take $x_\emptyset = 1$.)

3.7.2 Boolean polynomial threshold functions

Our main concern will be with Boolean polynomial threshold functions, which we shall refer to simply as polynomial threshold functions. As we have already noted, we can achieve a degree of simplification in the Boolean case by observing that, since $x^r = x$ for all $r \in \mathbb{N}$ if $x \in \{0,1\}$, we can dispense with terms x_S in which S contains repetitions. Let us denote by $[n]^{(m)}$ the set of all subsets of at most m objects from $[n] = \{1, 2, \ldots, n\}$. Thus, for example, $[3]^{(2)}$ consists of the sets

$$\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\},$$

and, in general, $[n]^{(m)}$ consists of $\sum_{i=0}^{m} \binom{n}{i}$ sets. As above, for each $S \in [n]^{(m)}$, and for any $x = (x_1, x_2, \ldots, x_n) \in \{0,1\}^n$, x_S will denote the product of the x_i for $i \in S$ (with the usual convention that $x_\emptyset = 1$). Then, a Boolean function f defined on $\{0,1\}^n$ is a polynomial threshold function of degree m if there are constants w_S, one for each $S \in [n]^{(m)}$, such that

$$f(x) = 1 \iff \sum_{S \in [n]^{(m)}} w_S x_S > 0$$

or, equivalently,

$$f(x) = \operatorname{sgn}\left(\sum_{S \in [n]^{(m)}} w_S x_S\right).$$

The set of all polynomial threshold functions on $\{0,1\}^n$ of degree m will be denoted by $T(n,m)$.

3.7.3 Closure under projections

Just as for linear threshold functions, we have closure under projection for the class of polynomial threshold functions.

Theorem 3.10 *The projection of any Boolean polynomial threshold function of degree m is a Boolean polynomial threshold function of degree m.*

The proof is straightforward: it follows from the fact that if $p(x_1, x_2, \ldots, x_n)$ is a polynomial of n variables, then fixing some of these variables equal to either 0 or 1 gives another polynomial in the remaining variables. The degree of this polynomial is no more than that of p (and it can, in fact, be of a considerably lower degree).

3.8 Polynomial representation of Boolean functions

An important observation is that every Boolean function is a polynomial threshold function. This follows almost immediately from the existence of a DNF formula for each Boolean function. Explicitly, suppose that $f : \{0,1\}^n \to \{0,1\}$ and that f has DNF representation

$$T_1 \vee T_2 \vee \cdots \vee T_k,$$

where each T_l is a term of the form

$$T_l = \left(\bigwedge_{i \in P} u_i \right) \bigwedge \left(\bigwedge_{j \in N} \bar{u}_j \right)$$

for some disjoint subsets P, N of $\{1, 2, \ldots, n\}$. For each term T_l, let us form a polynomial $p_l(x)$ by replacing u_i (for $i \in P$) by x_i and $\bar{u}_j$ (for $j \in N$) by $(1 - x_j)$. Consider $p(x) = p_1(x) + p_2(x) + \cdots + p_k(x)$. This is a polynomial expression in the variables $x_1, x_2, \ldots, x_n$ and it is easy to see that $p(x) \geq 1$ if and only if at least one of the terms T_l is true for x; in other words, $p(x) \geq 1$ if and only if $f(x) = 1$. It follows that

$$f(x) = \text{sgn}(p(x) - 1),$$

showing that f is a polynomial threshold function. This construction also demonstrates that if f is an l-DNF function (so that the number of literals in each term is no more than l), then f is a polynomial threshold function of degree l.

Example 3.11 The function f with DNF representation

$$\bar{u}_1 u_4 \vee u_1 u_2 \bar{u}_3$$

may be written as a polynomial threshold function as follows:

$$\begin{aligned} f(x_1, x_2, x_3, x_4) &= \text{sgn}\left((1 - x_1)x_4 + x_1 x_2 (1 - x_3) - 1\right) \\ &= \text{sgn}\left(x_4 - x_1 x_4 + x_1 x_2 - x_1 x_2 x_3 - 1\right). \end{aligned}$$

If f is a polynomial threshold function of degree m, but is not a polynomial threshold function of degree $(m-1)$, then we say that f has *threshold order* m. We have seen that every Boolean function is a polynomial threshold function. The threshold order of a Boolean function f is the minimal degree of a polynomial threshold function representing f. The analysis above shows that the threshold order is always no more than n if the function is defined on $\{0,1\}^n$, and that if the function is an l-DNF function, then its threshold order is no more than l. In a later chapter, we shall explore further the threshold order of a Boolean function.

We can also draw a connection between decision lists and polynomial threshold functions. We have already seen (Theorem 3.9) that 1-decision lists are (linear) threshold functions. It can be proved in an analogous way that if a function f is a decision list based on monomials of length no more than l, then f is a polynomial threshold function of degree l. In other words, if f is an l-DL, then the threshold order of f is at most l.

3.9 Geometrical interpretation of polynomial threshold functions

We have already observed that a function is a linear threshold function if and only if the true points can be separated from the false points by a hyperplane. For a polynomial threshold function of degree m we have the corresponding geometrical characterization that the true points can be separated from the false points by a surface whose equation is a polynomial of degree m.

It is possible to relate such polynomial separation to linear separation in a higher-dimensional space, as we now explore. We focus on the Boolean case, though a similar analysis applies to the case of real polynomial threshold functions.

For $x \in \{0,1\}^n$, we define the *m-augment*, $x^{(m)}$, of x to be the $\{0,1\}$-vector of length $\sum_{i=1}^{m} \binom{n}{i}$ whose entries are x_S for $S \in [n]^{(m)}$ in some prescribed order. To be precise, we shall suppose the entries are in order of increasing degree and that terms of the same order are listed in *lexicographic (dictionary) order*. Thus, for example, when $n = 5$ and $m = 2$,

$$x^{(5)} = (x_1, x_2, x_3, x_4, x_5, x_1x_2, x_1x_3, x_1x_4, x_1x_5, x_2x_3, x_2x_4, x_2x_5, x_3x_4, x_3x_5, x_4x_5).$$

We observe that a Boolean function f is a polynomial threshold function of degree m if and only if there is some linear threshold function h_f, defined on $\{0,1\}$ vectors of length $r = \sum_{i=1}^{m} \binom{n}{i}$, such that

$$f(x) = 1 \iff h_f(x^{(m)}) = 1;$$

that is, if and only if the m-augments of the true points of f and the m-augments of the false points of f can be separated by a hyperplane in the higher-dimensional space $\mathbb{R}^r$, where $r = \sum_{i=1}^{m} \binom{n}{i}$.

3.10 Asummability and polynomial threshold functions

The m-augments can be used to provide an asummability criterion similar to Theorem 3.7.

We say that f is m-asummable if for any $k \in \mathbb{N}$, for any sequence $x_1, x_2, \ldots, x_k$ of (not necessarily distinct) true points of f and any sequence $y_1, y_2, \ldots, y_k$ of (not necessarily distinct) false points of f,

$$\sum_{i=1}^{k} x_i^{(m)} \neq \sum_{i=1}^{k} y_i^{(m)}.$$

Note that if f is m-asummable then f is m'-asummable for any $m' > m$. The following result holds.

Theorem 3.12 *The Boolean function f is a threshold function of degree m if and only if f is m-asummable.*

3.11 Additional remarks and bibliographical notes

It has been known for some time that there are 2-monotonic nonthreshold functions; see [75]. Example 3.5 is described by Taylor and Zwicker [99] as arising from a model of the Canadian electoral system.

We mentioned earlier that Minsky and Papert's notion of perceptron included Boolean polynomial threshold functions. The class $T(n, m)$ is, in their terminology [72], the set of functions computable by a *mask perceptron* of order at most m, on *retina* $\{0,1\}^n$.

The idea of transforming polynomial separation into linear separation of augmented vectors may be found in [29]. More generally, the idea of linear separation of the "transforms" of points in higher-dimensional spaces is central to the Support Vector Machine; see [32], for example.

The proof that 1-decision lists are threshold functions follows a similar proof in [10].

The characterization of threshold functions in terms of asummability can be found in [75]. Its generalization, Theorem 3.12, is due to Wang and Williams [104].

This page intentionally left blank

Chapter 4

Number of Threshold Functions

4.1 Introduction

We have seen that threshold functions form a very special class of Boolean function, and the question of how extensive a subclass of Boolean functions they constitute naturally arises. In this chapter, we discuss some well-known upper and lower bounds on the number of threshold functions, and also some more recent results on the number of polynomial threshold functions of a given degree.

4.2 Number of threshold functions: Upper bounds

In this section we shall derive an upper bound on the number of threshold functions. Recall that for any $f \in T_n$, we can form the *up-projection* $f\uparrow$ and the *down-projection* $f\downarrow$ as follows:

$$f\uparrow(x_1, x_2, \ldots, x_{n-1}) = f(x_1, x_2, \ldots, x_{n-1}, 1),$$

$$f\downarrow(x_1, x_2, \ldots, x_{n-1}) = f(x_1, x_2, \ldots, x_{n-1}, 0).$$

Recall also that these are both threshold functions, and that if $f \leftarrow [w, \theta]$, where $w = (w^*, w_n)$, $w^* \in \mathbb{R}^{n-1}$, and $w_n \in \mathbb{R}$, then $f\uparrow \leftarrow [w^*, \theta - w_n]$ and $f\downarrow \leftarrow [w^*, \theta]$.

Consider any threshold function $f : \{0,1\}^n \to \{0,1\}$, and suppose that $f \leftarrow [w, \theta]$, so that

$$f(x) = \operatorname{sgn}\left(\sum_{i=1}^{n} w_i x_i - \theta\right).$$

Suppose $g \leftarrow [(w, -\theta), 0]$ is the homogeneous threshold function defined on $\{0,1\}^{n+1}$ by

$$g(x) = g(x_1, x_2, \ldots, x_n, x_{n+1}) = \operatorname{sgn}\left(\sum_{i=1}^{n} w_i x_i - \theta x_{n+1}\right).$$

Then f is the up-projection $g\uparrow$; in other words,

$$f(x_1, x_2, \ldots, x_n) = g(x_1, x_2, \ldots, x_n, 1).$$

Therefore, if we obtain an upper bound on the number of functions which are up-projections of homogeneous threshold functions of $n+1$ variables, then that upper bound is also an upper bound on the number of threshold functions on n variables.

A bound on the number of functions in T_n can be obtained using geometrical arguments. We have already noted that each $f \in T_n$ is the up-projection of some homogeneous threshold function in T_{n+1}. The set of possible weight-vectors for a homogeneous threshold function on $\{0,1\}^{n+1}$ is $\mathbb{R}^{n+1}$; in other words, the *parameter space* is $\mathbb{R}^{n+1}$. Consider the 2^n hyperplanes, one for each $x \in \{0,1\}^n$, with equations $(x^T,1)w' = 0$. Writing the typical $w' \in \mathbb{R}^{n+1}$ as $(w,-\theta)$, where $w \in \mathbb{R}^n$ and $\theta = -w_{n+1}$, we see that each such hyperplane divides parameter space $\mathbb{R}^{n+1}$ into two half-spaces: on one side, we have all weight-vectors $w' = (w,-\theta)$ such that $(x^T,1)w' > 0$, or $x^T w > \theta$, and on the other, all weight-vectors such that $(x^T,1)w' < 0$, or $x^T w < \theta$. Now, these 2^n hyperplanes divide $\mathbb{R}^{n+1}$ into a number of connected regions. Formally, these regions are the connected components of

$$\mathbb{R}^{n+1} \setminus \bigcup_{x \in \{0,1\}^n} \{w' \in \mathbb{R}^{n+1} : (x^T,1)w' = 0\}.$$

Suppose that $w' = (w,-\theta)$ and $u' = (u,-\phi)$ in $\mathbb{R}^{n+1}$ belong to the same region, so that for each $x \in \{0,1\}^n$, $(x^T,1)w'$ and $(x^T,1)u'$ are either both positive or both negative. Then, $\mathrm{sgn}(x^T w - \theta) = \mathrm{sgn}(x^T u - \phi)$ for all $x \in \{0,1\}^n$. But this means that the homogeneous threshold functions represented by weight-vectors w' and u' have exactly the same up-projection. It follows that to bound the number of different threshold functions on $\{0,1\}^n$, it suffices to bound the number of such regions. The following classical result can be used. (This result is, in fact, more general than we need, but we state it in this generality because it is quite easy to prove, and because we shall use the general result later in the book.)

Theorem 4.1 *The maximum number of connected regions into which $\mathbb{R}^d$ can be partitioned by N hyperplanes passing through the origin is bounded above by*

$$C(N,d) = 2\sum_{k=0}^{d-1} \binom{N-1}{k},$$

with the usual convention that $\binom{a}{b} = 0$ if $b > a$, and $\binom{0}{b} = 1$.

Proof. We prove this by induction on N. For the case $N = 1$, $C(1,d) = 2$, and this is precisely the number of regions into which one hyperplane divides $\mathbb{R}^d$. Suppose the bound holds for $N-1$ planes, and consider the introduction of an additional, Nth plane. Let's call this plane P. Now, P intersects the other $N-1$ hyperplanes in at most $N-1$ hyperplanes of dimension $d-2$. These $(d-2)$-dimensional planes pass through the origin. By the inductive assumption (identifying P with $\mathbb{R}^{d-1}$), these $(d-2)$-dimensional hyperplanes partition P into no more than $C(N-1,d-1)$ regions. Consider what happens to the regions defined by the initial $N-1$ hyperplanes. (There are at most $C(N-1,d)$ of these, by the inductive assumption.) Some of these regions might remain intact, but some will be subdivided into new regions by the introduction of P. However, each initial region can be split into at most two parts by P. Since the number of regions of P defined by the $N-1$ initial hyperplanes is bounded by $C(N-1,d-1)$, the number of regions so divided is no more than $C(N-1,d-1)$. Hence, the number of additional regions created by introducing P is no more than $C(N-1,d-1)$. Thus, the maximum number of regions into which $\mathbb{R}^d$ can be partitioned by N hyperplanes is no more than

$$\begin{aligned} C(N-1,d)+C(N-1,d-1) &= 2\sum_{k=0}^{d-1}\binom{N-2}{k}+2\sum_{k=0}^{d-2}\binom{N-2}{k} \\ &= 2\left(\binom{N-2}{0}+\sum_{i=1}^{d-1}\left(\binom{N-2}{i}+\binom{N-2}{i-1}\right)\right) \\ &= 2\left(1+\sum_{i=1}^{d-1}\binom{N-1}{i}\right) \\ &= C(N,d). \end{aligned}$$

□

In fact, the bound of Theorem 4.1 is tight, in the sense that equality is obtained if the N hyperplanes have normal vectors with the property that no $d+1$ of them lie in a $(d-1)$-dimensional hyperplane; that is, if they are in *general position.*

As a corollary of Theorem 4.1, we have the following upper bound on the number $|T_n|$ of threshold functions on $\{0,1\}^n$.

Theorem 4.2 *The number, $|T_n|$, of threshold functions mapping from $\{0,1\}^n$ to $\{0,1\}$ is bounded as follows:*

$$|T_n| \leq 2\sum_{k=0}^{n}\binom{2^n-1}{k}.$$

Proof. We have already observed that the number of threshold functions on $\{0,1\}^n$ is bounded by the number of connected components of $\mathbb{R}^{n+1}$ defined by the 2^n planes with normal vectors $(x^T,1)$, for $x \in \{0,1\}^n$. By Theorem 4.1, this is bounded by $C(2^n,n+1)$, from which the result follows. □

The following looser bound is useful in providing an indication of how relatively few threshold functions there are.

Theorem 4.3 *The number $|T_n|$ of threshold functions on $\{0,1\}^n$ satisfies*

$$|T_n| \leq 2^{n^2},$$

for all n.

Proof. A useful bound, which we shall use later, is the following: for $m \geq d$,

$$\sum_{i=0}^{d}\binom{m}{i} < \left(\frac{em}{d}\right)^d.$$

To prove this, we have

$$\begin{aligned} \sum_{i=0}^{d}\binom{m}{i} &< \left(\frac{m}{d}\right)^d\sum_{i=0}^{d}\binom{m}{i}\left(\frac{d}{m}\right)^d \\ &\leq \left(\frac{m}{d}\right)^d\sum_{i=0}^{d}\binom{m}{i}\left(\frac{d}{m}\right)^i \\ &\leq \left(\frac{m}{d}\right)^d\sum_{i=0}^{m}\binom{m}{i}\left(\frac{d}{m}\right)^i \\ &= \left(\frac{m}{d}\right)^d\left(1+\frac{d}{m}\right)^m. \end{aligned}$$

Now, for all $x > 0$, $(1 + (x/m))^m < e^x$, so this is bounded by $(m/d)^d e^d = (em/d)^d$, giving the bound.

Now, applying this bound with $m = 2^n - 1$ and $d = n$, we obtain

$$|T_n| \leq 2 \sum_{k=0}^{n} \binom{2^n - 1}{k} < 2 \left(\frac{e(2^n - 1)}{n} \right)^n .$$

For $n \geq 4$ this is less than $2(e/4)^4 2^{n^2} < 2^{n^2}$, and it is easily seen that when $n = 1, 2, 3$, the bound is also less than 2^{n^2}. Thus, $|T_n| < 2^{n^2}$. □

Note that, since there are 2^{2^n} Boolean functions on $\{0,1\}^n$, the threshold functions form a very small subclass of all Boolean functions.

The following slightly more general result is immediate from the analysis of this section.

Theorem 4.4 *Suppose that $S \subseteq \{0,1\}^n$, and let T_S be the set of functions in T_n restricted to domain S. Then,*

$$|T_S| \leq C(|S|, n+1) = 2 \sum_{k=0}^{n} \binom{|S| - 1}{k} < 2 \left(\frac{|S|e}{n} \right)^n .$$

4.3 Number of threshold functions: Lower bound

We have an upper bound on $|T_n|$ of 2^{n^2}. The following result gives a lower bound of the order of $\sqrt{2^{n^2}}$.

Theorem 4.5 *For all n, the number, $|T_n|$, of threshold functions on $\{0,1\}^n$ satisfies*

$$|T_n| \geq 2^{n(n-1)/2}.$$

Proof. The crucial part of this proof is showing that $|T_n| \geq (2^{n-1} + 1)|T_{n-1}|$ for all $n \geq 1$. To obtain this bound, we consider, for any fixed $f \in T_{n-1}$, how many $g \in T_n$ are such that $g\downarrow = f$. Suppose that $f \in T_{n-1}$. Suppose that the weight vector $w \in \mathbb{R}^{n-1}$ and threshold θ are such that $f \leftarrow [w, \theta]$ and also such that the numbers $w^T x$, as x ranges through $\{0,1\}^{n-1}$, are distinct. (This second condition is achievable by the discreteness of $\{0,1\}^{n-1}$: if it does not hold, we may perturb the weights in w slightly so that it does, but without changing the *sign* of $w^T x - \theta$ for any $x \in \{0,1\}^{n-1}$.) Suppose that $w' = (w, w_n) \in \mathbb{R}^n$, where $w_n \in \mathbb{R}$, and let g be the threshold function in T_n represented by weight-vector w' and threshold θ. Then it is easily seen that $g\downarrow = f$. Let the points of $\{0,1\}^{n-1}$ be denoted $x^{(1)}, x^{(2)}, \ldots, x^{(2^{n-1})}$, listed in such a way that

$$w^T x^{(1)} < w^T x^{(2)} < \cdots < w^T x^{(2^{n-1})}.$$

(This can be done by our assumption that the numbers $w^T x$ are distinct.) For any $x \in \{0,1\}^{n-1}$, let $x' = (x, 1) = (x_1, x_2, \ldots, x_{n-1}, 1) \in \{0,1\}^{n-1} \times \{1\}$. Then,

$$(w')^T x' = w^T x + w_n,$$

so

$$\operatorname{sgn}\left((w')^T x' - \theta\right) = \operatorname{sgn}\left(w^T x + w_n - \theta\right).$$

If we choose $w^T x^{(j)} < w_n < w^T x^{(j+1)}$ for some $j \in \{1, \ldots, 2^{n-1} - 1\}$, then $g \leftarrow [w', \theta]$ satisfies $g((x^{(i)}, 1)) = 0$ for $i \leq j$, and $g((x^{(i)}, 1)) = 1$ for $i \geq j + 1$. Similarly, choosing $w_n < w^T x^{(1)}$, we obtain a function in T_n which is identically 0 on $\{0,1\}^{n-1} \times \{1\}$, and if we choose $w_n > w^T x^{(2^{n-1})}$, we obtain a function in T_n which is identically 1 on $\{0,1\}^{n-1} \times \{1\}$. We see then that the flexibility in the choice of w_n leads to $2^{n-1} + 1$ different functions g in T_n, each of which satisfies $g \downarrow = f$. The fact that $|T_n| \geq (2^{n-1} + 1)|T_{n-1}|$ follows. Now we have, by iteration, and the fact that $|T_1| = 2$,

$$\begin{aligned} |T_n| &\geq \prod_{k=1}^{n} 2^{k-1} |T_1| \\ &\geq \prod_{k=1}^{n} 2^{k-1} \\ &= 2^{\sum_{k=1}^{n}(k-1)} \\ &= 2^{n(n-1)/2}, \end{aligned}$$

and the result follows. □

4.4 Asymptotic number of threshold functions

The upper and lower bounds obtained above imply that

$$\frac{n^2}{2} - \frac{n}{2} < \log_2 |T_n| < n^2.$$

In fact, the upper bound is tighter than the lower, as shown by the following improved lower bound.

Theorem 4.6 *Let T_n be the set of threshold functions on $\{0,1\}^n$. Then, for sufficiently large n,*

$$\log_2 |T_n| > n^2 \left(1 - \frac{10}{\ln n}\right).$$

The proof of the weaker lower bound, Theorem 4.5, was reasonably straightforward. By contrast, the proof of Theorem 4.6 is quite involved, and is omitted.

Using this new lower bound, we have (for n sufficiently large)

$$n^2 \left(1 - \frac{10}{\ln n}\right) < \log_2 |T_n| < n^2,$$

and so we obtain the asymptotic result

$$\frac{\log_2 |T_n|}{n^2} \to 1 \text{ as } n \to \infty,$$

or

$$\log_2 |T_n| \sim n^2 \text{ as } n \to \infty.$$

4.5 Number of polynomial threshold functions: Upper bounds

We can obtain an upper bound on the number of polynomial threshold functions of a given degree by using Theorem 4.1 and the proof of Theorem 4.2, together with the fact that a Boolean function is a polynomial threshold function of degree m if and only if the m-augments of true points and the m-augments of the false points are linearly separable.

Theorem 4.7 *The number, $T(n,m)$ of polynomial threshold functions of degree m on $\{0,1\}^n$ satisfies*

$$|T(n,m)| \leq C\left(2^n, \sum_{i=0}^{m}\binom{n}{i}\right) = 2\sum_{k=0}^{\sum_{i=1}^{m}\binom{n}{i}}\binom{2^n-1}{k}$$

for all m,n with $1 \leq m \leq n$.

Proof. We observed earlier that f is a polynomial threshold function of degree m if and only if there is some linear threshold function h_f, defined on $\{0,1\}$ vectors of length $r = \sum_{i=1}^{m}\binom{n}{i}$, such that

$$f(x) = 1 \iff h_f(x^{(m)}) = 1.$$

By Theorem 4.1 and by an argument similar to that given in the proof of Theorem 4.2 (and in the discussion surrounding these theorems), the number of polynomial threshold functions is therefore bounded above by $C(2^n, r+1)$, as required. □

It is fairly easy to deduce from this that $\log_2|T(n,m)|$ is at most $n\binom{n}{m} + O(n^m)$ as $n \to \infty$, with $m = o(n)$.

4.6 Number of polynomial threshold functions: Lower bounds

The following result can be used to obtain a lower bound on the number of threshold functions of a given degree.

Theorem 4.8 *Let $T(n,m)$ denote the set of all polynomial threshold functions of degree m on $\{0,1\}^n$. Then*

$$|T(n,m)| \geq |T(n-1,m)||T(n-1,m-1)|$$

for $2 \leq m \leq n-1$.

Proof. Let $f \in T(n-1,m)$ and $g \in T(n-1,m-1)$. Then there are polynomials p_f, p_g, on variables $x_1,\ldots,x_{n-1}$ and of degrees $m, m-1$, respectively, such that for

all $x \in \{0,1\}^{n-1}$, $f(x) = \text{sgn}(p_f(x))$ and $g(x) = \text{sgn}(p_g(x))$. We may also assume that $p_f(x), p_g(x)$ are never 0 for $x \in \{0,1\}^{n-1}$. (This is possible by the discreteness of $\{0,1\}^{n-1}$.) Consider the function h on $\{0,1\}^n$ given by $h(x) = \text{sgn}(p_h(x))$ where, writing $x \in \{0,1\}^n$ as $x = (x', x_n)$ with $x' \in \{0,1\}^{n-1}$,

$$p_h(x) = p_f(x') + Kx_n g(x')$$

for some positive K. If K is large enough, then, as can be checked, $h(x) = f(x')$ if $x_n = 0$ and $h(x) = g(x')$ if $x_n = 1$. (That is, the down-projection of h is f and the up-projection of h is g.) Now, p_h is a polynomial of degree at most m. This argument shows that for any $f \in T(n-1, m)$ and any $g \in T(n-1, m-1)$, there is a corresponding function $h \in T(n, m)$. Furthermore, since, in this correspondence, no h will arise from more than one (f, g) pair, it follows that $|T(n,m)| \geq |T(n-1,m)||T(n-1,m-1)|$, as required. □

Theorem 4.9 *The number, $|T(n,m)|$, of polynomial threshold functions of degree m on $\{0,1\}^n$ satisfies*

$$|T(n,m)| \geq 2^{\binom{n}{m+1}}$$

for all m, n with $1 \leq m \leq n-1$.

Proof. Let $\phi(n,m) = \log_2 |T(n,m)|$. By Theorem 4.8 and Theorem 4.5,

$$\phi(n,m) \geq \phi(n-1,m) + \phi(n-1,m-1)$$

and

$$\phi(n,1) \geq n(n-1)/2 = \binom{n}{2}.$$

The fact that $\phi(n,m) \geq \binom{n}{m+1}$ for all $1 \leq m \leq n-1$ can be proved by induction on $m+n$. The base case of the induction is $m+n = 3$, corresponding to the case $n = 2, m = 1$, and the result is true in this case, because $\log_2 |T(2,1)| = \log_2 2 = 1$, which equals $\binom{2}{2}$. For the general inductive step, consider a pair (m,n), with $1 \leq m \leq n-1$ and suppose that the lower bound on ϕ holds for the pairs $(m-1,n)$ and $(m-1,n-1)$, so that

$$\phi(n-1,m) \geq \binom{n-1}{m+1}, \quad \phi(n-1,m-1) \geq \binom{n-1}{m}.$$

(If $m = n-1$, then we interpret the first of these two binomial coefficients as 1, and we note that it is certainly true that $\phi(n-1,n-1) \geq 1$.) Then

$$\begin{aligned} \phi(n,m) &\geq \phi(n-1,m) + \phi(n-1,m-1) \\ &\geq \binom{n-1}{m+1} + \binom{n-1}{m} \\ &= \binom{n}{m+1}, \end{aligned}$$

and the result follows. □

Note that this lower bound is not at all tight for $m > n/2$. However, for constant m it provides a good match for the upper bound of Theorem 4.7. Taken together, the results imply that, for fixed m, for some positive constants c, k, $\log_2 |T(n,m)|$ is between cn^{m+1} and kn^{m+1}. Noting that n^{m+1} is substantially less than 2^n for large n, we see that, for fixed m, the class $T(n,m)$ forms a very small subclass of all Boolean functions.

4.7 Additional remarks and bibliographical notes

Theorem 4.1, the bound on the number of cells in a hyperplane arrangement goes back to work by Schläfli in the last century; see [92]. It also appears in [29]. The upper bound has been observed by a number of researchers; see [29]. The lower bound on $|T_n|$ in Theorem 4.5 is due to Muroga [75, 74]. The asymptotic behavior of $\log_2 |T_n|$ was determined by Zuev [106].

The inequality $\sum_{i=0}^{d} \binom{m}{i} < (\frac{em}{d})^d$ used in Theorem 4.3 appears in [20], and the proof presented here is from [26].

For Theorem 4.7, see [13, 5]. The lower bound, Theorem 4.8, is due to Saks [90].

Chapter 5

Sizes of Weights for Threshold Functions

5.1 Introduction

A weight-vector and threshold are said to be *integral* if the threshold and each entry of the weight-vector are integers. Any Boolean threshold function can be represented by an integral weight-vector and threshold. To see this, note first that, by the discreteness of $\{0,1\}^n$, any Boolean threshold function can be represented by a rational threshold and weight-vector. Scaling these by a suitably large integer yields integral threshold and weight-vector representing the same function. A natural question is how large the integer weights (including the threshold) have to be. In this chapter, we present some classic results from the 1960s: we give a general upper bound on the size of integer weights, and we show that integral weights must in some cases be exponentially large in n (proving this for a particular threshold function).

5.2 A general upper bound on size of weights

Suppose we are given a threshold function f and we want to find weights $w_1, \ldots, w_n$ and threshold θ such that $t \leftarrow [w, \theta]$. Then, any solutions w_i and θ to the following system of linear inequalities will suffice:

$$\begin{aligned} \langle w, x \rangle &\geq \theta, \quad x \in T(f), \\ \langle w, x \rangle &< \theta, \quad x \in F(f). \end{aligned}$$

For an equivalent set of linear inequalities, suppose first that we transform the points of $\{0,1\}^n$ as follows: $x \mapsto x^*$, where $x^* = x$ if $x \in T(f)$ and $x^* = -x$ if $x \in F(f)$. Then, for each x, let $x' = (x^*, 1) \in \{0,1\}^n \times \{1\}$ be x^* augmented by 1. Then, given any solution $w' \in \mathbb{R}^{n+1}$ to the system

$$\langle w', x' \rangle \geq 1, \quad x \in \{0,1\}^n, \tag{5.1}$$

we have a solution to the first system. Explicitly, if $w' = (w_1, w_2, \ldots, w_n, w_{n+1})$ satisfies system (5.1), we may take $w = (w_1, w_2, \ldots, w_n)$ and $\theta = -w_{n+1}$ to obtain a solution to the first system, because $\langle w', x' \rangle \geq 1$ implies $\langle w, x^* \rangle + w_{n+1} \geq 1$. For $x \in T(f)$,

this means $\langle w, x\rangle \geq 1 - w_{n+1} > \theta$. For $x \in F(f)$, it means $\langle w, -x\rangle \geq 1 - w_{n+1}$, or $\langle w, x\rangle \leq \theta - 1 < \theta$. Thus, to find integral weights and threshold, it suffices to find an integer solution to system (5.1). With this observation, we can now proceed with the main theorem.

Theorem 5.1 *For any Boolean threshold function f on $\{0,1\}^n$, there is an integral weight-vector w and an integral threshold θ such that $t \leftarrow [w, \theta]$ and such that*

$$\max\{|\theta|, |w_1|, \ldots, |w_n|\} \leq (n+1)n^{n/2}.$$

Proof. Consider the region of $\mathbb{R}^{n+1}$ consisting of the solutions w' to the system (5.1) of linear inequalities. This is a convex region in $\mathbb{R}^{n+1}$, the extreme points of which satisfy $n+1$ of the inequalities with equality. Let w' be any such extreme point. Then it satisfies a system of equations $Mw' = \mathbf{1}$, where M is an $(n+1) \times (n+1)$ matrix with entries in $\{0,1\}$, corresponding to $n+1$ of the inequalities in system (5.1) (which, in turn, correspond to $n+1$ of the x'). Here, $\mathbf{1}$ denotes the all-1 vector of length $n+1$. Now, by Cramer's rule, this means that for $i = 1, 2, \ldots, n+1$,

$$w'_i = \frac{\det(M_i)}{\det(M)},$$

where M_i is obtained from M by replacing the ith column with a column of ones. Now, expanding $\det(M_i)$ by its ith column,

$$\det(M_i) = \sum_{j=1}^{n+1} (-1)^{i+j} \det(M_{ij}),$$

where the matrices M_{ij} are $n \times n$ submatrices of M_i. Noting that the denominators in our expressions for each w'_i are the same number, $\det(M)$, if we multiply each w'_i by $\det(M)$, we obtain a solution w'' of system (5.1) in which

$$w''_i = \sum_{j=1}^{n+1} (-1)^{i+j} \det(M_{ij})$$

is an integer (as the sum of determinants of $\{0,1\}$-matrices, which are clearly integers). Furthermore, since Hadamard's inequality shows that the determinant of an $n \times n$ $\{0,1\}$-matrix is bounded by $n^{n/2}$, we have

$$|w''_i| \leq \sum_{j=1}^{n+1} |\det(M_{ij})| \leq (n+1)n^{n/2}.$$

Thus we obtain an integral representation of f in which the threshold and each weight are bounded in absolute value by $(n+1)n^{n/2}$. The result follows. □

5.3 Exponential lower bound

5.3.1 Existence of large-weight threshold functions

It is easy to show that exponential-sized integer weights are sometimes necessary just by a simple counting argument. We know from Theorem 4.5 that there are at least $2^{n(n-1)/2}$

threshold functions on $\{0,1\}^n$. For $B \in \mathbb{N}$, the number of pairs (w,θ) of integer weight-vector and threshold which satisfy $|w_i| \leq B$ for $i = 1,2,\ldots,n$, and $|\theta| \leq B$, is at most B^{n+1}. So, for example, the number of threshold functions representable with integer weights and threshold bounded in magnitude by $2^{n/6}$ is no more than $2^{n(n+1)/6}$. But this is less than $2^{n(n-1)/2}$ for $n \geq 2$, so there must be some threshold functions in which, using integer weights, we would need weights greater than $2^{n/6}$ in magnitude.

5.3.2 A specific example

The simple argument given above establishes the need for large weights, but it does not provide a concrete example of a threshold function requiring such large weights. The example we give now is not the simplest possible, but the function concerned is one we shall reconsider later in the book.

We shall consider, for n even, the Boolean function f_n on variables with formula

$$f_n = u_n \wedge (u_{n-1} \vee (u_{n-2} \wedge (u_{n-3} \vee (\ldots (u_2 \wedge u_1))\ldots)).$$

Thus, for example,

$$f_6 = u_6 \wedge (u_5 \vee (u_4 \wedge (u_3 \vee (u_2 \wedge u_1)))).$$

This function may also be represented as a 1-decision list. We have

$$f_n = (\bar{u}_n, 0), (u_{n-1}, 1), (\bar{u}_{n-2}, 0), \ldots, (\bar{u}_2, 0), (u_1, 1).$$

Since, as we have seen, all 1-decision lists are threshold functions, f_n is a Boolean threshold function.

Recall that the Fibonacci numbers F_n $(n \geq 1)$ are given by $F_1 = 1$, $F_2 = 1$ and, for $n \geq 3$, $F_n = F_{n-1} + F_{n-2}$. Solving the recurrence gives a formula for the Fibonacci numbers:

$$F_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n.$$

An explicit integral weight-vector and threshold representing f_n can be constructed from the Fibonacci numbers. It can be checked that if we take $\theta = F_{n+1}$ and $w = (F_1, F_2, \ldots, F_n)$, then $f_n \leftarrow [w,\theta]$. The following result shows that we do indeed require weights of this size to represent f_n.

Theorem 5.2 *Let $f_n = u_n \wedge (u_{n-1} \vee (u_{n-2} \wedge (u_{n-3} \vee (\ldots (u_2 \wedge u_1))\ldots))$. Then, if w is any integral weight-vector in a threshold representation of f_n, we have $w_i \geq F_i$ for $1 \leq i \leq n$, where F_i is the ith Fibonacci number.*

Proof. Suppose that $f_n \leftarrow [w,\theta]$. Since the function f_n is increasing, the weights are nonnegative. From

$$f_n(11\underbrace{0101\ldots01}_{n-2}) = 1, \quad f_n(01\underbrace{0101\ldots01}_{n-2}) = 0, \quad f_n(10\underbrace{0101\ldots01}_{n-2}) = 0,$$

it follows that w_1, w_2 are positive and hence, since they are integers, $w_1 \geq 1$ and $w_2 \geq 1$. So $w_1 \geq F_1$ and $w_2 \geq F_2$. This observation forms the base case for a proof by induction. Suppose now that $1 \leq j \leq n/2 - 1$. Let

$$x_j = \underbrace{11\ldots1}_{2j-1}00\underbrace{1010\ldots101}_{n-2j-1}, \quad y_j = \underbrace{00\ldots0}_{2j-1}01\underbrace{1010\ldots101}_{n-2j-1}.$$

Then $f_n(x_j) = 0$ and $f_n(y_j) = 1$. It follows that $\langle w, x_j \rangle < \theta \leq \langle w, y_j \rangle$, which, since the inner products are integers, means $\langle w, y_j \rangle \geq 1 + \langle w, x_j \rangle$. Expanding the inner products, this implies

$$w_{2j+1} \geq 1 + \sum_{i=1}^{2j-1} w_i.$$

Assuming as inductive hypothesis that $w_i \geq F_i$ for $1 \leq i \leq 2j-1$, we obtain

$$w_{2j+1} \geq 1 + \sum_{i=1}^{2j-1} F_i.$$

It is straightforward to prove that for any k,

$$F_k = 1 + \sum_{i=1}^{k-2} F_i.$$

So, we obtain $w_{2j+1} \geq F_{2j+1}$. To deal with even indices, we note that for $2 \leq j \leq n/2-1$, since $f_n(x_j) = 0$ and $f_n(y_{j-1}) = 1$, we must have $\langle w, y_{j-1} \rangle \geq 1 + \langle w, x_j \rangle$. Now,

$$x_j = \underbrace{11\ldots11}_{2j-2}10\underbrace{0101\ldots01}_{n-2j}, \quad y_{j-1} = \underbrace{00\ldots00}_{2j-2}11\underbrace{0101\ldots01}_{n-2j},$$

so by the relationship between the inner products and the inductive assumption, we have

$$w_{2j} \geq 1 + \sum_{i=1}^{2j-2} F_i = F_{2j}.$$

We have shown that $w_k \geq F_k$ for $1 \leq k \leq n-1$. To prove $w_n \geq F_n$, in order to complete the proof, we note that

$$f_n(11\ldots110) = 0, \quad f_n(00\ldots0011) = 1,$$

so we must have

$$w_n \geq 1 + \sum_{k=1}^{n-2} w_k \geq 1 + \sum_{k=1}^{n-2} F_k = F_n.$$

□

It is easy to check that

$$F_n \geq \frac{\sqrt{5}}{6}\left(\frac{1+\sqrt{5}}{2}\right)^n$$

for all n. So this result gives an explicit example of a threshold function on $\{0,1\}^n$ requiring integer weights exponential in n.

5.3.3 Tightness of general upper bound

The general upper bound on integral weights given in Theorem 5.1 is $(n+1)n^{n/2}$, whereas the specific lower bound exhibited by the function f_n is (merely) exponential in n. The question arises as to whether the general upper bound might be quite loose and could be considerably improved. In fact, although we shall not provide the details, the upper bound is quite tight, in that there are functions requiring superexponential integer weights.

Theorem 5.3 *There are constants $k > 0$ and $c > 1$ such that, for n a power of 2, there is a threshold function f on $\{0,1\}^n$ so that any integral weight-vector representing f has a weight at least $kc^{-n}n^{n/2}$.*

5.4 Additional remarks and bibliographical notes

Exponential lower bounds on the sizes of weights can be found in [74, 75]. (See also [81], on which our proof is based.) The tighter lower bound, Theorem 5.3, is due to Håstad [41].

The general upper bound on weight sizes is due to Muroga [75].

Some differences are experienced in bounding weights when we use the $\{-1,1\}$ convention instead of the standard $\{0,1\}$; see [21], for example. Using $\{-1,1\}^n$ as the domain, we obtain an improvement in the general upper bound (due to Håstad [41]): the weights can then be bounded by $2^{-n+1}(n+1)n^{n/2}$, because any $n \times n$ $\{-1,1\}$ matrix has determinant divisible by 2^n.

Bohossian and Bruck [21] considered in some detail the size of integer weights for threshold functions. They have proved the following result, for domain $\{-1,1\}^n$: for a threshold function $f \in T_n$, let $S(f)$ denote the minimum over all integral thresholds θ and weight-vectors w representing f, of $|\theta| + \sum_{i=1}^{n} |w_i|$. Then, for all pairs (n,s) of integers such that $s \geq n$ is even and $s \leq 2^{(n-1)/2}$, there exists $f \in T_n$ such that $S(f) = s$.

This page intentionally left blank

Chapter 6

Threshold Order

6.1 Introduction

As we have seen, the threshold order of a Boolean function is defined to be the least degree of an equivalent polynomial threshold function. A natural question is whether we can determine whether a function is a linear threshold function, given a particular type of formula for it. More generally, we might ask if the threshold order can be determined or bounded. We present some positive and negative results along these lines. We then look at the "typical" threshold order of a Boolean function.

6.2 Algorithms and complexity

6.2.1 Introduction

We now turn our attention to the issue of how to determine the threshold order of a function, given a DNF formula for the function. We shall show that, under some standard complexity-theoretic assumptions, this cannot be done efficiently, in general. We then show, however, that if the function f in question is known to be increasing, and if the formula presented is the irredundant DNF for f (or, equivalently, a list of the MTPs of f), then the issue of whether f is a *linear* threshold function *can* be resolved efficiently.

The reader is probably familiar with the basics of complexity theory, and there is no intention here to present a rigorous account of these basics. Nonetheless, we shall give a very informal (and very sketchy) account of what we mean by an efficient algorithm and what we mean by a "hard" problem. This will suffice for our purposes.

6.2.2 Efficient algorithms and "hardness" of problems

Complexity Theory deals with the relationship between the size of the input to an algorithm and the time required for the algorithm to produce its output. In particular, it is concerned with the question of when the algorithm can be described as "efficient." The size of an input to an algorithm can be measured in various ways. We shall mainly be concerned with algorithms in which the input is either a Boolean formula or a set of $\{0,1\}$-vectors, and it is natural to take as the size of input the total number of literals

appearing in the formula (that is, the length of the formula) or, respectively, the total number of bits in the set of vectors.

The "running time" of an algorithm is, of course, dependent on the speed with which the underlying calculations can be carried out. For a device-independent definition, running time is measured by the number of operations needed rather than the actual time involved. Furthermore, since we shall only be interested in the form of the dependence on input size, not the exact details, we shall make use of the O-notation. Let A be an algorithm which accepts inputs of varying size s. We say that the running time of A is $O(f(s))$ if, for large enough s and for any input of size s, the number of operations required to produce the output of A is at most $Kf(s)$, where K is some fixed constant. Note that this involves the *worst-case* running time: that is, we consider the maximum possible number of operations, taken over all inputs of a given size.

A *polynomial-time* algorithm is one with running time $O(s^r)$, for some fixed integer $r \geq 1$, and we shall regard such algorithms as efficient. We shall be concerned largely with *decision problems*, in which an *instance* of the problem is given, and the task is to determine whether the answer to a given question is "yes" or "no." For example, a classic decision problem is the "DNF non-tautology" problem, defined as follows.

DNF NON-TAUTOLOGY
Instance: A disjunctive normal formula ϕ in n variables.
Question: Is there $x \in \{0,1\}^n$ such that $f(x) = 0$, where f is the function represented by ϕ?

Problems which can be solved by a polynomial time algorithm are usually regarded as "easy." To show that a problem is "hard," we would ideally like to show that no polynomial-time algorithm for its solution exists. However, in practice, what one shows is that the problem is at least as hard as one of a range of standard "NP-complete" problems, all of which are believed not to be solvable by polynomial time algorithms. (Roughly speaking, this belief is known as the "P$\neq$ NP conjecture.") More precisely, one shows that a polynomial-time algorithm for the problem could be used to produce a polynomial-time algorithm for the NP-complete problem. DNF NON-TAUTOLOGY is one of the standard NP-complete problems. So also, for example, is graph three-colorability, in which the instance is a graph, and the decision to be taken is whether it has a proper vertex-coloring with three colors. Suppose Π is a decision problem and that Π_0 is a decision problem which is known to be NP-complete. If we can demonstrate that the existence of a polynomial time algorithm for Π implies the existence of one for Π_0, then we say that Π is *NP-hard*. If the P $\neq$ NP conjecture is true, then proving that a problem Π is NP-hard establishes that there is no polynomial time algorithm for Π.

6.3 Complexity of determining threshold order

6.3.1 The problem MEMBERSHIP(C)

For a nonnegative integer m, let us consider the problem of determining whether a function represented by a given DNF formula is of threshold order at most m; that is, whether it belongs to $T(n,m)$. Formally, we have the following decision problem.

THRESHOLD ORDER $\leq m$
Instance: A disjunctive normal formula ϕ in n variables.
Question: Does the function f represented by ϕ have threshold order at most m?

This is an example of a "membership" problem for a class of Boolean functions. Generally, by a *class* $\mathcal{C}$ of Boolean functions, we mean a sequence $\{C_n\}_{n\geq 1}$ where, for each $n \in \mathbb{N}$, C_n is a set of Boolean functions on $\{0,1\}^n$. For example, the class of threshold functions is $\mathcal{T} = \{T_n\}_{n\geq 1}$. We shall denote by $\mathcal{T}(m) = \{T(n,m)\}_{n\geq 1}$ the class of Boolean functions of threshold order at most m (that is, the polynomial threshold functions of degree m).

For a class $\mathcal{C}$ of functions, the *membership problem* for $\mathcal{C}$ is as follows.

MEMBERSHIP($\mathcal{C}$)
Instance: A disjunctive normal form formula ϕ in n variables.
Question: Does the function f represented by ϕ belong to C_n?

We shall show that MEMBERSHIP($\mathcal{C}$) is NP-complete for all classes $\mathcal{C}$ satisfying certain properties. The following definition describes these properties.

Definition 6.1 *Suppose that $\mathcal{C} = \{C_n\}$ is a class of Boolean functions. Then $\mathcal{C}$ is* closed under projections *if for all n and, for all $f \in C_n$, every projection f_a of f belongs to $\mathcal{C}$ (explicitly, it belongs to C_k, where k is the number of entries of $a \in \{0,1,\star\}^n$ equal to $\star$). Furthermore, we say that a class $\mathcal{C}$* has the projection property *if*

(i) *$\mathcal{C}$ is closed under projections;*

(ii) *for every $n \in \mathbb{N}$, the identically-1 function belongs to C_n;*

(iii) *there exists $k \in \mathbb{N}$ such that some Boolean function on $\{0,1\}^k$ does* not *belong to C_k.*

Then we have the following result.

Theorem 6.2 *Suppose that $\mathcal{C}$ is a class of Boolean functions having the projection property. Then* MEMBERSHIP*($\mathcal{C}$) is NP-hard.*

Proof. We show that any efficient algorithm for MEMBERSHIP($\mathcal{C}$) would lead to one for the NP-complete DNF NON-TAUTOLOGY problem. Now, since $\mathcal{C}$ satisfies the projection property, there is some $k \in \mathbb{N}$ such that some $h : \{0,1\}^k \to \{0,1\}$ is *not* in $\mathcal{C}$. Let ϕ be an instance of DNF NON-TAUTOLOGY on n variables, so ϕ is a DNF formula, representing some function $f : \{0,1\}^n \to \{0,1\}$. Construct the DNF formula ψ representing the function $g : \{0,1\}^{n+k} \to \{0,1\}$ given by

$$g(x,y) = f(x) \vee h(y).$$

Note that, since k is fixed, the resulting formula ψ is still of size proportional to the size of ϕ, the given instance of DNF NON-TAUTOLOGY. (As usual in proofs of this type, we have to check that the constructed instance is of size polynomial in the size of the given instance of the NP-complete problem.) Suppose now that $f(x) = 1$ for all x. Then $g(x,y) = 1$ for all x and y; that is, g is the identically-1 function. Since $\mathcal{C}$ has the projection property, C_{n+k} contains the identically-1 function, so in this case, an algorithm for MEMBERSHIP($\mathcal{C}$) will answer "yes" on instance ψ. Suppose that there is some $x^* \in \{0,1\}^n$ such that $f(x^*) = 0$. Then $g(x^*,y) = h(y)$, which is not in C_k. But, since $\mathcal{C}$ is closed under projections, and $g(x^*,y)$ is a projection of g, this means that $g \notin C_{n+k}$, so an algorithm for MEMBERSHIP($\mathcal{C}$) would, in this case, return the answer "no" on instance ψ. We see, therefore, that if we had an algorithm for MEMBERSHIP($\mathcal{C}$),

we would have one for DNF NON-TAUTOLOGY. For, we would simply take an instance ϕ of DNF NON-TAUTOLOGY, transform it as indicated in this proof to an instance ψ of MEMBERSHIP($\mathcal{C}$), and apply the algorithm for the latter. The analysis shows that if the answer to MEMBERSHIP($\mathcal{C}$) on ψ is "yes," then the answer to DNF NON-TAUTOLOGY on ϕ must be "no," and that if the answer is "no," then the answer to DNF NON-TAUTOLOGY is "yes." Moreover, if the MEMBERSHIP($\mathcal{C}$) algorithm is a polynomial-time algorithm, then since the size of ψ is polynomial in the size of ϕ, this procedure for DNF NON-TAUTOLOGY is polynomial-time. This establishes the NP-hardness. □

6.3.2 Complexity of determining threshold order

For fixed m, the class $\mathcal{T} = \{T(n,m)\}_{n\geq 1}$ of Boolean functions of threshold order at most m is easily seen to have the projection property. We have already observed (Theorem 3.10) that it is closed under projection. Furthermore, the identically-1 function has threshold order 0 and so belongs to $T(n,m)$ for all n and all m. We know (Theorem 4.7) that the number of functions of degree m on n variables is at most $2^{O(n^{m+1})}$, so for all sufficiently large n, this is less than 2^{2^n}, and we can deduce that there exists k such that $T(k,m)$ does not contain all Boolean functions on $\{0,1\}^k$. By Theorem 6.2, we therefore have the following result.

Theorem 6.3 *For any nonnegative integer m,* THRESHOLD ORDER $\leq m$ *is NP-hard.*

6.4 Threshold synthesis

6.4.1 Introduction

The problem MEMBERSHIP($\mathcal{T}$) of recognizing whether a Boolean function, given a formula for it, is a linear threshold function is a long-studied problem in "threshold logic." The process of answering this question, and, if the answer is affirmative, finding weights and threshold representing the function, is known as "threshold synthesis." We have already seen, from Theorem 6.3, that the problem is NP-hard in general, even if the formula given is required to be a DNF. But we now show that if we restrict attention to *increasing* Boolean functions and require that the formula be the irredundant DNF formula, then the problem can be solved efficiently. (Although we restrict attention to increasing functions, it should be noted that any function which is a threshold function will be increasing when some of the variables are negated, simply because any such function is unate.) Recall that for an increasing function f, the irredundant DNF formula is the disjunction of all the prime implicants of f, and that this is a positive DNF. We have noted earlier that the prime implicants of an increasing function f correspond directly with the MTPs (maximal true points) of f, so presenting the irredundant DNF is entirely equivalent to presenting a list of the MTPs of f. For simplicity, we assume that it is this list that is given. We now outline a polynomial-time algorithm for the following problem.

LINEAR THRESHOLD RECOGNITION
Instance: The MTPs of an increasing function $f : \{0,1\}^n \to \{0,1\}$.
Question: Is f a linear threshold function?

In fact, it will be apparent that the procedure we describe not only solves this decision problem but will find suitable weights and threshold when appropriate; that is, it is a procedure for threshold synthesis.

6.4.2 Overview of the method

Before giving the details, it is useful first to have an overview of the procedure for solving LINEAR THRESHOLD RECOGNITION. There are three key stages to the procedure (each of which we will consider in detail shortly) as follows.

1. First, we apply a *recognition* algorithm to determine whether f is 2-monotonic. Recall that a function is 2-monotonic if and only if variables $x_1, x_2, \ldots, x_n$ can be permuted in such a way that the function becomes regular. (This stage of the process determines what this permutation should be.) Since any linear threshold function is 2-monotonic, if we determine at this stage that f is not 2-monotonic, then we can immediately deduce that f is not a threshold function. (However, as we have seen, not every 2-monotonic function is a threshold function.)

2. Next, if the function *is* 2-monotonic, we permute the variables (according to the permutation found in the first stage) to obtain a regular function. We then apply a *dualization* algorithm which takes the MTPs of f and produces the complete list of MFPs (minimal false points) of f. (The term "dualization" arises from the fact, noted earlier, that the MFPs of an increasing function are the negations of the MTPs of the dual function f^d.)

3. Lastly, we use a *linear programming* algorithm to determine whether there is a hyperplane strictly separating the MTPs of f from the MFPs of f. If there is, then we deduce that f is a threshold function; if not, we deduce that it is not. (In the case that the answer is positive, the linear programming algorithm also returns representative weights and thresholds.)

6.4.3 Recognition algorithm for 2-monotonic functions

A function is 2-monotonic if and only if its variables can be permuted to obtain a regular function; that is, if and only if there is a permutation $\sigma : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$ such that f^σ is regular, where

$$f^\sigma(x_1, x_2, \ldots, x_n) = f(x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)}).$$

The following result provides a method of determining, from the MTPs of a 2-monotonic function, an appropriate permutation σ. The result involves the *lexicographic order* (or *dictionary order*) $\preceq_{\mathrm{L}}$ on n-vectors. For $x, y \in \mathbb{R}^n$, we have $x \preceq_{\mathrm{L}} y$ if and only if $x = y$, or $x_k < y_k$, where $1 \leq k \leq n$ is the first coordinate in which x and y are not equal. We also write $y \succeq_{\mathrm{L}} x$. For example, $(1, 1, 2, 4) \preceq_{\mathrm{L}} (1, 1, 3, 1)$.

Theorem 6.4 *Suppose f is a 2-monotonic Boolean function on $\{0,1\}^n$. For x in $\{0,1\}^n$, let $\|x\|$ be the number of ones in x. Define vectors $z^{(1)}, z^{(2)}, \ldots, z^{(n)} \in \mathbb{N}^n$ by*

$$z_i^{(j)} = |\{x \in \mathrm{MTP}(f) : x_j = 1, \|x\| = i\}|$$

for $i = 1, 2, \ldots, n$. Suppose that

$$z^{(j_1)} \succeq_{\mathrm{L}} z^{(j_2)} \succeq_{\mathrm{L}} \cdots \succeq_{\mathrm{L}} z^{(j_n)},$$

and let the permutation σ be given by $\sigma(j_i) = i$. Then f^σ, given by $f^\sigma(x_1, x_2, \ldots, x_n) = f(x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)})$, is regular.

Example 6.5 As an easy example, suppose f is the 2-monotonic function represented by formula $u_4 \wedge (u_3 \vee (u_2 \wedge u_1))$. Then the MTPs of f are 0011 and 1101 and the vectors $z^{(i)}$ are

$$z^{(1)} = (0,0,1,0),\ z^{(2)} = (0,0,1,0),\ z^{(3)} = (0,1,0,0),\ z^{(4)} = (0,1,1,0),$$

so

$$z^{(4)} \succeq_{\mathrm{L}} z^{(3)} \succeq_{\mathrm{L}} z^{(2)} = z^{(1)}.$$

Taking σ to be

$$4 \mapsto 1,\ 3 \mapsto 2,\ 2 \mapsto 3,\ 1 \mapsto 4$$

(or, alternatively, we could take $2 \mapsto 4$ and $1 \mapsto 3$), we should have that f^σ is regular, where

$$f^\sigma(x_1, x_2, x_3, x_4) = f(x_4, x_3, x_2, x_1),$$

and this is indeed the case.

Now that we know how to transform a given 2-monotonic function into a regular one, one way to test whether any increasing function f is 2-monotonic is to apply this technique, and then determine whether the resulting function f^σ is regular. If it is, then we have verified that f is indeed 2-monotonic; if it is not, then we can deduce that f is not 2-monotonic. We therefore present a technique for determining whether an increasing function g is regular, given its MTPs. (Note that the MTPs of f^σ are immediately obtained from those of f.) This technique is based on the fact (noted in Section 2.4.4) that an increasing function g is regular if and only if for all pairs (v, i), where v is an MTP of g, $1 \leq i \leq n$, and $v_i = 0, v_{i+1} = 1$, we have $g(v + e_i - e_{i+1}) = 1$. To check, then, whether g is regular, we simply take each MTP in turn, run through the coordinates to find any indices i such that $v_i = 0, v_{i+1} = 1$, and then check whether $v + e_i - e_{i+1}$ is a true point of g. This can be done by comparing $v + e_i - e_{i+1}$ with each MTP: $v + e_i - e_{i+1}$ will be a true point of g if and only if there is an MTP x such that $x \preceq v + e_i - e_{i+1}$.

6.4.4 Dualization algorithm for regular functions

Assuming that f has been transformed into a regular function $g = f^\sigma$, the next stage is to find the MFPs of g by a "dualization" algorithm. The technique we describe is based on the following two theorems. (The proof of the second is straightforward, and is omitted.)

Theorem 6.6 *For a regular function g on $\{0,1\}^n$, x is an MFP of g with $x_n = 0$ if and only if $x + e_n$ is an MTP of g.*

Proof. Suppose that $x_n = 0$ and x is an MFP of g. Then, by minimality of x, $g(x + e_n) = 1$. Suppose that $y \preceq x + e_n$ and $y \neq x + e_n$. Then $y \preceq x + e_n - e_i$ for some i

such that $x_i = 1$. By the regularity of g, and since $g(x) = 0$, we have $g(x + e_n - e_i) = 0$. But since g is increasing, this implies $g(y) = 0$, and hence y is not a true point. It follows that $x + e_n$ is a *minimal* true point.

Conversely, suppose $x + e_n$ is an MTP of g. Then $x_n = 0$ and $x \preceq x + e_n$, so $g(x) = 0$ (by the minimality of $x + e_n$). If x is not a maximal false point, then there is $i < n$ such that $x + e_i$ is a false point. But then, by the regularity of g, $(x + e_i) + e_n - e_i = x + e_n$ is a false point, which is a contradiction. So for any MTP of g of the form $x + e_n$, x is an MFP of g. □

Theorem 6.7 *For an increasing function g on $\{0,1\}^n$, let $g\uparrow$ be the up-projection of g, defined by $g\uparrow(x_1, x_2, \ldots, x_{n-1}) = g(x_1, x_2, \ldots, x_{n-1}, 1)$. Then*

(i) *x is an MFP of $g\uparrow$ if and only if $(x, 1)$ is an MFP of g;*

(ii) *x is an MTP of $g\uparrow$ if and only if either $(x, 1)$ is an MTP of g, or $(x, 0)$ is an MTP of g and for all $y \preceq x$, $(y, 1)$ is* not *an MTP of g;*

(iii) *$g\uparrow$ is regular if g is regular.*

Given the MTPs of a regular function g, we may therefore determine the MFPs of g as follows. First, we apply Theorem 6.6 to determine all MFPs with last entry 0. By Theorem 6.7 (part (i)), if we had the MFPs of $g\uparrow$, then the MFPs of g with last entry equal to 1 would be immediately determined. Now, $g\uparrow$ is regular and, by part (ii), we can determine its MTPs from those of g, so we can iterate the procedure, finding the MFPs of $g\uparrow$ from its MTPs, and so on.

A useful consequence of Theorems 6.6 and 6.7 is an upper bound on the number of MFPs in terms of the number of MTPs.

Theorem 6.8 *Suppose that f is a regular Boolean function on $\{0,1\}^n$ and that f has m MTPs. Then the number of MFPs is no more than $m(n-1)+1$.*

Proof. To prove this, assume, inductively on k, that the number of MFPs of a k-variable regular function is bounded by $m(k-1)+1$, where m is the number of MTPs of the function. (The base case, $k = 1$, is easily seen to be true.) Let f be a k-variable regular function with m MTPs and let $f\uparrow$ be the up-projection, a regular function of $k-1$ variables. By Theorem 6.6, the number of MFPs of f with last entry 0 is bounded above by m, and by part (i) of Theorem 6.7, the number of MFPs of f with last entry 1 equals the number of MFPs of $f\uparrow$. By part (ii) of Theorem 6.7, the number of MTPs of $f\uparrow$ is no more than m, so by the inductive hypothesis, the number of MFPs of $f\uparrow$ is no more than $m(k-2)+1$ and, hence, the number of MFPs of f is no more than $m + m(k-2) + 1 = m(k-1) + 1$ (since m is an upper bound on the number of MFPs with last entry 0 and $m(k-2)+1$ an upper bound on the number with last entry 1). □

6.4.5 Linear programming to determine separability

Suppose that f is an increasing Boolean function on $\{0,1\}^n$. Then, f is a threshold function if and only if there is a solution $w \in \mathbb{R}^n, \theta \in \mathbb{R}$ to the system

$$\begin{aligned} \langle w, x\rangle &\geq \theta, \quad x \in T(f), \\ \langle w, x\rangle &\leq \theta - 1, \quad x \in F(f), \\ w_1, w_2, \ldots, w_n &\geq 0. \end{aligned}$$

(The nonnegativity constraints can be imposed because f is increasing, and the discreteness of the domain means that we may demand $\langle w, x\rangle \leq \theta - 1$ for false points, rather than simply $\langle w, x\rangle < \theta$.) Since f is increasing, it is enough to consider only those inequalities corresponding to the set MTP(f) of MTPs of f and the set MFP(f) of MFPs of f. That is, to determine whether f is a threshold function, we determine whether the system of inequalities

$$\begin{aligned} \langle w, x\rangle &\geq \theta, \quad x \in \mathrm{MTP}(f), \\ \langle w, x\rangle &\leq \theta - 1, \quad x \in \mathrm{MFP}(f), \\ w_1, w_2, \ldots, w_n &\geq 0 \end{aligned}$$

is feasible. To see why, suppose that w, θ satisfy the second system of inequalities and that, for example, x is in $T(f)$ but is not an MTP of f. Then there will be some MTP y such that $y \preceq x$. The fact that $w_i \geq 0$ for all i implies $\langle w, x\rangle \geq \langle w, y\rangle \geq \theta$. In this way, we see that the inequalities of the first system involving points which are not MTPs or MFPs are implied by those of the second system.

6.4.6 Time complexity of synthesis procedure

To show that the threshold synthesis procedure is efficient, we want to prove that its worst-case running time is polynomial in the size of the instance of the LINEAR THRESHOLD RECOGNITION problem. This instance is a list of m MTPs of f, each of which is a $\{0,1\}$-vector of length n, and it therefore has size mn. We therefore show that the running time of the synthesis procedure is polynomial in mn.

The first stage involves the determination of the appropriate permutation σ and the implementation on f^σ of the technique for checking regularity. Now, the vectors $z^{(i)}$ can be determined in time $O(mn^2)$, the lexicographic ordering can be found in time $O(n^2)$, the transformation of the MTPs of f into those of f^σ can be done in time $O(mn)$, and the regularity checking can be executed in time $O(m^2n^2)$. The first stage therefore has time complexity $O(m^2n^2)$, which is polynomial in mn.

We now bound the time-complexity of the dualization technique. First, we find all MFPs of g of the form $(x, 0)$, by running through the MTPs of g, noting that any of the form $(x, 1)$ immediately give the MFP $(x, 0)$. This takes time $O(mn)$. The calculation of the MTPs of $g\uparrow$ from those of g (making use of part (ii) of Theorem 6.7) takes time $O(m^2n)$. It follows that the running time of this iterative procedure on g is bounded by $O(m^2n) + T$, where T is the time taken for the determination of the MFPs of $g\uparrow$ from its MTPs (by the same technique, applied to $g\uparrow$). Noting that the number of MTPs of $g\uparrow$ is also bounded above by m, this implies an $O(m^2n^2)$ bound on the running-time of the procedure. Again, this is polynomial in mn.

We have seen (Theorem 6.8) that the number of MFPs of a regular function with m MTPs is $O(mn)$. It follows that the system of linear inequalities corresponding to the MTPs and MFPs involves no more than $O(mn)$ inequalities in $n + 1$ variables. It may therefore be solved by any polynomial-time linear programming algorithm (such as Karmarker's algorithm or the ellipsoid method), in time polynomial in $mn(n + 1)$, and hence polynomial in mn.

It follows that we have a polynomial-time algorithm for LINEAR THRESHOLD RECOGNITION.

6.5 Expected threshold order

We have seen that the number of linear threshold functions is extremely small when compared with the number of all Boolean functions, and in Chapter 4 we determined some bounds on the number of Boolean functions of a given degree. A natural question is what the threshold order of a "typical" Boolean function is.

A very precise behavior of the "expected" threshold order has been conjectured. Roughly speaking, the conjecture says that, for large even numbers n, almost all the Boolean functions on $\{0,1\}^n$ have threshold order equal to $n/2$; and that for large odd n, almost every function has threshold order $(n-1)/2$ or $(n+1)/2$, with an equal split between these. To make this precise, we introduce some notation. Let $\sigma(n,m)$ denote the proportion of Boolean functions of n variables with threshold order m; thus,

$$\sigma(n,m) = \frac{|T(n,m)| - |T(n,m-1)|}{2^{2^n}}.$$

It has been conjectured (and computational results seem to provide numerical evidence for this) that for even values of n, $\sigma(n,n/2) \to 1$ as $n \to \infty$ and that for odd values of n, $\sigma(n,(n-1)/2) \to 1/2$ and $\sigma(n,(n+1)/2) \to 1/2$ as $n \to \infty$. The following result provides a partial proof of this.

Theorem 6.9 *For all n,*

$$\lim_{n\to\infty} \frac{|T\left(n, \lfloor \frac{n}{2} \rfloor - 1\right)|}{2^{2^n}} = 0,$$

and so, for $m = m(n) \leq \lfloor n/2 \rfloor - 1$, $\sigma(n, m(n)) \to 0$ as $n \to \infty$. Furthermore, for odd n,

$$\frac{|T\left(n, \lfloor \frac{n}{2} \rfloor\right)|}{2^{2^n}} \leq \frac{1}{2}.$$

Proof. We know from Theorem 4.7 that $|T(n,m)| \leq 2\sum_{i=0}^{d(m)} \binom{2^n-1}{i}$, where $d(m) = \sum_{i=1}^{m} \binom{n}{i}$. Now,

$$d(\lfloor n/2 \rfloor - 1) = \sum_{i=1}^{\lfloor n/2 \rfloor - 1} \binom{n}{i} \leq \frac{2^n}{2} - \frac{1}{2}\binom{n}{\lfloor \frac{n}{2} \rfloor} \leq \frac{2^n}{2} - c\frac{2^n}{\sqrt{n}},$$

where c is a constant independent of n. Now, for any N and any $\lambda \leq N/2$, by a standard inequality of probability theory,

$$\sum_{i < \frac{N}{2} - \lambda} \binom{N}{i} < 2^N \exp(-2\lambda^2/N).$$

It follows that

$$|T\left(n, \lfloor n/2 \rfloor - 1\right)| < 2 \sum_{i=0}^{d(\lfloor \frac{n}{2} \rfloor - 1)} \binom{2^n}{i} < 2\, 2^{2^n} \exp\left(-2(c2^n/\sqrt{n})^2/2^n\right),$$

from which we obtain

$$\frac{|T\left(n, \lfloor \frac{n}{2} \rfloor - 1\right)|}{2^{2^n}} < 2\exp\left(-2c^2 2^n/n\right) \to 0,$$

as $n \to \infty$. Furthermore, for n odd, we have $d(\lfloor n/2 \rfloor) = 2^n/2 - 1$ and so

$$\left| T\left(n, \left\lfloor \frac{n}{2} \right\rfloor\right) \right| < 2 \sum_{i=0}^{2^n/2-1} \binom{2^n - 1}{i} = 2 \frac{2^{2^n-1}}{2} = \frac{1}{2} 2^{2^n},$$

and the result follows. □

This result shows, among other things, that the representational power of $T(n, m)$ is limited unless m is of the same order as n. The analysis shows that if a Boolean function is chosen uniformly at random from the set of all Boolean functions on n variables, then

$$\text{Prob}(\text{threshold order of } f < \lfloor n/2 \rfloor) < \exp(-K(2^n/n))$$

for a positive constant K. Thus it might be said that the "typical" Boolean function has threshold order at least $\lfloor n/2 \rfloor$.

The conjecture that

$$\text{Prob}(\text{threshold order of } f > \lceil n/2 \rceil) \to 0,$$

as $n \to \infty$, remains. The following result, which moves some way toward it, has, however, been obtained.

Theorem 6.10 *There exists a fixed constant $\epsilon > 0$, such that almost all Boolean functions of n variables have threshold order less than $(1 - \epsilon)n$; that is,*

$$\sigma(n, (1 - \epsilon)n) \to 0 \text{ as } n \to \infty.$$

We mention that there are always (exactly) two functions with threshold order n, namely PARITY_n and its negation.

6.6 Additional remarks and bibliographical notes

Threshold synthesis (and related problems) have been studied for a long time; see, for example, the book by Muroga [75]. Peled and Simeone [83] were the first to produce a polynomial time algorithm for the threshold recognition problem. They also proved that the general MEMBERSHIP($\mathcal{T}$) problem is NP-hard.

The general hardness result on MEMBERSHIP($\mathcal{C}$) is due to Hegedűs and Megiddo [46]. Wang and Williams [104] had shown that determining whether the threshold order equals m is NP-hard.

Lexicographic ordering of n vectors of length k can be done in time $O(n(\ln n + k))$; see [15]. Consequently, the lexicographic ordering of the $z^{(i)}$ can be done in time $O(n^2)$.

The recognition algorithm for 2-monotonic functions is due to Makino [65], and Theorem 6.4 is due to Winder [105]. The dualization algorithm is Crama's [31]. The algorithms we presented for determining regularity and for dualizing regular functions are not the fastest, but they have the advantage of being simple to understand. Makino [65] has developed an $O(mn)$ algorithm for determining whether an increasing function is regular (improving upon the $O(m^2n^2)$ of the simpler method presented here). Furthermore, Crama [31] has devised an $O(mn^2)$ dualization algorithm for regular functions (improving upon the simple $O(m^2n^2)$ algorithm described above).

The conjecture on the asymptotic behavior of the expected threshold order is due to Wang and Williams [104], and essentially the same conjecture was made by Aspnes et al. [12]. Theorem 6.9 is due, independently, to Alon (see [90]) and Anthony [5].

The "standard inequality" from probability theory referred to in the proof of Theorem 6.9 may be found in [73] and is a direct consequence of a bound of Chernoff [27, 4] (and also of Hoeffding's inequality; see [71]).

Theorem 6.10 has been reported by Saks [90] and is due to Noga Alon, who obtained it using a result of Gotsman concerning the harmonic analysis of Boolean functions.

The fact that there are precisely two functions of threshold order n can be found in [104].

This page intentionally left blank

Chapter 7

Threshold Networks and Boolean Functions

7.1 Introduction

In this chapter, we consider the representation of Boolean functions by feed-forward linear threshold networks. We have seen that single linear threshold units have very limited computational abilities. However, we will easily see that any Boolean function can be represented by a linear threshold network with one hidden layer. We shall describe different general methods for obtaining such threshold network representations of Boolean functions, one of which draws links between decision lists and threshold networks. It is natural to ask how small a network can be used for particular functions or types of functions. Questions like this bring us into the realm of circuit complexity, a large area which we will only very briefly touch on here. Throughout this chapter, we shall be concerned only with feed-forward linear threshold networks, and will use the terminology "threshold network" for these. (In the circuit complexity literature, the terminology "threshold circuit" is used.)

7.2 DNFs and threshold networks

The existence of a DNF formula for every Boolean function can be used to show that any Boolean function can be computed by a two-layer feed-forward threshold network.

Theorem 7.1 *Any Boolean function on $\{0,1\}^n$ can be computed (with suitable weights and threshold) by a two-layer threshold network having n inputs, 2^n units in the hidden layer, and one output unit.*

Proof. Suppose that $f : \{0,1\}^n \to \{0,1\}$, and let ϕ be the DNF formula obtained as the disjunction of the prime implicants of f. Suppose $\phi = T_1 \vee T_2 \vee \cdots \vee T_k$, where each T_j is a term of the form $T_j = (\bigwedge_{i \in P_j} u_i) \bigwedge (\bigwedge_{j \in N_j} \bar{u}_j)$ for some disjoint subsets P_j, N_j of $\{1, 2, \ldots, n\}$. Suppose that the network has 2^n hidden units, and let us set the weights to and from all but the first k of these to equal 0, and the corresponding thresholds equal to 1 (so the effect is as if these units were absent). Then for each of the first k units, let the weight-vector $\alpha^{(j)}$ from the inputs to unit j correspond directly to

T_j, in that $\alpha_i^{(j)} = 1$ if $i \in P_j$, $\alpha_i^{(j)} = -1$ if $i \in N_j$, and $\alpha_i^{(j)} = 0$ otherwise. We take the threshold on unit j to be $|P_j|$, the weight on the connection between the unit and the output unit to be 1, and the threshold on the output unit to be $1/2$. It is clear that unit j outputs 1 on input x precisely when x satisfies T_j and that the output unit computes the "or" of all the outputs of the hidden units. Thus, the output of the network is the disjunction of the terms T_j and hence equals f. □

7.3 A geometric approach: "Chopping"

7.3.1 The chopping procedure

We now describe a geometrical approach to representing Boolean functions by threshold networks. This makes use of what we call a "chopping" procedure. We imagine the points of the Boolean hypercube as labeled either negative (false) or positive (true), according to the Boolean function f. We first use a hyperplane to separate off a set of points all having the same classification. (Thus, either all are positive examples or all are negative examples.) If the function f is a linear threshold function, there is a way of doing this so that all positive points (or all negative points) have been chopped off on the first chop. Generally, though, what will remain after the first chop is part of the Boolean hypercube, with some points labeled as positive and some as negative. The chopping procedure is simply iterated until either there are no positive points remaining, or there are no points remaining. For simplicity, we can (by the discreteness of the domain) assume that at each stage no point lies on the hyperplane.

Example 7.2 Consider the parity function PARITY_n. We first find a hyperplane such that all points on one side of the plane are either positive or negative. It is clear that all we can do at this first stage is chop off one of the points since the nearest neighbors of any given point have the opposite classification. Let us suppose that we decide to chop off the origin. We may take as the first hyperplane the plane with equation $y_1 + y_2 + \cdots + y_n = 1/2$. (Of course, there are infinitely many other choices of hyperplane which would achieve the same effect.) We then ignore the origin and consider the remaining points. We can next chop off all neighbors of the origin, namely all the points which have precisely one entry equal to 1. All of these are positive points and the hyperplane $y_1 + y_2 + \cdots + y_n = 3/2$ will separate them from the other points. These points are then deleted from consideration. We may continue in this manner. The procedure iterates n times, and at stage i in the procedure we "chop off" all data points having precisely $(i-1)$ ones, by using the hyperplane $y_1 + y_2 + \cdots + y_n = i - 1/2$, for example. (These hyperplanes are parallel, but this is not necessary, or possible, in general.)

To explain how this procedure establishes the existence of a certain type of threshold network representation of Boolean functions, we show first, as an intermediate step, how the procedure results in a certain type of decision list representation.

7.3.2 "Chopping" and threshold decision lists

Recall that for a set K of Boolean functions, $DL(K)$, the set of *decision lists based on* K, is the the set of finite sequences

$$f = (f_1, c_1),\ (f_2, c_2), \ldots,\ (f_r, c_r),$$

such that $f_i \in K$ and $c_i \in \{0,1\}$ $(1 \leq i \leq r)$. The values of f are defined by

$$f(y) = \begin{cases} c_j & \text{if } j = \min\{i : f_i(y) = 1\} \text{ exists,} \\ 0 & \text{otherwise.} \end{cases}$$

We may regard the chopping procedure as constructing a decision list based on threshold functions. To see this, suppose that the chopping procedure applied for the function f involves k chops. Let $f \in DL(T_n)$ be defined as follows. If, at stage i of the procedure, the hyperplane with equation $\sum_{i=1}^{n} w_i y_i = \theta$ is used to chop off positive (respectively, negative) points and these lie on side of the hyperplane with equation $\sum_{i=1}^{n} w_i y_i > \theta$, then we take as the ith term of the decision list the pair $(f_i, 1)$ (respectively, $(f_i, 0)$), where $f_i \leftarrow [w, \theta]$; otherwise we take the ith term to be $(g_i, 1)$ (respectively, $(g_i, 0)$), where $g_i \leftarrow [-w, -\theta]$. Then, as can easily be seen, the function f is represented by the decision list constructed. We call a decision list based on threshold functions a *threshold decision list.*

7.3.3 Threshold decision lists and threshold networks

Theorem 3.9 showed that any decision list based over the set of single literals is a threshold function. A more general version of this result can be given. To state this, we first define, for a set K of functions from $\{0,1\}$ to $\{0,1\}$, the notion of a *threshold function over* K: this is defined to be a function f of the form

$$f(y) = \begin{cases} 1 & \text{if } \sum_{i=1}^{l} \alpha_i f_i(y) > \theta, \\ 0 & \text{if } \sum_{i=1}^{l} \alpha_i f_i(y) < \theta \end{cases}$$

for some constants $\alpha_1, \alpha_2, \ldots, \alpha_l$ and θ and for some functions $f_1, f_2, \ldots, f_l$ in K. We write $g \leftarrow [(f_1, f_2, \ldots, f_l), \alpha, \theta]$.

The following result is similar to Theorem 3.9.

Theorem 7.3 *If the Boolean function f is a decision list over a set K of Boolean functions, then it is a linear threshold function over K.*

Proof. The proof is similar to that of Theorem 3.9. Suppose the decision list is

$$f = (f_1, c_1),\ (f_2, c_2), \ldots,\ (f_r, c_r).$$

We iteratively construct a threshold function over K which computes the same function as the decision list. Since a decision list gives, by default, output 0 if the given input y fails all tests, we may assume that $c_r = 1$. Therefore the last term, in isolation, since it is a one-term decision list, computes exactly the function f_r, which has the representation $f_r \leftarrow [(f_r), 1, \frac{1}{2}]$ as a threshold function over K. We then use an inductive procedure to construct a linear threshold function computing the same function as the decision list comprising the last m terms of decision list f for each m from 1 through to r; the final threshold function is the required one. Suppose then that $f^m = (f_{r-m+1}, c_{r-m+1}), \ldots, (f_r, c_r)$ is the decision list over K consisting of the last m terms of f and that we have constructed a linear threshold function representation

$$f^m \leftarrow [(f_{r-m+1}, \ldots, f_r), \alpha, \theta]$$

of f^m. We denote by $\|\alpha\|_1$ the 1-norm of α, $\|\alpha\|_1 = \sum_{i=1}^{m} |\alpha_i|$, and we set $M = \|\alpha\|_1 + |\theta| + 1$. Suppose that $c_{r-m} = 1$. Then it can be seen that the threshold function

$$g^{m+1} \leftarrow [(f_{r-m}, f_{r-m+1}, \ldots, f_r), (M, \alpha), \theta]$$

is exactly the same function as the decision list

$$f^{m+1} = (f_{r-m}, c_{r-m}), \ldots, (f_r, c_r).$$

Next, if $c_{r-m} = 0$, then the linear threshold function

$$g^{m+1} \leftarrow [(f_{r-m}, f_{r-m+1}, \ldots, f_r), (-M, \alpha), \theta]$$

computes the same function as f^{m+1}. (Verification of these statements can be carried out as in the proof of Theorem 3.9.) The result follows. □

This result show that any threshold decision list can be computed by a two-layer threshold network with one output unit, having one hidden unit for each term of the decision list. For, the threshold functions in the decision list can be realized by hidden threshold units, and the output unit can then implement the desired threshold function over these threshold functions. So we have another procedure for representing a given Boolean function by a two-layer threshold network. First, we use the chopping procedure to construct a threshold decision list for the function. Then, Theorem 7.3 provides a recursive procedure for translating this into a threshold function over the threshold functions in the threshold decision list. That is, we may determine suitable weights for the connections between the hidden units and the output unit. The number of units in the hidden layer is certainly no more than 2^n, since each "chop" slices off at least one example.

7.4 Universal networks

7.4.1 Universal networks for Boolean functions

A *universal network* for Boolean functions on $\{0,1\}^n$ is a threshold network which is capable of computing every Boolean function of n variables. Theorem 7.1 shows that the two-layer threshold network with n inputs, 2^n units in the hidden layer, and one output unit is universal. The chopping procedure also establishes the existence of a universal network. We have seen that it implies that a Boolean function can be represented by a two-layer threshold network with k hidden units, where $k \leq 2^n$ is the number of chops required in the chopping procedure. By setting the weights to and from $2^n - k$ of the hidden units to be 0, and the thresholds to be 1 (so that these units are effectively absent and there are only k "active" hidden units), we see again that the two-layer network described above is universal. It should be noted, however, that the representations of a particular function resulting from the DNF-based approach and the chopping procedure approach can be very different. For instance, we have seen from Example 7.2 that the chopping procedure employed on PARITY_n requires at most n chops, and this implies that we can represent parity by the threshold network, with only n "active" hidden units. On the other hand, since parity has 2^{n-1} prime implicants (one for each positive example), the DNF-based approach requires 2^{n-1} active hidden units.

7.4.2 Sizes of universal networks

The observations made above imply the existence of a two-layer universal network for Boolean functions having $O(2^n)$ threshold units. The question arises as to whether this can be bettered, perhaps by using more layers. By an easy counting argument, we can

obtain a lower bound on the size of *any* universal network, regardless of its structure. Here, we use the Ω-notation, which is similar in nature to the O-notation used earlier. We say that $g(n)$ is $\Omega(h(n))$ if there is a positive constant K such that, for all n greater than some fixed $n_0 \in \mathbb{N}$, $g(n) \geq K\, h(n)$. (Note that this notation is sometimes used to indicate the weaker assertion that there are infinitely many n for which $g(n) \geq K\, h(n)$.)

Theorem 7.4 *Any universal network for Boolean functions has at least* $\Omega(2^{n/2}/\sqrt{n})$ *threshold units.*

Proof. Suppose we have a threshold network with N threshold units. The output of the network on a given input depends on the outputs of the threshold units. Suppose the threshold units are labeled $1, 2, \ldots, N$ and that the number of connections into unit i is d_i. As the inputs to the network range through all of $\{0,1\}^n$, the resulting inputs to unit i range through some subset of $\{0,1\}^{d_i}$. By Theorem 4.4, the number of functions computable by unit i on this set of inputs is no more than $C(2^n, d_i+1)$. Assume $d_i < 2^n$ (which we can do, because $d \geq 2^n$ would imply that the size of the network is at least $\Omega(2^n)$, implying the result immediately). Then

$$C(2^n, d_i + 1) < 2^{nd_i+1}.$$

(This follows, for $d_i \geq e$, from $C(2^n, d_i + 1) \leq 2\,(e2^n/d_i)^{d_i}$ and from the precise expression for $C(2^n, d_i + 1)$, when $d_i = 1$ or 2.) The total number of functions computable by the network is therefore bounded by

$$\prod_{i=1}^{N} 2^{nd_i+1} = 2^{N+n\sum_{i=1}^{N} d_i}.$$

Now, by the directed graph version of the "handshaking lemma" of graph theory, $\sum_{i=1}^{N} d_i$ equals the number of connections in the network, which is at most $\binom{N}{2} + Nn$ (the first contribution being a bound on the connections between threshold units, and the second the maximum number of connections from the inputs to the threshold units). For $N > 2n$, this is no more than N^2. So the network can compute at most 2^{N+nN^2} functions. In order to be universal, however, it must compute all 2^{2^n} Boolean functions. Thus, $N+nN^2 \geq 2^n$, from which we obtain $N^2 = \Omega(2^n/n)$ and hence $N = \Omega(2^{n/2}/\sqrt{n})$, as required. □

This result applies to networks of any shape, that is, any number of layers. The following slightly better bound can be given for the special case of two-layer networks.

Theorem 7.5 *Any two-layer universal network for Boolean functions has at least* $\Omega(2^n/n^2)$ *threshold units.*

Proof. We modify slightly the previous proof. In this special case, we may bound the number of connections by $nN + N$ since there are at most nN connections from the inputs to the threshold units, and N connections from the threshold units in the hidden layer to the output unit. So the number of functions computable by the net is no more than $2^{N+n(nN+N)}$, and hence, if the network is universal, $N + +nN + n^2N \geq 2^n$, and $N = \Omega(2^n/n^2)$. □

$k-1$. *If m_k is the number of elements of rank P, then any antichain A in P satisfies $|A| \leq B = \max_k m_k$.*

Proof. A chain in P is said to be *maximal* if no elements of P can be added to it to obtain another chain. Given the properties of the rank function, and assuming the minimal rank is 0 and the maximal rank is n, a maximal chain takes the form

$$x_0 \leq x_1 \leq x_2 \leq \cdots \leq x_{n-1} \leq x_n,$$

where each x_i is of rank i. Let P_k be the set of all elements of rank k. For $x \in P_k$, the number of maximal chains containing x is $N_k = b_k b_{k-1} \ldots b_1 a_k a_{k+1} \ldots a_{n-1}$. If we denote by N the total number of maximal chains, then, since each such chain contains precisely one member of P_k, $N = N_k|P_k| = N_k m_k$. Let $s_k = |A \cap P_k|$ be the number of elements of the antichain A belonging to P_k. Since every chain meets the antichain A in at most one element (no two elements of the antichain being comparable), we must have

$$\sum_{k=0}^{n} s_k N_k \leq N,$$

which, given that $N_k = N/m_k \geq N/B$, implies

$$(N/B) \sum_{k=0}^{n} s_k \leq \sum_{i=1}^{k} s_k N_k \leq N,$$

so

$$|A| = \sum_{k=1}^{n} s_k \leq B,$$

as required. □

An important special case of this applies to the poset $(\{0,1\}^m, \preceq)$. (This poset may equivalently be thought of as the poset of all subsets of an n-set under the ordering of set-inclusion: $x \in \{0,1\}^n$ represents the set $\{i : x_i = 1\}$.) A suitable rank function here is given by $r(x) = \|x\| = |\{i : x_i = 1\}|$. (This is often called the "weight" of x: it is not to be confused with the 1-norm or 2-norm of x.) The corresponding a_k and b_k are $a_k = n - k$ and $b_k = k$. Then, $m_k = \binom{n}{k}$, and this is maximized when $k = \lfloor n/2 \rfloor$. Thus, we have the following theorem.

Theorem 7.8 *For any antichain A in the poset $(\{0,1\}^n, \preceq)$,*

$$|A| \leq \binom{n}{\lfloor \frac{n}{2} \rfloor}.$$

Note that this bound is tight: the set of all x of rank $\lfloor n/2 \rfloor$ (that is, all x with this many entries equal to 1) is an antichain of this cardinality.

7.5.3 Proof of Theorem 7.6

The output of a two-layer stratified network on a particular input is determined by the outputs of the hidden threshold units. Each of these hidden threshold units computes the characteristic function of a hyperplane. We have seen that the Boolean hypercube may be regarded as a graph, where x and y are adjacent if and only if they

differ in exactly one coordinate. Now, for each edge xy of the Boolean hypercube, $\text{PARITY}_n(x) \neq \text{PARITY}_n(y)$, so the network must have different outputs on inputs x and y; in particular, therefore, at least one of the hidden threshold units must have different outputs on x and y. Geometrically, this means that the hyperplane separates x from y; that is, it "cuts" the edge of $\{0,1\}^n$ joining x and y. The following result is the key to the proof.

Theorem 7.9 *Any hyperplane can cut at most $C(n)$ edges of $\{0,1\}^n$, where*

$$C(n) = \left(n - \left\lfloor \frac{n}{2} \right\rfloor\right) \binom{n}{\lfloor \frac{n}{2} \rfloor}.$$

Proof. First, we show that, for the purposes of upper bounding the number of edges cut, we may assume that all the coefficients (weights) of the normal vector to the hyperplane are nonnegative. To see this, we indicate why if H is a hyperplane with some negative weights, and H cuts m edges, then there is a hyperplane H' with nonnegative weights cutting at least m edges. Suppose, without loss of generality, that H is given by the equation $\langle w, x \rangle = 1$, where $w = (w_1, w_2, \ldots, w_n)$, with $w_i < 0$ for $i = 1, 2, \ldots, k$ and $w_i \geq 0$ for $i > k$. We may suppose, by the discreteness of $\{0,1\}^n$, that no $x \in \{0,1\}^n$ lies on the hyperplane. For $x \in \{0,1\}^n$, define x' as

$$x'_i = \begin{cases} 1 - x_i & \text{for } i = 1, 2, \ldots, k, \\ x_i & \text{for } i = k+1, \ldots, n, \end{cases}$$

and define w' by

$$w'_i = \begin{cases} -w_i & \text{for } i = 1, 2, \ldots, k, \\ w_i & \text{for } i = k+1, \ldots, n. \end{cases}$$

Let H' be the hyperplane with equation

$$\langle w', x \rangle = 1 - \sum_{i=1}^{k} w_i.$$

If the edge xy is cut by H, then $\langle w, x \rangle > 1 > \langle w, y \rangle$ or $\langle w, x \rangle < 1 < \langle w, y \rangle$. Assume the former, for convenience. Then it can be shown that

$$\langle w', x' \rangle > 1 - \sum_{i=1}^{k} w_i > \langle w', y' \rangle.$$

Moreover, since xy is an edge, x and y differ only in one coordinate, and hence so do x' and y'. It follows that $x'y'$ is an edge of the hypercube, and it is cut by H'. Clearly, the mapping $xy \mapsto x'y'$ is injective, so H' cuts at least as many edges as H does. Note that the "threshold" $1 - \sum_{i=1}^{k} w_i$ is positive since the weights w_i are negative for $i \leq k$. Therefore, by dividing each entry of w' by this threshold, we obtain a weight-vector w'' with nonnegative entries, such that H' has an equation $\langle w'', x \rangle = 1$.

Let E be the set of edges of the hypercube, where we write a typical edge as xy where $x \preceq y$ and x, y differ only in one coordinate. Define a partial order $\leq$ on E by $xy \leq x'y'$ if and only if $y \preceq x'$. Now we show that the set S of edges cut by H is an antichain in $(E, \leq)$. Recall that H has an equation of the form $\langle w, x \rangle = 1$, where w has nonnegative entries, and that no points of the hypercube lie on H. Suppose that

$e = xy$ and $e' = x'y'$ both belong to S and that $e \leq e'$. Then, recalling that edges xy are written in such a way that $x \preceq y$, and using the fact that all the entries of w are nonnegative, we must have $\langle w, x\rangle < 1 < \langle w, y\rangle$ and $\langle w, x'\rangle < 1 < \langle w, y'\rangle$. Because $e \leq e'$, $y \preceq x'$ and hence $\langle w, x' - y\rangle \geq 0$, since the entries of w are nonnegative. It follows that

$$1 < \langle w, y\rangle \leq \langle w, y\rangle + \langle w, x' - y\rangle = \langle w, x'\rangle,$$

a contradiction, since $\langle w, x'\rangle < 1$.

Define the rank function $r : E \to \{0, 1, \ldots, n\}$ by

$$r(xy) = \|x\| = |\{i : x_i = 1\}|.$$

A typical edge $e = xy$ of rank k has $\|x\| = k$, $\|y\| = k + 1$, and $x \preceq y$. This is covered by edge $e' = x'y'$ (with respect to $\leq$) precisely when $y = x'$ (and then, $x'y'$ being an edge, we have $y \preceq y'$ and $\|y'\| = k+2$). Each e of rank k is therefore covered by exactly the same number $(n - k - 1)$ of elements of E, and these are all of rank $k + 1$; equally, any e of rank k covers the same number $(n - k)$ of elements of E, each of rank $k - 1$. It follows from this that the poset $(E, \leq)$ satisfies the conditions of Theorem 7.7, and since the set S of edges cut by H is an antichain, we have $|S| \leq \max_k m_k$, where m_k is the number of edges of rank k. Now, $e = xy$ is of rank k if and only if x is a $\{0, 1\}$-vector with k entries equal to 1. Because xy is an edge, we also have that y equals x but for one entry, in a place where x has 0, which in y is equal to 1. Hence $m_k = (n - k)\binom{n}{k}$. Now,

$$m_k = (n - k)\binom{n}{k} = n\binom{n-1}{k},$$

and this is clearly maximized when $k = \lfloor n/2 \rfloor$, so

$$|S| \leq m_{\lfloor n/2 \rfloor} = C(n),$$

as required. □

We can now complete the proof of Theorem 7.6. By the discussion above, every edge of $\{0, 1\}^n$ must be cut by at least one of the hyperplanes determined by the hidden units in the two-layer network. There are $n2^n$ edges, and, by Theorem 7.9, each hyperplane can cut at most $C(n)$ edges. Therefore, the number of hyperplanes, which is the number of hidden units, must be at least $n2^n/C(n)$. Now,

$$C(n) = O\left(n\binom{n}{\lfloor \frac{n}{2} \rfloor}\right) = O\left(n\frac{2^n}{\sqrt{n}}\right) = O\left(\sqrt{n}2^n\right),$$

where we have used Stirling's formula,

$$n! \simeq \sqrt{2\pi n}n^n e^{-n} \quad \text{as } n \to \infty.$$

Therefore, the number N of hyperplanes is $\Omega(n2^n/(\sqrt{n}2^n)) = \Omega(\sqrt{n})$, as required.

7.6 Additional remarks and bibliographical notes

The chopping procedure discussed here is similar in nature to one set forth by Jeroslow [51], but at each stage in his procedure, only positive examples may be "chopped off" (not positive *or* negative). Jeroslow's method requires 2^{n-1} iterations to define the

parity function, since at each stage it can "chop off" only one positive point. If we apply this construction to the series of hyperplanes resulting from the Jeroslow method, we get a restricted form of decision list—one in which all terms are of the form $(f_i, 1)$. But such a decision list is quite simply the *disjunction* $f_1 \vee f_2 \vee \ldots \vee f_k$. The decision lists arising from the chopping procedure described here are less restricted and may provide a more convenient representation of a given function. (Since fewer hyperplanes might be used, the decision list could be smaller.)

The idea of threshold decision lists appears in papers of Marchand et al. [66, 67], who termed them "neural decision lists." In [66], a construction of a "cascade" network from a threshold decision list is presented. The construction of a threshold network outlined in this chapter is from [6]. The papers [66, 67] consider the algorithmic implementations of constructing a threshold decision list representation, and the subsequent network representation.

Theorem 7.4 is due to Nechiporuk [77]. A related result, bounding the number of weights required to represent all Boolean functions on a given domain (not necessarily all of $\{0,1\}^n$) can similarly be obtained; see [95, 30].

Section 7.5 closely follows the presentation given in [95]. Theorem 7.9 is due to O'Neil [80]. For the details involving the extension of Theorem 7.6 to the nonstratified case, see [95] (Exercise 3.15). Our proof of Theorem 7.7 follows that given by Bollobás [22]. (See [2] for related results.)

There are many publications on the circuit complexity of feed-forward linear threshold networks (or "threshold circuits"). The book by Siu, Roychowdhury, and Kailath [95] provides an excellent overview of the area. For the results presented here, we used some fairly elementary techniques, but many useful results have recently been obtained on the powers of networks (and of polynomial threshold functions) using fairly sophisticated techniques. In particular, ideas from the *harmonic analysis* of Boolean functions (see [95, 25, 90], for instance) have proven extremely useful, as have ideas concerning the approximation of functions by rational functions (see [82, 95], for example). The size of networks representing the parity function has been investigated in some depth; see [95] and the references therein, particularly [82, 96, 49].

Other, more restricted types of threshold network have been considered. Mayoraz [69], for example, has examined the power of "majority" networks, in which each unit computes the majority function of some of its inputs. On the other hand, more powerful types of network have also been studied. In particular, sigmoid networks have been shown explicitly to be more powerful than threshold networks. Maass, Schnitger, and Sontag [63] and DasGupta and Schnitger [33] have proved that there are functions easily computable by small sigmoid networks but which can be computed only by much larger threshold networks.

Chapter 8

Specifying Sets

8.1 Introduction

Given a class H of Boolean functions on n variables, a *specifying set* (or *test set*) S for a function f in the class is a subset of $\{0,1\}^n$ with the property that if $g \in H$ and g equals f on S, then, necessarily, $g = f$. This notion is useful for two reasons. First, the minimal size of a specifying set gives a lower bound on the amount of training data needed in the worst case when learning by any "consistent" learning algorithm, when the aim is to find an unknown target function exactly. (See the next chapter.) Second, one might want to be sure that a neural network computes exactly the correct function. This can be "tested" using a specifying set: we simply check that the output is what it ought to be for each input in a specifying set.

In this chapter, we shall focus mainly on the specification of Boolean threshold functions. Some tight extremal results on the size of specifying sets are presented.

8.2 Upper bounds on specification numbers of threshold functions

8.2.1 Introduction

Suppose that H is a set of Boolean functions. Then for $f \in H$, the *specification number* of f (in H) is the least cardinality of a specifying set for f in H. The specification number of f in H will be denoted by $\sigma_H(f)$, or simply by $\sigma(f)$ when H is clear. We will be dealing with the class of threshold functions, and for convenience and also to emphasize dependence on n, we shall denote $\sigma_{T_n}(f)$ by $\sigma_n(f)$.

In general, since a Boolean function is uniquely determined by its value on all points of $\{0,1\}^n$, the specification number of any function of n variables, within any class of Boolean functions, is no more than 2^n. It is, in fact, the case that the specification number of threshold functions can be this large. For example, to distinguish the identically-0 function from the functions having precisely one positive example, one needs to present every single example $x \in \{0,1\}^n$: if x is not presented, then the possibility that the function in question is that which has x as its single positive example has not been ruled out. (Note that all these functions are indeed threshold functions.) In this section, we

shall present some general approaches to bounding the specification number of threshold functions, and we shall also obtain some extremal results.

8.2.2 The boundary approach

For our first approach to specifying threshold functions, we regard the points in $\{0,1\}^n$ as the vertices of the n-dimensional Boolean hypercube. For any $t \in T_n$, $T(t)$, the set of true points (or, positive examples) of t and $F(t)$, the set of false points (negative examples) of t, are connected subsets (in the graph-theoretic sense) of the hypercube. Let $\Delta(t)$ be the set of examples x such that there is y adjacent to x with $t(x) \neq t(y)$. Thus, $\Delta(t)$ may be thought of as the examples on the *boundary* between $T(t)$ and $F(t)$. Denoting the cardinality of $\Delta(t)$ by $\partial(t)$, we have the following bound.

Theorem 8.1 (Boundary result) *For $t \in T_n$, not the identically-0 function or the identically-1 function, $\sigma_n(t) \leq \partial(t)$, where $\partial(t)$ is the number of boundary examples of t.*

Proof. If t is nonconstant, then it has boundary examples. Any set which contains the examples in $\Delta(t)$ is a specifying set for t. This follows directly from the connectedness of $T(t)$ and $F(t)$. For, $\Delta(t)$ specifies edges of the hypercube whose deletion results in two connected components, one of which is $T(t)$ and one of which is $F(t)$. Any other threshold function agreeing with t on $\Delta(t)$ must therefore have exactly the same set of true and false points, and hence equals t. □

The following consequence of this result is useful for functions with a small number of true points or false points.

Theorem 8.2 *For $t \in T_n$, not the identically-0 or the identically-1 function, let*

$$m = \min\left(|T(t)|, |F(t)|\right).$$

Then $\sigma_n(t) \leq m(n-1)+2$.

Proof. Suppose, without loss, that $m = |T(t)|$. Let P be the subgraph of the Boolean hypercube vertex-induced by $T(t)$. Then P is connected, and so has at least $m-1$ edges. It follows, since each positive example has n neighbors, that the number of boundary vertices satisfies $\partial(t) \leq m + mn - 2(m-1) = m(n-1)+2$. □

We have seen that the identically-0 function has specification number 2^n. Equally, so does the identically-1 function. Consider the functions of the form

$$t(x_1, x_2, \ldots, x_n) = 1 \iff x_i = b,$$

where i is an integer between 1 and n and b is 0 or 1. We call such functions *hyperface functions*. It is easily seen that any hyperface function has specification number 2^n. As an application of the boundary result, we can characterize the functions of T_n which have largest possible specification numbers.

Theorem 8.3 *If $t \in T_n$ has $\sigma_n(t) = 2^n$ then t is either the identically-0 function, the identically-1 function, or a hyperface function.*

Proof. Suppose t is neither the identically-0 function nor the identically-1 function and that $\sigma_n(t) = 2^n$. Then, by Theorem 8.1, $\partial(t) = 2^n$, and all 2^n examples are boundary

examples of $t \leftarrow [w, \theta]$. Without loss, assume $w_i \geq 0$ for $1 \leq i \leq n$ and that $\theta > 0$. (This is an assumption we can make since it is not important which point we define to be the origin; thus, we may take as the origin the negative example furthest from the defining hyperplane.) The negative example $00\ldots00$ is a boundary example of t, and so for some i the example e_i is a positive example, from which we deduce $w_i \geq \theta$. Also, since the positive example $11\ldots11$ is a boundary example, for some j, the example $11\ldots101\ldots1$ with a 0 in the jth coordinate is a negative example of t. Since $w_i \geq \theta$, we must have $i = j$. Then $w_i \geq \theta$ and $\sum_{j \neq i} w_i < \theta$, so that t is the hyperface function $x_i = 1$. □

8.2.3 Using the partial order

For our second approach to bounding specification numbers, we use the partial order on $\{0,1\}^n$. Recall that $x \preceq y$ if $x_i = 1$ implies $y_i = 1$. We need one more important idea before proceeding. We say that t *depends on coordinate* i if there are $x^{(0)}, x^{(1)}$ differing only in their ith entries, such that $t(x^{(0)}) = 0, t(x^{(1)}) = 1$. In this case, the sign of w_i can be determined from $x^{(0)}$ and $x^{(1)}$ since $\langle w, x^{(1)} \rangle \geq \theta > \langle w, x^{(0)} \rangle$. Suppose that t depends on all the coordinates. Then we may, without loss, suppose that t is increasing, so that $t(x) = 1$ and $x \preceq y$ imply $t(y) = 1$. (We can assume that the function is increasing because the origin can be taken to be any point and we can determine from t which point is acting as the origin. Equivalently, recalling that any threshold function is unate, we may suppose that certain variables have been negated to obtain an increasing function.) Recall that, for an increasing function t, $\mathrm{MFP}(t)$ is the set of maximal false points and $\mathrm{MTP}(t)$ is the set of minimal true points of t. Then, we have the following result.

Theorem 8.4 *Suppose $t \in T_n$ is increasing and depends on all the coordinates. Then the set $\mathrm{MFP}(t) \cup \mathrm{MTP}(t)$ specifies t.*

Proof. Suppose $t \leftarrow [w, \theta]$ and that, without loss of generality, $0 < w_1 \leq w_2 \leq \ldots \leq w_n$. We claim that the signs of $w_1, w_2, \ldots, w_n$ can be deduced solely from the classification of $\mathrm{MFP}(t) \cup \mathrm{MTP}(t)$ and the fact that $t \in T_n$. Proceeding inductively, suppose that $w_1, \ldots, w_{m-1}$ are known to be positive; we show that it can be deduced that w_m is positive. Since t depends on coordinate m, there are y, y', differing only in that $y_m = 0$ and $y'_m = 1$, such that $\langle w, y \rangle < \theta$ and $\langle w, y' \rangle \geq \theta$. Then $y \preceq x$ for some $x \in \mathrm{MFP}(t)$ and $x_m = 0$ since otherwise $y' \preceq x$. Let x' be the example equal to x except that $x'_m = 1$; then x' is a positive example, since $x \preceq x'$, $x \neq x'$, and $x \in \mathrm{MFP}(t)$. Take $z' \in \mathrm{MTP}(t)$ with $z' \preceq x'$. Then,

$$\langle w, z' \rangle \geq \theta > \langle w, x \rangle = \langle w, x' \rangle - w_m.$$

So, $\langle w, x' - z' \rangle < w_m$, and hence x' and z' do not differ in coordinates $m+1, m+2, \ldots, n$. Let C be the set of coordinates $i < m$ such that $x'_i \neq z'_i$. Now we have

$$0 < \langle w, z' \rangle - \langle w, x \rangle = w_m - \sum_{i \in C} w_i.$$

The above inequality can be deduced solely from the facts that $z' \in \mathrm{MTP}(t)$ and $x \in \mathrm{MFP}(t)$. Thus, given the additional information that $w_i > 0$ for $i \in C \subseteq \{1, \ldots, m-1\}$, it can be deduced that $w_m > 0$ also. Therefore, given only the clas-

sification of $\mathrm{MFP}(t) \cup \mathrm{MTP}(t)$, one can deduce that t is increasing. Furthermore, t is specified by $\mathrm{MFP}(t) \cup \mathrm{MTP}(t)$: for any example y, there will be a point x in $\mathrm{MFP}(t)$ with $y \preceq x$ or there will be $z \in \mathrm{MTP}(t)$ with $z \preceq y$. In the first case, $t(y) = 0$ and in the second $t(y) = 1$. □

As an immediate corollary of this result, we have the following bound.

Theorem 8.5 *If $t \in T_n$ depends on all the coordinates, then*

$$\sigma_n(t) \leq \binom{n+1}{\lfloor \frac{n+1}{2} \rfloor}.$$

Proof. We may, without loss, suppose that t is increasing. (Clearly if t depends on all the coordinates but is not increasing, one may simply shift the origin to yield an analogous specifying set. Equivalently, transform the order $\preceq$ and produce as a specifying set the minimal positive examples and maximal negative examples with respect to the transformed ordering.) Then $\sigma_n(t) \leq |\mathrm{MFP}(t)| + |\mathrm{MTP}(t)|$. Now form a set A consisting of all points $x1$ for $x \in \mathrm{MFP}(t)$ and all points $y0$, for $y \in \mathrm{MTP}(t)$. We now show that A is an antichain in the poset $(\{0,1\}^{n+1}, \preceq)$. It is clear that, since the elements of $\mathrm{MFP}(t)$ are incomparable, so are the elements of the form $x1$, where $x \in \mathrm{MFP}(t)$. Similarly, the points $y0$ for $y \in \mathrm{MTP}(t)$ are incomparable. Also, for any $x \in \mathrm{MFP}(t)$ and $y \in \mathrm{MTP}(t)$, $x1$ and $y0$ are incomparable since it cannot be true that $y \preceq x$ (because t is increasing). Sperner's theorem shows that the maximal size of an antichain in $(\{0,1\}^{n+1}, \preceq)$ is the quantity stated, and the result follows since $|A| = |\mathrm{MFP}(t) \cup \mathrm{MTP}(t)|$. □

Let g_n^k be the function which has as positive examples the points with at least k ones. Then $g_n^k \leftarrow [(1,1,\ldots,1),k] \in T_n$, and we shall call it the *weight-at-least-k* function.

Theorem 8.6 *The weight-at-least-k function g_n^k has specification number $\binom{n+1}{k}$.*

Proof. If x, y are adjacent vertices of the hypercube then their weights (number of ones) differ by 1. Hence if $g_n^k(x) = 1$ and $g_n^k(y) = 0$, then x has weight k and y has weight $k-1$. It follows that the set $\Delta(g_n^k)$ consists of all examples with weight k and all examples of weight $k-1$, so that

$$\sigma_n(g_n^k) \leq \partial(g_n^k) = \binom{n}{k-1} + \binom{n}{k} = \binom{n+1}{k}.$$

We now show that all examples of weight k and of weight $k-1$ must be presented to specify g_n^k. (Then every specifying set must contain all such examples, yielding the lower bound.) Let x be an example of weight $k-1$, which we may suppose is $x = 11\ldots10\ldots0$. Let $w = 11\ldots11$, and let $\theta = k$. Then $g_n^k \leftarrow [w, \theta]$. Now let

$$w' = (\underbrace{1+1/k, 1+1/k, \ldots, 1+1/k}_{k-1}, 1, 1, \ldots, 1)$$

and note that $g_n^k \leftarrow [w', k]$. Now, $h \leftarrow [w', k-1/k]$ misclassifies x as a positive example and correctly classifies all other examples. Hence x must be presented if g_n^k is to be specified. The treatment for examples of weight k is similar. □

This shows that we can have equality in Theorem 8.5, achieved by the weight-at-least-$\lfloor (n+1)/2 \rfloor$ function. In fact, we can characterize precisely those threshold functions which depend on all the variables and have highest specification number.

Theorem 8.7 *Suppose $t \in T_n$ depends on all the coordinates. Then t has maximum possible specification number for such a function if and only if one of the following holds:*

(i) *n is odd and there is $v \in \{0,1\}^n$ such that $t(x) = 1$ if and only if x and v agree on at least $(n+1)/2$ entries;*

(ii) *n is even and there is $v \in \{0,1\}^n$ such that $t(x) = 1$ if and only if x and v agree on at least $n/2$ entries;*

(iii) *n is even and there is $v \in \{0,1\}^n$ such that $t(x) = 1$ if and only if x and v agree on at least $(n/2+1)$ entries.*

Proof. It is known that if n is even, then there is exactly one antichain of the maximum possible size $\binom{n}{\lfloor n/2 \rfloor}$ in the poset $(\{0,1\}^n, \preceq)$—namely, the set of all x having $n/2$ entries equal to 1 (that is, having weight $n/2$). For n odd, the only antichains of size $\binom{n}{\lfloor n/2 \rfloor}$ are the collection of all points of weight $(n-1)/2$, and the collection of all points of weight $(n+1)/2$. Consider the proof of Theorem 8.5 in the case n odd. Then we have equality in the bound if and only if the antichain A is the set of all examples of weight $(n+1)/2$, from which it follows that the maximal false points of t are all the points of weight $(n-1)/2$ and the minimal true points are all those of weight $(n+1)/2$. But this means that if n is odd, if t is increasing and depends on all the coordinates and we have equality in Theorem 8.5 then t must be the weight-at-least-$(n+1)/2$ function. Conversely, such a function meets the upper bound. If t is not increasing, then we may take some other point y as the origin to transform t to an increasing function. Then t is as described in the statement of this result, with $v = \mathbf{1} - y$, where $\mathbf{1}$ is the all-1 vector. The proof for n even is similar. □

8.3 A lower bound and its attainment

We have characterized the functions in T_n with largest specification numbers. Now we turn our attention to those functions with the lowest possible specification numbers. We say that a set of $n+1$ points in $\mathbb{R}^n$ is in *general position* if the points do not all lie on a hyperplane.

Theorem 8.8 *For any $t \in T_n$, any specifying set for t must contain $n+1$ examples in general position. In particular, $\sigma_n(t) \geq n+1$. Furthermore, equality can hold in this bound.*

Proof. Suppose that T is a set of examples not containing $(n+1)$ points in general position. Then all the points of T lie in some hyerplane with equation $\langle w', x \rangle = c$ for some $w' \in \mathbb{R}^n$ and $c \in \mathbb{R}$. Let $t \leftarrow [w, \theta]$ be any function in T_n and let η be any real number. Then if $h_\eta \leftarrow [w + \eta w', \theta + \eta c]$, h_η agrees with t on T because if $x \in T$, then $\langle w', x \rangle = c$ and

$$\langle w + \eta w', x \rangle = \langle w, x \rangle + \eta \langle w', x \rangle = \langle w, x \rangle + \eta c,$$

so that, for $x \in T$,

$$t(x) = 1 \iff \langle w, x \rangle \geq \theta \iff \langle w + \eta w', x \rangle \geq \theta + \eta c.$$

Now, choose $y \in \{0,1\}^n$ which does not lie on the flat determined by T, so that $\langle w', y \rangle \neq c$. Then for some values of η, y is a negative example of h_η, and for some other values, y

is a positive example of h_η. In other words, the set T does not specify t, since there are at least two distinct functions consistent with t on the set. To see that the lower bound can be attained, note that if t has just one positive example, then the set consisting of this positive example and its n neighbors is a set of length $n+1$ which specifies t (because these $n+1$ points are the boundary points). □

The functions having exactly one positive example or one negative example have the least possible specification number $n+1$. But these are not the only such hypotheses, as we now show.

Let us recursively define a Boolean function to be *nested* by the following: both functions of 1 variable are nested, and t_n, a function of n variables, is nested if $t_n = u_n \star t_{n-1}$ or $t_n = \bar{u}_n \star t_{n-1}$, where $\star$ is $\vee$ (the OR connective) or $\wedge$ (the AND connective) and t_{n-1} is a nested function of $n-1$ variables. (Here, we mean that t_{n-1} acts on the first $n-1$ entries of its argument.) Any nested Boolean function is a threshold function, because if $t_{n-1} \leftarrow [w, \theta]$ is nested and in T_{n-1}, then

$$u_n \wedge t_{n-1} \leftarrow [(w, M), \theta + M], \quad u_n \vee t_{n-1} \leftarrow [(w, M), \theta],$$

$$\bar{u}_n \wedge t_{n-1} \leftarrow [(w, -M), \theta], \quad \bar{u}_n \vee t_{n-1} \leftarrow [(w, -M), \theta - M]$$

for a suitably large M. Examples of nested functions include the functions with formulas $u_1 \wedge u_2 \wedge \ldots \wedge u_n$ and $u_1 \vee u_2 \vee \ldots \vee u_n$, with only one positive example, and one negative example (respectively), and (for n even) the function

$$f_n = u_n \wedge (u_{n-1} \vee (u_{n-2} \wedge (u_{n-3} \vee (\ldots (u_2 \wedge u_1)) \ldots)),$$

which we met earlier in considering the sizes of weights for threshold functions. The next result shows that all nested functions are easily specified.

Theorem 8.9 *The specification number of any nested function in T_n is $n+1$.*

Proof. It suffices to prove that $|\mathrm{MFP}(t) \cup \mathrm{MTP}(t)| \leq n+1$ for any increasing nested function t in T_n. This is clearly true when $n = 1$, for in this case the total number of examples is 2. Suppose the statement is true for all nested functions of $(n-1)$ variables. Suppose t_n is a nested function in T_n. If $t_n = u_n \vee t_{n-1}$, then $\mathrm{MFP}(t_n)$ consists of all examples $x0$ where $x \in \mathrm{MFP}(t_{n-1})$, and $\mathrm{MTP}(t_n)$ consists of all examples $y0$, where $y \in \mathrm{MTP}(t_{n-1})$ together with the single example $00\ldots01$. If $t_n = u_n \wedge t_{n-1}$ then $\mathrm{MFP}(t_n)$ consists of all examples $x1$ where $x \in \mathrm{MFP}(t_{n-1})$, together with the example $11\ldots10$, and $\mathrm{MTP}(t_n)$ consists of the examples $y1$ where $y \in \mathrm{MTP}(t_{n-1})$. In either case, $\sigma_n(t_n) \leq |\mathrm{MFP}(t_n)| + |\mathrm{MTP}(t_n)| \leq 1 + |\mathrm{MFP}(t_{n-1})| + |\mathrm{MTP}(t_{n-1})| = 1 + n$, as required. □

We may extend the definition of nested function by allowing the variables to be permuted (or relabeled), so that we would say for example that the function $u_2 \wedge (u_3 \vee u_1)$ is nested. Clearly the above result is true for this more general definition of a nested function. One may relate nested functions to particular types of decision lists. It is straightforward to see that any nested function can be realized as a 1-decision list of length n in which, for each i between 1 and n, precisely one term of the form (u_i, b) or $(\bar{u}_i, b)$ occurs (for some $b \in \{0, 1\}$) (and vice versa).

8.4 Signatures

In calculating the specification number of the weight-at-least-k function, we used the fact that if x has the property that there is $h \in T_n$ with $h(y) = t(y)$ for all $y \neq x$ and $h(x) \neq t(x)$ then x must belong to any specifying set for t. We shall say that an example with this property is *essential* for t. Clearly any specifying set for t must contain all examples which are essential for t. We now give a simple proof of the fact that the essential examples alone are sufficient to specify a threshold function.

Theorem 8.10 *Let $t \in T_n$ and let $S(t)$ be the set of examples essential for t. Then $S(t)$ specifies t.*

Proof. Suppose not. Then there is h agreeing with t on $S(t)$ but disagreeing on some other examples. Let's say $t \leftarrow [w, \theta]$ and $h \leftarrow [w', \phi]$, where no example lies on the defining hyperplanes. For $0 \leq \lambda \leq 1$, consider the function

$$\lambda t + (1-\lambda)h \leftarrow [\lambda w + (1-\lambda)w', \lambda\theta + (1-\lambda)\phi].$$

The function $\lambda t + (1-\lambda)h$ correctly classifies any example in $S(t)$ since each of h, t correctly classifies such examples. Suppose y is misclassified by h. Then $\langle w, y\rangle > \theta$ and $\langle w', y\rangle < \phi$, or $\langle w, y\rangle < \theta$ and $\langle w', y\rangle > \phi$. The function

$$f(\lambda) = \lambda\langle w, y\rangle + (1-\lambda)\langle w', y\rangle - \lambda\theta - (1-\lambda)\phi$$

is continuous and strictly increasing or strictly decreasing and $f(0)$, $f(1)$ are of opposite signs, so there is a unique $0 < \lambda_y < 1$ such that $f(\lambda_y) = 0$; that is,

$$\lambda_y\langle w, y\rangle + (1-\lambda_y)\langle w', y\rangle = \lambda_y\theta + (1-\lambda_y)\phi.$$

Furthermore, it is easy to see that w, w' could have been chosen in such a way that if $y \neq z$ then $\lambda_y \neq \lambda_z$. Observe that if $\lambda_y > \lambda_z$ then the function $\lambda_y t + (1-\lambda_y)h$ correctly classifies z. Now, since the λ_x are distinct, there is some example v such that $\lambda_v > \lambda_y$ for all $y \neq v$ misclassified by h. Thus (by taking a value of λ very close to λ_v), there is a function $\lambda t + (1-\lambda)h$ which classifies all examples but v correctly. Therefore $v \in S(t)$. But this is a contradiction, since we assumed h to be consistent with t on $S(t)$ (in which case, any such convex combination of t and h would classify v correctly). □

We therefore have the following corollary.

Theorem 8.11 *For $t \in T_n$, there is precisely one set of $\sigma_n(t)$ examples which specifies t. This set is $S(t)$.*

We shall call the set $S(t)$ of all examples essential for t the *signature* of t. Any specifying set contains these examples, and so the signature is the unique minimal specifying set for t.

8.5 Projections

Recall that, for $t \in T_n$, the up-projection and down-projection $t\uparrow, t\downarrow$ are the functions in T_{n-1} given by

$$t\uparrow(x_1, x_2, \ldots, x_{n-1}) = t(x_1, x_2, \ldots, x_{n-1}, 1),$$

$$t\downarrow(x_1, x_2, \ldots, x_{n-1}) = t(x_1, x_2, \ldots, x_{n-1}, 0).$$

Thus, $t\uparrow$ is the restriction of t to the hyperface $x_n = 1$ of the Boolean hypercube and $t\downarrow$ is its restriction to the hyperface $x_n = 0$. Note that if $t \leftarrow [w, \theta]$, where $w = (w', d)$, $w' \in \mathbb{R}^{n-1}$ and $d \in \mathbb{R}$, then $t\uparrow \leftarrow [w', \theta - d]$ and $t\downarrow \leftarrow [w', \theta]$.

Theorem 8.12 (Projection result) *For $t \in T_n$,*

$$\sigma_n(t) \leq \sigma_{n-1}(t\uparrow) + \sigma_{n-1}(t\downarrow),$$

and equality holds when $t\uparrow = t\downarrow$.

Proof. It is easy to see that the inequality holds. Let $S(t\downarrow)$ be the signature of $t\downarrow$ and $S(t\uparrow)$ the signature of $t\uparrow$. For each $s = (a_1, a_2, \ldots, a_{n-1})$ in $S(t\downarrow)$, form the example $s0 = (a_1, a_2, \ldots, a_{n-1}, 0)$ and for each $s = (a_1, a_2, \ldots, a_{n-1}) \in S(t\uparrow)$, form the example $s1 = (a_1, a_2, \ldots, a_{n-1}, 1)$. Then it is clear that these examples specify t, so that

$$\sigma_n(t) \leq |S(t\downarrow)| + |S(t\uparrow)| = \sigma_{n-1}(t\downarrow) + \sigma_{n-1}(t\uparrow).$$

In order to prove equality when $t\uparrow = t\downarrow$, we first prove that if z is any point in the signature $S(t)$ of t and $t(z) = 1$ (respectively, $t(z) = 0$) then there are w, θ such that $t \leftarrow [w, \theta]$ and for any other positive (respectively, negative) example x of t, $\langle w, x\rangle > \langle w, z\rangle$ (respectively, $\langle w, x\rangle < \langle w, z\rangle$). Suppose $z \in S(t)$ is a positive example of t. Then there is $h \leftarrow [w', \phi]$ such that h agrees with t on all examples except z, and such that no example x satisfies $\langle w', x\rangle = \phi$. We may assume that there is $c > 0$ such that $\langle w, z\rangle \geq \theta + c$, $\langle w', z\rangle \leq \phi - c$ and such that for $x \in F(t)$, $\langle w, x\rangle \leq \theta - c$ and $\langle w', x\rangle \leq \phi - c$, and for $z \neq x \in T(t)$, $\langle w, x\rangle \geq \theta + c$ and $\langle w', x\rangle \geq \phi + c$. Let λ be such that

$$\lambda\langle w, z\rangle + (1 - \lambda)\langle w', z\rangle = \lambda\theta + (1 - \lambda)\phi.$$

Let $w'' = \lambda w + (1 - \lambda)w'$ and $\psi = \lambda\theta + (1 - \lambda)\phi$. Then, as can easily be checked, $[w'', \psi]$ represents t. Further, for $z \neq x \in T(t)$,

$$\langle w'', x\rangle = \langle \lambda w + (1 - \lambda)w', x\rangle \geq \lambda\theta + (1 - \lambda)\phi + c > \lambda\theta + (1 - \lambda)\phi = \langle w'', z\rangle.$$

The argument when z is a negative example is similar.

Now we show that the set S consisting of all examples $z1$ and $z0$ for $z \in S(t\downarrow)$ is the signature of t. As mentioned above, the points of S specify t. We prove that all are essential for t, from which it follows that $S = S(t)$. Without loss of generality, suppose $s = z1$, where z is a positive example of $t\downarrow$. We wish to find $h \leftarrow [w', \phi]$ where $w' \in \mathbb{R}^n, \phi \in \mathbb{R}$ such that $h(z0) = 1$, $h(z1) = 0$, $h(x1) = h(x0) = 1$ for all $z \neq x \in T(t\downarrow)$, and $h(x1) = h(x0) = 0$ for all $x \in F(t\downarrow)$. By the above, we may assume that $t\downarrow \leftarrow [w, \theta]$, where for some $c > 0$,

$$z \neq x \in T(t\downarrow) \Longrightarrow \langle w, x\rangle \geq \langle w, z\rangle + c.$$

Let $w' = (w, -c)$ and let $\phi = \langle w, z\rangle$. Then $\langle w', z1\rangle = \langle w, z\rangle - c < \phi$ and $\langle w', z0\rangle = \langle w, z\rangle$, so that $h(z1) = 0$ and $h(z0) = 1$. For $x \in T(t\downarrow), x \neq z$, we have

$$\langle w', x0\rangle \geq \langle w', x1\rangle = \langle w, x\rangle - c \geq \langle w, z\rangle = \phi,$$

whence $h(x0) = h(x1) = 1$. Also, for $x \in F(t\downarrow)$,

$$\langle w', x1\rangle \leq \langle w', x0\rangle = \langle w, x\rangle < \theta,$$

which is less than ϕ since $\langle w, z\rangle > \theta$ and so $h(x1) = h(x0) = 0$. The result follows. □

We now turn our attention to functions in T_n which depend on a particular number, k, of the coordinates. Such a function has $n-k$ "irrelevant attributes." Suppose that t depends on coordinates 1 to k only and denote by t_k the function in T_k defined by $t_k(x_1, x_2, \ldots, x_k) = t(x_1, x_2, \ldots, x_k, 0, 0, \ldots, 0)$, which is obtained by projecting t down $n-k$ times. Then we have the following result, an immediate consequence of the projection result.

Theorem 8.13 *If $t \in T_n$ and t depends only on coordinates $1, 2, \ldots, k$, then the specification number of t equals $2^{n-k}\sigma_k(t_k)$.*

As an example of this, consider the function $g \in T_n$ defined by $g(x) = 1$ if and only if, of the first k entries of x, at least r are equal to 1. Then g is the *r-out-of-k* function and is easily seen to be a threshold function. Clearly, g depends only on the first k coordinates and $g_k \in T_k$ is the weight-at-least-r function , so that $\sigma_n(g) = 2^{n-k}\sigma_k(g_k) = 2^{n-k}\binom{k+1}{r}$.

We have the following tight bound, from Theorem 8.5, Theorem 8.8, and the projection result.

Theorem 8.14 *Suppose $t \in T_n$ depends on exactly k coordinates. Then*

$$2^{n-k}(k+1) \leq \sigma_n(t) \leq 2^{n-k}\binom{k+1}{\lfloor \frac{k+1}{2} \rfloor},$$

and equality is possible in both cases.

From Theorem 8.7, it is easy to obtain a characterization of those t meeting the upper bound above. Furthermore, our results on nested functions enable us to generate a class of functions meeting the lower bound. By the relationship (mentioned earlier) between 1-decision lists and nested functions, if f is a 1-decision list of length k in which each term is of the form (u_i, b) or $(\bar{u}_i, b)$ (for some $b \in \{0, 1\}$) and in which no literal occurs (negated or nonnegated) in more than one term, then f has specification number $2^{n-k}(k+1)$ (in the class of threshold functions).

8.6 The expected specification number

We have now seen the extreme values that specification numbers in T_n can take. A natural problem is to determine the *average* or *expected* specification number, by which we mean the quantity

$$\overline{\sigma_n(t)} = \frac{1}{|T_n|}\sum_{t \in T_n} \sigma_n(t).$$

8.6.1 A general bound

First, we present a general upper bound for the average specification number of any class of Boolean functions.

Theorem 8.15 *Let H be any set of Boolean functions and let $\overline{\sigma_H(f)}$ be the average specification number, meaning $(1/|H|)\sum_{f \in H}\sigma_H(f)$. Then*

$$\overline{\sigma_H(f)} \leq 2\sqrt{|H|}.$$

Proof. We first obtain a useful little result. This is simply that if a class H of functions has k different members, then there is a set $T \subseteq \{0,1\}^n$ of cardinality no more than $k-1$ which is, simultaneously, a specifying set for every function in the class. That is, for every two functions $f, g \in H$, there is some $x \in T$ such that $f(x) \neq g(x)$. This can easily be proved by induction on the cardinality of H. It is trivially true when the class contains only two functions. Suppose, as an inductive hypothesis, that it is true for all classes of cardinality no more than k, and suppose that H has cardinality $k+1$. Let h_0 be any particular function in H, and let $H' = H \setminus \{h_0\}$. By the inductive hypothesis, there is $T \subseteq \{0,1\}^n$ such that $|T| \leq k-1$ and T simultaneously specifies all functions in H'. (Thus, restricting H' to domain T still gives k distinct functions.) Either T distinguishes h_0 from the functions in H' (in which case the result is immediate) or there is precisely one function $g \in H'$ such that g equals h_0 on T. In the second case, we need only add to T some example y for which $h_0(y) \neq g(y)$ to obtain a set specifying all functions in H simultaneously. The cardinality of this set is no more than k, and the result follows.

We now move on to the proof of the theorem itself. Denote the cardinality of H by r, and suppose we enumerate the functions in H in order of nonincreasing specification number, so that $H = \{f_1, f_2, \ldots, f_r\}$, where

$$\sigma_H(f_1) \geq \sigma_H(f_2) \geq \cdots \geq \sigma_H(f_r).$$

Let $H_k = \{f_1, f_2, \ldots, f_k\} \subseteq H$, where k is some integer between 1 and r. By the result just obtained above, there is a set T of at most $k-1$ examples which simultaneously specifies all functions in H_k. Fix attention on a particular $f \in H_k$. For each $g \in H \setminus H_k$ such that f, g are equal on T, we add to T precisely one example distinguishing g from f. The resulting set is a specifying set for f (in H). Now, no function in $H \setminus H_k$ can agree with two different functions of H_k, since the functions in H_k do not agree on T. It follows that the total number of additional examples added to T to obtain specifying sets for the functions in H_k (with respect to H) is no more than $r-k$. Therefore,

$$\sum_{i=1}^{k} \sigma_H(f_i) \leq (r-k) + k|T| \leq (r-k) + k(k-1) = k^2 - 2k + r.$$

From this, and the fact that, for $1 \leq i \leq k-1$, $\sigma_H(f_i) \geq \sigma_H(f_k)$, we also have

$$\sigma_H(f_k) \leq \frac{k^2 - 2k + r}{k} = k - 2 + \frac{r}{k}.$$

Thus, for all $i > k$, $\sigma_H(f_i) \leq k - 2 + (r/k)$. Hence,

$$\sum_{f \in H} \sigma_H(f) = \sum_{i=1}^{r} \sigma_H(f_i) \leq (k^2 - 2k + r) + (r-k)\left(k - 2 + \frac{r}{k}\right) = rk - 2r + \frac{r^2}{k}.$$

Choosing $k = \lceil \sqrt{r} \rceil$, this is at most

$$r\lceil \sqrt{r} \rceil - 2r + \frac{r^2}{\lceil \sqrt{r} \rceil} \leq r(\sqrt{r}+1) - 2r + r^{3/2} < 2r^{3/2}.$$

This completes the proof, since we then have $\overline{\sigma_H(f)} < 2r^{3/2}/r = 2\sqrt{r}$. □

8.6.2 Expected specification number of T_n

Theorem 8.15 provides an upper bound of $2\sqrt{|T_n|}$ on $\overline{\sigma_n(t)}$, the average specification number of T_n. Given the bounds we have obtained on $|T_n|$, this is of the order of $2^{n^2/2}$. However, the following much better bound can be obtained. Its proof is rather technical and is omitted.

Theorem 8.16 *Let T_n be the set of Boolean threshold functions on $\{0,1\}^n$. Then the average specification number of T_n may be bounded as follows:*

$$\overline{\sigma_n(t)} \leq \log_2 |T_n|.$$

We therefore have the following result, by Theorem 4.3.

Theorem 8.17 *For the set T_n of Boolean threshold functions on $\{0,1\}^n$, the average specification number $\overline{\sigma_n(t)}$ satisfies*

$$\overline{\sigma_n(t)} = \frac{1}{|T_n|} \sum_{t \in T_n} \sigma_n(t) \leq n^2$$

for all n.

Given that specification numbers can be exponential in n, this bound is surprisingly close to the absolute lower bound of $n + 1$.

8.7 Additional remarks and bibliographical notes

We have mentioned that one possible motivation for the types of problems studied here is concerned with the number of examples which will guarantee exact learning by any "consistent" learning algorithm (as defined in the next chapter). However, similar problems have been investigated in the context of fault detection in circuits. (In particular, the central problem of this chapter might be thought of as concerning the "testing," by examples, that a perceptron computes the function it ought to.) Hegedűs [45] cites relevant work, largely published in Russian, from around the 1960s. More recently, there have been a number of papers in computational learning theory concerned with "learning from a helpful teacher"; see [39, 10, 40, 50], for instance. Quantifying the number of examples such a "helpful" teacher should provide is equivalent to bounding specification numbers. A related idea involves learners capable of asking membership queries (a model developed by Angluin [3]); see [45] for a discussion.

The results on the specification numbers of Boolean threshold functions are from [10], and the proof of the average specification number bound for the class of threshold functions may be found in that paper. The general bound on the average specification number is due to Kushilevitz et al. [59]. The preliminary result used in the proof of this bound is due to Bondy [23]; see [22].

Given any set X of points of $\mathbb{R}^n$ in general position, we may define a set of $\{0,1\}$-valued functions on X by the same method we used to define the class of Boolean threshold functions; that is, for each hyperplane in $\mathbb{R}^n$, assign 1 to the points of X on one side of this hyperplane, and 0 to the others. Cover [29] has investigated such sets of functions and has proved that, asymptotically, the average specification number (in our terminology) of the class of functions behaves asymptotically like $2n$. But Cover's

analysis cannot be carried over to T_n, for here $X = \{0,1\}^n$, which is far from being in general position. Cover's paper also uses the notion of what we have termed essential examples. He obtains the analogue to Theorem 8.11, using results of Mays [70] on boundary matrices. Hu [48] provides a proof based on Mays's work.

Shinohara and Miyano [94] consider the subset of T_n in which the nonthreshold weights are permitted to be only 0 or 1. They show a $2n$ bound on the specification numbers within this class, and they give an algorithm for producing specifying sets.

Chapter 9

Neural Network Learning

9.1 Introduction

One of the main reasons why neural networks have proved so attractive is that they are, in a sense, capable of "learning." In a formal mathematical or engineering approach to neural networks, "learning" simply means the changing of weights of the network in response to some input data. When "successful" or "convergent" learning is possible, there is no need to program the network explicitly to perform a particular task; in other words, we need not know in advance how to set the weights. The neural network adjusts its weights according to a *learning algorithm*, in response to some classified training examples, with (roughly speaking) the state of the network converging to the "correct" one. In this sense, the neural network "learns from experience."

9.2 Supervised learning

9.2.1 A learning framework

In our model of supervised learning, we assume that there is some *target function* (or *target concept*) t, which is the function to be learned. The target function is to be thought of as the "correct" function we want the network to compute. It simplifies matters greatly if we assume that there *is* a correct function t and that this function can be computed by the network with some set of weight assignments. We shall assume that the set of all possible target functions, C, is a subset of the set of all functions computable by the neural network. A *labeled example* for the target function t is a pair $(x, t(x))$, where x is an input pattern to the network; for instance, if the neural network has n input units accepting real inputs, then $x \in \mathbb{R}^n$. The network is given a *training sample*, a sequence of such labeled examples: this constitutes its "experience." In response to this, its weights are altered by applying a learning algorithm. More formally, suppose that the set of all possible examples is $X \subseteq \mathbb{R}^n$ (where X will usually be $\mathbb{R}^n$ or $\{0,1\}^n$), where n is the number of inputs to the network, and that the target function t can be computed by the neural network in some state. A *training sample for* t of length m is an element $\mathbf{s}$ of $(X \times \{0,1\})^m$, of the form

$$\mathbf{s} = ((x_1, t(x_1)), (x_2, t(x_2)), \ldots, (x_m, t(x_m))) .$$

A *learning algorithm* accepts as input the training sample $\mathbf{s}$ and alters the state of the network in some way in response to the information provided by the sample. The function $L(\mathbf{s})$ that is computed by the network after "learning" is called the *output function* or *hypothesis*, and we want it to be a good approximation to the target function, or to be "closer" to the target function than the function computed before learning. Of course, we shall have to formalize what is meant by "close," and this is the subject of a later chapter. Although we use the term "learning algorithm," we shall sometimes simply regard a learning algorithm as a function from training samples to states of the network. At times, though, we will be interested in the algorithmic aspects of the learning algorithm, such as its running time. Formally, the output $L(\mathbf{s})$ of the algorithm is a state of the network, but, as indicated above, we shall find it useful to identify the state with the function it represents.

9.2.2 Consistent algorithms

There is one very important property that a learning algorithm may possess. We say that learning algorithm L is *consistent* if, the output function of the algorithm always agrees with the target function on the examples given in the training sample. Explicitly, L is consistent if, for all $m \in \mathbb{N}$ and for all $t \in C$,

$$\mathbf{s} = ((x_1, t(x_1)), (x_2, t(x_2)), \ldots, (x_m, t(x_m)))$$

is any training sample for t, and if $h = L(\mathbf{s})$ is the output function of L on $\mathbf{s}$, then $h(x_i) = t(x_i)$ for $i = 1, 2, \ldots, m$. Intuitively, this seems like a reasonable requirement: all it says, roughly speaking, is that the learning algorithm takes full account of its experience. We shall see that not all neural network learning algorithms have this property but that they can sometimes be modified to obtain, or be used as the basis for, consistent learning algorithms.

We commented in the previous chapter on a connection between specifying sets and consistent algorithms. Recall that a set S of examples specifies a function t in a set H of Boolean functions if for every other function $f \in H$ there is some $x \in S$ such that $f(x) \neq t(x)$. Suppose that $S = \{x_1, x_2, \ldots, x_m\}$, and let $\mathbf{s}$ be the corresponding training sample for t,

$$\mathbf{s} = ((x_1, t(x_1)), (x_2, t(x_2)), \ldots, (x_m, t(x_m))).$$

Then it is clear that, for any consistent learning algorithm L which returns a function in H, $L(\mathbf{s}) = t$. Thus, a specifying set yields a training sample which will ensure *exact* learning of the target function by any consistent learning algorithm, provided that the algorithm has output function in H.

9.3 Consistent algorithms for the perceptron

As an explicit example of the general framework for supervised learning, we consider the case in which the neural network is simply a (real) perceptron, consisting of a single linear threshold unit and taking real-valued inputs.

9.3.1 Linear programming method

Suppose we are given the training sample

$$\mathbf{s} = ((x_1, b_1), (x_2, b_2), \ldots, (x_m, b_m)),$$

where each x_i is in $\mathbb{R}^n$ and, for some function t computable by the perceptron, $b_i = t(x_i)$ $(i = 1, 2, \ldots, m)$. To find a perceptron function consistent with $\mathbf{s}$, we consider the problem of producing $w \in \mathbb{R}^n$ which satisfies the following m inequalities:

$$\langle w, x_i \rangle \geq 1 \text{ if } b_i = 1, \quad \langle w, x_j \rangle < 1 \text{ if } b_j = 0.$$

To see that it suffices to consider this problem, we can argue as follows. Suppose that there is a consistent function, represented by weight-vector w and threshold θ. If $\theta > 0$, we can simply divide all entries of the weight-vector by θ to obtain a representation with threshold equal to 1. If, however, θ must be negative in order to obtain a representation of a consistent function, then this is not true, since on division by θ the inequalities would change direction. However, in that case, the following procedure will fail, we would realize this, and we would then simply rerun it, reversing the inequalities on the constraints or, equivalently, swapping the negative and positive examples in the sample. By the finiteness of the sample, if there is a solution to this system of inequalities, then there will be one to the system

$$\langle w, x_i \rangle \geq 1 + c \text{ if } b_i = 1, \quad \langle w, x_j \rangle \leq 1 - c \text{ if } b_j = 0$$

for some $c > 0$. Therefore, any feasible solution to the linear programming problem

$$\text{maximize } c \text{ subject to}$$

$$\langle w, x_i \rangle - c - 1 \geq 0 \text{ if } b_i = 1, \quad \langle w, x_j \rangle + c - 1 \leq 0 \text{ if } b_j = 0$$

will correspond to a consistent function computed by the perceptron. The feasible region is nonempty, by the assumption that the function t generating the labels in the sample is a perceptron function. (However, as noted above, we may have to swap negative and positive examples first to ensure feasibility. If the system is infeasible, then it becomes feasible on performing this swap.) By solving the resulting linear programming problem, we therefore obtain a function $L(\mathbf{s})$ consistent with $\mathbf{s}$. Thus, linear programming can be used as the basis for a consistent learning algorithm for the perceptron.

9.3.2 The incremental perceptron learning algorithm

We now describe an "incremental" learning algorithm for the perceptron, often known simply as the perceptron learning algorithm. As we shall see, it differs substantially from the algorithm based on linear programming.

The *incremental perceptron learning algorithm* L acts on the training sample in the following manner. The algorithm L maintains at each stage a "current" weight-vector and threshold, which are updated on the basis of a labeled example $(x, t(x))$. The initial weight-vector and threshold are chosen to be the all-0 vector and 0. Suppose that the current weight-vector is w, the current threshold is θ, and a labeled example $(x, t(x))$ is presented. Denote by h the function computed by the network in its current state. Then the algorithm forms the new weight-vector w' and threshold θ' as follows:

$$\begin{aligned} w' &= w + (t(x) - h(x))\, x, \\ \theta' &= \theta - (t(x) - h(x)). \end{aligned}$$

Let's unravel this slightly. What it means is that, for example, if $h(x) = 0$ and $t(x) = 1$, then the weight-vector is moved slightly in the direction of the example x, and the

threshold is decreased slightly. What is the effect of this? The change in w increases the inner product of the weight-vector with the example x, since

$$\langle w', x\rangle = \langle w + x, x\rangle = \langle w, x\rangle + \|x\|_2^2,$$

where $\|x\|_2$ denotes the Euclidean 2-norm, or length, of x. (We shall denote this simply by $\|x\|$ in what follows: this is not to be confused with the weight of x.) At the same time, the threshold is decreased. Thus, the effect is to decrease the gap between the inner product and the threshold it must exceed in order to give output 1. Note, though, that $\langle w', x\rangle$ may not exceed θ': we simply move in that direction. Our learning algorithm, which we shall simply call the incremental algorithm, will repeatedly cycle through the labeled examples in the sample. The fact that it will eventually end up in a state in which no further changes are needed—that is, which yields a consistent hypothesis—is implied by the following version of the *perceptron convergence theorem*. In general, the theorem asserts that no matter how many examples are presented, the incremental algorithm makes only a finite number of changes, or updates. In particular, if the examples are those in $\mathbf{s}$, presented repeatedly, then eventually we find a consistent hypothesis.

Theorem 9.1 *Suppose that a training sample* $\mathbf{s} = ((x_1, b_1), \ldots, (x_m, b_m))$ *is given, and that there is a weight vector* w^* *and threshold* θ^* *such that* $\|w^*\|^2 + \theta^{*2} = 1$ *and* $b_i = \operatorname{sgn}(\langle w^*, x_i\rangle - \theta^*)$ *for* $i = 1, \ldots, m$. *Define*

$$\gamma = \min\{|\langle w^*, x_i\rangle - \theta^*| : i = 1, \ldots, m\},$$

and suppose $\gamma > 0$. *Then the incremental perceptron algorithm* L *gives a weight-vector and threshold yielding a function consistent with* $\mathbf{s}$ *after no more than* $(R^2 + 1)/\gamma^2$ *updates, where* $R = \max_i \|x_i\|$ *is the maximum Euclidean 2-norm of the examples in* $\mathbf{s}$.

Proof. Let (w_i, θ_i) be the pair consisting of the current weight-vector and threshold immediately after the ith update. Suppose that the ith update occurs on example (x_j, b_j). This example must have been misclassified by the perceptron when in state (w_{i-1}, θ_{i-1}), so $\operatorname{sgn}(\langle w_{i-1}, x_j\rangle - \theta_{i-1}) = 1 - b_j$, and hence the new weight-vector $w_i = w'_{i-1}$ is given by

$$\begin{aligned} w_i &= w_{i-1} + (b_j - \operatorname{sgn}(\langle w_{i-1}, x_j\rangle - \theta_{i-1}))\, x_j \\ &= w_{i-1} + (2b_j - 1)x_j. \end{aligned}$$

Similarly, $\theta_i = \theta_{i-1} - (2b_j - 1)$.

We consider the way the quantity $\|w_i\|^2 + \theta_i^2$, changes as the algorithm proceeds. First, we note that it increases with each update performed. We have

$$\begin{aligned} \langle w^*, w_i\rangle + \theta^*\theta_i &= \langle w^*, w_{i-1}\rangle + \theta^*\theta_{i-1} + (2b_j - 1)(\langle w^*, x_j\rangle - \theta^*) \\ &\geq \langle w^*, w_{i-1}\rangle + \theta^*\theta_{i-1} + \gamma, \end{aligned}$$

and so $\langle w^*, w_i\rangle + \theta^*\theta_i \geq i\gamma$. We now use the Cauchy–Schwarz inequality, which states that $|\langle a, b\rangle| \leq \|a\|\|b\|$. Applying this to the vectors $a = (w_i, \theta_i)$ and $b = (w^*, \theta^*)$, and noting that $\|b\| = 1$, we obtain

$$i\gamma \leq \langle w^*, w_i\rangle + \theta^*\theta_i = \langle (w_i, \theta_i), (w^*, \theta^*)\rangle \leq \|(w_i, \theta)\|,$$

and hence $\|(w_i, \theta)\|^2 = \|w_i\|^2 + \theta_i^2 \geq (i\gamma)^2$.

On the other hand, we have

$$\begin{aligned}
&\|w_i\|^2 + \theta_i^2 \\
&\quad = \langle w_{i-1} + (2b_j - 1)x_j, w_{i-1} + (2b_j - 1)x_j \rangle + (\theta_{i-1} - (2y_j - 1)x_j)^2 \\
&\quad = \|w_{i-1}\|^2 + \theta_{i-1}^2 + \left(\|x_j\|^2 + 1\right) + 2(2b_j - 1)\left(\langle w_{i-1}, x_j \rangle - \theta_{i-1}\right) \\
&\quad \leq \|w_{i-1}\|^2 + \theta_{i-1}^2 + \left(\|x_j\|^2 + 1\right) \\
&\quad \leq \|w_{i-1}\|^2 + \theta_{i-1}^2 + \left(R^2 + 1\right)
\end{aligned}$$

and so $\|w_i\|^2 + \theta_i^2 \leq i(R^2 + 1)$. Combining these inequalities shows that

$$(i\gamma)^2 \leq \|w_i\|^2 + \theta_i^2 \leq i(R^2 + 1),$$

and so $i \leq (R^2 + 1)/\gamma^2$ as required. □

9.4 Efficiency of Boolean perceptron learning

We consider the efficiency of the two procedures described above, when they are applied in the case where each input to the linear threshold unit is 0 or 1. (This restriction to binary inputs makes the analysis easier, but similar considerations apply without it.)

The input to a learning algorithm is a sequence $\mathbf{s}$ of labeled examples. If the examples x_i are in $\{0,1\}^n$, and the length of the sample is m, then $\mathbf{s}$ has size $m(n+1)$, which is of order mn. For a consistent learning algorithm to be efficient, we therefore should like the algorithm to find a consistent output function in time polynomial in mn.

9.4.1 The linear programming approach

Recall that the linear programming approach proceeds by maximizing c subject to the constraints

$$\langle w, x_i \rangle \geq 1 + c \text{ if } b_i = 1, \quad \langle w, x_j \rangle \leq 1 - c \text{ if } b_j = 0.$$

The number of variables is $n+1$ and the number of constraints is m, so any polynomial-time linear programming algorithm may be used to find a consistent hypothesis in time polynomial in mn. Thus, this consistent algorithm is algorithmically efficient.

9.4.2 The incremental approach

Although the incremental consistent algorithm has some attractive features, it is not in fact generally efficient. If the "margin" γ in the statement of Theorem 9.1 can be taken to be sufficiently large (for instance, if $1/\gamma$ is no larger than some polynomial in n), then the algorithm would be efficient, since the number of updates required would be no more than of order n^2/γ^2, by the bound in the proof of Theorem 9.1 (noting that the examples in question are all from $\{0,1\}^n$ in this case). However, for samples of size polynomial in n, the margin γ can be exponentially small in n. We shall show that the algorithm is not efficient by proving that for a sample of size polynomial in n, the number of complete cycles required can be exponential in n.

To establish this, we consider again, for even n, the function f_n with Boolean formula

$$u_n \wedge (u_{n-1} \vee (u_{n-2} \wedge (u_{n-3} \vee (\ldots (u_2 \wedge u_1)) \ldots)).$$

We have already encountered two results concerning this function. First, Theorem 5.2 shows that if w is any integral weight-vector in a threshold representation of f_n, then $w_i \geq F_i$ for $1 \leq i \leq n$, where F_i is the ith Fibonacci number. We have also seen that f_n is nested and that its specification number is, by Theorem 8.9, $n+1$. Thus, there is some set of examples $S \subseteq \{0,1\}^n$ such that $|S| = n+1$ and S specifies f_n.

Combining the two previous results, we have the following.

Theorem 9.2 *The incremental consistent learning algorithm for the perceptron is not a polynomial-time algorithm.*

Proof. We show that for each even n, there is a sample of $n+1$ labeled examples such that the time taken by L to produce a consistent function is exponential in n. Suppose that $t = f_n$ is the target function and let $S(f_n)$ be the signature of f_n, of cardinality $n+1$. Let $\mathbf{s}$ be the corresponding training sample, of size $(n+1)^2$. Recall that, initially, the weight-vector and threshold are the identically-0 vector, and 0. Let u be the number of updates made before a function $L(\mathbf{s})$, consistent with the sample, is produced. Because $S(f_n)$ specifies f_n in the class of all Boolean threshold functions, the only consistent function is f_n itself. Suppose w and θ are the weight-vector and threshold produced by the algorithm, and representing $L(\mathbf{s}) = f_n$. Because all the x_i in the sample are from $\{0,1\}^n$, and by the way the algorithm works, w and θ will be integral. It follows that $w_n \geq F_n$. But each update increases w_n by at most 1, and the initial value of w_n is 0, so the number of updates is at least F_n. Now, F_n is at least $(\sqrt{5}/6)\xi^n$, where $\xi = (1+\sqrt{5})/2$. The number of updates is therefore exponential in n, whereas the input sample has size $O(n^2)$. Thus the algorithm is not a polynomial-time algorithm. □

9.5 Additional remarks and bibliographical notes

The notion of "Hebbian learning" (from Hebb [44]) can be seen as motivation for the incremental perceptron learning algorithm, which is due to Rosenblatt [89] (see also [72]). There are many proofs of the perceptron convergence theorem. The proof given here is due to Novikoff [79] (see also [34]), and our presentation is as in [7].

Variants of the perceptron learning algorithm have been studied. (See [24], for instance.) In the versions often discussed, there is a "learning constant" ν, and the corresponding update rule is

$$\begin{aligned} w' &= w + \nu\,(t(x) - h(x))\,x, \\ \theta' &= \theta - \nu\,(t(x) - h(x))\,. \end{aligned}$$

The results presented here still hold for such variants.

Littlestone [60] has developed an algorithm similar to the perceptron learning algorithm, particularly useful in the Boolean case. Here, the updates are multiplicative rather than additive; that is, instead of adding or subtracting a constant ν to some weights (including the threshold), the algorithm multiplies or divides by ν.

Theorem 9.2 is a slightly modified version of a result from [11].

Chapter 10

Probabilistic Learning

10.1 Introduction

In this chapter, we briefly discuss a probabilistic model of neural network learning known as the "PAC" model. We present results which show that, within this model, the amount of training data which should be used is linked to a combinatorial parameter of the network known as the Vapnik–Chervonenkis dimension, or *VC-dimension*. We give some basic combinatorial results on the VC-dimension. In the next chapter, we bound the VC-dimension of various simple types of neural network.

10.2 The PAC learning model

10.2.1 The learning framework

The basic "probably approximately correct" (PAC) model of learning is applicable to neural networks with one output unit which outputs either the value 0 or 1; thus, it applies to *classification* problems. In the PAC model, it is assumed that the neural network receives a sequence of *examples* x, each labeled with the value $t(x)$ of the particular *target function* t which is being "learned." A fundamental assumption of this model is that these examples are presented independently and at random according to some fixed (but unknown) probability distribution on the set of all examples.

Recall that a *training sample for* t of length m is an element $\mathbf{s}$ of $(X \times \{0,1\})^m$, of the form

$$\mathbf{s} = ((x_1, t(x_1)), (x_2, t(x_2)), \ldots, (x_m, t(x_m))) .$$

We shall denote by $S(m,t)$ the set of all training samples of length m for t. A learning algorithm accepts the training sample $\mathbf{s}$ and produces a state of the network, and hence an output function $L(\mathbf{s})$, computable by the network. It is to be hoped that this function is a good approximation to the target function.

10.2.2 Probability and approximation

If $L(\mathbf{s})$ is the function computed by the network after training sample $\mathbf{s} \in S(m,t)$ has been presented and learning algorithm L has been applied, one way in which to assess the success of the learning process is to measure how close $L(\mathbf{s})$ is to t. Since

there is assumed to be some probability distribution, P, on the set of all examples, and since t takes only the values 0 or 1, we may define the *error*, $\mathrm{er}(h,t)$, of a function h (with respect to t) to be the P-probability that a randomly chosen example is classified incorrectly by h. In other words,

$$\mathrm{er}(h,t) = P(\{x \in X : h(x) \neq t(x)\}).$$

The aim is to ensure that the error of $L(\mathbf{s})$ is "usually small." Since each of the m examples in the training sample is drawn randomly and independently according to P, the sample vector $\mathbf{x}$ is drawn randomly from X^m according to the product probability distribution P^m. Thus, more formally, we want it to be true that with high P^m-probability the sample $\mathbf{s}$ arising from $\mathbf{x}$ is such that the output function $L(\mathbf{s})$ has small error with respect to t. This leads us to the following formal definition of PAC learning.

Definition 10.1 (PAC learning) *The learning algorithm L is a* PAC-learning algorithm *for the network if for any given $\delta, \epsilon > 0$ there is a sample length $m_0(\delta,\epsilon)$ such that for all target functions t computable by the network and for all probability distributions P on the set of examples, we have*

$$m \geq m_0(\delta,\epsilon) \Rightarrow P^m(\{\mathbf{s} \in S(m,t) : \mathrm{er}(L(\mathbf{s}),t) > \epsilon\}) < \delta.$$

(We should note that the product probability distribution P^m is really defined not on subsets of $S(m,t)$ but on sets of vectors $\mathbf{x} \in X^m$. However, this abuse of notation is convenient and is unambiguous: for a fixed t, there is a clear one-to-one correspondence between vectors $\mathbf{x} \in X^m$ and training samples $\mathbf{s} \in S(m,t)$.)

Informally, a learning algorithm is PAC if, provided a random sample is long enough (where "long enough" is independent of P and t), then it is "probably" the case that after training on that sample, the function computed by the network is "approximately" correct. We often refer to ϵ as the *accuracy parameter*, because it quantifies the desired bound on the error of the output function, and δ as the *confidence parameter*, because it quantifies the probability with which we want the output function to be accurate.

Note that the probability distribution P occurs twice in the definition: first in the requirement that the P^m-probability of a sample be small and secondly through the fact that the error of $L(\mathbf{s})$ is measured with reference to P. The crucial feature of the definition is that we require that the sample length $m_0(\delta,\epsilon)$ be independent of P and of t. It is not immediately clear that this is possible, but the following informal arguments explain why it can be done. If a particular example has not been seen in a large sample $\mathbf{s}$, the chances are that this example has low probability (with respect to P) and therefore misclassification of that example contributes little to the error of the function $L(\mathbf{s})$. In other words, since the penalty paid for misclassification of a particular example is its probability, we can imagine that the two occurrences of the probability distribution in the definition can therefore "balance" or "cancel" each other.

10.2.3 The finite case

We shall use the notation H for the set of all functions computable by the network, and will call this the *hypothesis space*. (It represents all "hypotheses" as to what the true, hidden target function might be.) Our first general result on PAC learning shows that if a network computes only a finite number of functions (for example, when it receives only binary inputs, or when the weights are restricted to a finite set of allowed values), then any consistent learning algorithm for the network will be a PAC learning algorithm for the network.

Theorem 10.2 *Suppose that the hypothesis space is finite. Then any consistent learning algorithm is probably approximately correct. Moreover, a sufficient sample length is*

$$m_0(\delta, \epsilon) = \frac{1}{\epsilon} \ln \left(\frac{|H|}{\delta} \right).$$

Proof. Suppose $|H| = S$ and that $t \in H$ is any target function. If h is computable by the network and has error $\epsilon_h \geq \epsilon$ with respect to t and P (the underlying probability distribution), then the probability (with respect to the product distribution P^m) that h agrees with t on a random sample is clearly at most $(1 - \epsilon_h)^m$. This is at most $\exp(-\epsilon_h m)$, using a standard approximation. Thus, since there are certainly at most S such functions h, the probability that *some* function computable by the network has error at least ϵ *and* is consistent with a randomly chosen sample $\mathbf{s}$ is at most $S \exp(-\epsilon m)$. For any fixed positive δ, this probability is less than δ provided

$$m \geq m_0(\delta, \epsilon) = \frac{1}{\epsilon} \ln \left(\frac{S}{\delta} \right),$$

a bound independent of both the distribution and the target function. The result then follows. □

Note that the proof just presented fails completely if the hypothesis space under consideration is infinite, as is the case, for instance, even for the perceptron with real inputs. It is not immediately clear that PAC learning is possible in such circumstances. In the next section, however, we begin to develop a theory which will enable us to show that, in many such cases, it is possible.

10.3 The growth function and VC-dimension

10.3.1 The growth function of a hypothesis space

Suppose that H is a set of functions from a set X to $\{0, 1\}$. (When H is the hypothesis space of an n-input neural network, X will usually be $\mathbb{R}^n$ or $\{0, 1\}^n$.) Let $\mathbf{x} = (x_1, x_2, \ldots, x_m)$ be a sample (unlabeled) of length m of examples from X. We define $\Pi_H(\mathbf{x})$, the *number of classifications of* $\mathbf{x}$ *by* H, to be the number of distinct vectors of the form

$$\mathbf{x}^*(f) = (f(x_1), f(x_2), \ldots, f(x_m)),$$

as f runs through all functions of H. Although H may be infinite, $H_{\mathbf{x}}$, the set of functions obtained by restricting the functions of H to domain $E = \{x_1, x_2, \ldots, x_m\}$, is finite and is of cardinality $\Pi_H(\mathbf{x})$. Note that for any sample $\mathbf{x}$ of length m, $\Pi_H(\mathbf{x}) \leq 2^m$. An important quantity, and one which turns out to be crucial in PAC learning theory, is the maximum possible number of classifications by H of a sample of a given length. We define the *growth function* Π_H by

$$\Pi_H(m) = \max \{\Pi_H(\mathbf{x}) : \mathbf{x} \in X^m\}.$$

We have used the notation Π_H for both the number of classifications and the growth function, but this should not cause confusion.

10.3.2 VC-dimension

We noted that the number of possible classifications by H of a sample of length m is at most 2^m, this being the number of binary vectors of length m. We say that a sample $\mathbf{x}$ of length m is *shattered* by H, or that H *shatters* $\mathbf{x}$, if this maximum possible value is attained; that is, if H gives all possible classifications of $\mathbf{x}$. We shall also find it useful to talk of a set of points, rather than a sample, being shattered. The notion is the same: the set is shattered if and only if a sample with those entries is shattered. Note that if the examples in $\mathbf{x}$ are not distinct then $\mathbf{x}$ cannot be shattered by any H. When the examples are distinct, $\mathbf{x}$ is shattered by H if and only if for each subset S of E, there is some function f_S in H such that for $1 \leq i \leq m$, $f_S(x_i) = 1 \iff x_i \in S$.

Consistent with the intuitive notion that a set H of functions has high expressive power if it can achieve all possible classifications of a large set of examples, we use as a measure of this power the *Vapnik–Chervonenkis dimension*, or *VC-dimension*, of H, which is defined to be the maximum length of a sample shattered by H. (If there is no such maximum, we say that the VC-dimension of H is infinite.) Using the notation introduced above, we can say that the VC-dimension of H, denoted $\mathrm{VCdim}(H)$, is given by

$$\mathrm{VCdim}(H) = \max\{m : \Pi_H(m) = 2^m\},$$

where we take the maximum to be infinite if the set is unbounded. We state this definition formally, and in a slightly different form, for future reference.

Definition 10.3 (VC-dimension) *Let H be a set of functions from a set X to $\{0,1\}$. The* VC-dimension *of H is (infinite, or) the maximal size of a subset E of X with the property that for each $S \subseteq E$, there is $f_S \in H$ with $f_S(x) = 1$ if $x \in S$ and $f_S(x) = 0$ if $x \in E \setminus S$.*

Recall that, for a binary-output neural network $\mathcal{N}$ with set of possible inputs X, the hypothesis space H consists of all functions from X to $\{0,1\}$ which can be computed by the network in some state. As a set of $\{0,1\}$-valued functions, this hypothesis space has a VC-dimension. When X is clear from the context, we refer to this as the *VC-dimension of the neural network* and denote it simply by $\mathrm{VCdim}(\mathcal{N})$. When X is not clear, we shall use the notation $\mathrm{VCdim}(\mathcal{N}, X)$, and refer to this quantity as the *VC-dimension of $\mathcal{N}$ on example set X*. (For instance, we may wish to consider the VC-dimension of a particular network first on real inputs and then on only binary inputs, and these quantities might well be different.) We state this definition explicitly.

Definition 10.4 (VC-dimension of neural network) *Let $\mathcal{N}$ be a binary-output neural network capable of taking on a number of states, and let Ω denote the set of all possible states. Suppose that X is the set of all possible inputs. Let h_ω be the function from X to $\{0,1\}$ computed by $\mathcal{N}$ when its state is ω, and let*

$$H_{\mathcal{N}} = \{h_\omega : \omega \in \Omega\}$$

be the hypothesis space of functions computable by $\mathcal{N}$. Then the VC-dimension of $\mathcal{N}$ on example set X, *denoted* $\mathrm{VCdim}(\mathcal{N}, X)$ *(or* $\mathrm{VCdim}(\mathcal{N})$ *if X is clear), is defined to be the VC-dimension of $H_{\mathcal{N}}$.*

The following result is often useful.

Theorem 10.5 *If the set H of $\{0,1\}$-valued functions is finite, then it has finite VC-dimension, and* $\mathrm{VCdim}(H) \leq \log_2 |H|$.

Proof. If d is the VC-dimension of H and $\mathbf{x} \in X^d$ is shattered by H, then $|H| \geq |H_{\mathbf{x}}| = 2^d$. (Here, $H_{\mathbf{x}}$ denotes the restriction of H to domain $E = \{x_1, x_2, \ldots, x_d\}$.) It follows that $d \leq \log_2 |H|$. □

Since any given class of Boolean functions is finite, it necessarily will have finite VC-dimension. Thus, for example, if we consider the hypothesis space corresponding to a neural network accepting only binary-valued inputs, then it will have finite VC-dimension. It should be noted that, although Theorem 10.5 provides a bound on the VC-dimension in such cases, the bound may be quite loose. For example, we shall see in the next chapter that the set T_n of all Boolean threshold functions (the hypothesis space of the perceptron with n binary inputs) has VC-dimension $n + 1$, whereas (as we have seen) $\log_2 |T_n|$ is of order n^2.

10.4 Relating growth function and VC-dimension

The growth function $\Pi_H(m)$ of a hypothesis space of finite VC-dimension is a measure of how many different classifications of an m-sample into positive and negative examples can be achieved by the functions of H, while the VC-dimension of H is the maximum value of m for which $\Pi_H(m) = 2^m$. Thus, the VC-dimension is defined in terms of the growth function. But there is a converse relationship: the growth function $\Pi_H(m)$ can be bounded by a polynomial function of m, and the degree of the polynomial is the VC-dimension d of H. Explicitly, we have the following theorem, usually known as Sauer's lemma.

Theorem 10.6 (Sauer's lemma) *Let $d \geq 0$ and $m \geq 1$ be given integers and let H be a set of $\{0,1\}$-valued functions with $\mathrm{VCdim}(H) = d \geq 1$. Then*

$$\Pi_H(m) \leq \sum_{i=0}^{d} \binom{m}{i} < \left(\frac{em}{d}\right)^d,$$

where the second inequality holds for $m \geq d$. (Here, e is the base of natural logarithms.)

Proof. We prove the first inequality. (A proof of the second can be found in the proof of Theorem 4.3.) For $m \leq d$, the inequality is trivially true since in that case the sum is 2^m. Assume that $m > d$ and fix a set $S = \{x_1, \ldots, x_m\} \subseteq X$. We will make use of the correspondence between $\{0,1\}$-valued functions on a set and subsets of that set by defining the set system (or family of sets)

$$\mathcal{F} = \{\{x_i \in S : f(x_i) = 1\} : f \in H\}.$$

The proof proceeds by first creating a transformed version $\mathcal{F}^*$ of $\mathcal{F}$ that is a down-set with respect to the partial order induced by set-inclusion, and which has the same cardinality as $\mathcal{F}$. (To say that $\mathcal{F}^*$ is a down-set means that if $A \in \mathcal{F}^*$ and $B \subseteq A$, then $B \in \mathcal{F}^*$.)

For an element x of S, let T_x denote the operator that, acting on a set system, removes the element x from all sets in the system, unless that would give a set that is already in the system:

$$T_x(\mathcal{F}) = \{A \setminus \{x\} : A \in \mathcal{F}\} \cup \{A \in \mathcal{F} : A \setminus \{x\} \in \mathcal{F}\}.$$

Note that $|T_x(\mathcal{F})| = |\mathcal{F}|$. Consider now $\mathcal{F}^* = T_{x_1}(T_{x_2}(\cdots T_{x_m}(\mathcal{F})\cdots))$. Clearly, $|\mathcal{F}^*| = |\mathcal{F}|$. Furthermore, for all x in S, $T_x(\mathcal{F}^*) = \mathcal{F}^*$. It's clear that $\mathcal{F}^*$ is a down-set. If it was not, there would be some $C \in \mathcal{F}^*$ and some $x \in C$ such that $C \setminus \{x\} \notin \mathcal{F}^*$. But we have applied the operator T_x to obtain $\mathcal{F}^*$; thus, if $C \in \mathcal{F}^*$, then this is only because $C \setminus \{x\}$ is also in $\mathcal{F}^*$.

We can define the notion of shattering for a family of subsets, in the same way as for a family of $\{0,1\}$-valued functions. For $R \subseteq S$, we say that $\mathcal{F}$ shatters R if $\mathcal{F} \cap R = \{A \cap R : A \in \mathcal{F}\}$ is the set of all subsets of R.

We next show that, whenever $\mathcal{F}^*$ shatters a set, so does $\mathcal{F}$. It suffices to show that, for any $x \in S$, if $T_x(\mathcal{F})$ shatters a set, so does $\mathcal{F}$. So suppose that x in S, $R \subseteq S$, and $T_x(\mathcal{F})$ shatters R. If x is not in R, then, trivially, $\mathcal{F}$ shatters R. If x is in R, then for all $A \subseteq R$ with x not in A, since $T_x(\mathcal{F})$ shatters R we have $A \in T_x(\mathcal{F}) \cap R$ and $A \cup \{x\} \in T_x(\mathcal{F}) \cap R$. By the definition of T_x, this implies $A \in \mathcal{F} \cap R$ and $A \cup \{x\} \in \mathcal{F} \cap R$. This argument shows that $\mathcal{F}$ shatters R. It follows that $\mathcal{F}^*$ can only shatter sets of cardinality at most d. Since $\mathcal{F}^*$ is a down-set, this means that the largest set in $\mathcal{F}^*$ has cardinality no more than d. (For, if there were a set of cardinality $d+1$ in $\mathcal{F}^*$, all its subsets would be in $\mathcal{F}^*$ too, because $\mathcal{F}^*$ is a down-set, and it would therefore be shattered.) We therefore have $|\mathcal{F}^*| \leq \sum_{i=0}^{d} \binom{m}{i}$, this expression being the number of subsets of S containing no more than d elements. The result follows, because $|\mathcal{F}| = |\mathcal{F}^*|$, and because S was chosen arbitrarily. □

The first inequality of this theorem is tight. If H corresponds to the set system $\mathcal{F}$ consisting of all subsets of $[n]$ of cardinality at most d, then $\mathrm{VCdim}(H) = d$ and $|\mathcal{F}|$ meets the upper bound.

10.5 VC-dimension and PAC learning

It turns out that the existence of a PAC learning algorithm is dependent on the VC-dimension being finite. In fact, something stronger is true: if the VC-dimension is finite, then *any* consistent learning algorithm is PAC. Conversely, if there exists a PAC learning algorithm, then the VC-dimension has to be finite.

10.5.1 Upper bound on sample length

We now indicate why finite VC-dimension is a sufficient condition for there to exist a PAC learning algorithm, and we indicate how the sufficient sample length can be bounded in terms of the VC-dimension. The following result is central.[2]

Theorem 10.7 *Suppose that H is a hypothesis space and that $t \in H$, probability distribution P, and $\epsilon > 0$ are arbitrary, but fixed. Let*

$$\mathcal{P}_{m,\epsilon} = P^m\left(\{\mathbf{x} \in X^m : \exists h \in H, h(x_i) = t(x_i) \text{ for } i = 1, \ldots, m, \text{ with } \mathrm{er}(h) > \epsilon\}\right)$$

be the probability that, for a randomly chosen training sample $\mathbf{s} \in S(m,t)$, there is some $h \in H$ which, although consistent with t, has error greater than ϵ. Then

$$\mathcal{P}_{m,\epsilon} < 2\,\Pi_H(2m)\,2^{-\epsilon m/2}$$

for all positive integers $m \geq 8/\epsilon$.

[2]This result requires some measure-theoretic assumptions about H and P, which we shall not explore: these assumptions are, however, very plausible.

The proof of this theorem is given shortly, but we first explore its implications for PAC learning. Consider the upper bound on $\mathcal{P}_{m,\epsilon}$, the probability that some consistent function has error greater than ϵ. Suppose that H has finite VC-dimension. By Sauer's lemma, the growth function is bounded by a polynomial in m. Hence, as $m \to \infty$, for fixed ϵ, we have $\mathcal{P}_{m,\epsilon} \to 0$. In particular, therefore, given $\delta > 0$, there is some $m_0(\delta, \epsilon)$ such that for $m \geq m_0$, $\mathcal{P}_{m,\epsilon} < \delta$. The following result provides such an m_0. (The proof, which isn't hard, is omitted.) Better (that is, smaller) bounds on sufficient sample length m_0 can often be obtained if one knows bounds on the growth function directly. However, it is often easier to work with the VC-dimension. The following results make it quite explicit that finite VC-dimension is sufficient for PAC learnability.

Theorem 10.8 *Suppose that H is a hypothesis space of finite VC-dimension $d \geq 1$ and that $0 < \delta, \epsilon < 1$. Let*

$$m_0 = m_0(\delta, \epsilon) = \frac{4}{\epsilon}\left(d \log_2\left(\frac{12}{\epsilon}\right) + \log_2\left(\frac{2}{\delta}\right)\right).$$

Then, in the notation of Theorem 10.7, for any $m \geq m_0$, $\mathcal{P}_{m,\epsilon} < \delta$ for any probability distribution P on X.

Immediately, therefore, we obtain the following corollary.

Theorem 10.9 *Suppose that hypothesis space H has VC-dimension $d \geq 1$. Then any consistent learning algorithm L is a PAC learning algorithm, with a sufficient sample length being*

$$m_0(\delta, \epsilon) = \frac{4}{\epsilon}\left(d \log_2\left(\frac{12}{\epsilon}\right) + \log_2\left(\frac{2}{\delta}\right)\right).$$

This result is an extension to infinite H of Theorem 10.2.

10.5.2 Proof of Theorem 10.7

The proof of Theorem 10.7 is rather involved, and it is worth giving first a very informal explanation of the method. We aim to bound the probability that a given sample $\mathbf{x}$ of length m is "bad," in the sense that there is some function in H which is consistent with the target concept t on the sample but which has error greater than ϵ. We transform this problem into a slightly more manageable one involving samples of length $2m$. For such a sample, the subsample $\mathbf{x}$ comprising the first half of the sample may be thought of as a randomly drawn sample of length m, while the second half may be thought of as a "testing" sample on which to evaluate the performance of a function consistent with the target concept on $\mathbf{x}$. We obtain a bound on the probability that some function in H consistent with the target on the first half of the sample is "bad," in the sense that it has "observed error" greater than $\epsilon/2$ on the second half of the sample. A given example is just as likely to occur in the first half as in the second half. A "group action" based on this idea enables us to find the required bound by solving a simple counting problem.

Suppose $t \in H$ is a fixed, but arbitrary, function from H in what follows. We require an important definition: for any $\mathbf{z} \in X^m$, the *observed error* $\mathrm{er}_{\mathbf{z}}(h)$ of $h \in H$ on $\mathbf{z}$ (with respect to t) is simply the proportion of entries z_i of $\mathbf{z}$ on which h and t disagree; that is,

$$\mathrm{er}_{\mathbf{z}}(h) = \frac{1}{m}\left|\{i : h(z_i) \neq t(z_i)\}\right|.$$

Given samples $\mathbf{x}, \mathbf{y} \in X^m$, let $\mathbf{xy} \in X^{2m}$ denote the sample of length $2m$ obtained by concatenating $\mathbf{x}$ and $\mathbf{y}$. With this notation, define

$$R_m^\epsilon = \left\{ \mathbf{xy} \in X^{2m} : \exists h \in B_\epsilon \text{ for which } \mathrm{er}_{\mathbf{x}}(h) = 0 \text{ and } \mathrm{er}_{\mathbf{y}}(h) > \frac{\epsilon}{2} \right\}.$$

The following result relates the probability we want to bound to the probability of an event involving observed errors and samples of length $2m$.

Lemma 10.10 *For all $m \geq 8/\epsilon$,*

$$\mathcal{P}_{m,\epsilon} \leq 2P^{2m}(R_m^\epsilon).$$

Proof. Let us denote by $H[\mathbf{x}]$ the set of functions h consistent with t on $\mathbf{x}$, that is, the functions h such that $h(x_i) = t(x_i)$ for $i = 1, \ldots, m$. Let B_ϵ, the set of "ϵ-bad" functions, be those h for which $\mathrm{er}(h) > \epsilon$. Then $\mathcal{P}_{m,\epsilon}$ is the P^m probability of the set

$$Q_m^\epsilon = \{\mathbf{x} \in X^m : H[\mathbf{x}] \cap B_\epsilon \neq \emptyset\}.$$

Let χ_Q be the characteristic function of Q_m^ϵ; that is, $\chi_Q(\mathbf{x}) = 1$ if $\mathbf{x} \in Q_m^\epsilon$ and $\chi_Q(\mathbf{x}) = 0$ otherwise. If we define the characteristic function χ_R similarly, then

$$\chi_R(\mathbf{xy}) = \chi_Q(\mathbf{x})\psi_{\mathbf{x}}(\mathbf{y}),$$

where

$$\psi_{\mathbf{x}}(\mathbf{y}) = \begin{cases} 1 & \text{if } \exists h \in H[\mathbf{x}] \cap B_\epsilon \text{ with } \mathrm{er}_{\mathbf{y}}(h) > \epsilon/2, \\ 0 & \text{otherwise.} \end{cases}$$

Now we have

$$P^{2m}(R_m^\epsilon) = \int \chi_R(\mathbf{xy}) = \int \left(\chi_Q(\mathbf{x}) \int \psi_{\mathbf{x}}(\mathbf{y}) \right),$$

where the integrals are taken over the whole of the relevant sets, with respect to the product measures. The inner integral is the probability that, given $\mathbf{x}$, there is some $h \in B_\epsilon$ which is consistent with $\mathbf{x}$ and satisfies $\mathrm{er}_{\mathbf{y}}(h) > \epsilon/2$. This is certainly not less than the probability that, for a particular $h \in B_\epsilon$ consistent with $\mathbf{x}$, we have $\mathrm{er}_{\mathbf{y}}(h) > \epsilon/2$. Thus it suffices to show that this latter probability is at least $\frac{1}{2}$, for then we have

$$P^{2m}(R_m^\epsilon) \geq \int \frac{1}{2}\chi_Q(\mathbf{x}) = \frac{1}{2}P^m(Q_m^\epsilon).$$

In order to prove this, we use Chebyshev's inequality. This states that if a random variable Z has mean μ and variance σ^2 then $\mathrm{Prob}(|Z - \mu| > a) < \sigma^2/a^2$, a bound which follows immediately from the observation that σ^2 is the expected value of $|Z - \mu|^2$, and hence

$$\sigma^2 > a^2 \,\mathrm{Prob}(|Z - \mu| > a).$$

Let $h \in B_\epsilon$, so that $\mathrm{er}(h) = \epsilon_h > \epsilon$. For $\mathbf{y} \in X^m$, $m\,\mathrm{er}_y(h)$ is the number of components of $\mathbf{y}$ on which h and t disagree, and so it is a binomially distributed random variable, with mean $\mu = m\,\epsilon_h$ and variance $\sigma^2 = m\,\epsilon_h(1 - \epsilon_h)$. Now, applying Chebyshev's inequality, we have

$$\begin{aligned} P^m\left(\mathrm{er}_{\mathbf{y}}(h) \leq \epsilon/2\right) &= P^m\left(|m\,\mathrm{er}_{\mathbf{y}}(h) - m\,\epsilon_h| > m\,\epsilon_h - m\,\epsilon/2\right) \\ &\leq \frac{\epsilon_h(1 - \epsilon_h)m}{m^2(\epsilon_h - (\epsilon/2))^2} \\ &\leq \frac{\epsilon_h}{m(\epsilon_h/2)(\epsilon/2)} \\ &= \frac{4}{\epsilon m}, \end{aligned}$$

which is at most $1/2$ for $m \geq 8/\epsilon$. □

The next stage is to bound the probability of R_m^ϵ by making use of a group action on X^{2m} to transform the problem of bounding $P^{2m}(R_m^\epsilon)$ into a simple counting problem.

For $i \in \{1, \ldots, m\}$ let τ_i be the permutation of $\{1, \ldots, 2m\}$ which switches i and $m+i$. There is an induced transformation of X^{2m} defined by letting τ_i act on the coordinates, and we use τ_i to denote this transformation also. Thus, for example, if $m = 4$,

$$\tau_2(z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8) = (z_1, z_6, z_3, z_4, z_5, z_2, z_7, z_8).$$

Let G_m be the group generated by the permutations τ_i $(1 \leq i \leq m)$. Note that $|G_m| = 2^m$.

Lemma 10.11 *Given $\mathbf{z} \in X^{2m}$, let $\Gamma(\mathbf{z})$ denote the number of $\sigma \in G_m$ for which $\sigma\mathbf{z}$ is in R_m^ϵ. Then*

$$|G_m|\, P^{2m}(R_m^\epsilon) \;\leq\; \max \Gamma(\mathbf{z}),$$

where this maximum is taken over all $\mathbf{z} \in X^{2m}$.

Proof. Let χ_R be the characteristic function of R. Since $G = G_n$ is finite we can interchange summation and integration as follows (where the integral sign represents integration over the entire space with respect to the product measure derived from P):

$$\sum_{\sigma \in G} \int \chi_R(\sigma\mathbf{z}) \;=\; \int \sum_{\sigma \in G} \chi_R(\sigma\mathbf{z}).$$

The left-hand side is the sum over σ of the measure of $\sigma^{-1}(R)$, which is the same as the measure of R, since coordinate-permutations preserve the product measure. Hence the left-hand side is just $|G|\, P^n(R)$. The integrand on the right-hand side is just the number of σ in G for which $\sigma\mathbf{z} \in R$. Since the total weight of a probability measure is 1, the integral is bounded by the maximum of this quantity, taken over $\mathbf{z}$. □

Given any $h \in B_\epsilon$, let

$$R_m^\epsilon(h) = \left\{ \mathbf{xy} \in X^{2m} : \mathrm{er}_{\mathbf{x}}(h) = 0 \text{ and } \mathrm{er}_{\mathbf{y}}(h) > \frac{\epsilon}{2} \right\}.$$

Also, for $\mathbf{z} \in X^{2m}$, let $\Gamma(h, \mathbf{z})$ denote the number of $\sigma \in G_m$ which transform $\mathbf{z}$ to a vector in $R_m^\epsilon(h)$.

Lemma 10.12 *Suppose that m is any positive integer and that $h \in B_\epsilon$. Then*

$$\Gamma(h, \mathbf{z}) < 2^{m(1-\epsilon/2)}$$

for all $\mathbf{z} \in X^{2m}$.

Proof. Suppose that $\Gamma(h, \mathbf{z}) \neq 0$. If $\mathbf{z} \notin R_m^\epsilon(h)$, then for some $\tau \in G_m$, $\tau\mathbf{z} \in R_m^\epsilon(h)$. But the number of σ such that $\sigma\mathbf{z} \in R_m^\epsilon(h)$ is precisely the number of σ for which $\sigma\tau\mathbf{z} \in R_m^\epsilon(h)$ (since G_m is a group). Hence, we may, without loss of generality, suppose that $\mathbf{z} \in R_m^\epsilon(h)$.

Now, $\mathbf{z} = \mathbf{xy}$ where $\mathrm{er}_x(h) = 0$ and $\mathrm{er}_y(h) > \epsilon/2$. To simplify notation, let us suppose that the $r > m\epsilon/2$ entries of $\mathbf{z}$ on which h and t disagree are

$$z_{m+1}, z_{m+2}, \ldots, z_{m+r}.$$

Recall that a transformation $\sigma \in G_m$ interchanges some pairs (z_j, z_{m+j}). If $\sigma\mathbf{z}$ is in $R_m^\epsilon(h)$ then σ does not interchange (z_j, z_{m+j}) for $1 \leq j \leq r$. Conversely, any σ which satisfies this condition is in $R_m^\epsilon(h)$. Since σ is uniquely determined by the set of j for which $\sigma(z_j) = z_{m+j}$, the number of such σ is just the number of subsets of $\{r+1, r+2, \ldots, m\}$; that is, $\Gamma(h, \mathbf{z}) = 2^{m-r}$. Since $r > \epsilon m/2$, we have

$$\Gamma(h, \mathbf{z}) < 2^{m-\epsilon m/2},$$

as required. □

Lemma 10.13 *For any positive integer m,*

$$P^{2m}(R_m^\epsilon) < \Pi_H(2m)\, 2^{-\epsilon m/2}.$$

Proof. Let $\mathbf{z} \in X^{2m}$ be fixed but arbitrary, and let $s = \Pi_{B_\epsilon}(\mathbf{z})$ be the number of classifications of $\mathbf{z}$ by B_ϵ. Then there are functions $h_1, \ldots, h_s$ in B_ϵ which give s different classifications of $\mathbf{z}$ and, further, any classification of $\mathbf{z}$ by a function in B_ϵ is one of these s classifications. We have

$$s = \Pi_{B_\epsilon}(\mathbf{z}) \leq \Pi_H(\mathbf{z}) \leq \Pi_H(2m).$$

Suppose $\sigma\mathbf{z} = \mathbf{ab}$ is in R_m^ϵ. This means that there is some $h \in H$ such that $\mathrm{er}_\mathbf{a}(h) = 0$ and $\mathrm{er}_\mathbf{b}(h) > \epsilon/2$. Since all classifications of $\mathbf{z}$, and hence of its rearrangement $\sigma\mathbf{z} = \mathbf{ab}$, are realized by some h_i $(1 \leq i \leq s)$, it follows that $\sigma\mathbf{z}$ is in one of the sets $R_m^\epsilon(h_i)$. Thus the set of σ for which $\sigma\mathbf{z}$ is in R_m^ϵ is the union of the sets of those σ for which $\sigma\mathbf{z}$ is in $R_m^\epsilon(h_i)$. We therefore have

$$\Gamma(\mathbf{z}) \leq \sum_{i=1}^{s} \Gamma(h_i, \mathbf{z}).$$

The last expression is the sum of $s \leq \Pi_H(2m)$ terms and, by Lemma 10.12, each of them is bounded above by $2^{m(1-\epsilon/2)}$. Thus, from Lemma 10.11, we have

$$P^{2m}(R_m^\epsilon) \leq |G_m|^{-1} \max_{\mathbf{z}} \Gamma(\mathbf{z}) \leq 2^{-m}\Pi_H(2m)\, 2^{m(1-\epsilon/2)} = \Pi_H(2m)\, 2^{-m\epsilon/2},$$

as claimed. □

Theorem 10.7 now follows from Lemmas 10.10 and 10.13.

10.5.3 Lower bound on sample length

We now explain why finite VC-dimension is a necessary condition for there to exist a PAC learning algorithm. In fact, we present a result which establishes something stronger, namely a lower bound on the sufficient sample length m_0 in terms of the VC-dimension.

Theorem 10.14 *Let H be a hypothesis space of VC-dimension at least 1 and consisting of at least three distinct concepts. Suppose that L is any PAC learning algorithm. Then*

the sufficient sample length $m_0(\delta, \epsilon)$ in the definition of PAC learnability must satisfy

$$m_0(\delta, \epsilon) > \max\left(\frac{\mathrm{VCdim}(H) - 1}{32\epsilon}, \frac{1}{\epsilon}\ln\left(\frac{1}{\delta}\right)\right)$$

for all $\epsilon \leq 1/8$ and $\delta \leq 1/100$.

It follows immediately from this result that if the VC-dimension is infinite, there can be no PAC learning algorithm.

10.5.4 VC-dimension characterizes PAC learnability

To sum up the results of this section, we see that if H is a hypothesis space (the set of functions computable by a neural network), then finite VC-dimension of H is both a necessary and sufficient condition for there to exist a PAC learning algorithm. It should be noted, though, that at this level of analysis, by a learning algorithm we simply mean a function from training samples to hypotheses: questions of algorithmic efficiency remain and will be discussed in a later chapter. Not only is finiteness of the VC-dimension a characterization for PAC learnability but the VC-dimension provides bounds on the sufficient sample length $m_0(\delta, \epsilon)$ which one should use for learning with the required accuracy and confidence. The upper bounds and lower bounds we have presented imply the existence of constants c, k such that the minimal sufficient sample length m_0 satisfies

$$\frac{k}{\epsilon}\left(\mathrm{VCdim}(H) + \ln\left(\frac{1}{\delta}\right)\right) \leq m_0(\delta, \epsilon) \leq \frac{c}{\epsilon}\left(\mathrm{VCdim}(H)\ln\left(\frac{1}{\epsilon}\right) + \ln\left(\frac{1}{\delta}\right)\right)$$

for all ϵ and δ sufficiently small.

10.6 Additional remarks and bibliographical notes

The model of PAC learning was introduced by Valiant [101, 100], and it is now the subject of several books, including [8, 56, 7]. The VC-dimension was first used in probability theory to develop "uniform convergence" results [103, 102]. Since then it has found many other applications, from learning theory (as discussed here) to logic (see, for instance, [93]) and computational geometry (see [68]). The relationship between VC-dimension and learnability was established in [20]. Theorem 10.9 is from [20]. The proof of Theorem 10.7 uses techniques similar to those used in [103], and the proof presented here makes use of the "swapping group" as used by Pollard [85]. Discussion of the measurability conditions required for Theorem 10.7 may be found in [20, 85]. Proofs of Theorem 10.9 and some improvements of it may be found, for example, in [9, 8, 56]. Theorem 10.14 follows from two lower bound results. The main one, of order d/ϵ, is due to Ehrenfeucht et al. [36]; the other, of order $(1/\epsilon)\ln(1/\delta)$, is from [20].

Sauer's lemma, Theorem 10.6, is due to Sauer [91] and, independently, to Shelah [93]. The proof presented here is from [7], and it follows a proof of Steele [98, 22].

A number of extensions of the basic PAC model have been made. In particular, the model can be adapted to remove the assumption that the labeled examples are classified by a fixed target concept belonging to the class and, more significantly, a PAC-type learning framework can be developed for neural networks whose output is a real number, not necessarily 0 or 1. See [42], for example. These and other recent developments are discussed in the book [7].

This page intentionally left blank

Chapter 11

VC-dimensions of Neural Networks

11.1 Introduction

We have seen that the VC-dimension and the growth function are central to PAC learnability. A PAC learning algorithm for a neural network exists if and only if the network has finite VC-dimension. Moreover, the sample complexity (that is, the minimum sufficient sample length for PAC learning) can be quite tightly bounded in terms of the VC-dimension. In this chapter, therefore, our aim is to present some bounds on the VC-dimension of certain types of neural network.

11.2 VC-dimension and linear dimension

In this section, we present a useful result relating linear (vector-space) dimension to the VC-dimension. We will find this useful when it comes to bounding the VC-dimension of certain neural networks.

Recall that a homogeneous threshold function on $\mathbb{R}^n$ is a real linear threshold function with threshold 0. Thus, the set of positive examples is $\{x \in \mathbb{R}^n : \sum_{i=1}^n w_i x_i \geq 0\}$, for some constants w_i, $(1 \leq i \leq m)$. We have the following characterization of the sets which are shattered by homogeneous threshold functions.

Theorem 11.1 *A set $E = \{x_1, x_2, \ldots, x_k\}$ of points of $\mathbb{R}^n$ can be shattered by the set of homogeneous threshold functions if and only if the points of E are linearly independent.*

Proof. Suppose that the points are linearly dependent. Then at least one is a linear combination of the others. Without loss of generality, suppose $x_1 = \sum_{i=2}^k \lambda_i x_i$ for some constants λ_i, $(1 \leq i \leq k)$, not all zero. Suppose the vector w is such that for $2 \leq j \leq k$, the inner product $\langle w, x_j \rangle$ is nonnegative if $\lambda_j > 0$ and negative if $\lambda_j \leq 0$. Then $\langle w, x_1 \rangle = \sum_{i=2}^k \lambda_i \langle w, x_j \rangle \geq 0$. It follows that there is no homogeneous threshold function such that the following holds: x_1 is a negative example and, for $2 \leq j \leq k$, x_j is a positive example if and only if $\lambda_j > 0$. Thus the set E is not shattered.

For the converse, it suffices to prove the result when $k = n$. Let A be the matrix whose rows are the row vectors $x_1, x_2, \ldots, x_n$ and let $v \in \{-1, 1\}^n$. Then A is nonsingular and so the matrix equation $Aw = v$ has a solution. The homogeneous threshold

function t defined by this solution weight-vector w satisfies $t(x_j) = 1$ if and only if entry j of v is 1. Thus, all possible classifications of the set of vectors can be realized, and the set is shattered. □

Suppose $\mathcal{V}$ is a set of real functions defined on some set X. For $f, g \in \mathcal{V}$ and $\lambda \in \mathbb{R}$, we can form the function $f + g : X \to \mathbb{R}$ by *pointwise addition* and the function $\lambda f : X \to \mathbb{R}$ by *pointwise scalar multiplication*, as follows:

$$(f+g)(x) = f(x) + g(x), \quad (\lambda f)(x) = \lambda f(x), \quad (x \in X).$$

If $\mathcal{V}$ is closed under these operations, then it is a vector space of functions. Then, in $\mathcal{V}$, we say that the set $\{f_1, f_2, \ldots, f_k\}$ of functions is *linearly dependent* if there are constants λ_i $(1 \leq i \leq k)$, not all zero, such that, *for all* $x \in X$,

$$\lambda_1 f_1(x) + \lambda_2 f_2(x) + \cdots + \lambda_k f_k(x) = 0;$$

that is, some nontrivial linear combination of the functions is the zero function on X. The set is *linearly independent* if it is not linearly dependent. The vector space $\mathcal{V}$ is finite-dimensional, of linear dimension d, if the maximum cardinality of a linearly independent set of functions in $\mathcal{V}$ is d. We have the following connection between linear dimension and VC-dimension.

Theorem 11.2 *Let $\mathcal{V}$ be a real vector space of real-valued functions defined on a set X. Suppose that $\mathcal{V}$ has linear dimension d. For any $f \in \mathcal{V}$, define the $\{0,1\}$-valued function f_+ on X by*

$$f_+(x) = \begin{cases} 1 & \text{if } f(x) \geq 0, \\ 0 & \text{if } f(x) < 0, \end{cases}$$

and let $\operatorname{sgn}(\mathcal{V}) = \{f_+ : f \in \mathcal{V}\}$. *Then the VC-dimension of* $\operatorname{sgn}(\mathcal{V})$ *is d.*

Proof. Suppose that $\{f_1, f_2, \ldots, f_d\}$ is a basis for $\mathcal{V}$ and, for $x \in X$, let

$$x^{\mathcal{V}} = (f_1(x), f_2(x), \ldots, f_d(x)) \in \mathbb{R}^d.$$

Suppose the subset E of X is shattered by $\operatorname{sgn}(\mathcal{V})$. Then for each $S \subseteq E$ there is $f_S \in \mathcal{V}$ such that $f_S(x) \geq 0$ if $x \in S$ and $f_S(x) < 0$ if $x \in E \setminus S$. Now, since $\{f_1, \ldots, f_d\}$ is a basis of $\mathcal{V}$, there are constants w_i $(1 \leq i \leq d)$ such that $f_S = \sum_{i=1}^{d} w_i f_i$. Then the condition $f_S(x) \geq 0$ for $x \in S$ and $f(x) < 0$ for $x \in E \setminus S$ is equivalent to

$$\sum_{i=1}^{d} w_i f_i(x) \geq 0 \text{ if } x \in S, \quad \sum_{i=1}^{d} w_i f_i(x) < 0 \text{ if } x \in E \setminus S.$$

But this is true if and only if there is a homogeneous threshold function (given by the weight-vector whose entries are the w_i) such that $t(x^{\mathcal{V}}) = 1$ if $x \in S$ and $t(x^{\mathcal{V}}) = 0$ if $x \in E \setminus S$. Thus, E is shattered by $\operatorname{sgn}(\mathcal{V})$ if and only if the set $E^{\mathcal{V}} = \{x^{\mathcal{V}} : x \in E\}$ is shattered by the set of homogeneous threshold functions. It follows, first, that the VC-dimension of $\operatorname{sgn}(\mathcal{V})$ is at most d. Second, it is equal to d if and only if there is a set E of cardinality d such that $E^{\mathcal{V}}$ is a linearly independent set. Suppose that this is not so. Then the vector subspace spanned by the set $\{x^{\mathcal{V}} : x \in X\}$ is of dimension at most $d-1$ and therefore is contained in some hyperplane. Hence there are constants $\lambda_1, \lambda_2, \ldots, \lambda_d$, not all zero, such that for every $x \in X$, $\sum_{i=1}^{d} \lambda_i (x^{\mathcal{V}})_i = 0$; that is, $\sum_{i=1}^{d} \lambda_i f_i(x) = 0$ for all x. But this contradicts the fact that the functions $f_1, \ldots, f_d$ are linearly independent. It follows that the VC-dimension of $\operatorname{sgn}(\mathcal{V})$ is d. □

11.3 VC-dimension of the perceptron

The VC-dimension of the perceptron with real inputs can be bounded fairly directly using Theorem 11.2 because the class of real linear threshold functions is precisely $\text{sgn}(\mathcal{V})$, where $\mathcal{V}$ is the set of affine functions, of the form $x \mapsto w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$ for some constants $w_0, w_1, \ldots, w_n$. The set $\mathcal{V}$ is easily seen to be a vector space of linear dimension $n+1$, and hence the perceptron has VC-dimension $n+1$. In fact, this is so even if we restrict the inputs of the perceptron to be binary.

Theorem 11.3 *Let P_n denote the perceptron (single linear threshold unit) on n inputs. Then* $\text{VCdim}(P_n, \{0,1\}^n) = \text{VCdim}(P_n, \mathbb{R}^n) = n+1$.

Proof. We have already indicated why $\text{VCdim}(P_n, \mathbb{R}^n) = n+1$. Certainly, we must have $\text{VCdim}(P_n, \{0,1\}^n) \leq \text{VCdim}(P_n, \mathbb{R}^n)$, so the result will follow if we show that $\text{VCdim}(P_n, \{0,1\}^n) \geq n+1$. We do so by proving that a particular subset of $\{0,1\}^n$ of cardinality $n+1$ is shattered by the hypothesis space of P_n. Let $\mathbf{0}$ denote the all-0 vector and, for $1 \leq i \leq n$, let e_i be the point with a 1 in the ith coordinate and all other coordinates 0. We shall show that P_n shatters the sample $\mathbf{x} = (\mathbf{0}, e_1, e_2, \ldots, e_n)$. Suppose that S is any subset of $E = \{\mathbf{0}, e_1, \ldots, e_n\}$. For $i = 1, 2, \ldots, n$, let

$$w_i = \begin{cases} 1 & \text{if } e_i \in S, \\ -1 & \text{if } e_i \notin S, \end{cases}$$

and let

$$\theta = \begin{cases} -1/2 & \text{if } \mathbf{0} \in S, \\ 1/2 & \text{if } \mathbf{0} \notin S. \end{cases}$$

Then it is straightforward to verify that if ω is the state

$$\omega = (w_1, w_2, \ldots, w_n, \theta)$$

of P_n, then the set of positive examples of h_ω in E is precisely S. Therefore $\mathbf{x}$ is shattered by P_n and, consequently, $\text{VCdim}(P_n) \geq n+1$. The result follows. □

Given the type of analysis carried out in Chapter 4, it is fairly easy to bound directly the growth function of the perceptron. Theorem 4.4 shows that for any $S \subseteq \{0,1\}^n$, if T_S is the set of functions in T_n restricted to domain S,

$$|T_S| \leq C(|S|, n+1) = 2\sum_{k=0}^{n} \binom{|S|-1}{k} < 2\left(\frac{|S|e}{n}\right)^n.$$

A closer look at the analysis in Chapter 4 shows, however, that the same bound holds for any subset S of $\mathbb{R}^n$. (That is, S does not have to be a subset of $\{0,1\}^n$.) Now, this means that for all m, if H is the hypothesis space of the perceptron with real inputs, we have

$$\Pi_H(m) \leq 2\sum_{k=1}^{n} \binom{m-1}{k}.$$

Since this is less than 2^m for $m > n+1$, we see again that the VC-dimension is at most $n+1$.

In fact, one can characterize fairly precisely the subsets of $\mathbb{R}^n$ which are shattered by the perceptron. In a manner similar to the proof of Theorem 11.1, the following can be shown.

Theorem 11.4 *A set $E = \{x_1, x_2, \ldots, x_k\}$ of points of $\mathbb{R}^n$ can be shattered by the perceptron if and only if E is an affinely independent set, meaning that $\{x_2 - x_1, x_3 - x_1, \ldots, x_k - x_1\}$ is a linearly independent set of vectors.*

Since the largest affinely independent set in $\mathbb{R}^n$ has cardinality $n + 1$, the VC-dimension result follows immediately from this characterization.

11.4 VC-dimension of class of polynomial threshold functions

We now bound the VC-dimension of the classes of real and Boolean polynomial threshold functions. Recall that a function $f : \mathbb{R}^n \to \{0, 1\}$ is a real polynomial threshold function of degree m if

$$f(x) = \operatorname{sgn}\left(\sum_{S \in [n]^m} w_S x_S\right),$$

for some $w_S \in \mathbb{R}$, where $[n]^m$ is the set of multisets of at most m elements from $\{1, 2, \ldots, n\}$. Here, x_S denotes the product of the x_i for $i \in S$ (with repetitions as required).

Let $B(n, m) = \{x_S : S \in [n]^m\}$, regarded as a set of real functions on $\mathbb{R}^n$. (Thus, for example, we identify $x_{\{1,2\}}$ with the function $x \mapsto x_1 x_2$.)

Theorem 11.5 *For all n and m, the set $B(n, m)$ is a linearly independent set of real functions on $\mathbb{R}^n$.*

Proof. We prove by induction on n that for all m, $B(n, m)$ is a linearly independent set of real functions on $\mathbb{R}^n$. The base case $n = 1$ is straightforward; it is well known that the functions $1, x, x^2, x^3, \ldots, x^m$ are linearly independent. Suppose now that the assertion is true for a value of $n \geq 1$ and let m be any positive integer. By the inductive assumption, the set $\{x_S : S \in [n]^m\}$ is a linearly independent set. For $0 \leq k \leq m$, let $S_k \subseteq [n+1]^m$ be the set of multisets containing $n + 1$ exactly k times. Suppose that for some constants c_S, for all $x \in \mathbb{R}^{n+1}$,

$$\sum_{S \in [n+1]^m} c_S x_S = 0.$$

Then

$$\sum_{k=0}^{m} x_{n+1}^k \sum_{S \in S_k} c_S x_S^* = 0$$

for all x, where, for $S \in S_k$, x_S^* is x_S with the k factors equal to x_{n+1} deleted. (So, x_S^* is of the form x_T for some $T \in [n]^m$; that is, $x_S^* \in B(n, m)$.) It follows, from the linear independence of $1, x_{n+1}, x_{n+1}^2, \ldots, x_{n+1}^m$, that for all $x_1, x_2, \ldots, x_n$, we have

$$\sum_{S \in S_k} c_S x_S^* = 0$$

for each k. But the inductive assumption then implies that for all k and for all $S \in S_k$, $c_S = 0$; that is, all the coefficients c_S are zero. Hence the functions are linearly independent. □

Turning now to the Boolean polynomial threshold functions, we recall that these have the form

$$f(x) = \text{sgn}\left(\sum_{S \in [n]^{(m)}} w_S x_S\right)$$

for some $w_S \in \mathbb{R}$ $(S \subseteq [n]^{(m)})$, where $[n]^{(m)}$ is the set of subsets of $[n]$ of cardinality at most m. For $m \leq n$, let $C(n,m) = \left\{x_S : S \in [n]^{(m)}\right\}$, regarded as a set of real functions on domain $\{0,1\}^n$.

Theorem 11.6 *For all n, m with $m \leq n$, $C(n,m)$ is a linearly independent set of real functions defined on $\{0,1\}^n$.*

Proof. Let $n \geq 1$ and suppose that for some constants c_S and for all $x \in \{0,1\}^n$,

$$A(x) = \sum_{S \in [n]^{(m)}} c_S x_S = 0.$$

Set x to be the all-0 vector to deduce that $c_\emptyset = 0$. Let $1 \leq k \leq m$ and assume inductively that $c_S = 0$ for all $S \subseteq [n]$ with $|S| < k$. Let $S \subseteq [n]$ with $|S| = k$. Setting $x_i = 1$ if $i \in S$ and $x_j = 0$ if $j \notin S$, we deduce that $A(x) = c_S = 0$. Thus, for all S of cardinality k, $c_S = 0$. Hence $c_S = 0$ for all S, and the functions are linearly independent. □

We therefore obtain the following VC-dimension results.

Theorem 11.7 *Let $P(n,m)$ be the set of real polynomial threshold functions of degree m, and let $T(n,m)$ be the set of Boolean polynomial threshold functions of degree m. Then, for all n, m,*

$$\text{VCdim}(P(n,m)) = \binom{n+m}{m},$$

and for all n, m with $m \leq n$,

$$\text{VCdim}(T(n,m)) = \sum_{i=0}^{m} \binom{n}{i}.$$

Proof. Let $\mathcal{V}$ be the vector subspace of the space of all real functions on $\mathbb{R}^n$ spanned by $B(n,m)$. Since $B(n,m)$ is a linearly independent set of $\binom{n+m}{m}$ functions, it follows that $\mathcal{V}$ has vector-space dimension $|B(n,m)| = \binom{n+m}{m}$. But, clearly, $P(n,m) = \text{sgn}(\mathcal{V})$, in the notation of Theorem 11.2. By that result, $P(n,m)$ has the given VC-dimension.

In a similar way, let $\mathcal{W}$ be the vector subspace of the space of all real functions defined on $\{0,1\}^n$ spanned by $C(n,m)$. Then $\mathcal{W}$ has vector-space dimension $|C(n,m)| = \sum_{i=0}^{m} \binom{n}{i}$. However, $T(n,m) = \text{sgn}(\mathcal{W})$ and therefore $T(n,m)$ has VC dimension $\sum_{i=0}^{m} \binom{n}{i}$. □

Note that if all inputs are binary, the VC dimension is lower than if the inputs are allowed to be arbitrary real numbers. We remark that, as we saw earlier, the VC-dimensions coincide for $m = 1$, the case of linear threshold functions.

11.5 VC-dimension of threshold networks

We now provide a simple bound on the VC-dimension of feed-forward linear threshold networks.

Theorem 11.8 *Suppose that $\mathcal{N}$ is a feed-forward linear threshold network having a total of W variable weights and thresholds, and n inputs. Then*

$$\text{VCdim}(\mathcal{N}, \{0,1\}^n) \leq \text{VCdim}(\mathcal{N}, \mathbb{R}^n) < 6W \log_2 W.$$

Proof. The proof is similar to the proof of Theorem 7.4. Let $X = \mathbb{R}^n$ and suppose that $\mathbf{x} \in X^m$ is a sample of m points from X. We bound the growth function of hypothesis space $H = H_{\mathcal{N}}$ by bounding $\Pi_H(\mathbf{x})$ independently of $\mathbf{x}$. Denote by N the number of computation units (that is, the number of linear threshold neurons) in the network. Since the network is a feed-forward network, the computation units may be labeled with the integers $1, 2, \ldots, N$ so that if the output of threshold unit i feeds into unit j then $i < j$. Consider any particular threshold unit, i. Denote the in-degree of i by d_i. The set of functions computable by unit i (in isolation) has VC-dimension $d_i + 1$, by Theorem 11.3. It follows, by a simple consequence of Sauer's lemma, that if $\mathbf{y}$ is any sample of length m of points in $\{0,1\}^{d_i}$, then the number of ways in which unit i can classify $\mathbf{y}$ is at most m^{d_i+2} for $m \geq d_i + 1$. It follows that (if $m > \max_i d_i + 1$) the number of classifications $\Pi_H(\mathbf{x})$ of $\mathbf{x} \in X^m$ by the network is bounded by

$$m^{d_1+2} m^{d_2+2} \ldots m^{d_N+2},$$

which, since $W = d_1 + d_2 + \cdots + d_N + N$, the total number of weights and thresholds, is at most m^{W+N}. Since $W \geq N$ (there being a threshold for each threshold unit), this is at most m^{2W}. Now, $m^{2W} < 2^m$ if $m = 6W \log_2 W$, from which it follows that the VC-dimension of the network is less than $6W \log_2 W$. □

We know from Theorem 10.9 that, since the VC-dimension is finite, any consistent learning algorithm for a threshold network will be PAC. Moreover, we can provide a sufficient sample length. For PAC learning with accuracy ϵ and confidence δ, it suffices to use a sample length of order $\epsilon^{-1}(W \log_2 W \log_2(1/\epsilon) + \log_2(1/\delta))$. (See Theorem 10.9.) However, a slightly better bound may be obtained by using Theorem 10.7 together with the explicit estimate we have obtained for the growth function, namely $\Pi_H(m) < m^{2W}$ for $m > W$. If we do this, we obtain a sample length bound of order $\epsilon^{-1}W(\ln(W/\epsilon) + \ln(1/\delta))$.

With more careful use of Sauer's lemma, the VC-dimension of feed-forward linear threshold networks can be bounded above by $2W \log_2(eN)$. This upper bound is of order $W \ln N$, where W is the total number of weights and thresholds; that is, the total number of variable parameters determining the state of the network. We have already seen that the perceptron on n inputs has VC-dimension $n + 1$, which is exactly the number of variable parameters in this case. We have also seen that for polynomial threshold functions, the VC-dimension is precisely the number of variable parameters. The question therefore arises as to whether the $O(W \ln N)$ bound is of the best possible order or whether in this case, too, the VC-dimension is of order W. In fact, the $\ln N$ factor cannot, in general, be removed, as the following result shows. (Its proof, which is quite technical, is omitted here.)

Theorem 11.9 *Let W be any positive integer greater than 32. Then there is a three-layer feed-forward linear threshold network $\mathcal{N}_W$ with at most W weights and thresholds,*

for which $\mathrm{VCdim}(\mathcal{N}_W, \{0,1\}^n) > (1/132)W\log_2(N/16)$, *where* N *is the number of computation units.*

11.6 VC-dimension of sigmoid networks

When it comes to activation functions other than the simple linear threshold activation function, bounding the VC-dimension becomes rather more complicated. In fact, there exist activation functions such that some very small networks making use of these activation functions have infinite VC-dimension. Fortunately, for the standard sigmoid activation function $\sigma(x) = 1/(1+e^x)$, there are finite, and reasonably small, upper bounds on the VC-dimension. The proof of the following result, which is omitted, is extremely involved: it brings together techniques from logic and algebraic geometry.

Theorem 11.10 *Let* $\mathcal{N}$ *be a sigmoid network, that is, a feed-forward network with linear threshold output unit and the standard sigmoid function* $\sigma(x) = 1/(1+e^{-x})$ *as activation function on the hidden units. Suppose that the total number of adjustable weights and thresholds is* W*, that the number of inputs is* n*, and that there are* N *computation units. Then*

$$\mathrm{VCdim}(\mathcal{N}, \mathbb{R}^n) \leq (WN)^2 + 11WN\log_2(18WN^2).$$

Note that this bound, which is $O(W^4)$, is polynomial in the number of weights and thresholds. It has been shown that the VC-dimension of (unbounded depth) sigmoid nets can be as large as W^2. There is thus, generally, a strict separation between the VC-dimension of threshold networks (with VC-dimension $O(W\ln W)$) and sigmoid networks.

11.7 Additional remarks and bibliographical notes

Relationships between linear dimension and VC-dimension have been explored by Dudley, and Theorem 11.2 is due to him [35]. The proof given here is taken from [5].

The VC-dimension of the perceptron is well-known, and was shown in [20], for example. The bounds on the VC-dimensions of the sets of polynomial threshold functions are from [5].

As noted, Theorem 11.8 on the VC-dimension of threshold networks is a slightly weaker version of a bound due to Baum and Haussler [14], with a simpler proof that makes use, as in [62], of a coarse form of Sauer's inequality. (Baum and Haussler's upper bound is $2W\log_2(eN)$.) The sample length bound obtained for threshold networks by using the growth function directly is as in [14].

The fact that there are $\Omega(W\log_2 N)$ lower bounds on the VC-dimension of threshold networks is a result of Maass [61].

Sontag showed (see [97]) that there is a neural network $\mathcal{N}$ of *infinite* VC-dimension, having two real inputs, two hidden units with sigmoid activation function

$$f(y) = \frac{1}{\pi}\tan^{-1} y + \cos y/(7+7y^2) + 1/2,$$

and a linear threshold output unit. See also [7] for another example. Finiteness of the VC-dimension of sigmoid networks was established by Macintyre and Sontag [64],

using deep results from logic. In view of Sontag's result, this was an important step forward. The explicit polynomial bound, Theorem 11.10, was obtained by Karpinski and MacIntyre [54, 55]. The lower bound of order W^2 on the VC-dimension of some sigmoid networks is due to Koiran and Sontag [58, 57].

Chapter 12

The Complexity of Learning

12.1 Introduction

Thus far, a learning algorithm has been mainly thought of as a function which maps training samples into output functions (or hypotheses). In this chapter, we shall be more specific about the computational effectiveness of learning algorithms. If the process of PAC learning by an algorithm L is to be of practical value, it should be possible to implement the algorithm "quickly." We discuss what should be meant by an efficient PAC learning algorithm, and we highlight an important connection between the existence of efficient PAC learning algorithms and the existence of efficient procedures for producing consistent hypotheses.

12.2 Efficient PAC learning algorithms

12.2.1 Definition of efficient PAC algorithm

In thinking about how the complexity of a learning algorithm for a certain type of neural network "scales" with the size of the network, it is useful to think about *classes* of neural networks. For example, suppose we want to analyze the efficiency of a learning algorithm L for the perceptron. Then, denoting the n-input perceptron by P_n, we think about the performance of L on each network in the *class* $\{P_n\}$. Generally, a neural network class is a set $\{\mathcal{N}_n : n \in \mathbb{N}\}$ of neural networks, $\mathcal{N}_n$ having n inputs. Let us denote by H_n the hypothesis space of $\mathcal{N}_n$, and X_n the set of all possible inputs. (In considering binary-input networks, for example, H_n is defined on $X_n = \{0,1\}^n$.) A learning algorithm L for the class (rather than just for one specific $\mathcal{N}_n$) can be thought of as a mapping from training samples, for all possible $t \in H_n$, for all n, to $\bigcup_{n=1}^{\infty} H_n$, with the (obvious) property that if $\mathbf{s}$ is a training sample for $t \in H_n$, then $L(\mathbf{s}) \in H_n$.

We wish to consider how the running time of a learning algorithm varies with the number n of inputs to the network. For a PAC learning algorithm to be efficient, the time taken to produce a PAC output function should increase polynomially with n. However, there is another factor to consider. Until now, we have regarded the accuracy parameter ϵ as fixed but arbitrary. It is clear that decreasing this parameter makes the learning task more difficult, and therefore the time taken to produce a probably approximately correct output function should be allowed to increase, but not too rapidly, as ϵ decreases. We shall ask that the running time be polynomial in $1/\epsilon$.

A PAC learning algorithm receives as input some random training sample $\mathbf{s}$, of length m, say, and hence of "size" $m(n+1)$. To produce a PAC output function, we should make m sufficiently large. That is, we should have $m \geq m_0(n, \delta, \epsilon)$, where, in a slight modification of the notation used in the definition of a PAC algorithm, $m_0(n, \delta, \epsilon)$ denotes a sufficient sample length for L to learn with accuracy ϵ and confidence δ when the target function is computable by $\mathcal{N}_n$. If L is a polynomial-time algorithm, then the time it takes to produce its output function is polynomial in mn. Taking m to be m_0, the total time taken to produce the required PAC output function is therefore polynomial in $m_0(\delta, \epsilon)n$. This will be polynomial in n and $1/\epsilon$ provided $m_0(\delta, \epsilon)$ is polynomial in n and $1/\epsilon$. We therefore make the following definition.

Definition 12.1 *Suppose that H is a hypothesis space and L is a PAC learning algorithm for H. Then, L is efficient (with respect to accuracy ϵ, example size n and sample length m) if its running time is polynomial in the "size" mn of the training sample and if there is a value of $m_0(\delta, \epsilon)$ sufficient for PAC learning which is polynomial in n and ϵ^{-1}.*

12.2.2 Sufficient conditions for efficient algorithms

Consider the consistent learning algorithm for the (binary-input) perceptron based on linear programming. We have seen, in subsection 9.4.1, that for a perceptron on n inputs, and for a sample of length m, the algorithm finds a consistent hypothesis in time polynomial in mn. Furthermore, since the VC-dimension of the perceptron is $n+1$, we know (by Theorem 10.9) that this algorithm is a PAC algorithm and that there is a sufficient m_0 of order $(n/\epsilon)\ln(1/\epsilon)$. Since this is polynomial in $1/\epsilon$ and n, the algorithm is an efficient PAC learning algorithm. More generally, we have the following easy result.

Theorem 12.2 *Suppose that L is a consistent learning algorithm for a class of neural networks $\{\mathcal{N}_n\}$. If $\mathrm{VCdim}(\mathcal{N}_n)$ is polynomial in n and L is polynomial-time, then L is an efficient PAC learning algorithm.*

When considering network classes with binary-valued inputs, we may use the bound $\mathrm{VCdim}(H_n) \leq \log_2 |H_n|$ to see that $\mathrm{VCdim}(H_n)$ will be polynomial provided $\log_2 |H_n|$ is. So, the condition that $\mathrm{VCdim}(\mathcal{N}_n)$ be polynomial may be replaced in this case by polynomiality of $\log_2 |H_n|$.

12.3 PAC algorithms and randomized consistent algorithms

We have seen that, provided the VC-dimension of H_n is polynomial, a consistent learning algorithm for a neural network class is an efficient PAC learning algorithm. There is a partial converse of this. First, by Theorem 10.14, the VC-dimension of H_n must be polynomial in n if m_0 is to be polynomial in n. Furthermore, the existence of an efficient PAC learning algorithm implies the existence of a *randomized* polynomial-time algorithm for the following "consistency" problem. (Recall that H_n denotes the hypothesis space of $\mathcal{N}_n$, and is defined on X_n.)

$\{\mathcal{N}_n\}$-CONSISTENCY
Instance: $\mathbf{s} \in (X_n \times \{0,1\})^m$.
Question: Is there $h \in H_n$ such that h is consistent with $\mathbf{s}$?

The consistency problem is that of recognizing whether a labeled sample is in fact a training sample for some function t computable by $\mathcal{N}_n$.

For our purposes, a randomized algorithm $\mathcal{A}$ has access to a random number generator and is allowed to use these random numbers as part of its input. The computation carried out by the algorithm is determined by its input and on the particular sequence produced by the random number generator. For this reason, we can speak of the probability that $\mathcal{A}$ has a given outcome on a particular input. By a randomized polynomial-time algorithm for the consistency problem, we mean a randomized algorithm $\mathcal{A}$ which always halts and produces an output ("yes" or "no"), in time polynomial in mn and such that

- if the answer to $\{\mathcal{N}_n\}$-CONSISTENCY on the instance s is "no," then the output of $\mathcal{A}$ is "no;"
- if the answer to $\{\mathcal{N}_n\}$-CONSISTENCY on the instance s is "yes," then with probability at least $1/2$, the output of $\mathcal{A}$ is "yes."

The following result will enable us to link the existence of efficient learning algorithms to the complexity of the consistency problem.

Theorem 12.3 *Suppose $\{\mathcal{N}_n\}$ is a neural network and that there is a PAC learning algorithm for the class which is efficient (with respect to accuracy, example size, and sample length). Then there is a polynomial-time randomized algorithm which solves* $\{\mathcal{N}_n\}$-CONSISTENCY.

Proof. Suppose that s is an instance of the consistency problem and that it contains $r \leq m$ distinct labeled examples. Define a probability distribution P on X_n by $P(x) = 1/r$ if x occurs in s and by $P(x) = 0$ otherwise. We can use a random number generator with output values i in the range 1 to r to select an example from X according to this distribution. Thus the selection of a training sample of any length m' for t, according to the probability distribution P, can be simulated by generating a sequence of m' random numbers in the required range.

Suppose L is an efficient PAC learning algorithm. Then, given δ, ϵ, there is an integer $m_0(n, \delta, \epsilon)$ for which the probability, for a training sample s' of length m_0, that the error of $L(\mathrm{s}')$ is less than ϵ is greater than $1 - \delta$. Suppose we take the confidence and accuracy parameters to be $\delta = 1/2$ and $\epsilon = 1/r$. Then if we run the given algorithm L on a training sample s' of length $m_0(n, 1/2, 1/r)$, drawn randomly according to the distribution P, the probability that the error of the output is less than $1/r$ is greater than $1 - 1/2 = 1/2$. Since there are no examples with probability strictly between 0 and $1/r$, this means that the probability that the output function is consistent with the training sample is greater than $1/2$. Given this, a randomized algorithm $\mathcal{A}$ for $\{\mathcal{N}_n\}$ consistency is as follows.

- Evaluate $m_0 = m_0(n, 1/2, 1/r)$.
- Construct a sample s' of length m_0, according to the specified probability distribution P.
- Compute $L(\mathrm{s}')$.
- Check $L(\mathrm{s}')$ explicitly to determine whether or not it agrees with s.
- If $L(\mathrm{s}')$ does not agree with s, output *no*. If it does, output "yes."

Algorithm $\mathcal{A}$ will always answer "no" if there is no consistent hypothesis. Furthermore, as we noted, because L is PAC, $\mathcal{A}$ will output "yes" with probability at least $1/2$ when there is a consistent hypothesis. Because L is efficient, it is a polynomial-time algorithm, and $m_0(n, 1/2, 1/r)$ can be chosen to be polynomial in n and $r = 1/\epsilon \leq m$. Thus, the running time of $\mathcal{A}$ is polynomial in mn. □

12.4 Learning can be hard

Theorem 12.3 enables us to move from hardness results for the consistency problem to hardness results for PAC learning. The class RP denotes the class of problems which can be solved by polynomial-time randomized algorithms. It is conjectured that no NP-complete problems can be solved by a polynomial-time randomized algorithm. This is the "RP $\neq$ NP" conjecture, and is widely believed to be true. We have the following result.

Theorem 12.4 *Suppose $\{\mathcal{N}_n\}$ is a neural network class and that the $\{\mathcal{N}_n\}$-*CONSISTENCY *problem is NP-hard. Then, unless RP equals NP, there is no efficient PAC learning algorithm for the class.*

We now prove a specific "hardness" result for a relatively simple class of neural networks.

Suppose that $k \geq 3$ is some fixed integer. Let $\mathcal{N}_n^k$ be the two-layer feed-forward threshold network with n inputs and $k+1$ threshold units, k of these in the hidden layer, one of them the single member of the output layer, and with no connections directly from the inputs to the output. In $\mathcal{N}_n^k$, we fix the weights of all connections from hidden units to the output unit to be 1, and the threshold of the output unit is fixed at k. (This means that the output unit computes the "and" of the outputs of the hidden units.)

We shall prove that the consistency problem for $\mathcal{N}^k = \bigcup \mathcal{N}_n^k$ is NP-hard for $k \geq 3$ by reducing the problem to the well-known NP-complete graph coloring problem.

GRAPH k-COLORABILITY
Instance: A graph $G = (V, E)$ on vertex set $V = \{1, 2, \ldots, n\}$.
Question: Is there a proper vertex-coloring of G by k colors?

Given an instance $G = (V, E)$, we construct an instance of $\{\mathcal{N}_n^k\}$-CONSISTENCY $\mathbf{s}(G)$ as follows. For each vertex $i \in V$ we take as a negative example the vector e_i which has 1 in the ith coordinate position, and 0 elsewhere. For each edge $ij \in E$ we take as a positive example the vector $e_i + e_j$. We also take the all-0 vector $\mathbf{0}$ to be a positive example.

Theorem 12.5 *Let G be a graph. There is a function computable by $\mathcal{N}_n^k$ which is consistent with $\mathbf{s}(G)$ if and only if G is k-colorable.*

Proof. First, we introduce some notation. A state ω of $\mathcal{N}_n^k$ is described by the thresholds θ_l for $1 \leq l \leq k$ of the first k computation units and the weights w_{il} on the connections (i, l) linking input i to computation unit l. We shall use the notation $w^{(l)}$ for the n-vector of weights on the arcs terminating at l, so that $w_i^{(l)} = w_{il}$. Suppose h is computable by $\mathcal{N}_n^k$ and is consistent with the training sample. By the construction

of the network, h is a conjunction $h = h_1 \wedge h_2 \wedge \cdots \wedge h_k$ of linear threshold functions. (That is, $h(x) = 1$ if and only if $h_i(x) = 1$ for all i between 1 and k.) Specifically, there are weight-vectors $w^{(1)}, w^{(2)}, \ldots, w^{(k)}$ and thresholds $\theta_1, \theta_2, \ldots, \theta_k$ such that

$$h_l(y) = 1 \iff \langle w^{(l)}, y \rangle \geq \theta_l \quad (1 \leq l \leq k).$$

Note that, since $\mathbf{0}$ is a positive example, we have $0 = \langle w^{(l)}, \mathbf{0} \rangle \geq \theta_l$ for each l between 1 and k. For each vertex i, $h(e_i) = 0$, and so there is at least one function h_f $(1 \leq f \leq k)$ for which $h_f(e_i) = 0$. Thus we may define $\chi : V \to \{1, 2, \ldots, k\}$ by

$$\chi(i) = \min\{f : h_f(e_i) = 0\}.$$

It remains to prove that χ is a coloring of G. Suppose that $\chi(i) = \chi(j) = f$, so that $h_f(e_i) = h_f(e_j) = 0$. Then we have

$$\langle w^{(f)}, e_i \rangle < \theta_f, \quad \langle w^{(f)}, e_j \rangle < \theta_f.$$

Recalling that $\theta_f \leq 0$, this implies $\langle w^{(f)}, e_i + e_j \rangle < \theta_f + \theta_f \leq \theta_f$. It follows that $h_f(e_i + e_j) = 0$ and $h(e_i + e_j) = 0$. Now if ij were an edge of G, then we should have $h(e_i + e_j) = 1$, because we assumed that h is consistent with the training sample. Thus ij is not an edge of G, and χ is a coloring, as claimed.

Conversely, suppose we are given a coloring $\chi : V \to \{1, 2, \ldots, k\}$. For $1 \leq l \leq k$ define the weight-vector $w^{(l)}$ as follows: $w_i^{(l)} = -1$ if $\chi(i) = l$ and $w_i^{(l)} = 1$ otherwise. Define the threshold θ_l to be $-1/2$. Let $h_1, h_2, \ldots, h_k$ be the corresponding linear threshold functions, and let h be their conjunction. We claim that h is consistent with $\mathbf{s}(G)$. Since $0 \geq \theta_l = -1/2$ it follows that $h_l(\mathbf{0}) = 1$ for each l, and so $h(\mathbf{0}) = 1$. In order to evaluate $h(e_i)$, note that if $\chi(i) = f$, then $\langle w^{(f)}, e_i \rangle = w_i^{(f)} = -1 < -1/2$, so $h_f(e_i) = 0$ and $h(e_i) = 0$, as required. Finally, for any color l and edge ij we know that at least one of $\chi(i)$ and $\chi(j)$ is not l. Hence $\langle w^{(l)}, e_i + e_j \rangle = w_i^{(l)} + w_j^{(l)}$, where either both of the terms on the right-hand side are 1 or one is 1 and the other is -1. In any case the sum exceeds the threshold $-1/2$, and $h_l(e_i + e_j) = 1$. Thus $h(e_i + e_j) = 1$. □

The proof that the consistency problem for $\mathcal{N}^k$ is NP-hard for $k \geq 3$ follows directly from this result. We have shown that if there is a polynomial-time algorithm for $\{\mathcal{N}_n^k\}$-CONSISTENCY, then there is one for GRAPH k-COLORABILITY. But GRAPH k-COLORABILITY is NP-complete for $k \geq 3$, and so $\{\mathcal{N}_n^k\}$-CONSISTENCY is NP-hard if $k \geq 3$.

Theorem 12.4 tells us that, since (for any $k \geq 3$) $\{\mathcal{N}_n^k\}$-CONSISTENCY is NP-hard, unless RP equals NP there can be no computationally efficient PAC learning algorithm for the class $\{\mathcal{N}_n^k\}$.

12.5 Additional remarks and bibliographical notes

In his initial papers introducing the PAC model [101, 100], Valiant placed great stress on the efficiency of learning algorithms. The computational complexity of PAC learning was further studied in a number of papers, including [20, 84, 43]. The definition of efficient learning given here is standard, and most other variants of the definition turn out on close analysis to be equivalent; see [43].

Theorem 12.4 is from [84]. (See also [76, 43].) The proof given here is from [8]. For a discussion of randomized algorithms; see, for example, [28].

The fact that computational complexity-theoretic hardness results hold for neural networks was first shown by Judd [52, 53]. The hardness result for $\{\mathcal{N}_n^k\}$ is a slight variant, from [8], of a result due to Blum and Rivest [18, 19]. Blum and Rivest showed the corresponding hardness result for the networks in which the weights into the output, and its threshold, are not fixed. They also showed that the hardness result holds in the case $k = 2$, by relating it to an NP-complete problem known as SET-SPLITTING.

Chapter 13

Boltzmann Machines and Combinatorial Optimization

13.1 Introduction

In this chapter, we briefly touch on a type of "stochastic" neural network known as the *Boltzmann machine*. This is quite different from the previous types of neural networks considered in the book (hence its deferral to this late stage). In particular, a Boltzmann machine is not feed-forward, and it exhibits elements of what might be described as "self-organization." We describe how Boltzmann machines "evolve" and we present some convergence results. We then describe how these results may be applied to obtain heuristics for certain graph-theoretical optimization problems.

13.2 Boltzmann machines

A Boltzmann machine is a type of *stochastic* artificial neural network, in which the underlying directed graph $G = (V, A)$ has symmetric connections: that is, if $(i, j) \in A$, then $(j, i) \in A$. Furthermore, the weights are always constrained so that the weight on the connection joining unit i to unit j equals the weight on the connection joining j to i; that is, $w_{(i,j)} = w_{(j,i)} = w_{ij}$. It is perhaps easier, therefore, to think of a Boltzmann machine M as having the structure of an undirected graph (which may have loops), and having weighted edges. Each computation unit in a Boltzmann machine is, in contrast to the types of computation unit considered up to now, "stochastic," as shall be explained shortly. We define a Boltzmann machine M to be a pair (G, Ω), where $G = (V, E)$ is a graph with, say, n vertices and m edges, and $\Omega \subseteq \mathbb{R}^m \times \{0, 1\}^n$ is a set of allowable *states*. For $\omega = (w_{e_1}, w_{e_2}, \ldots w_{e_m}, S_1, \ldots, S_n) \in \Omega$, the vector $w = (w_{e_1}, w_{e_2}, \ldots w_{e_m})$ describes the weights assigned to the edges $e_1, e_2, \ldots, e_m$ of the graph, and the tail of ω, $(S_1, S_2, \ldots, S_n)$ describes which of the vertices (units) $1, 2, \ldots, n$ are "on": vertex i is "on" in state ω if and only if $S_i = 1$. The *consensus function* of the Boltzmann machine M is the function $C : \Omega \to \mathbb{R}$ given by

$$C(\omega) = \sum_{ij \in E} w_{ij} S_i S_j.$$

Computation proceeds in the machine in a stochastic manner in such a way as to

increase consensus. Thus, if w_{ij} is positive, there is a tendency for units i and j to be either both on or both off, while if the weight is negative, there is a tendency for them to have different activations. When a weight is positive, we refer to it as *excitatory* and, when negative, *inhibitory*.

We now describe how the state of the network evolves when the weights are fixed. This is, of course, a very restricted analysis since, in general, the weights may also change through "learning." However, for the applications we have in mind, the weights will be determined explicitly by an instance of a combinatorial optimization problem.

If i is a vertex and ω a state, $\omega[i \to \bar{i}]$ is the state obtained from ω by changing S_i to $1 - S_i$, that is, by "flipping" the activation of i. If such a flip is made, then the resulting change in the consensus function is

$$\Delta C_\omega(i) = C(\omega[i \to \bar{i}]) - C(\omega).$$

If this is fairly large, it is advantageous to flip the activation of i, and if it is negative, with large absolute value, it is disadvantageous to do so. The decision on whether to flip the activation of unit i is made stochastically, based on $\Delta C_\omega(i)$. The simplest model of computation in a Boltzmann machine is the sequential model. Here, the machine has an internal clock, and on the tth tick of the clock, a vertex i_t is chosen, uniformly at random. The activation of i_t is then flipped, and the state changed to $\omega[i_t \to \bar{i}_t]$, with probability

$$\text{Prob}\,(\omega \to \omega[i_t \to \bar{i}_t]) = \frac{1}{1 + \exp(-\gamma \Delta C_\omega(i_t))}$$

for some constant $\gamma > 0$. A more complicated procedure is that in which the machine's computations are in parallel. Then, at time t, a subset S^t of the vertices is chosen according to some probability distribution on the subsets, and for each $i \in S^t$, a decision is made, as described above, whether to flip the activation of that unit. It should be noted that although this may involve making a large number of such decisions simultaneously, the computations involved in this procedure are local, since $\Delta C_\omega(i)$ depends only on the activations of the units j adjacent to i and on the weights w_{ij}. The result is some state ω' which is obtained from ω by flipping some of the S_i, for $i \in S^t$. It can be proved that the sequential mode of computation results in a stationary distribution on the set of all states, in which the probability that the state of the machine is ω is proportional to $\exp(\gamma C(\omega))$. The same conclusion holds if the computations are *synchronous, with limited parallelism*. This is the special case of parallel computation, in which only independent sets of vertices are generated, and each independent set is generated with equal probability. In both cases, in the stationary distribution, states of high consensus are more likely.

The parameter γ has an interesting interpretation. Its reciprocal $T = 1/\gamma$ is often called *temperature*. There are indications that the process of *simulated annealing*, whereby the temperature is slowly decreased as computation proceeds (so that γ is no longer constant, but increases with time), may be helpful. It turns out that, as $T \to 0$ and as time tends to infinity, the limiting stationary distribution of states is uniform over all states that maximize the consensus function.

A number of combinatorial optimization problems can be realized as the problem of maximizing consensus in Boltzmann machines. Here, we shall describe the general approach and give just two examples.

13.3 Combinatorial optimization

Suppose we have a combinatorial optimization problem. It is then often possible to construct a Boltzmann machine with fixed weights determined by the instance of the problem so that maximizing consensus in the machine is equivalent to solving the optimization problem. A general approach is as follows. First, phrase the optimization problem as a $\{0,1\}$-valued quadratic programming problem. Second, construct a Boltzmann machine with vertices (nodes) corresponding to the variables in the quadratic programming problem. Third, define the edges and weights of the machine in such a way that the following *correspondence conditions* hold:

- local maxima of the resulting consensus function correspond to feasible solutions of the optimization problem;
- if $\mathcal{S}_1, \mathcal{S}_2$ are two feasible solutions of the optimization problem, with $\mathcal{S}_1$ better than $\mathcal{S}_2$, then the consensus of the state corresponding to $\mathcal{S}_1$ is higher than that corresponding to $\mathcal{S}_2$.

Note that we may always take the underlying graph of the Boltzmann machine to be complete, since setting a weight to zero is equivalent to omitting an edge. If the procedure just described can be carried out, then, in theory, one way to attempt to solve the optimization problem is to construct the Boltzmann machine with these properties and let it evolve, perhaps decreasing the temperature parameter γ^{-1} as time progresses. We have already mentioned that such an approach may be promising, since the machine has certain convergence properties. We should remark, however, that even when we know from the theory that the state of the machine will converge to one maximizing consensus, it may not do so feasibly fast.

13.4 Maximum cuts

To illustrate the Boltzmann machine approach, we start with the problem of finding a maximum cut in a weighted graph. Given a graph $G = (V, E)$ of order n having weighted edges, the problem is to determine a partition of the vertex-set into subsets V_1 and V_2 in such a way that the total sum of the weights of the edges that join vertices in the two sets is maximum.

This may easily be phrased as a $\{0,1\}$ quadratic programming problem: introduce a variable x_i for each vertex $i \in V$, and take $x_i = 1$ to mean that $i \in V_2$ and $x_i = 0$ to mean that $i \in V_1$. Then the problem is to maximize the objective function

$$F = \sum_{i=1}^{n} \sum_{j=i+1}^{n} W_{ij} \left((1 - x_i)x_j + (1 - x_j)x_i \right),$$

where the W_{ij} are the weights on the edges of the graph. Our Boltzmann machine will have as its underlying graph the instance graph together with loops on all the vertices. We assign weights as follows: for $ij \in E$, set the weight w_{ij} on this edge in the Boltzmann machine to be $-2W_{ij}$, where W_{ij} is the corresponding weight in the original graph. For $i \in V$, set the weight w_{ii} on the loop to be $\sum_{j=1}^{n} W_{ij}$. The resulting Boltzmann machine then satisfies the correspondence conditions.

13.5 The traveling salesman problem

An instance of the traveling salesman problem is a weighted graph $G = (V, E)$, where the vertices represent cities and the weight d_{ij} on edge ij is the distance from i to j. The aim is to find a Hamiltonian cycle of minimum total length. A number of neural approaches to this problem have been made. We describe here a fairly simple Boltzmann machine implementation. Following the general approach outlined above, we first describe the problem as a quadratic programming problem. Suppose that the vertex-set of the graph is $V = \{1, 2, \ldots, n\}$ and, for i and a between 1 and n, introduce a variable x_{ia}, which, for a particular tour, equals 1 if and only if vertex i is the ath vertex in the tour. For i, j, a, b between 1 and n, let

$$u_{ijab} = \begin{cases} d_{ij} & \text{if } a \equiv b + 1 \,(\text{mod } n), \\ 0 & \text{otherwise.} \end{cases}$$

The traveling salesman problem is then equivalent to minimizing

$$F = \sum_{i,j,a,b} u_{ijab} x_{ia} x_{jb}$$

subject to the constraints

$$\sum_{i=1}^{n} x_{ia} = 1 \;\; (1 \le a \le n) \;\text{ and }\; \sum_{a=1}^{n} x_{ia} = 1 \;\; (1 \le i \le n).$$

We now construct a Boltzmann machine whose consensus function satisfies the properties described at the beginning of this section. It has n^2 vertices, one for each pair (i, a), corresponding to x_{ia}. Define the weights as follows:

$$\begin{aligned} w_{(i,a)(i,a)} &= 1 + \max\{d_{ik} + d_{il} : l \neq k\}, \\ w_{(i,a)(j,b)} &= -d_{ij} \text{ for } i \neq j, \text{ and } a \equiv b + 1 \,(\text{mod } n), \\ w_{(i,a)(i,b)} &= -1 - \min(w_{(i,a)(i,a)}, w_{(i,b)(i,b)}) \text{ for } a \neq b, \\ w_{(i,a)(j,a)} &= -1 - \min(w_{(i,a)(i,a)}, w_{(j,a)(j,a)}) \text{ for } i \neq j. \end{aligned}$$

Then the correspondence conditions are satisfied, and the Boltzmann machine will 'solve' the traveling salesman problem, although not generally in polynomial time.

13.6 Additional remarks and bibliographical notes

This chapter has only just skimmed the surface of a very interesting area. The book by Aarts and Korst [1] proves the convergence results alluded to, and also presents the general approach to solving combinatorial optimization problems using Boltzmann machines. (See also the references contained therein.) This book also contains a wide-ranging discussion of simulated annealing and related topics. The implementation of the traveling salesman problem presented here is based on [38]; see [1]. Bose and Liang [24] discuss Boltzmann machines and other types of neural network, exploring the use of networks in combinatorial optimization.

Bibliography

[1] E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Networks.* Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley, New York, 1989.

[2] I. Anderson. *Combinatorics of Finite Sets.* Oxford University Press, Oxford, UK, 1987.

[3] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.

[4] D. Angluin and L. G. Valiant. Fast probabilistic algorithms for Hamiltonian circuits and matchings. *Journal of Computer and System Sciences*, 18(2):155–193, 1979.

[5] M. Anthony. Classification by polynomial surfaces. *Discrete Applied Mathematics*, 61:91–103, 1995.

[6] M. Anthony. Threshold functions, decision lists, and the representation of Boolean functions. Technical Report NC-TR-96-028, Neurocolt Technical Reports, 1996.

[7] M. Anthony and P. Bartlett. *Neural Network Learning: Theoretical Foundations.* Cambridge University Press, Cambridge, UK, 1999.

[8] M. Anthony and N. Biggs. *Computational Learning Theory.* Cambridge Tracts in Theoretical Computer Science (30). Cambridge University Press, 1992. Cambridge, UK, Reprinted 1997.

[9] M. Anthony, N. Biggs, and J. Shawe-Taylor. The learnability of formal concepts. In *Proceedings of the 3rd Annual Workshop on Computational Learning Theory*, pages 246–257, Morgan Kaufmann, San Mateo, CA, 1990.

[10] M. Anthony, G. Brightwell, and J. Shawe-Taylor. On specifying Boolean functions by labelled examples. *Discrete Applied Mathematics*, 61:1–25, 1995.

[11] M. Anthony and J. Shawe-Taylor. Using the perceptron algorithm to find consistent hypotheses. *Combinatorics, Probability and Computing*, 4(2):385–387, 1993.

[12] J. Aspnes, R. Beigel, M. Furst, and S. Rudich. The expressive power of voting polynomials. In *23rd ACM Symposium on Theory of Computing*, pages 402–409. ACM Press, New York, 1991.

[13] P. Baldi. Neural networks, orientations of the hypercube, and algebraic threshold functions. *IEEE Transactions on Information Theory*, 34(3):523–530, 1988.

[14] E. Baum and D. Haussler. What size net gives valid generalization? *Neural Computation*, 1(1):151–160, 1989.

[15] J. Bentley and R. Sedgewick. Fast algorithms for sorting and searching strings. In *Proceedings of the Annual ACM-SIAM Symposium of Discrete Algorithms*, pages 360–369, 1997.

[16] C. M. Bishop. *Neural networks for Pattern Recognition.* Oxford University Press, Oxford, UK, 1995.

[17] A. Blake. *Canonical Expressions in Boolean algebra.* Ph.D. thesis, University of Chicago, 1937.

[18] A. Blum and R. L. Rivest. Training a 3-node neural network is NP-complete. In *Proceedings of the First ACM Workshop on Computational Learning Theory*, pages 9–18, Cambridge, MA, 1988.

[19] A. Blum and R. L. Rivest. Training a 3-node neural network is NP-complete. *Neural Networks*, 5(1):117–127, 1992.

[20] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(4):929–965, 1989.

[21] V. Bohossian and J. Bruck. Algebraic techniques for constructing minimal-weight threshold function. Technical Report Paradise (Parallel and Distributed Computing Group) ETR 015, California Institute of Technology, 1996.

[22] B. Bollobás. *Combinatorics: Set systems, Hypergraphs, Families of Vectors, and Combinatorial Probability.* Cambridge University Press, Cambridge, UK, 1986.

[23] A. Bondy. Induced subsets. *Journal of Combinatorial Theory (B)*, 12:201–202, 1972.

[24] N. K. Bose and P. Liang. *Neural Network Fundamentals, with Graphs, Algorithms, and Applications.* Electrical Engineering Series. McGraw-Hill, New York, 1996.

[25] J. Bruck and R. Smolensky. Polynomial threshold functions, AC^0 functions, and spectral norms. *SIAM Journal on Computing*, 21(1):33–42, 1992.

[26] S. Chari, P. Rohatgi, and A. Srinivasan. Improved algorithms via approximations of probability distributions (extended abstract). In *Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of Computing*, pages 584–592. ACM Press, New York, 1994.

[27] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23:493–509, 1952.

[28] T. Cormen, C. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1990.

[29] T. M. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, EC-14:326–334, 1965.

[30] T. M. Cover. Capacity problems for linear machines. In L. Kanal, editor, *Pattern Recognition*, pages 283–289. Thompson Book Co., Washington, 1968.

[31] Y. Crama. Dualization of regular Boolean functions. *Discrete Applied Mathematics*, 16:79–85, 1987.

[32] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods.* Cambridge University Press, Cambridge, UK, 2000.

[33] B. DasGupta and G. Schnitger. The power of approximating: a comparison of activation functions. In S. Hanson, J. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems* 5, pages 615–622. Morgan Kaufmann, San Mateo, CA, 1993.

[34] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* Wiley, 1973.

[35] R. M. Dudley. Central limit theorems for empirical measures. *Annals of Probability*, 6(6):899–929, 1978.

[36] A. Ehrenfeucht, D. Haussler, M. Kearns, and L. G. Valiant. A general lower bound on the number of examples needed for learning. *Information and Computation*, 82:247–261, 1989. First appeared in Proceedings of 1st Annual Workshop on Computational Learning Theory, 1988.

[37] O. Ekin, P. L. Hammer, and A. Kogan. On connected boolean functions. Technical Report RRR 31-96, RUTCOR, Rutgers University, 1996.

[38] R. S. Garfinkel. Motivation and modeling. In E. H. Lawler, editor, *Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, pages 17–36. Wiley, New York, 1985.

[39] S. A. Goldman and M. J. Kearns. On the complexity of teaching. *Journal of Computer and System Sciences*, 50(1):20–31, 1995. An earlier version appeared in the 4th Workshop on Computational Learning Theory, COLT'91.

[40] S. A. Goldman and H. D. Mathias. Teaching a smarter learner. *Journal of Computer and System Sciences*, 52(2):255–267, 1996. An earlier version appeared in the 6th Workshop on Computational Learning Theory, COLT'93.

[41] J. Håstad. On the size of weights for threshold gates. *SIAM Journal on Discrete Mathematics*, 7(3):484–492, 1994.

[42] D. Haussler. Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Information and Computation*, 100(1):78–150, Sept. 1992.

[43] D. Haussler, M. J. Kearns, N. Littlestone, and M. K. Warmuth. Equivalence of models for polynomial learnability. *Information and Computation*, 95(2):129–161, Dec. 1991.

[44] D. Hebb. *The Organization of Behavior.* Wiley, New York, 1949.

[45] T. Hegedűs. Combinatorial results on the complexity of teaching and learning. In *Proceedings of the 19th International Symposium on Mathematical Foundations of Computer Science*, pages 393–402. Springer-Verlag, New York, 1994.

[46] T. Hegedűs and N. Megiddo. On the geometric separability of boolean functions. *Discrete Applied Mathematics*, 66:205–218, 1996.

[47] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation.* Addison-Wesley, Redwood City, CA, 1991.

[48] S.-T. Hu. *Threshold Logic.* University of California Press, Berkeley, CA, 1965.

[49] R. Impagliazzo, R. Paturi, and M. Saks. Size-depth tradeoffs for threshold circuits. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 541–550. ACM Press, New York, 1993.

[50] J. Jackson and A. Tomkins. A computational model of teaching. In *Proceedings of the 5th Annual Workshop on Computer Learning Theory*, pages 319–326. ACM Press, New York, 1992.

[51] R. G. Jeroslow. On defining sets of vertices of the hypercube by linear inequalities. *Discrete Mathematics*, 11:119–124, 1975.

[52] J. S. Judd. Learning in neural networks. In *Proceedings of the 1st Annual Workshop on Computer Learning Theory*, pages 2–8, Morgan Kaufmann, San Mateo, CA, 1988.

[53] J. S. Judd. *Neural Network Design and the Complexity of Learning.* MIT Press, Cambridge, MA, 1990.

[54] M. Karpinski and A. J. Macintyre. Polynomial bounds for VC dimension of sigmoidal neural networks. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, pages 200–208. ACM Press, New York, 1995.

[55] M. Karpinski and A. J. Macintyre. Polynomial bounds for VC dimension of sigmoidal and general Pfaffian neural networks. *Journal of Computer and System Sciences*, 54:169–176, 1997.

[56] M. J. Kearns and U. Vazirani. *Introduction to Computational Learning Theory.* MIT Press, Cambridge, MA, 1995.

[57] P. Koiran and E. D. Sontag. Neural networks with quadratic VC dimension. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 197–203. MIT Press, Cambridge, MA, 1996.

[58] P. Koiran and E. D. Sontag. Neural networks with quadratic VC dimension. *Journal of Computer and System Sciences*, 54(1):190–198, Feb. 1997.

[59] E. Kushilevitz, N. Linial, Y. Rabinovich, and M. Saks. Witness sets for families of binary vectors. *Journal of Combinatorial Theory (A)*, 73(2):376–380, 1996.

[60] N. Littlestone. Learning when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[61] W. Maass. Bounds for the computational power and learning complexity of analog neural nets. In *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing*, pages 335–344. ACM Press, New York, 1993.

[62] W. Maass. On the complexity of learning on feedforward neural nets. Manuscript, Institute for Theoretical Computer Science, Technische Universitaet Graz, 1993.

[63] W. Maass, G. Schnitger, and E. Sontag. On the computational power of sigmoid versus boolean threshold circuits. In *Proceedings of the 32nd Annual ACM Symposium on Foundations of Computer Science*, pages 767–776, 1991.

[64] A. Macintyre and E. D. Sontag. Finiteness results for sigmoid. In *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing*, pages 325–334. ACM Press, New York, 1993.

[65] K. Makino. A linear time algorithm for recognizing regular Boolean functions. Technical Report RRR 15-99, RUTCOR, Rutgers University, 1999. (Revision of RRR 32-98, 1998.)

[66] M. Marchand and M. Golea. On learning simple neural concepts: From halfspace intersections to neural decision lists. *Network: Computation in Neural Systems*, 4:67–85, 1993.

[67] M. Marchand, M. Golea, and P. Ruján. A convergence theorem for sequential learning in two-layer perceptrons. *Europhysics Letters*, 11(6):487–492, 1990.

[68] J. Matousek. Tight upper bounds for the discrepancy of half-spaces. *Discrete & Computational Geometry*, 13:593–601, 1995.

[69] E. Mayoraz. On the power of democratic networks. Technical Report RRR 4-95, RUTCOR, Rutgers University, 1995.

[70] C. H. Mays. Adaptive threshold logic. Technical Report 1557-1, Stanford Electronics Laboratories, Stanford University, Stanford, CA, 1963.

[71] C. McDiarmid. On the method of bounded differences. In J. Siemons, editor, *Surveys in Combinatorics*, 1989, London Mathematical Society Lecture Note Series (141). Cambridge University Press, Cambridge, UK, 1989.

[72] M. Minsky and S. Papert. *Perceptrons*. MIT Press, Cambridge, MA, 1969.

[73] J. Moon. Four combinatorial problems. In D. J. A. Welsh, editor, *Combinatorial Mathematics and Its Applications*, pages 185–190. Academic Press, London, 1971.

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

[75] S. Muroga. *Threshold Logic and its Applications*. Wiley, New York, 1971.

[76] B. K. Natarajan. On learning sets and functions. *Machine Learning*, 4(1):67–97, 1989.

[77] E. I. Nechiporuk. The synthesis of networks from threshold elements. *Problemy Kibernetiki*, 11:49–62, 1964.

[78] N. J. Nilsson. *Learning Machines.* McGraw-Hill, New York, 1965.

[79] A. B. Novikoff. On convergence proofs on perceptrons. In *Symposium on the Mathematical Theory of Automata*, volume 12, pages 615–622. Polytechnic Institute of Brooklyn, Brooklyn, NY, 1962.

[80] P. O'Neil. Hyperplane cuts of an n-cube. *Discrete Mathematics*, 1:193–195, 1971.

[81] I. Parberry. *Circuit Complexity and Neural Networks.* Foundations of Computing Series. MIT Press, Cambridge, MA, 1994.

[82] R. Paturi and M. Saks. On threshold circuits for parity. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, pages 397–404. IEEE Press, Washington DC, 1990.

[83] U. Peled and B. Simeone. Polyomial-time algorithms for regular set-covering and threshold synthesis. *Discrete Applied Mathematics*, 12:57–69, 1985.

[84] L. Pitt and L. Valiant. Computational limitations on learning from examples. *Journal of the ACM*, 35:965–984, 1988.

[85] D. Pollard. *Convergence of Stochastic Processes.* Springer-Verlag, New York, 1984.

[86] W. Quine. A way to simplify truth functions. *American Mathematical Monthly*, 62:627–631, 1955.

[87] B. D. Ripley. *Pattern Recognition and Neural Networks.* Cambridge University Press, Cambridge, UK, 1996.

[88] R. L. Rivest. Learning decision lists. *Machine Learning*, 2:229–246, 1987.

[89] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychology Review*, 65:386–407, 1958. (Reprinted in *Neurocomputing*, MIT Press, Cambridge, MA, 1988.)

[90] M. Saks. Slicing the hypercube. In *Surveys in Combinatorics (invited talks of the 1993 British Combinatorial Conference)*, pages 211–255. Cambridge University Press, Cambridge, UK, 1993.

[91] N. Sauer. On the density of families of sets. *Journal of Combinatorial Theory, Series A*, 13:145–147, 1972.

[92] L. Schläfli. *Gesammelte Mathematische Abhandlungen* I. Birkhäuser, Basel, 1950.

[93] S. Shelah. A combinatorial problem: Stability and order for models and theories in infinity languages. *Pacific Journal of Mathematics*, 41:247–261, 1972.

[94] A. Shinohara and S. Miyano. Teachability in computational learning. *New Generation Computing*, 8:337–347, 1991.

[95] K.-Y. Siu, V. Roychowdhury, and T. Kailath. *Discrete Neural Computation: A Theoretical Foundation.* Prentice Hall Information and System Sciences Series. Prentice Hall, Englewood Cliffs, NJ, 1995.

[96] K.-Y. Siu, V. P. Roychowdhury, and T. Kailath. Computing with almost optimal threshold circuits. In *IEEE International Symposium on Information Theory,* Budapest, Hungary, 1991.

[97] E. D. Sontag. Feedforward nets for interpolation and classification. *Journal of Computer and System Sciences*, 45:20–48, 1992.

[98] J. M. Steele. Existence of submatrices with all possible columns. *Journal of Combinatorial Theory, Series A*, 24:84–88, 1978.

[99] A. D. Taylor and W. S. Zwicker. *Simple Games: Desirability Relations, Trading, Pseudoweightings.* Princeton University Press, Princeton, NJ, 1999.

[100] L. G. Valiant. Deductive learning. *Philosophical Transactions of the Royal Society of London A*, 312:441–446, 1984.

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

[102] V. N. Vapnik. *Estimation of Dependences Based on Empirical Data.* Springer-Verlag, New York, 1982.

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

[104] C. Wang and A. Williams. The threshold order of a Boolean function. *Discrete Applied Mathematics*, 31:51–69, 1991.

[105] R. O. Winder. *Threshold Logic.* Ph.D. thesis, Princeton University, Princeton, NJ, 1962.

[106] Y. A. Zuev. Asymptotics of the logarithm of the number of threshold, functions of the algebra of logic. *Soviet Mathematics Doklady*, 39:512–513, 1989.

This page intentionally left blank

Index